

Solution exo1 TD1

La droite de la table de vérité est la somme binaire des 3 entrées :

r	b	a	rn0	s
e(2)	e(1)	e(0)	s(1)	s(0)
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

```
ENTITY adder IS PORT(  
    e : in BIT_VECTOR(2 DOWNTO 0); -- 3 entrées  
    s : out BIT_VECTOR(1 DOWNTO 0));  
    -- 2 sorties  
END demo;  
ARCHITECTURE aAdder of adder IS BEGIN  
    With e select  
        s <= "00" WHEN "000",  
            "01" WHEN "001",  
            "01" WHEN "010",  
            "10" WHEN "011",  
            "01" WHEN "100",  
            "10" WHEN "101",  
            "10" WHEN "110",  
            "11" WHEN OTHERS;  
END aAdder;
```

Solution exo2 TD1

```
library ieee;  
use ieee.std_logic_1164.all;  
ENTITY adder IS PORT(  
    e : in std_logic_VECTOR(2 DOWNTO 0); -- 3 entrées  
    s : out std_logic_VECTOR(1 DOWNTO 0)); -- 2 sorties  
END demo;  
ARCHITECTURE aAdder of adder IS BEGIN  
    With e select  
        s <= "00" WHEN "000",  
            "01" WHEN "001",  
            "01" WHEN "010",  
            "10" WHEN "011",  
            "01" WHEN "100",  
            "10" WHEN "101",  
            "10" WHEN "110",  
            "11" WHEN OTHERS;  
END aAdder;
```

Solution exo3 TD1

Utiliser la symétrie du code Gray pour le retrouver sur 3 bits

```
1 -- compteur Gray  
2 ENTITY cmptGray IS PORT (  
3 clock: IN BIT;  
4 q : OUT BIT_VECTOR(2 DOWNTO 0)); -- conforme aux conseils Xilinx  
5 END cmptGray;  
6 ARCHITECTURE mydemo OF cmptGray IS  
7 SIGNAL s_q : BIT_VECTOR(2 DOWNTO 0);  
8 BEGIN  
9 PROCESS(clock) BEGIN  
10 IF clock'EVENT AND clock='1' THEN  
11 CASE s_q IS --style case when  
12 WHEN "000" => s_q <= "001";  
13 WHEN "001" => s_q <= "011";  
14 WHEN "011" => s_q <= "010";
```

```

15         WHEN "010" => s_q <="110";
16         WHEN "110" => s_q <="111";
17         WHEN "111" => s_q <="101";
18         WHEN "101" => s_q <="100";
19         WHEN OTHERS => s_q <="000";
20     END CASE;
21 END IF;
22 END PROCESS;
23 q <= s_q;
24 END mydemo;

```

Solution exo1 TD2

Il possède 2^n états. Pour $n=16$ cela fait 65536 états donc autant de lignes dans votre compteur.

Solution exo2 TD2

Il y a une toute petite subtilité dans cet exercice car le poids 0 divise déjà par 2. Donc le poids 1 par 4 et le poids n par 2^{n+1} . Comme on veut une division par $32768=2^{15}$, il nous faut un poids de 14, d'où le programme ci-dessous :

```

1  -- compteur Gray
2  entity lent is
3      port(horloge : in std_logic;
4           h_lente : out std_logic);
5  end lent;
6
7  architecture alent of lent is
8      signal compteur : std_logic_vector(14 downto 0);
9  begin
10     --division de l'horloge par 32768
11     process(horloge) begin
12         if(horloge'event and horloge='1') then
13             compteur <= compteur + 1;
14         end if;
15     end process;
16     h_lente <= compteur(14);
17 end alent;

```

Solution exo3 TD2

Voici la solution asynchrone :

```

1  PROCESS(clk,raz,set) BEGIN
2      IF raz='1' THEN
3          q<=(OTHERS=>'0');
4      ELSIF set='1' THEN
5          q<=(OTHERS=>'1');
6      ELSIF clk'event and clk='1' THEN
7          q<=std_logic_vector(unsigned(q)+1);
8      END IF;
9  END PROCESS;

```

et le compteur qui compte jusqu'à 24 (compris) sur 5 bits donc :

```

1  PROCESS(clk,raz,set) BEGIN
2      IF raz='1' THEN
3          q<=(OTHERS=>'0');
4      ELSIF set='1' THEN
5          q<=(OTHERS=>'1');
6      ELSIF clk'event and clk='1' THEN
7          IF unsigned(q) <25 then
8              q<=std_logic_vector(unsigned(q)+1);
9          ELSE
10             q<=(OTHERS=>'0');
11         END IF;

```

```

12     END IF;
13     END PROCESS;

```

Solution exo4 VGA TD2

1°) $f_h = 25 \text{ Mhz} \Rightarrow T_h = 40 \text{ ns}$

2°) Nb compteur pour 25,6 us : $25,6 \text{ us} / 40 \text{ ns} = 640$

3°) Nb compteur pour 0,64 us : $0,64 \text{ us} / 40 \text{ ns} = 16$

Période 31,75 us / 40 ns = 793,75 arrondi à 800 donc XXX=799

Durée de la valeur 0 3,8 us soit 95

Donc en horizontal :

0 <----->639 <--->655 (639+16) <-->750(655+95)<--->799

donc ZZZ=654 et TTT=751 (car comparaisons strictes)

4°) Période verticale : $T_v = 40 \text{ ns} \times 800 = 32 \text{ us}$

5°) Le signal doit être périodique à 16,6 ms soit un compteur 518,75 arrondi à 520 donc YYY=519

480 lignes durée $480 \times 32 \text{ us} = 15,36 \text{ ms}$ (au lieu de 15,24 ms)

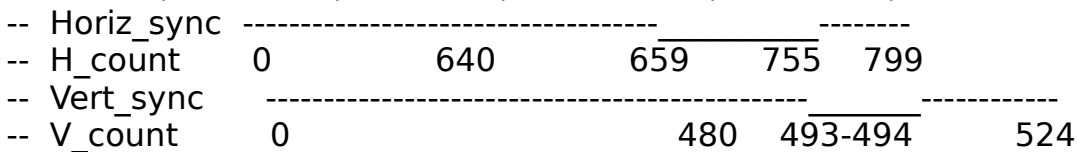
6°) Le temps de 0,35 ms est réalisé par comptage $0,35\text{ms}/32 \text{ us} = 11$

7°) Durée de la valeur 0 pendant 64 us est réalisée avec comptage de 2

0 <----->479 <----->490 (479+11) <----->492 (490+2)<----->519

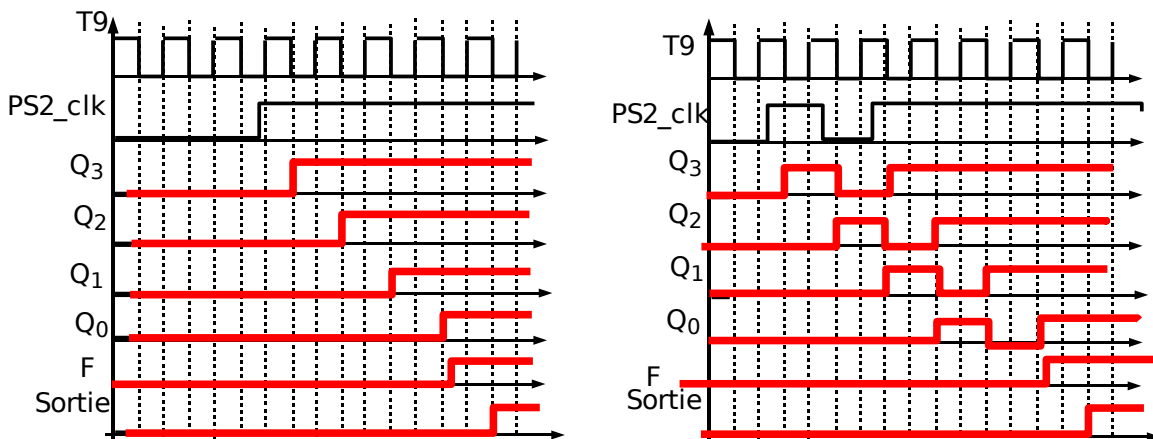
donc UUU=489 et VVV=493 (car comparaisons strictes).

Pour votre information, les valeurs typiques que j'utilise en projet sont :
 XXX=799, ZZZ=658, TTT=756, et YYY=524, UUU=492, VVV=495



Solution exo5 TD2

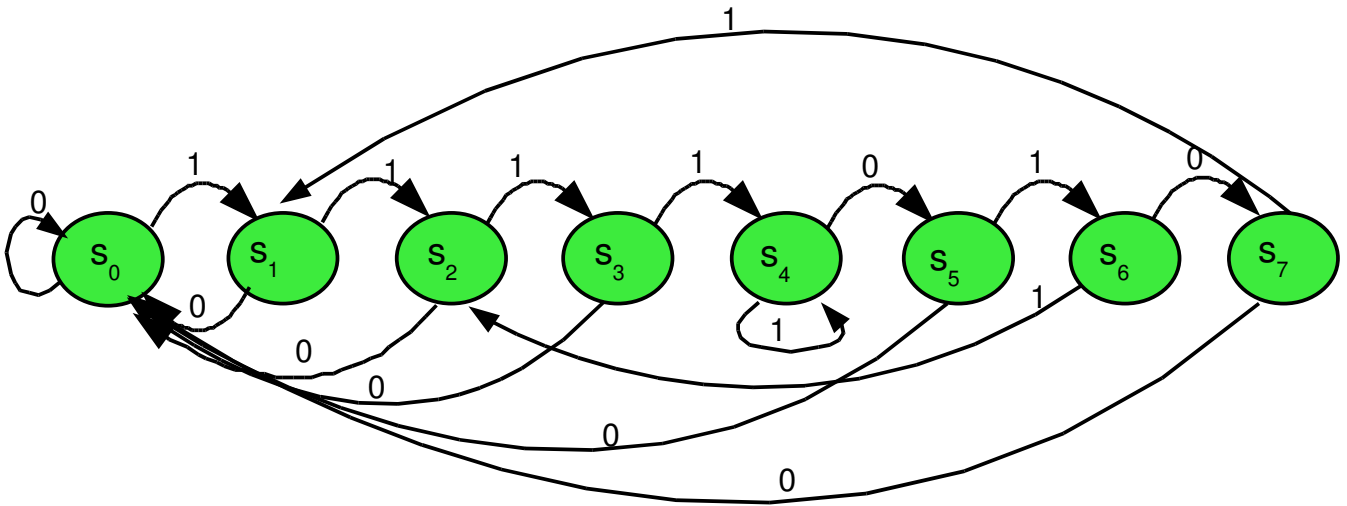
Cette correction correspond au schéma original avec registre à décalage sur front descendant.



Remarquez le décalage des signaux, caractéristique des registres à décalage.

Solution exo1 TD3

1°)



Certains aspects

de cet automate demandent réflexion.

2°) Les méthodes d'optimisation du codage d'états ne sont pas enseignées (Huffman). Mais il existe une méthode heuristique pour coder les états qui consiste à mettre le plus de zéros possible sur les états qui ont beaucoup de flèches de transitions qui s'y terminent. On peut constater qu'un codage en binaire naturel à partir de l'état de gauche sur la figure ci-dessus convient très bien (c'est un pur hasard). L'état s0 de gauche est codé "000" et cela tombe bien c'est celui qui a plus de flèches qui y arrivent (6 en tout)

Solution exo2 TD3

1°)

État présent s1 s0	Condition e0 e1 Init	État futur s1 ⁺ s0 ⁺
0 0	0 X 0	0 0
0 0	1 X 0	0 1
0 1	X 0 0	0 1
0 1	X 1 0	1 0
1 0	0 X 0	1 0
1 0	1 X 0	0 0
1 1	X X X	0 0

$$2^{\circ}) \quad s_0^{futur} = \bar{s} \cdot \bar{s}_0 \cdot e_0 \cdot \overline{Init} + \bar{s} \cdot s_0 \cdot e_1 \cdot \overline{Init}$$

$$s_1^{futur} = s \cdot \bar{s}_0 \cdot \bar{e}_0 \cdot \overline{Init} + \bar{s} \cdot s_0 \cdot e_1 \cdot \overline{Init}$$

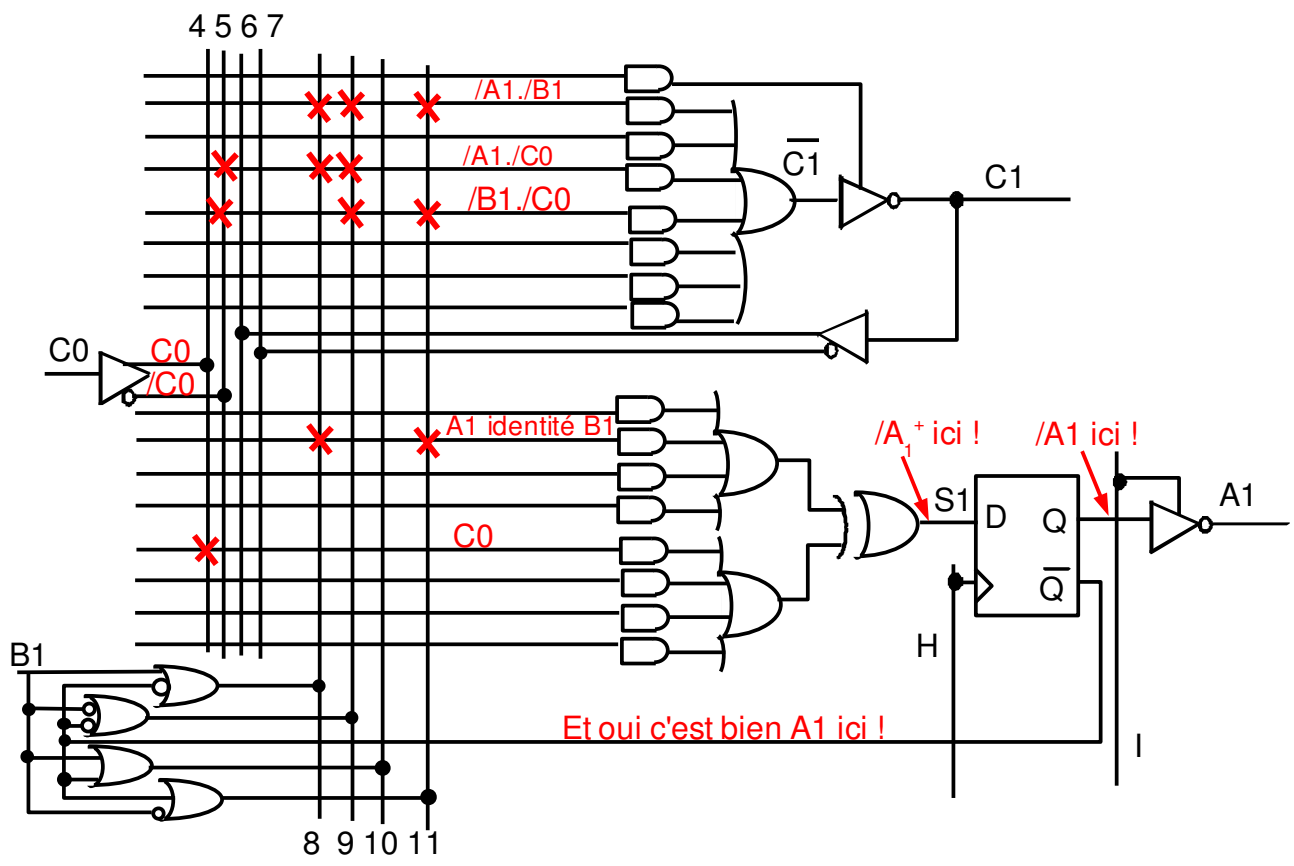
3°) la sortie est s0 => pas d'équation de sortie (ou s=s0 donc 1 fil sans porte).

Solution exo1 TD4

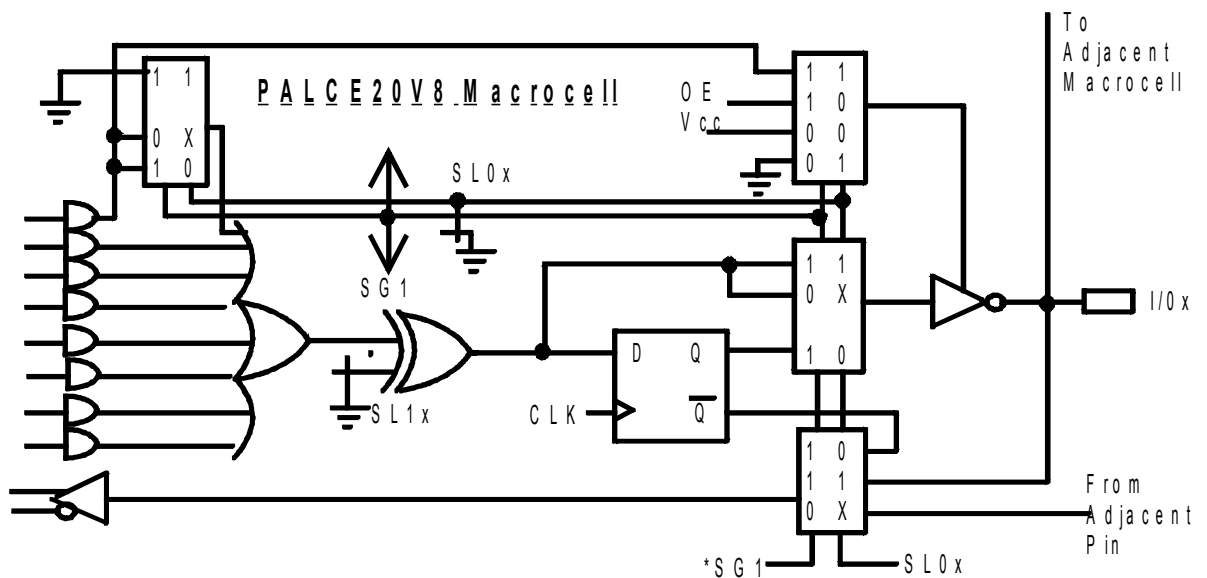
Le schéma donne Q⁺ = D = Toggle . /Q

Mais le schéma de la 16R8 nous montre immédiatement que ce n'est pas Q⁺ qu'il nous faut réaliser mais /Q⁺ à cause de l'inverseur présent sur la sortie.

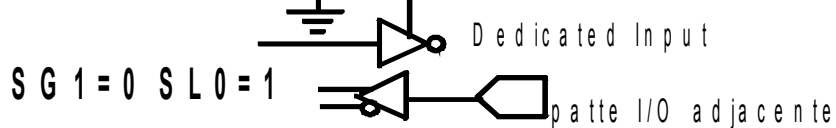
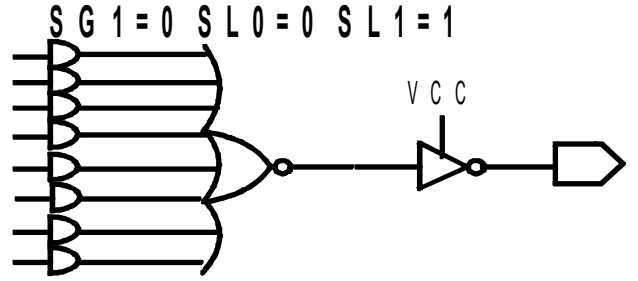
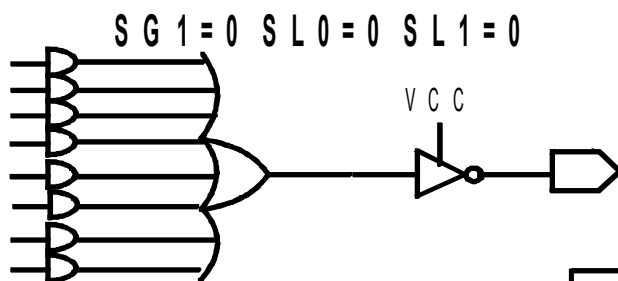
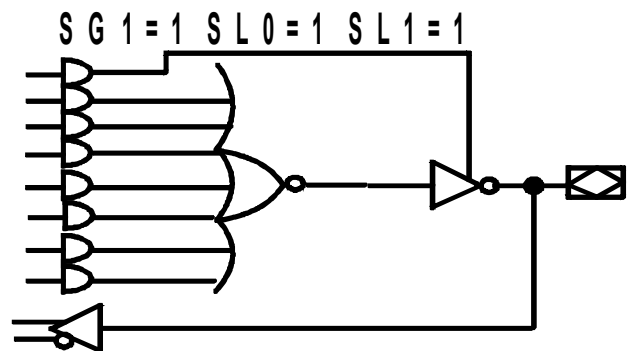
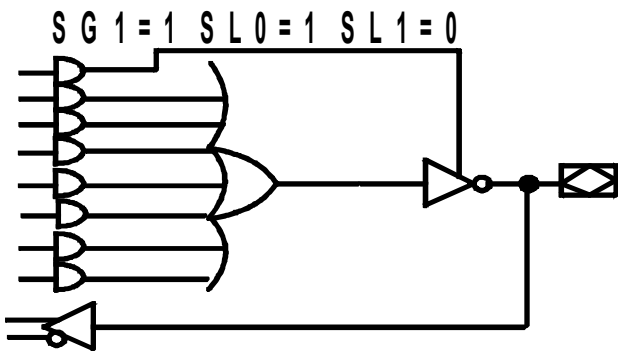
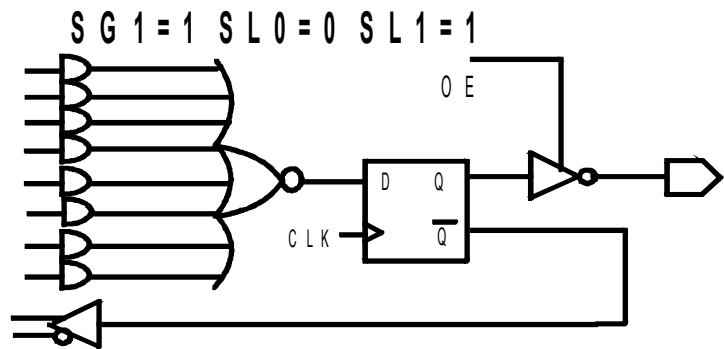
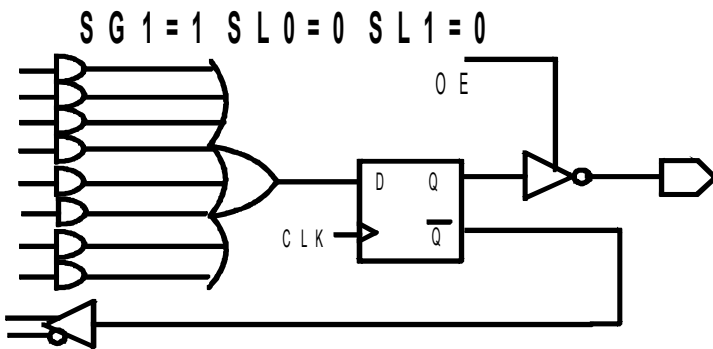
4°)



Solution exo 3 TD4



*In macrocell MC0 and MC7, SG1 is replaced by /SG0 on the feedback multiplexer.



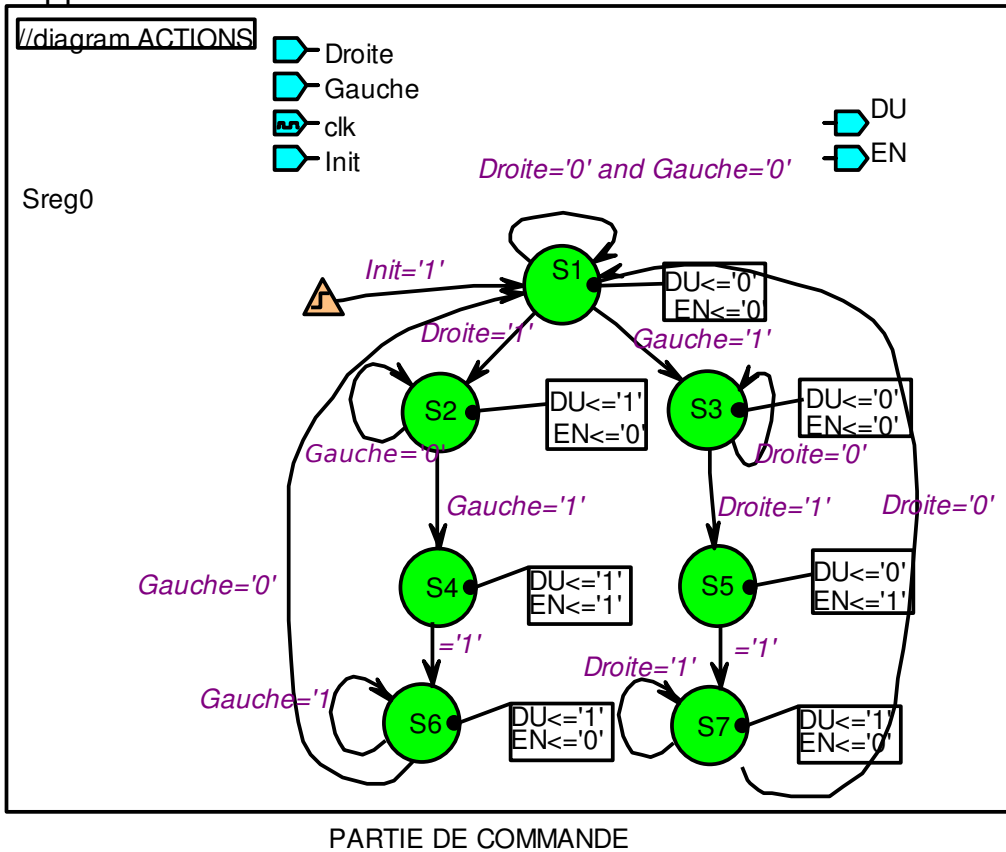
Solution exo 4 TD4

1°) C'est S0 qui détermine la polarité : quand il passe de 0 à 1 on

complémente tout.

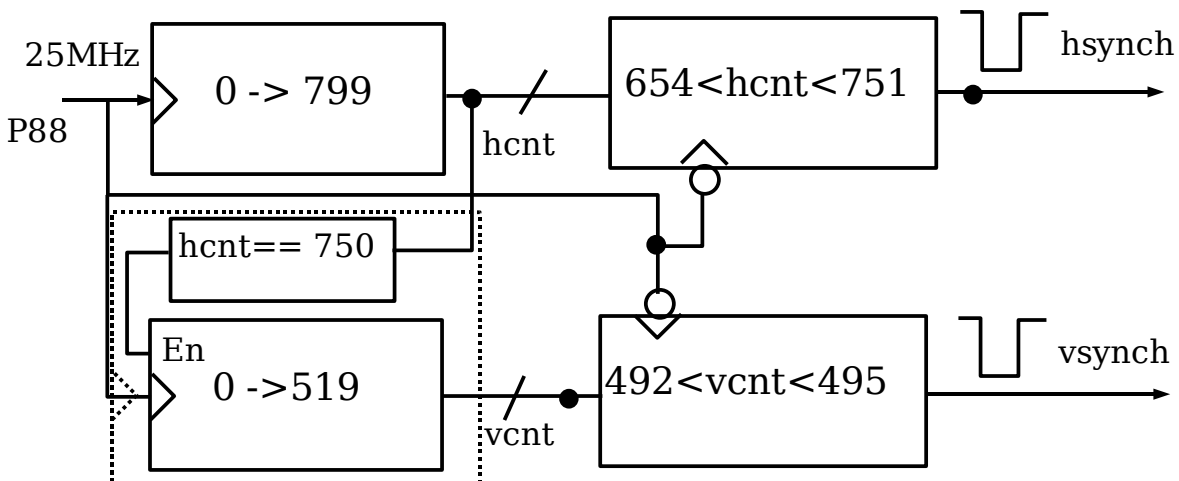
Solution exo 1 TD5

Il faut changer le chemin de données (pas présenté ici) en utilisant un compteur qui possède une entrée de validation En. On ne peut plus rester dans S4 et S5 plus d'une période d'horloge, autrement le chemin de données comptera ou décomptera sans arrêt ! D'où l'ajout de deux états supplémentaires.

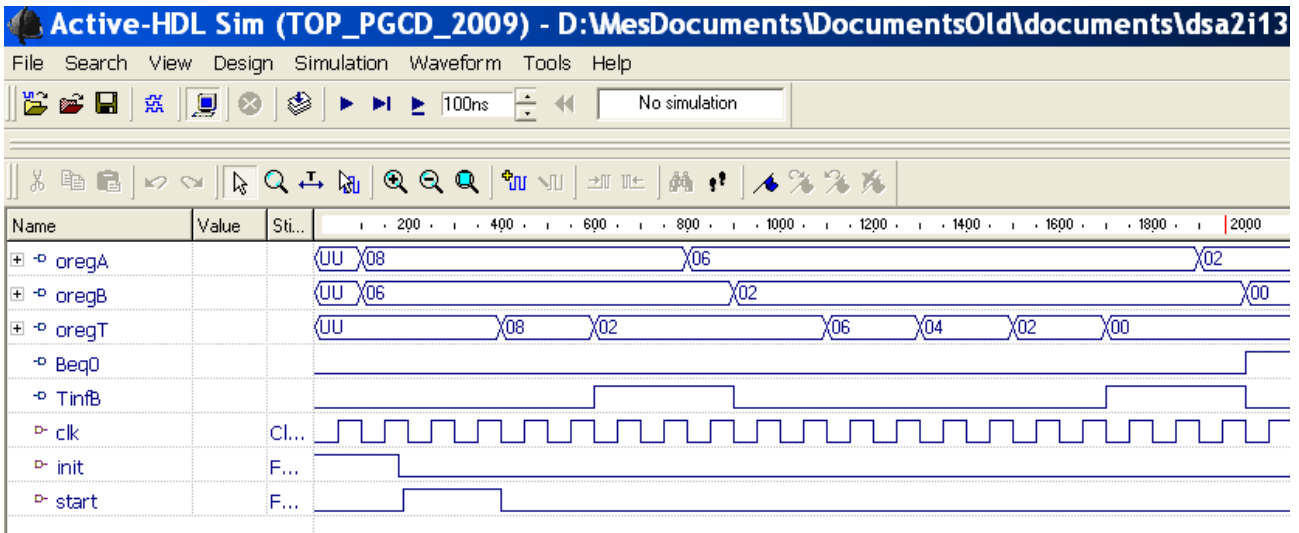


Solution exo 2 TD5

Le deuxième compteur est présenté ci-dessous en pointillés. Remarquez qu'il fonctionne maintenant avec la même horloge que le premier compteur.



Solution exo 3 PGCD TD5



Solution exo4 TD5

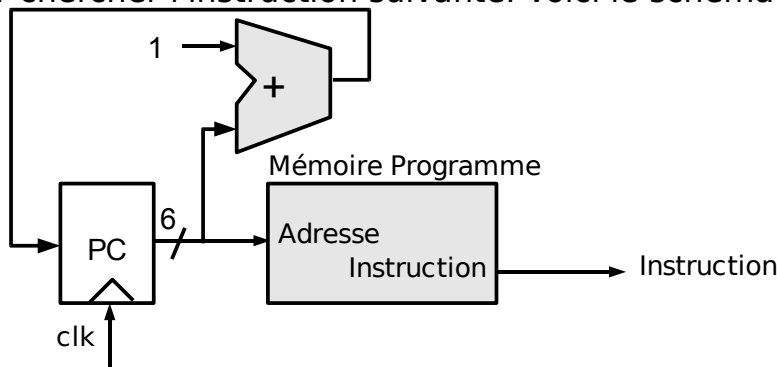
I) Réalisation en mono cycle

Dans une réalisation mono cycle un processeur doit accéder aux instructions et aux données indépendamment et chaque instruction se réalise en un cycle d'horloge.

Si l'on veut rapidement réaliser une architecture programmable mono cycle, il faut d'abord se pencher sur le chemin de données. Concevoir le chemin de données est un processus incrémental. On commence toujours par le séquençement des instructions (recherche des instructions en mémoire), puis à chaque étape nous examinons une classe d'instructions et nous essayons de construire une portion du chemin de donnée qui peut exécuter cette classe d'instructions.

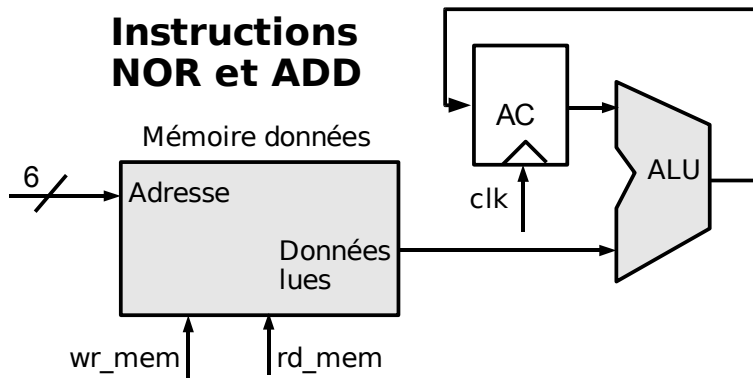
I-1) Séquençement du programme

Le déroulement d'un programme se fait par recherche de l'instruction "Instruction Fetch" et incrémentation du compteur programme pour pouvoir aller chercher l'instruction suivante. Voici le schéma correspondant :



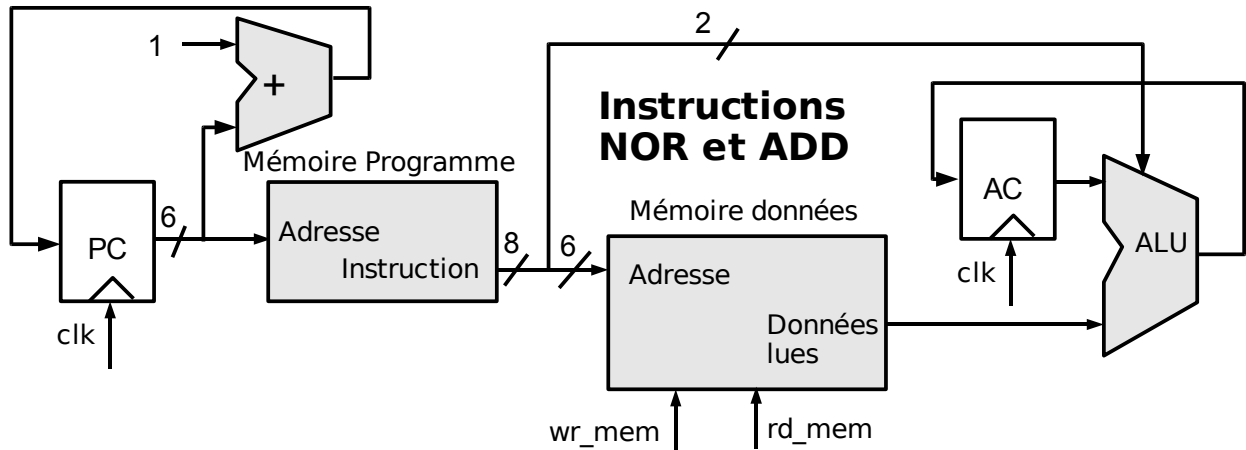
I-2) Instructions NOR et ADD

Ces deux instructions sont dans le même groupe car elles vont chercher toutes les deux une donnée en mémoire. Voici le schéma correspondant :

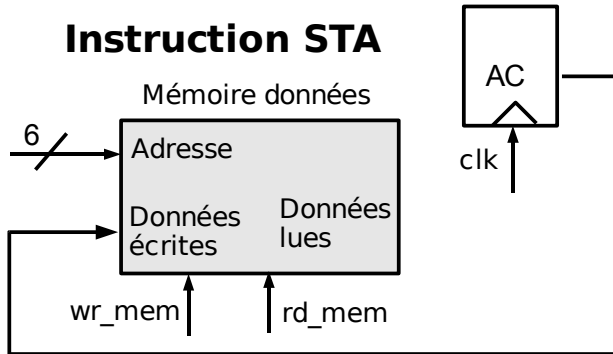


I-3) Séquencement de NOR et ADD

Cela consiste tout simplement à associer les deux schémas précédents :



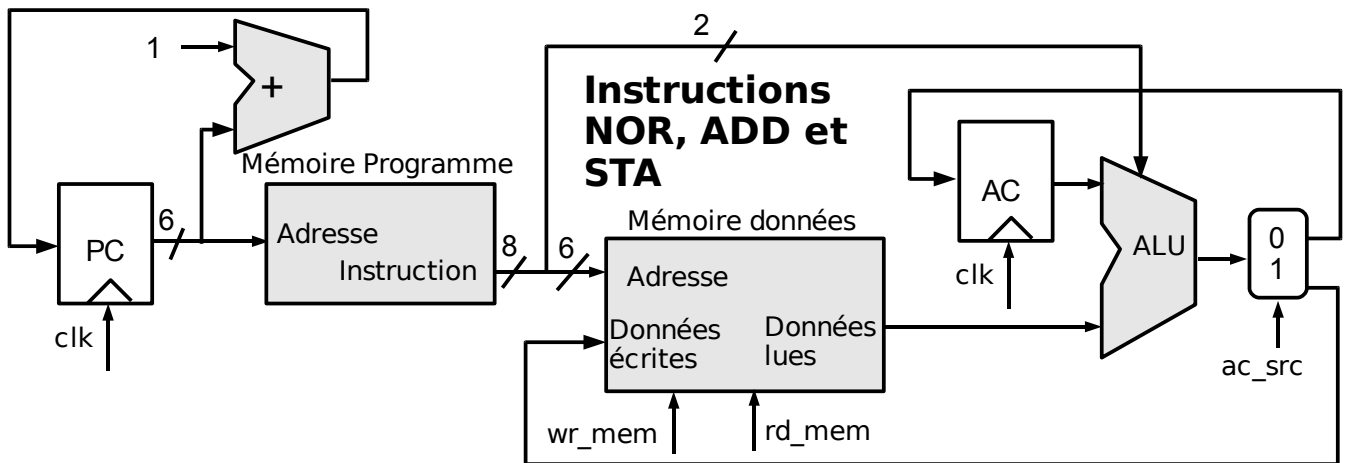
I-4) Instruction STA ou écriture dans la mémoire données



Évidemment l'adresse provient de l'instruction.

I-5) Séquencement des 3 instructions

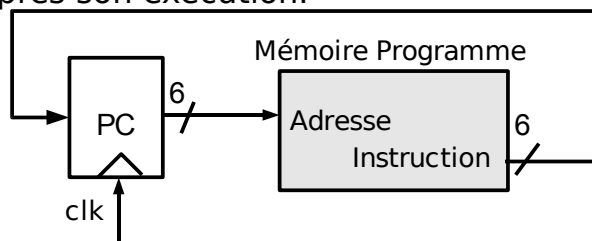
On assemble les deux derniers schémas :



Un soin tout particulier devra être pris pour l'ensemble AC + ALU + DMUX si l'on veut garder la valeur dans AC (front d'horloge inévitable).

I-6) Instruction de contrôle et son chemin de donnée

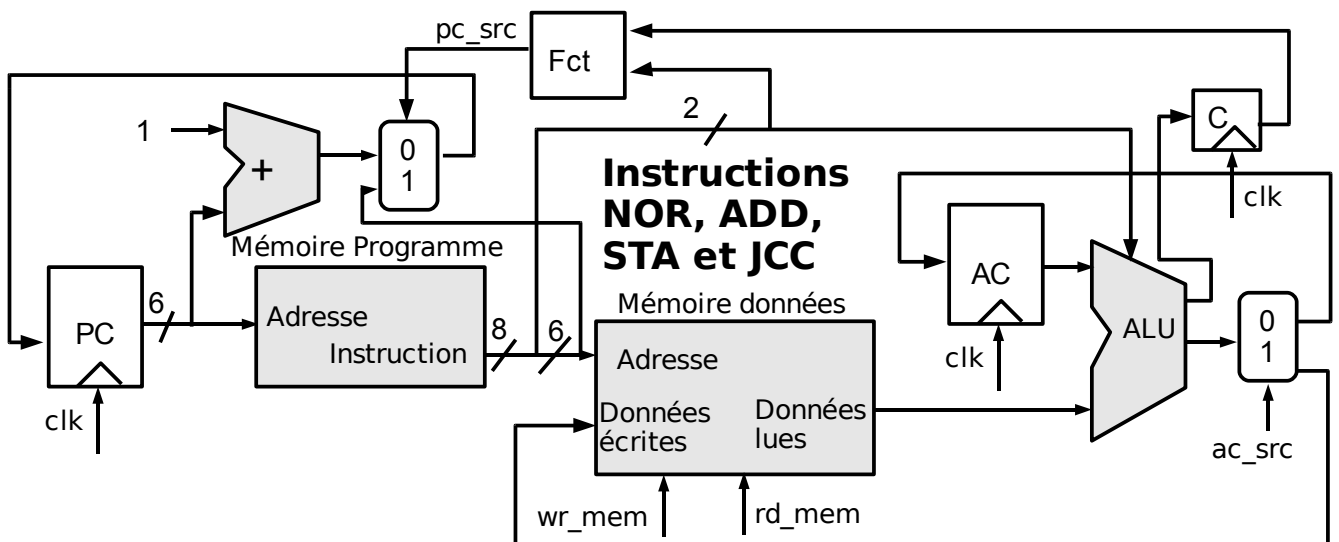
L'instruction de contrôle unique est JCC a la particularité de mettre la retenue à 0 après son exécution.



Jusqu'à présent, le bit de retenue n'a pas été ajouté au schéma pour raisons de simplifications.

I-7) Ensemble des instructions

L'ensemble des instructions peut être réalisé par le chemin de données suivant:



La fonction combinatoire "Fct" détecte l'instruction JCC et la retenue à 0.

I-8) Réalisation du contrôleur final

En fait le contrôleur final permet de générer "wr_mem", "rd_mem", "ac_src" et "pc_src" à partir des bits d'instructions et de la retenue. C'est donc tout simplement une super-fonction combinatoire remplaçant "Fct".

Solution exo5 TD5

Cet exercice n'a jamais été réalisé avec les étudiants. Historiquement il est dans le poly depuis le début. La solution proposée a été simulée. Si cela était à refaire j'essaierais de trouver une solution plus compacte.

Voici un programme VHDL :

```
ENTITY top IS
  PORT(
    Clk,reboot : IN BIT;
    I : IN BIT_VECTOR(3 DOWNTO 0);
    Q : INOUT BIT_VECTOR(3 DOWNTO 0);
    -- ajoute pour test
    SortieInstr : OUT BIT_VECTOR(2 DOWNTO 0);
    SortiesEn : OUT BIT_VECTOR(3 DOWNTO 0));
END Top;
ARCHITECTURE top of top IS
  COMPONENT u1
    PORT(
      A,B : IN BIT;
      S : OUT BIT;
      prog : IN BIT_VECTOR(2 DOWNTO 0));
  END COMPONENT;
  COMPONENT mux4_1
    PORT(
      e : IN BIT_VECTOR(3 DOWNTO 0);
      sel : IN BIT_VECTOR(1 DOWNTO 0);
      s : OUT BIT);
  END COMPONENT;
  COMPONENT basculeD
    PORT(
      clk : IN BIT;
      d,en : IN BIT;
      q: INOUT BIT);
  END COMPONENT;
  COMPONENT ou_ex
    PORT(
      s : OUT BIT;
      e1,e2 : IN BIT);
  END COMPONENT;
  COMPONENT pc
    PORT(
      clk : IN BIT;
      reset,en : IN BIT;
      q : INOUT BIT_VECTOR(2 DOWNTO 0));
  END COMPONENT;
  COMPONENT sequenceur
    PORT(
      -- horloge générale
      clk,reset : IN BIT;
      -- quelle sortie à selectionner
      sel : IN BIT_VECTOR(1 DOWNTO 0);
      -- instruction à exécuter
      instr : IN BIT_VECTOR(2 DOWNTO 0);
      -- H33,H32,H31,H30,H2,H1,H0
      QH : OUT BIT_VECTOR(7 DOWNTO 0);
      -- completer ou pas la sortie
      compl : OUT BIT);
  END COMPONENT;
  COMPONENT memoire
    PORT(
      -- bus d'adresse
      a : IN BIT_VECTOR(2 DOWNTO 0);
      -- bus de données
      d : OUT BIT_VECTOR(4 DOWNTO 0));
```

```

END COMPONENT;
signal BusSel : BIT_VECTOR(1 DOWNTO 0);
signal BusAdr : BIT_VECTOR(2 DOWNTO 0);
signal BusInstr : BIT_VECTOR(2 DOWNTO 0);
signal
filEntr, filA, filB, filS, filSortie, filReset, En0, En1, En2, En30, En31, En32, En33 : BIT;
signal inv, filReset2 : BIT;

BEGIN
-- entrees
ic1 : mux4_1 PORT MAP(e=>I, sel=>BusSel, s=>filEntr);
ic2 : basculeD PORT MAP(d=>filEntr, en=>En0, clk=>Clk, q=>filA);

-- Unite logique
ic3 : u1 PORT MAP(A=>filA, B=>filB, S=>filS, prog=>BusInstr);
ic4 : basculeD PORT MAP(d=>filS, en=>En1, q=>filB, clk=>Clk);
-- Compteur programme + memoire
ic5 : pc PORT MAP(clk=>clk, reset => filReset, en=>En2, q=>BusAdr);
ic6 : memoire PORT MAP(a=>BusAdr, d(4 downto 2)>BusInstr, d(1 downto
0)>BusSel);
-- sorties
--ic10 : dmux1_4 PORT MAP(e=>(filSortie XOR Inv), s=>Bus0, sel=>BusSel);
--ic7 : ou_ex PORT MAP(e1=>filB, e2=>Inv, s=>filSortie);
filSortie <= filB xor Inv;
ic8 : basculeD PORT MAP(d=>filSortie, en=>En30, clk=>Clk, q=>Q(0));
ic9 : basculeD PORT MAP(d=>filSortie, en=>En31, clk=>Clk, q=>Q(1));
ic10 : basculeD PORT MAP(d=>filSortie, en=>En32, clk=>Clk, q=>Q(2));
ic11 : basculeD PORT MAP(d=>filSortie, en=>En33, clk=>Clk, q=>Q(3));
-- sequenceur
ic12 : sequenceur PORT MAP(clk=>
Clk, sel=>BusSel, instr=>BusInstr, compl=>inv, QH(0)>En0,
QH(1)>En1, QH(2)>En2, QH(3)>En30, QH(4)>En31, QH(5)
=>En32, QH(6)>En33,
QH(7)>filReset, reset=>reboot);
filReset2 <= reboot OR filReset;
-- ajoute pour test
SortieInstr <= BusInstr;
sortiesEn(0) <= En30;
sortiesEn(1) <= En31;
sortiesEn(2) <= En32;
sortiesEn(3) <= En33;
END top;

-- unité logique
ENTITY u1 IS
PORT(
A, B : IN BIT;
S : OUT BIT;
prog : IN BIT_VECTOR(2 DOWNTO 0));
END u1;

ARCHITECTURE u1 OF u1 IS
BEGIN
WITH prog SELECT
S <= A WHEN "000", --LD
NOT A WHEN "001", --LDN
A AND B WHEN "010", --AND
NOT A AND B WHEN "011", --ANDN
A OR B WHEN "100", --OR
-- autrement on mémorise
B WHEN OTHERS; --ST, STN, EP
END u1;
--multiplexeur
ENTITY mux4_1 IS

```

```

PORT(
  e : IN BIT_VECTOR(3 DOWNTO 0);
  sel : IN BIT_VECTOR(1 DOWNTO 0);
  s : OUT BIT);
END mux4_1;
ARCHITECTURE mux4_1 OF mux4_1 IS
BEGIN
  WITH sel SELECT
    s<= e(0) WHEN "00",
        e(1) WHEN "01",
        e(2) WHEN "10",
        e(3) WHEN OTHERS;
END mux4_1;
-- bascule D
ENTITY basculeD IS
  PORT(
    clk : IN BIT;
    d,en: IN BIT;
    q: INOUT BIT);
END basculeD;

ARCHITECTURE abasculeD OF basculeD IS
BEGIN
  PROCESS (clk) BEGIN
    IF clk'event AND clk='1' THEN
      IF en='1' THEN
        q <= d;
      END IF;
    END IF;
  END PROCESS;
END abasculeD;
-- compteur programme
ENTITY pc IS
  PORT(
    clk : IN BIT;
    reset,en : IN BIT;
    q : INOUT BIT_VECTOR(2 DOWNTO 0));
END pc;
ARCHITECTURE pc OF pc IS BEGIN
  PROCESS(clk) BEGIN
    IF clk'event AND clk='1' THEN
      IF reset = '1' THEN
        q<="000";
      ELSIF en='1' THEN
        CASE q IS
          WHEN "000" => q <="001";
          WHEN "001" => q <="010";
          WHEN "010" => q <="011";
          WHEN "011" => q <="100";
          WHEN "100" => q <="101";
          WHEN "101" => q <="110";
          WHEN "110" => q <="111";
          WHEN "111" => q <="000";
        END CASE;
      END IF;
    END IF;
  END PROCESS;
END pc;

-- mémoire programme
ENTITY memoire IS
  PORT(
    -- bus d'adresse
    a : IN BIT_VECTOR(2 DOWNTO 0);

```

```

-- bus de données
  d : OUT BIT_VECTOR(4 DOWNTO 0));
END memoire;
ARCHITECTURE memoire OF memoire IS
BEGIN
  WITH a SELECT
    -- LD I0
    d<= "00000" WHEN "000",
    -- AND I1
    "01001" WHEN "001",
    -- ST Q2
    "10110" WHEN "010",
    -- OR I2
    "10010" WHEN "011",
    -- STN Q3
    "11011" WHEN "100",
    -- EP
    "11100" WHEN OTHERS;
END memoire;
--library work;use work.textio.all;-- avec warp2
--library std;use std.textio.all;
ENTITY sequenceur IS
  PORT(
    -- horloge générale
    clk,reset : IN BIT;
    -- quelle sortie à selectionner
    sel : IN BIT_VECTOR(1 DOWNTO 0);
    -- instruction à exécuter
    instr : IN BIT_VECTOR(2 DOWNTO 0);
    -- filReset,En33,En32,En31,En30,En3,En1,En0 : OUT BIT;
    QH : OUT BIT_VECTOR(7 DOWNTO 0);
    -- complements ou pas la sortie
    compl : OUT BIT);
END sequenceur;

ARCHITECTURE sequenceur OF sequenceur IS
  TYPE typetat IS (e0,e1,e2,e3,e4);
  SIGNAL etat : typetat;

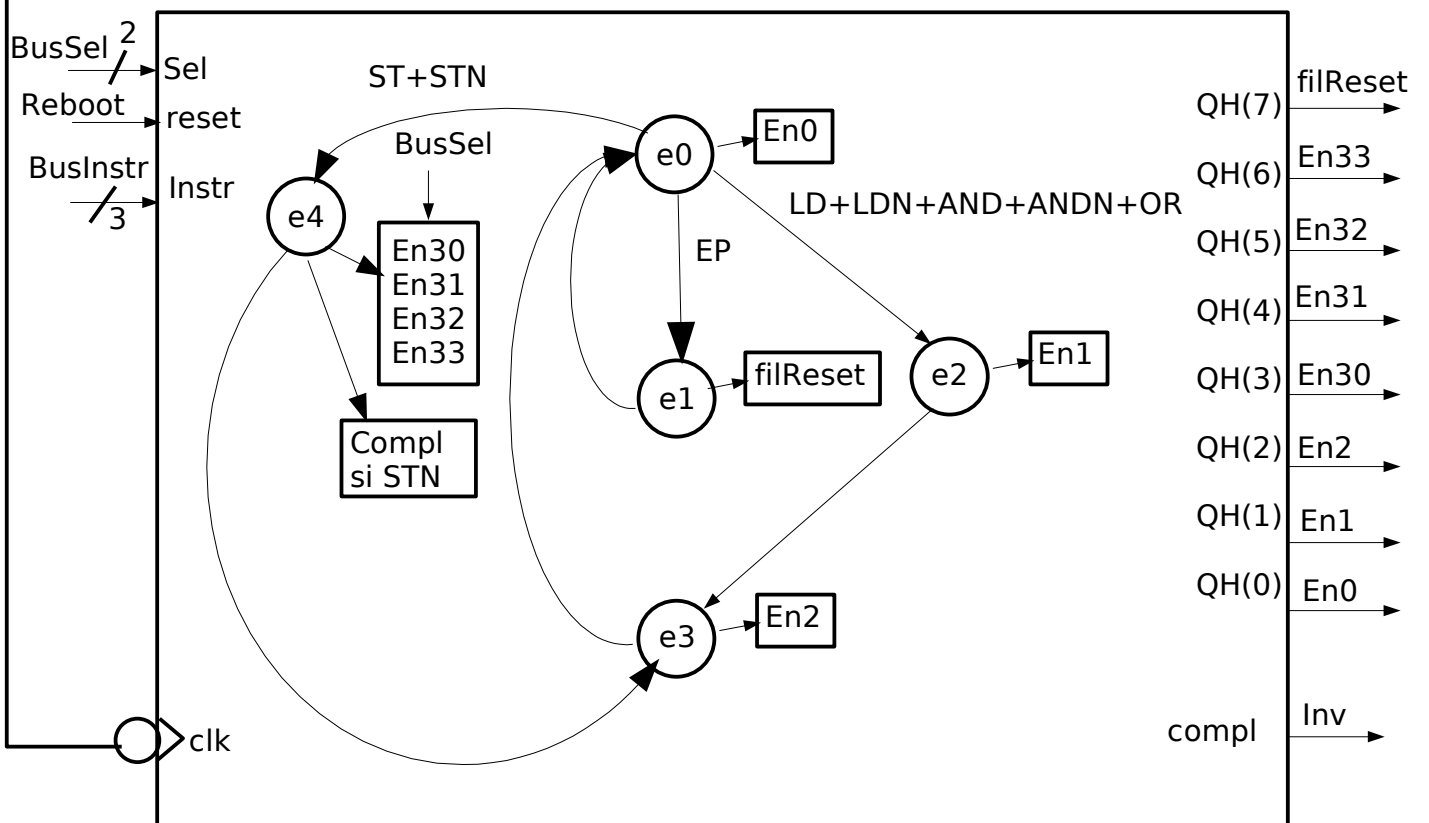
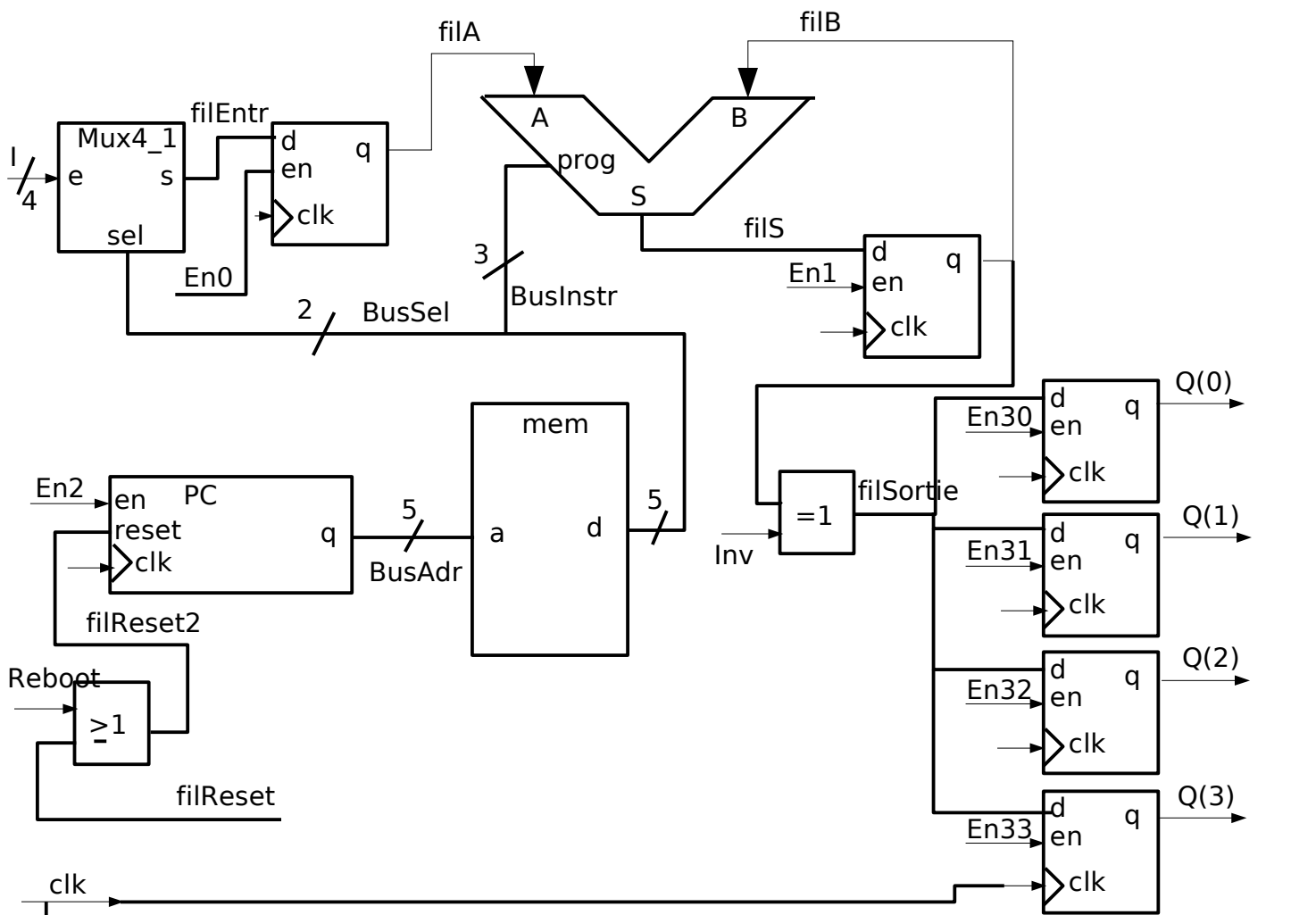
BEGIN
  --PROCESS(clk)variable L:LINE; BEGIN
  PROCESS(clk) BEGIN
    IF clk'event AND clk='0' THEN -- front descendant
      --write(L,now);
      --write(L,STRING("(" instr= "));write(L,instr);
    IF reset = '1' THEN
      etat <= e0; --retour au départ
    ELSE
      CASE etat IS
        WHEN e0 => IF instr = "000" OR instr ="001" --LD, LDN
                    OR instr ="010" -- AND
                    OR instr ="011" -- ANDN
                    OR instr ="100" THEN --OR
                      etat <= e2;
                    ELSIF instr = "111" THEN --EP
                      etat <=e1;
                    ELSE -- ST, STN
                      etat <= e4;
                    END IF;
        WHEN e1 => etat <= e0;
        WHEN e2 => etat <= e3;
        WHEN e3 => etat <= e0;
        WHEN e4 => etat <= e3;
      END CASE;
    END PROCESS;
  END ARCHITECTURE;

```

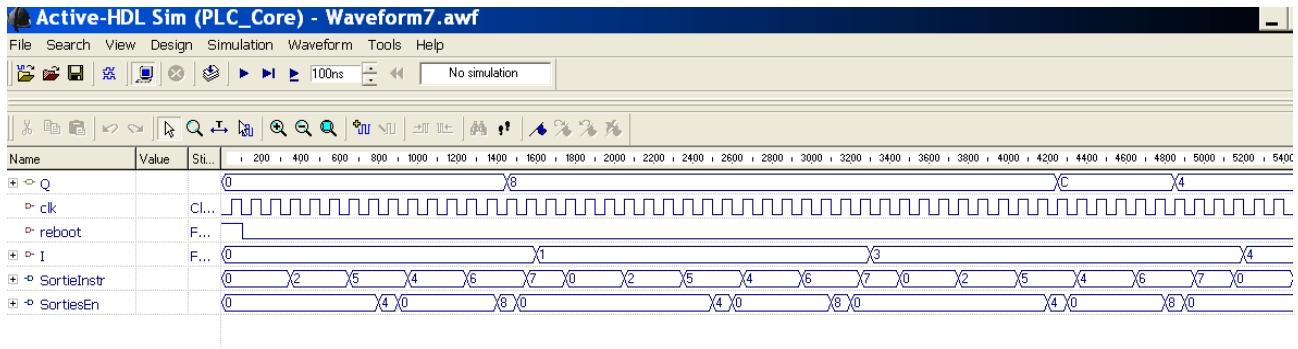
```

    END IF;
  END IF;
END PROCESS;
QH(0) <= '1' when (etat = e0) else
    '0' when (etat = e1) else
    '0' when (etat = e2) else
    '0' when (etat = e3) else
    '0' when (etat = e4) else
    '0';
QH(1) <= '0' when (etat = e0) else
    '0' when (etat = e1) else
    '1' when (etat = e2) else
    '0' when (etat = e3) else
    '0' when (etat = e4) else
    '0';
QH(2) <= '0' when (etat = e0) else
    '0' when (etat = e1) else
    '0' when (etat = e2) else
    '1' when (etat = e3) else
    '0' when (etat = e4) else
    '0';
QH(7) <= '0' when (etat = e0) else
    '1' when (etat = e1) else
    '0' when (etat = e2) else
    '0' when (etat = e3) else
    '0' when (etat = e4) else
    '0';
QH(3) <= '1' when (etat = e4) and sel = "00" else '0';
QH(4) <= '1' when (etat = e4) and sel = "01" else '0';
QH(5) <= '1' when (etat = e4) and sel = "10" else '0';
QH(6) <= '1' when (etat = e4) and sel = "11" else '0';
compl <= '1' when (instr="110") else '0';
END sequenceur;

```



Et le dessin du séquenceur :



Ce TD 6 n'a jamais été réalisé avec nos étudiants

Solution exo1 TD6

Question 1

Remplacer le process central par :

```
process(clk) begin
-- wait until (Clk='0');
  if clk'event and clk='0' then
    if (Count=7) then Count <=0;
    else Count <=Count+1;
    end if;
  end if;
end process;
```

Question 2

```
-- fichier demo1.vhdl unique
-- Compilation :
--[smoutou@p3 freehdl-20040113]$ cd myexamples/
--[smoutou@p3 myexamples]$ ../v2cc/gvhdl demo1.vhdl ../std/internal_textio.o
```

----- Composant à tester -----

```
entity addit is
  --generic (gen : integer := 10);
  port (
    a,b,c: in bit;
    s,r: out bit);
end addit;
architecture addit of addit is
begin
  s <= a xor b xor c;
  r <= ((a and b) or (c and (a xor b)));
end addit;
```

----- Test Bench -----

```
library std;
use std.textio.all;
use WORK.addit;
entity model is
end model;
architecture struct of model is
  signal asig,bsig,csig,sum,carry : bit:='0';
  component addit is port (
    a,b,c: in bit;
    s,r: out bit);
  end component;
```

```

begin
  ad1: addit port map (a=>asig,b=>bsig,c=>csig,s=>sum,r=>carry);
  asig <= not asig after 10 ns;
  bsig <= not bsig after 20 ns;
  csig <= not csig after 40 ns;
  TestBench:process(sum,carry) variable L:LINE; begin
    write(L,now);write(L,STRING'(" (r,s)= "));
    write(L,carry);write(L,sum);
    write(L,STRING'(" = "));write(L,asig);write(L,STRING'(" + "));
write(L,bsig);
    write(L,STRING'(" + "));write(L,csig);writeline(Output,L);
  end process;
end struct;

```

donne comme résultat :

```

Simulation time = 0 fs + 0d
> r 100ns
Run simulation for which time span?
Simulating model to time 100 ns
10 ns (r,s)= 01 = 1 + 0 + 0
30 ns (r,s)= 10 = 1 + 1 + 0
40 ns (r,s)= 01 = 0 + 0 + 1
50 ns (r,s)= 10 = 1 + 0 + 1
70 ns (r,s)= 11 = 1 + 1 + 1
80 ns (r,s)= 00 = 0 + 0 + 0
90 ns (r,s)= 01 = 1 + 0 + 0
Simulation time = 100 ns + 0d

```

On peut modifier comme suit pour obtenir un affichage décimal:

```

-- fichier demo1.vhdl unique
-- Compilation :
--[smoutou@p3 freehdl-20040113]$ cd myexamples/
--[smoutou@p3 myexamples]$ ../v2cc/gvhdl demo1.vhdl ../std/internal_textio.o

```

```

----- Composant à tester -----
entity addit is
  --generic (gen : integer := 10);
  port (
    a,b,c: in bit;
    s: out bit_vector(1 downto 0));
end addit;
architecture aaddit of addit is
begin
  s(0) <= a xor b xor c;
  s(1) <= ((a and b) or (c and (a xor b)));
end aaddit;

```

```

----- Test Bench -----
library std;
use std.textio.all;
use WORK.addit;
entity model is
end model;
architecture struct of model is
  signal asig,bsig,csig : bit:='0';
  signal sum : bit_vector(1 downto 0):="00";
  signal somme : integer :=0;
  component addit is port (
    a,b,c: in bit;
    s: out bit_vector(1 downto 0));
  end component;
begin
  ad1: addit port map (a=>asig,b=>bsig,c=>csig,s=>sum);

```

```

asig <= not asig after 10 ns;
bsig <= not bsig after 20 ns;
csig <= not csig after 40 ns;
TestBench:process(sum) variable L:LINE; begin
  if sum="00" then somme<=0;
    elsif sum="01" then somme<=1;
    elsif sum="10" then somme<=2;
  else somme<=3;
  end if;
  write(L,now);write(L,STRING'(" somme = "));
  write(L,somme);
  write(L,STRING'(" = "));write(L,asig);write(L,STRING'(" + "));
write(L,bsig);
  write(L,STRING'(" + "));write(L,csig);writeline(Output,L);
end process;
end struct;

```

Le point important est la conversion bit_vector integer ici réalisée manuellement mais qui peut être réalisée autrement si l'on a les bonnes librairies.

La simulation donne :

```

> r 100ns
Run simulation for which time span?
Simulating model to time 100 ns
10 ns somme = 0 = 1 + 0 + 0
30 ns somme = 1 = 1 + 1 + 0
40 ns somme = 2 = 0 + 0 + 1
50 ns somme = 1 = 1 + 0 + 1
70 ns somme = 2 = 1 + 1 + 1
80 ns somme = 3 = 0 + 0 + 0
90 ns somme = 0 = 1 + 0 + 0
Simulation time = 100 ns + 0d

```

A noter l'erreur que ce programme donne. Pour corriger cette erreur une variable sera plutôt utilisée à la place du signal « somme » :

```

TestBench:process(sum)
  variable L:LINE;
  variable somme : integer ;
begin
  if sum="00" then somme:=0;
    elsif sum="01" then somme:=1;
    elsif sum="10" then somme:=2;
  else somme:=3;
  end if;
  write(L,now);write(L,STRING'(" somme = "));
  write(L,somme);
  write(L,STRING'(" = "));write(L,asig);write(L,STRING'(" + "));
write(L,bsig);
  write(L,STRING'(" + "));write(L,csig);writeline(Output,L);
end process;

```

Donne le résultat correct :

```

> r 100 ns
Run simulation for which time span?
Simulating model to time 100 ns
10 ns somme = 1 = 1 + 0 + 0
30 ns somme = 2 = 1 + 1 + 0
40 ns somme = 1 = 0 + 0 + 1
50 ns somme = 2 = 1 + 0 + 1
70 ns somme = 3 = 1 + 1 + 1
80 ns somme = 0 = 0 + 0 + 0
90 ns somme = 1 = 1 + 0 + 0
Simulation time = 100 ns + 0d

```

Question 3

```

--library work;use work.textio.all;
library std;use std.textio.all;
PACKAGE test7seg IS
COMPONENT SeptSeg
  PORT (a : IN BIT_VECTOR(6 DOWNT0 0));
END COMPONENT;
END test7seg;
ENTITY SeptSeg IS
  PORT (a : IN BIT_VECTOR(6 DOWNT0 0));
END SeptSeg;
library std;use std.textio.all; --imperatif avant architecture
ARCHITECTURE aSeptSeg OF SeptSeg IS
BEGIN
  process (a) variable L:LINE;
  begin
    write(L,now);
    writeline(Output,L);
    if a(0)='1' then
      write(L,STRING'" *"*');
      writeline(Output,L);
    else
      write(L,STRING'" --"');
      writeline(Output,L);
    end if;
    if (a(5)='1' and a(1)='0') then
      write(L,STRING'"* |"');
      writeline(Output,L);
      write(L,STRING'"* |"');
      writeline(Output,L);
    elsif (a(5)='1' and a(1)='1') then
      write(L,STRING'"* *"*');
      writeline(Output,L);
      write(L,STRING'"* *"*');
      writeline(Output,L);
    elsif (a(5)='0' and a(1)='1') then
      write(L,STRING'"| *"*');
      writeline(Output,L);
      write(L,STRING'"| *"*');
      writeline(Output,L);
    else
      write(L,STRING'"| |"');
      writeline(Output,L);
      write(L,STRING'"| |"');
      writeline(Output,L);
    end if;
    if a(6)='1' then
      write(L,STRING'" *"*');
      writeline(Output,L);
    else
      write(L,STRING'" --"');
      writeline(Output,L);
    end if;
    if (a(4)='1' and a(2)='0') then
      write(L,STRING'"* |"');
      writeline(Output,L);
      write(L,STRING'"* |"');
      writeline(Output,L);
    elsif (a(4)='1' and a(2)='1') then
      write(L,STRING'"* *"*');
      writeline(Output,L);
      write(L,STRING'"* *"*');
      writeline(Output,L);
    elsif (a(4)='0' and a(2)='1') then
      write(L,STRING'"| *"*');

```

```

        writeline(Output,L);
        write(L,STRING'("| *"));
        writeline(Output,L);
    else
        write(L,STRING'("| |"));
        writeline(Output,L);
        write(L,STRING'("| |"));
        writeline(Output,L);
    end if;
    if a(3)='1' then
        write(L,STRING'(" **"));
        writeline(Output,L);
    else
        write(L,STRING'(" --"));
        writeline(Output,L);
    end if;
end process;
END aSeptSeg;

```

Un fichier test du style :

```

ENTITY transcod IS
    PORT(a:IN BIT_VECTOR(3 DOWNT0 0);s:OUT BIT_VECTOR(6 DOWNT0 0));
END transcod;
ARCHITECTURE atranscod OF transcod IS
BEGIN
    with a SELECT
    s <="0111111" WHEN "0000",
        "0000110" WHEN "0001",
        "1011011" WHEN "0010",
        "1001111" WHEN "0011",
        "1100110" WHEN "0100",
        "1101101" WHEN "0101",
        "1111101" WHEN "0110",
        "0000111" WHEN "0111",
        "1111111" WHEN "1000",
        "1101111" WHEN "1001",
        "0000000" WHEN OTHERS;
END atranscod;

ENTITY test7seg IS END;
library afficheur7seg;
use afficheur7seg.all;
ARCHITECTURE atest7seg OF test7seg IS
SIGNAL e4 :BIT_VECTOR(3 DOWNT0 0);
SIGNAL s7 : BIT_VECTOR(6 DOWNT0 0);
COMPONENT SeptSeg
    PORT (a : IN BIT_VECTOR(6 DOWNT0 0));
END COMPONENT;
COMPONENT transcod
    PORT(a:IN BIT_VECTOR(3 DOWNT0 0);s:OUT BIT_VECTOR(6 DOWNT0 0));
END COMPONENT;
BEGIN
    c1:transcod PORT MAP(e4,s7);
    c2:SeptSeg PORT MAP(s7);
    e4(0) <= not e4(0) after 100 ns;
    e4(1) <= not e4(1) after 200 ns;
    e4(2) <= not e4(2) after 400 ns;
    e4(3) <= not e4(3) after 800 ns;
END;

```

donnera un rapport : (** : segment allumé, -- segment éteint) pas très facile à

lire ...

```
0 ns
:  --
:  |  |
:  |  |
:  --
:  |  |
:  |  |
:  --
:  0 ns
:  **
:  *  *
:  *  *
:  --
:  *  *
:  *  *
:  **
: 100 ns
:  --
:  |  *
:  |  *
:  --
:  |  *
:  |  *
:  --
:  **
:  |  *
:  |  *
:  **
:  *  |
:  *  |
:  **
: 300 ns
:  **
:  |  *
:  |  *
:  **
:  |  *
:  |  *
:  **
: 400 ns
:  --
:  *  *
:  *  *
:  **
:  |  *
:  |  *
:  --
: 500 ns
:  **
:  *  |
:  *  |
:  **
:  |  *
:  |  *
:  **
: 600 ns
:  **
:  *  |
:  *  |
:  **
:  *  *
```

```
: * *  
: **  
: 700 ns
```

```
.....
```

```
KERNEL: stopped at time: 1600 ns
```