Corrections des TDs MCENL1 (2011/2012)

Solution exo1 TD1

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

rn	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;
ARCITECTURE 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;
ARCITECTURE 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
```

```
                WHEN "000" => s_q <="001";
                WHEN "001" => s_q <="011";
                WHEN "011" => s_q <="010";
                WHEN "010" => s_q <="110";
                WHEN "110" => s_q <="111";
                WHEN "111" => s_q <="101";
                WHEN "101" => s_q <="100";
                WHEN OTHERS => s_q <="000";
              END CASE;
            END IF;
          END PROCESS;
        q <= s_q;
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 :

```
-- compteur Gray
entity lent is
        port(horloge : in std_logic;
             h_lente : out std_logic);
end lent;

architecture alent of lent is
        signal compteur : std_logic_vector(14 downto 0);
        begin
        --division de l'horloge par 32768
        process(horloge) begin
                if(horloge'event and horloge='1') then
                        compteur <= compteur + 1;
                end if;
        end process;
        h_lente <= compteur(14);
end alent;
```

Solution exo3 TD2

Voici la solution asynchrone :

```
PROCESS(clk,raz,set) BEGIN
  IF raz='1' THEN
      q<=(OTHERS=>'0');
  ELSIF set='1' THEN
      q<=(OTHERS=>'1');
  ELSIF clk'event and clk='1' THEN
    q<=std_logic_vector(unsigned(q)+1);
  END IF;
END PROCESS;
```

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

```
PROCESS(clk,raz,set) BEGIN
  IF raz='1' THEN
      q<=(OTHERS=>'0');
  ELSIF set='1' THEN
      q<=(OTHERS=>'1');
  ELSIF clk'event and clk='1' THEN
    IF unsigned(q) <25 then
      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

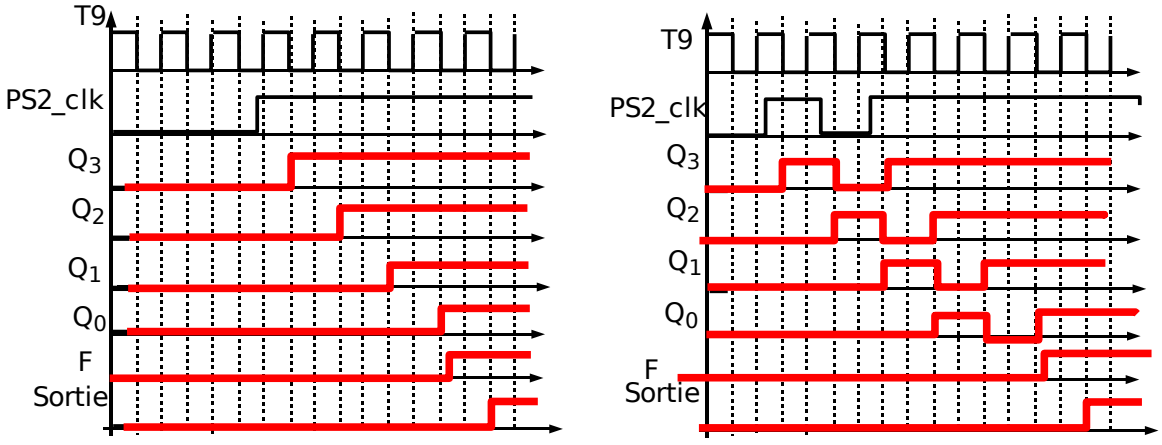
1°) fh = 25 Mhz => Th = 40 ns
2°) Nb compteur pour 25,6 us : 25,6 us/ 40 ns = 640
3°) Nb compteur pour 0,64 us : 0,64 us/ 40 ns = 16
Période 31,75 us / 40 ns = 793,75 arrondi à 800 donc XXX=799
Durée de la valeur 0 3,8 us soit 95
Donc en horizontal :
0 <--------->639 <--->655 (639+16) <-->750(655+95)<--->799
donc ZZZ=654 et TTT=751 (car comparaisons strictes)
4°) Période verticale : Tv = 40 ns x 800 = 32 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 x 32 us = 15,36 ms (au lieu de 15,24 ms)
6°) Le temps de 0,35 ms est réalisé par comptage 0,35ms/32 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

```
--  Horiz_sync  ------------------------------------_____________--------
--  H_count       0              640           659      755    799
--  Vert_sync     ------------------------------------------------_______-----------
--  V_count         0                                 480    493-494         524
```

Solution exo5 TD2

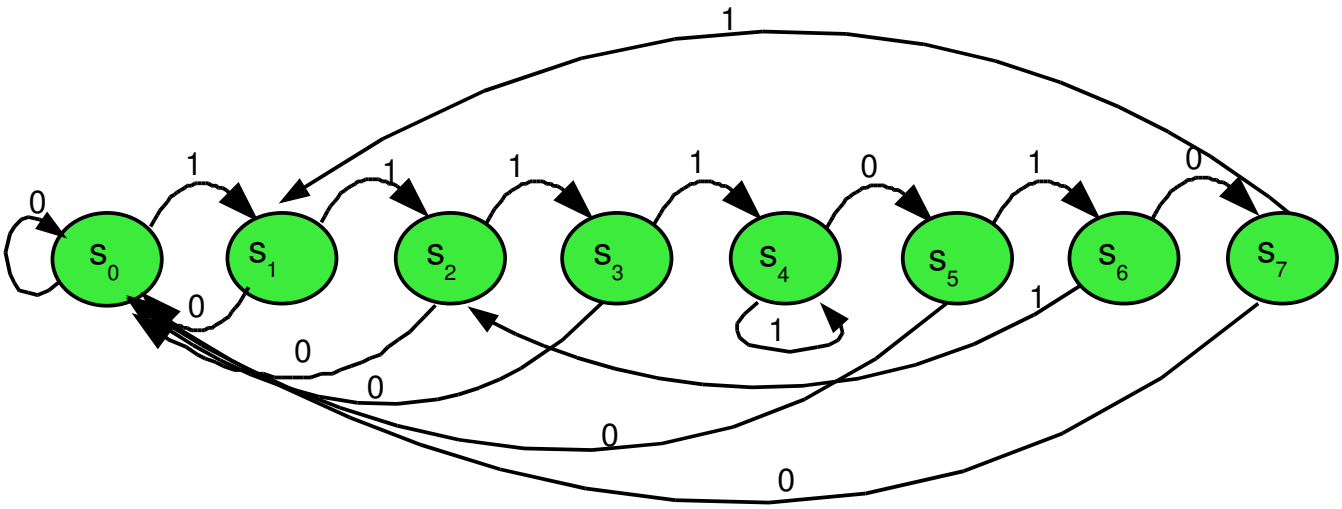
Cette correction correspond au schéma original avec registre à décalage sur front descendant.



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

Solution exo1 TD3

1°)



Certains aspects
de cet automate demandent reflexion.
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 s_0 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°) $s_0^{futur} = \overline{s} \cdot \overline{s_0} \cdot e_0 \cdot \overline{Init} + \overline{s} \cdot s_0 \cdot e_1 \cdot \overline{Init}$

$s_1^{futur} = s \cdot \overline{s_0} \cdot \overline{e_0} \cdot \overline{Init} + \overline{s} \cdot s_0 \cdot e_1 \cdot \overline{Init}$

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

Solution exo1 TD4

```
1         -- Xilinx specific
2         library IEEE;
3         use IEEE.STD_LOGIC_1164.ALL;
4         library unisim;
5         use unisim.vcomponents.all;
6         ENTITY transcodeur IS PORT(
7           e : in STD_LOGIC_VECTOR(3 DOWNTO 0);   -- 4 entrées
8           s : out STD_LOGIC_VECTOR(6 DOWNTO 0)); -- 7 sorties
```

```
9         END transcodeur;
10        ARCHITECTURE atranscodeur OF transcodeur IS BEGIN
11        i1 : LUT4 -- segment a
12          generic map (INIT => X"D7ED")
13          port map( I0 => e(0),
14                    I1 => e(1),
15                    I2 => e(2),
16                    I3 => e(3),
17                     O => s(0) );
18        i2 : LUT4 -- segment b
19          generic map (INIT => X"279F")
20          port map( I0 => e(0),
21                    I1 => e(1),
22                    I2 => e(2),
23                    I3 => e(3),
24                     O => s(1) );
25        i3 : LUT4 -- segment c
26          generic map (INIT => X"2FFB")
27          port map( I0 => e(0),
28                    I1 => e(1),
29                    I2 => e(2),
30                    I3 => e(3),
31                     O => s(2) );
32        i4 : LUT4 -- segment d
33          generic map (INIT => X"7B6D")
34          port map( I0 => e(0),
35                    I1 => e(1),
36                    I2 => e(2),
37                    I3 => e(3),
38                     O => s(3) );
39        i5 : LUT4 -- segment e
40          generic map (INIT => X"FD45")
41          port map( I0 => e(0),
42                    I1 => e(1),
43                    I2 => e(2),
44                    I3 => e(3),
45                     O => s(4) );
46        i6 : LUT4 -- segment f
47          generic map (INIT => X"DF71")
48          port map( I0 => e(0),
49                    I1 => e(1),
50                    I2 => e(2),
51                    I3 => e(3),
52                     O => s(5) );
53        i7 : LUT4 -- segment a
54          generic map (INIT => X"EFFC")
55          port map( I0 => e(0),
56                    I1 => e(1),
57                    I2 => e(2),
58                    I3 => e(3),
59                     O => s(6) );
60        end atranscodeur;
```

Solution exo2 TD4

```
1         library ieee;
2         use ieee.std_logic_1164.all;
3         entity top is port (
4           clk_50Mhz : in std_logic;
5           aff : out std_logic_vector(3 downto 0);
6           s_7segs : out std_logic_vector(6 downto 0));
7         end top;
8         architecture atop of top is
```

```
9           component rams_21a is
10            port (clk : in std_logic;
11                  en    : in std_logic;
12                  addr : in std_logic_vector(3 downto 0);
13                  data : out std_logic_vector(7 downto 0));
14          end component;
15          component div is
16          port( clk_50Mhz : in std_logic;
17                clk_slow : out std_logic);
18          end component;
19          component cmpt4bits is
20          port( clk,en : in std_logic;
21                q : out std_logic_vector(3 downto 0));
22          end component;
23          signal s_clk_slow, s_d : std_logic;
24          signal s_addr : std_logic_vector(3 downto 0);
25          signal sig_7segs : std_logic_vector(6 downto 0);
26        begin
27          i1 : div  port map(clk_50Mhz => clk_50Mhz,clk_slow => s_clk_slow);
28          i2 : cmpt4bits port map(clk => s_clk_slow,q => s_addr,en => '1');
29          i3 : rams_21a port map(clk => clk_50Mhz,addr => s_addr,
30               data(6 downto 0) => sig_7segs, data(7) => s_d, en => '1');
31          s_7segs <= not sig_7segs; -- remove the not if segment display with '1'
32          aff(0) <='0';
33          aff(1) <='1';
34          aff(2) <='1';
35          aff(3) <='1';
36        end atop;
37
38        --ROM With Registered Output VHDL Coding Example One
39        --
40        -- ROMs Using Block RAM Resources.
41        -- VHDL code for a ROM with registered output (template 1)
42        --
43        library ieee;
44        use ieee.std_logic_1164.all;
45        use ieee.std_logic_unsigned.all;
46        entity rams_21a is
47            port (clk : in std_logic;
48                  en    : in std_logic;
49                  addr : in std_logic_vector(3 downto 0);
50                  data : out std_logic_vector(7 downto 0));
51        end rams_21a;
52        architecture syn of rams_21a is
53          type rom_type is array (0 to 15) of std_logic_vector (7 downto 0);
54                  -- transcodage hexadécimal affiche 0 à F
55        signal ROM : rom_type:= (X"3F", X"06", X"5B", X"4F", X"66", X"6D",
56        X"7D",X"07", X"7F", X"6F", X"77", X"7C", X"39", X"5E", X"79", X"71");
57        begin
58            process (clk)
59            begin
60                if (clk'event and clk = '1') then
61                    if (en = '1') then
62                        data <= ROM(conv_integer(addr));
63                    end if;
64                end if;
65            end process;
66        end syn;
67
68        library ieee;
69        use ieee.std_logic_1164.all;
70        use ieee.std_logic_unsigned.all;
71        entity div is
72          port( clk_50Mhz : in std_logic;
```

```
73              clk_slow : out std_logic);
74      end div;
75      architecture adiv of div is
76      signal cmpt : std_logic_vector(23 downto 0);
77      begin
78        process(clk_50MHz) begin
79          if rising_edge(clk_50Mhz) then
80            cmpt <= cmpt + 1;
81          end if;
82        end process;
83        clk_slow <= cmpt(23);
84      end adiv;
85
86      library ieee;
87      use ieee.std_logic_1164.all;
88      use ieee.std_logic_arith.all;
89      use ieee.std_logic_unsigned.all;
90      entity cmpt4bits is
91        port( clk,en : in std_logic;
92              q : out std_logic_vector(3 downto 0));
93      end cmpt4bits;
94      architecture acmpt4bits of cmpt4bits is
95      signal s_q : std_logic_vector(3 downto 0);
96      begin
97        process(clk) begin
98          if rising_edge(clk) then
99              if en = '1' then
100                s_q <= s_q + 1;
101             end if;
102         end if;
103       end process;
104       q <= s_q;
105     end acmpt4bits;
```

Solution exo 1 TD5

1°) Le programme donné recopie les interrupteurs sur les segments.
Le programme demandé est :

```
1       ;;; Sortie sur deux digits
2       constant MAX, 255
3       namereg s0,i
4       NAMEREG s1, octet_lsb
5       NAMEREG s3, s7seg  ; rename register s3 as “s7seg”
6       NAMEREG s4, Aff
7       debut:
8               ; initialisation RAM : table de conversion
9               LOAD s0,01
10              STORE s0,00
11              LOAD s0,4F
12              STORE s0,01
13              LOAD s0,12
14              STORE s0,02
15              LOAD s0,06
16              STORE s0,03
17              LOAD s0,4C
18              STORE s0,04
19              LOAD s0,24
20              STORE s0,05
21              LOAD s0,20
22              STORE s0,06
23              LOAD s0,0F
```

```
24                STORE s0,07
25                LOAD s0,00
26                STORE s0,08
27                LOAD s0,04
28                STORE s0,09
29                LOAD s0,08
30                STORE s0,0A
31                LOAD s0,60
32                STORE s0,0B
33                LOAD s0,31
34                STORE s0,0C
35                LOAD s0,42
36                STORE s0,0D
37                LOAD s0,30
38                STORE s0,0E
39                LOAD s0,38
40                STORE s0,0F
41        boucle:
42                ;entree des 8 bits
43                INPUT octet_lsb,0
44                AND octet_lsb,0F
45                ;conversion par RAM
46                FETCH s7seg,(octet_lsb)
47                ;sortie 7 segs
48                OUTPUT s7seg,0
49                CALL wait
50                JUMP boucle
51        wait:
52                LOAD i,MAX
53        loop:   SUB i,01
54                JUMP NZ,loop
55                RETURN
```

2°) Voici une correction dans laquelle il manque le port d'entrée. Le port de sortie est réalisé par un process.

```
1         library ieee;
2         use IEEE.STD_LOGIC_1164.ALL;
3         entity tp10exo1 is
4           port (
5           clk,reset : in std_logic;
6           entrees : in std_logic_vector(7 downto 0);
7           sorties : out std_logic_vector(7 downto 0)
8           );
9         end tp10exo1;
10
11        architecture atp10 of tp10exo1 is
12        component kcpsm3
13            Port (      address : out std_logic_vector(9 downto 0);
14                    instruction : in std_logic_vector(17 downto 0);
15                        port_id : out std_logic_vector(7 downto 0);
16                   write_strobe : out std_logic;
17                       out_port : out std_logic_vector(7 downto 0);
18                    read_strobe : out std_logic;
19                        in_port : in std_logic_vector(7 downto 0);
20                      interrupt : in std_logic;
21                  interrupt_ack : out std_logic;
22                          reset : in std_logic;
23                            clk : in std_logic);
24        end component;
25        component mpu_rom
26            Port (      address : in std_logic_vector(9 downto 0);
27                    instruction : out std_logic_vector(17 downto 0);
```

```
28                          clk : in std_logic);
29      end component;
30
31      signal s_sorties, s_sorties2: std_logic_vector(7 downto 0);
32      signal s_write_strobe : std_logic;
33      signal s_address : std_logic_vector(9 downto 0);
34      signal s_instruction : std_logic_vector(17 downto 0);
35      begin
36        i1:kcpsm3 port map(address => s_address,
37                           instruction => s_instruction,
38              port_id => open,
39              in_port => entrees,
40              out_port => s_sorties,
41              read_strobe => open,
42              write_strobe => s_write_strobe,
43              interrupt =>'0',
44              interrupt_ack => open,
45              reset => reset,
46              clk => clk);
47        i2: mpu_rom port map(address => s_address,
48                             instruction => s_instruction,
49                    clk => clk);
50      -- mémorisation sorties = port de sortie

51      process(clk) begin
52        if rising_edge(clk) then
53          if s_write_strobe = '1' then
54             s_sorties2 <= s_sorties;
55         end if;
56        end if;
57      end process;
58      sorties <= s_sorties2;
59      end atp10;
```

Solution exo 2 TD5

```
1       constant MAX, 60
2       NAMEREG s0, nb
3       NAMEREG s1, i
4       NAMEREG s2, j
5       NAMEREG s3, sortie
6       NAMEREG s4, k
7
8       debut :
9       ; initialisation RAM : table de conversion
10        LOAD s0,01
11        STORE s0,00
12        LOAD s0,02
13        STORE s0,01
14        LOAD s0,04
15        STORE s0,02
16        LOAD s0,08
17        STORE s0,03
18        LOAD s0,10
19        STORE s0,04
20        LOAD s0,20
21        STORE s0,05
22        LOAD s0,40
23        STORE s0,06
24        LOAD s0,80
25        STORE s0,07
26        LOAD s0,40
```

```
27          STORE s0,08
28          LOAD s0,20
29          STORE s0,09
30          LOAD s0,10
31          STORE s0,0A
32          LOAD s0,08
33          STORE s0,0B
34          LOAD s0,04
35          STORE s0,0C
36          LOAD s0,02
37          STORE s0,0D
38          LOAD s0,01
39          STORE s0,0E
40          LOAD s0,00
41          STORE s0,0F
42        ;de la boucle 16 fois
43        init_1:
44          LOAD nb,0F
45        boucle:
46          FETCH sortie,(nb) ; conversion
47          OUTPUT sortie,0
48          CALL wait
49          SUB nb,01
50          JUMP NZ, boucle
51          JUMP init_1
52        wait:
53          LOAD i,MAX
54        loop:
55                LOAD j,MAX
56        loop_1:
57                      LOAD k,MAX
58        loop_2:
59                      SUB k,01
60                      JUMP NZ, loop_2
61                SUB j,01
62                JUMP NZ, loop_1
63          SUB i,01
64          JUMP NZ,loop
65          RETURN
```