

## I) Introduction

Nous allons partir d'une version corrigée du compteur de passages du TP6 (comme pour le médian) et en faire un certain nombre de modifications. Dans cette épreuve, on ne vous demande pas de vous poser des questions sur l'utilité ou non des modifications mais seulement de les exécuter et de montrer que le travail résultant fonctionne toujours. Le but de l'épreuve est ainsi d'évaluer comment vous appréhendez des modifications à partir d'un schéma complexe décrit en VHDL.

Vous disposez d'une feuille réponse sur laquelle vous répondez et sur laquelle l'enseignant notera ce qu'il a vu fonctionner et sinon, quelques indications sur l'état d'avancement de votre travail.

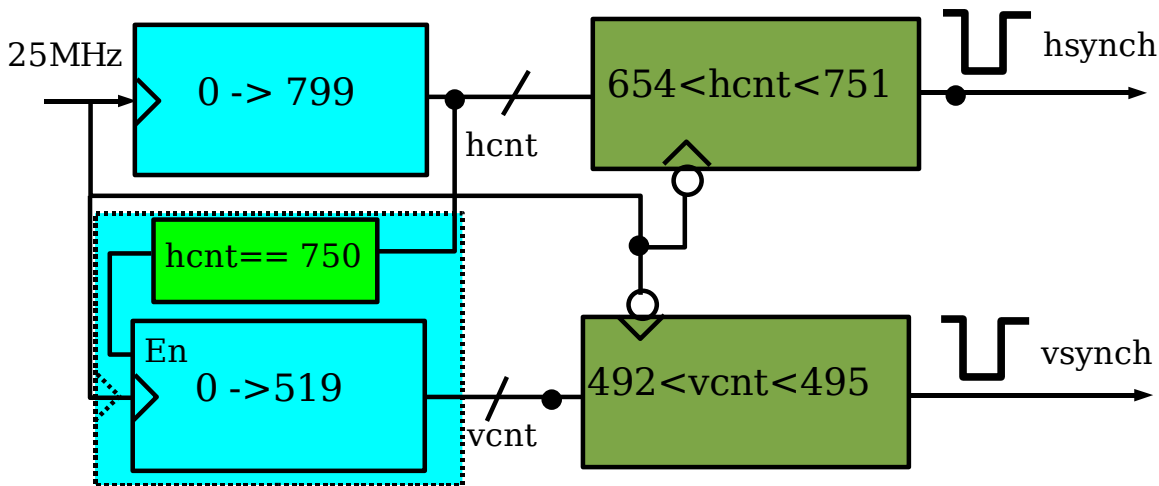
Le barème n'est donné qu'à titre indicatif.

## II) Affichage sur écran VGA en mode graphique

On désire réaliser un affichage sur écran VGA en mode graphique. Pour cela il nous faut gérer 5 signaux que l'on va décrire maintenant.

### Partie synchronisation

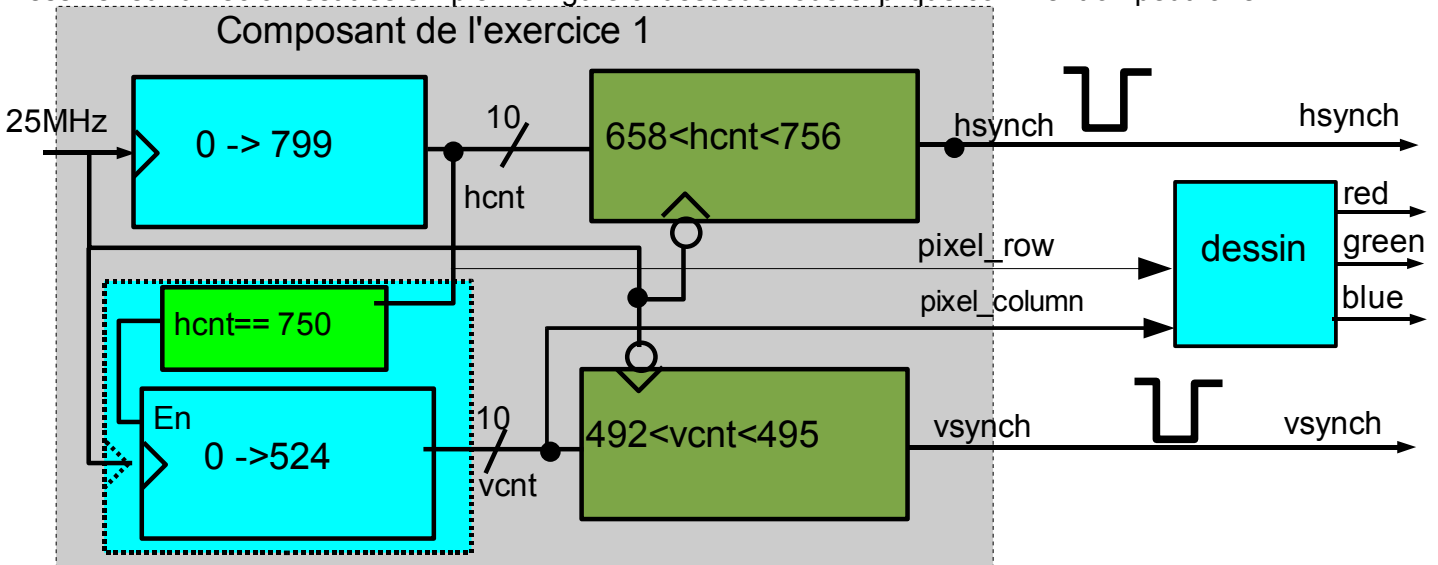
Son objectif est de gérer deux signaux appelés hsync et vsynch à l'aide de deux compteurs.



Le programme VHDL qui réalise cette partie peut être trouvé dans l'exercice 1 de la page [http://fr.wikiversity.org/wiki/Very\\_High\\_Speed\\_Integrated\\_Circuit\\_Hardware\\_Description\\_Language/Interfaces\\_VGA\\_et\\_PS/2](http://fr.wikiversity.org/wiki/Very_High_Speed_Integrated_Circuit_Hardware_Description_Language/Interfaces_VGA_et_PS/2)

### Partie dessin sur l'écran

Dessiner sur un écran est très simple. La figure ci-dessous vous explique comment on peut faire :



On a ajouté à droite un composant appelé dessin qui est combinatoire puisqu'il n'a pas d'horloge. Dessiner un rectangle consiste à faire des comparaisons entre `pixel_row` et `pixel_column` et des valeurs prédéfinies et sortir les couleurs correspondantes. Si vous voulez un deuxième rectangle, vous faites de même et vous faites un OU logique entre les couleurs avant de les sortir.

Le dessin d'un rectangle peut être trouvé dans l'exercice 2 de la page

[http://fr.wikiversity.org/wiki/Very\\_High\\_Speed\\_Integrated\\_Circuit\\_Hardware\\_Description\\_Language/Interfaces\\_VGA\\_et\\_PS/2](http://fr.wikiversity.org/wiki/Very_High_Speed_Integrated_Circuit_Hardware_Description_Language/Interfaces_VGA_et_PS/2)

### **Travail à réaliser (4 pts exo1)**

1°) A partir de l'exercice 1 et 2 (question 1 seule pour l'exercice 2) de la page

[http://fr.wikiversity.org/wiki/Very\\_High\\_Speed\\_Integrated\\_Circuit\\_Hardware\\_Description\\_Language/Interfaces\\_VGA\\_et\\_PS/2](http://fr.wikiversity.org/wiki/Very_High_Speed_Integrated_Circuit_Hardware_Description_Language/Interfaces_VGA_et_PS/2)

on vous demande d'écrire le fichier ucf et de montrer que l'on a bien dessiné un rectangle bleu sur notre écran VGA.

2°) Modifier le programme pour dessiner un rectangle vert en plus du bleu (un port map sans oublier le ou final)

### **Travail à réaliser (4 pts exo2)**

1°) Vous disposez lorsque vous arrivez à l'exercice 3 (question 1° de wikiversité) d'un afficheur sept segments sur écran VGA avec 4 entrées (donc le transcodage est fait) et une taille et une position à définir à l'aide d'un "generic map". Modifier la solution de l'exercice 3 question 2 pour n'avoir que deux afficheurs sept segments en milieu d'écran. Cela revient à retirer les deux raquettes, la balle et deux afficheurs et les signaux correspondants (pour les OU).

Votre entité globale à ce point sera donc :

```

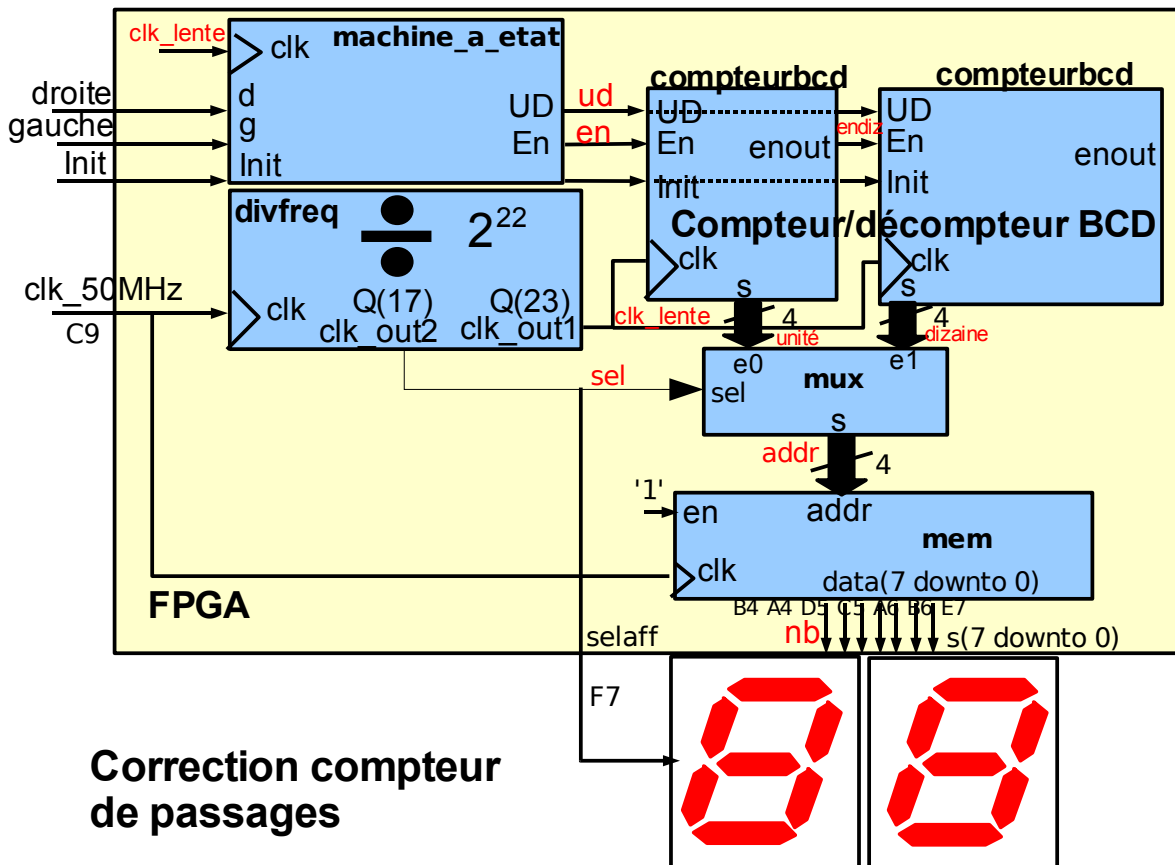
1      library IEEE;
2      use     IEEE.STD_LOGIC_1164.all;
3      --entité VGA globale exo2
4      ENTITY VGAtop IS
5          PORT (clk_50 : in STD_LOGIC; -- horloge 50MHz
6              -- valeurs à afficher sur deux digits
7                  DeuxDigits : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
8              -- en sortie nos cinq signaux VGA
9                  hsynch, vsynch, red, green, blue : out STD_LOGIC);
10     END VGAtop;
```

Que se passe-t-il au niveau de l'affichage après 9 ?

2°) Changer la couleur des afficheurs.

### **III) Compteur de passages sur écran VGA.**

Pour simplifier la lecture du fichier VHDL donné, on vous propose le schéma complet de la version corrigée (même figure que pour le médian).



Prenez le temps de remarquer les conventions du dessin, le nom des composants (en gras), le nom des entrées et sortie (à l'intérieur des rectangles bleus et du jaune) et le nom des signaux (en rouge).

### Travail à réaliser (4 pts exo3)

On vous demande

- de prendre le fichier source (en annexe plus loin) ou disponible sur internet (pour le médian)
- de retirer toute la partie après les compteurs/décompteurs BCD, c'est à dire le multiplexeur et le transcodage et de les remplacer par votre ensemble de la question précédente qui affiche sur écran VGA. On n'a plus besoin de multiplexeurs maintenant car on peut afficher les deux digits ensemble. On n'a plus besoin de selAff en sortie mais d'une sortie sur 8 bits qui existe mais qu'il faut reconnecter !
- de tester : le compteur de passages doit toujours fonctionner mais affiche ses valeurs sur l'écran VGA !

### IV) Le compteur de passages dans le picoBlaze

Nous allons maintenant interfacer le picoBlaze à un écran VGA. Notre objectif est donc de réaliser le schéma ci-après, c'est maintenant le picoBlaze qui va permettre de changer les valeurs affichées sur l'écran VGA.

### Travail à réaliser (3 pts exo4)

Réaliser la partie matérielle correspondant au schéma ci-après, sans oublier la mémoire "mpu\_rom" qui n'est pas dessinée.

Pour tester l'ensemble, on vous propose le programme picoBlaze suivant :

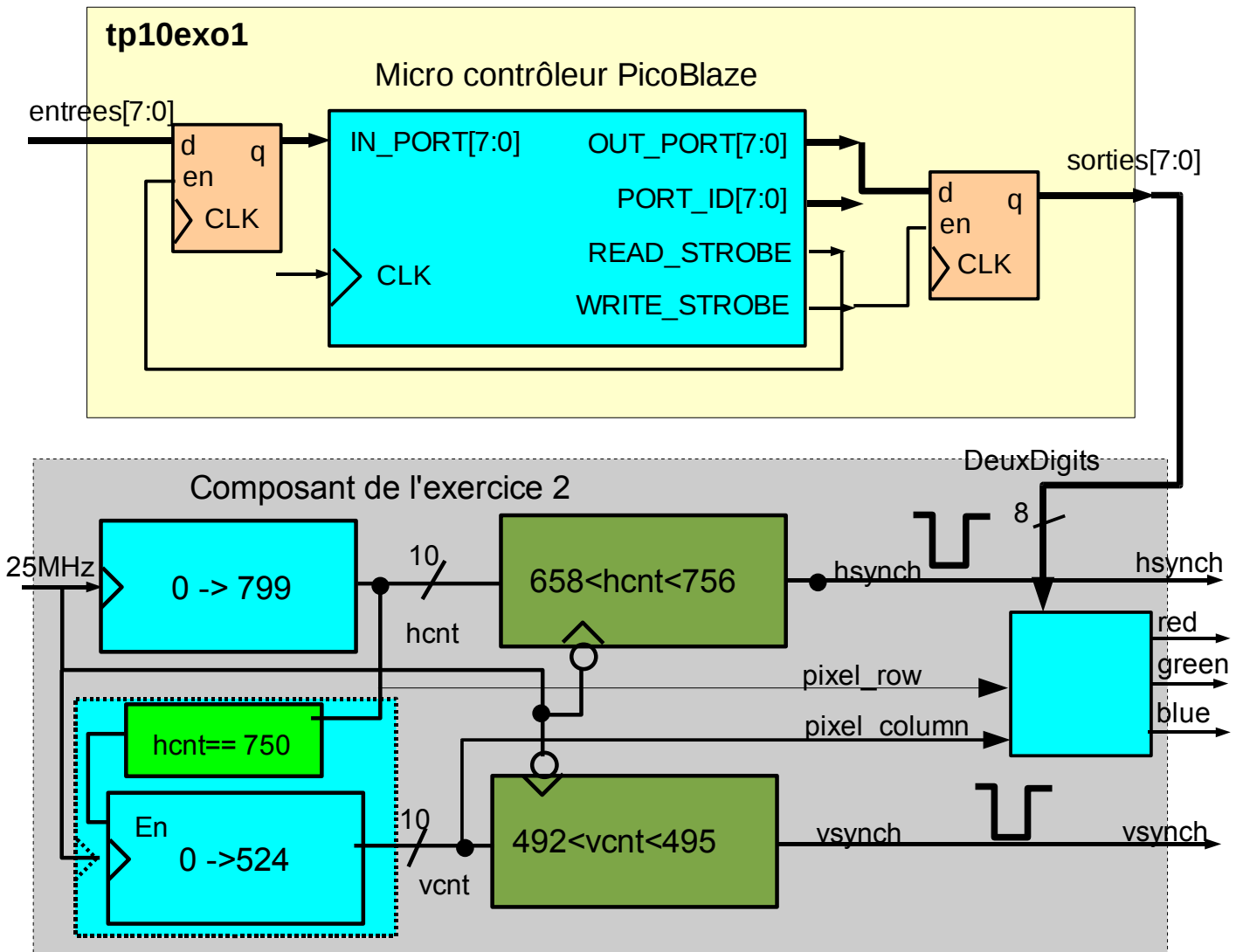
```

1           ;; examen final 2011
2           constant MAX, A0
3           namereg s0,i
4           NAMEREG s1, nb
5           namereg s2,j
6           namereg s3,k
7
8           init:

```

```
9          LOAD nb,00
10         debut:
11         ;lecture de l'état RS232
12         OUTPUT nb,0
13         CALL wait
14         ADD nb,01
15         JUMP debut
16         ;=== triple boucle d'attente===
17         wait:
18         LOAD i,MAX
19         loop:
20         LOAD j,MAX
21         loop_1:
22         LOAD k,MAX
23         loop_2:
24         SUB k,01
25         JUMP NZ, loop_2
26         SUB j,01
27         JUMP NZ, loop_1
28
29         SUB i,01
30         JUMP NZ,loop
31         RETURN
```

Vous constatez qu'à certains moments les afficheurs s'éteignent, c'est parce que les valeurs à afficher dépassent 9 (sur 4 bits) ! Modifier le programme pour qu'il compte en BCD. Le principe est que vous devez tester la partie basse. Si elle est plus grande que 9 on ajoute 6 au nombre. S'il le nombre est alors plus grand que 99 alors on le remet à 0.



La partie haute de cette figure **tp10exo1** correspond à l'exercice 1 du TP10 : cette partie est complètement donnée ci-après :

```

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 (
14                 address : out std_logic_vector(9 downto 0);
15                 instruction : in std_logic_vector(17 downto 0);
16                 port_id : out std_logic_vector(7 downto 0);
17                 write_strobe : out std_logic;
18                 out_port : out std_logic_vector(7 downto 0);
19                 read_strobe : out std_logic;
20                 in_port : in std_logic_vector(7 downto 0);
21                 interrupt : in std_logic;
22                 interrupt_ack : out std_logic;
23                 reset : in std_logic;
24                 clk : in std_logic);

```

```

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, s_entrees: std_logic_vector(7 downto 0);
32     signal s_write_strobe,s_read_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 => s_entrees,
40                     out_port => s_sorties,
41                     read_strobe => s_read_strobe,
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 des entrées
51     process(clk) begin
52         if rising_edge(clk) then
53             if s_read_strobe = '1' then
54                 s_entrees <= entrees;
55             end if;
56         end if;
57     end process;
58     -- mémorisation sorties
59     process(clk) begin
60         if rising_edge(clk) then
61             if s_write_strobe = '1' then
62                 s_sorties2 <= s_sorties;
63             end if;
64         end if;
65     end process;
66     sorties <= s_sorties2;
67 end atp10;

```

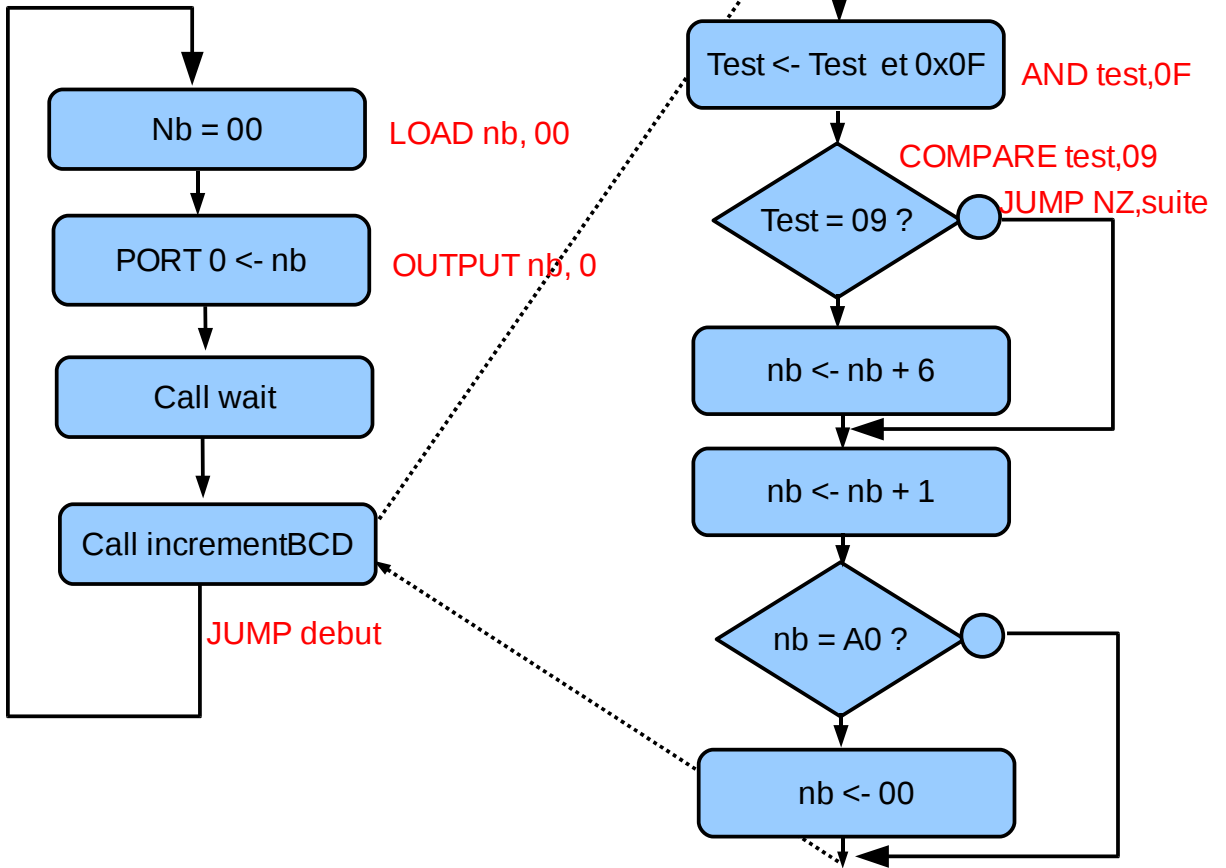
### **Travail à réaliser (5 pts exo5)**

1°) Modifier le programme donné pour avoir un compteur BCD correct. Il serait bon d'organiser votre programme de telle manière que l'incréméntation BCD soit un sous-programme. Tester.

**Indications** : organigramme en page suivante

### Sous-programme incrementBCD

### Programme principal



2°) Modifier le programme précédent pour avoir un sous-programme de décrémentation. Tester.

3°) Réaliser la partie logicielle qui réalise le séquenceur du compteur de passages et appelle les sous-programmes d'incrémentation ou de décrémentation suivant le cas. On ne s'intéressera pas à éviter les rebonds sur les interrupteurs.

## ANNEXE

```

--Version pour spartan 3E
-- Compteur de passage sur 2 digits
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

entity tp6_exo2 is port (
    gauche,droite: in std_logic;
    clk_50MHz : in std_logic;
    init : in std_logic;
    selaff : out std_logic; -- pour test sur carte spartan3E
    s: out std_logic_vector(7 downto 0)
);
end entity;

architecture behavior of tp6_exo2 is
    component machine_a_etat is port (
        clk,init : in std_logic;
        g,d : in std_logic;
        en,ud : out std_logic
    );
end component;
    component divfreq is port (
        clk : in std_logic;
        clk_out1,clk_out2 : out std_logic );
end component;
    component compteur is port (
        clk :in std_logic;
        init : in std_logic;
        s: out std_logic_vector(7 downto 0)
    );
end component;
    component compteurbcd is port (
        clk :in std_logic;
        en : in std_logic;
        ud: in std_logic;
        init : in std_logic;
        enout : out std_logic;
        s: out std_logic_vector(3 downto 0)
    );
end component;
    component mem is port (
        clk : in std_logic;
        en : in std_logic;
        addr : in std_logic_vector(3 downto 0);
        data : out std_logic_vector(7 downto 0)
    );
end component ;
    component mux is port (
        sel: in std_logic;
        e0,e1 : in std_logic_vector(3 downto 0);
        s : out std_logic_vector(3 downto 0)
    );
end component;
    signal clk_lente,sel,endiz : std_logic;
    signal addr,unite,dizaine : std_logic_vector(3 downto 0);
    signal nb : std_logic_vector(7 downto 0);
    signal en,ud : std_logic;

```

```

begin
  ic1: divfreq port map ( clk =>clk_50MHz, clk_out1=> clk_lente,clk_out2=>sel);
  ic2: compteurbcd port map( clk => clk_lente, ud=>ud,en=>en,init => init,
s=>unite,enout=>endiz);
  ic3: compteurbcd port map( clk => clk_lente, ud=>ud,en=>endiz,init => init,
s=>dizaine);
  ic4:mux port map ( sel=>sel, e0=>unite, e1=>dizaine, s=>addr);
  ic5 : mem port map ( clk=> clk_50MHz,en=>'1', addr => addr, data=> nb);
  ic6: machine_a_etat port map
(clk=>clk_lente,init=>init,g=>gauche,d=>droite,en=>en,ud=>ud);

  s <= nb; --- pour spartan3 ajouter not
  selaff <= sel; -- test spartan3E
end behavior;
----- couper le fichier ici -----
-- description du graphe
library ieee;
use ieee.std_logic_1164.all;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity machine_a_etat is port (
  clk,init : in std_logic;
  g,d : in std_logic;
  en,ud : out std_logic
);
end machine_a_etat;

architecture archi_machine of machine_a_etat is
  type type_etat is (s1,s2,s3,s4,s5,s6,s7);
  signal next_etat,reg_etat:type_etat;
begin
  valide_etat:process(clk)
    begin
      if rising_edge(clk) then
        if init='1' then
          reg_etat<=S1;
        else
          reg_etat<=next_etat;
        end if;
      end if;
    end process valide_etat;
  etat_suivant: process (reg_etat,d,g)
  begin
    next_etat<=reg_etat;
    case reg_etat is
      when S1=>
        if d='1' and g='0' then
          next_etat<=S2;
        elsif g='1' and d='0' then
          next_etat<=S3;
        else
          next_etat<=S1;
        end if;
      when S2=>
        if g='1' then
          next_etat<=S4;
        else
          next_etat<=S2;
        end if;
      when S3=>
        if d='1' then
          next_etat<=S5;

```

```

        else
            next_etat<=s3;
        end if;
    when S4=>next_etat<=S6;

    when S5=>next_etat<=S7;

    when S6=>
        if g='0' then
            next_etat<=S1;
        else
            next_etat<=S6;
        end if;
    when S7=>
        if d='0' then
            next_etat<=S1;
        else
            next_etat<=S6;
        end if;
    end case;
end process etat_suivant;
-- gestion des actions
process(reg_etat)
begin
    if reg_etat = s2 then
        ud <= '1';
    elsif reg_etat = s4 then
        ud <= '1';
    elsif reg_etat = s6 then
        ud <= '1';
    else
        ud <= '0';
    end if;
end process;
process(reg_etat)
begin
    if reg_etat = s4 then
        en <= '1';
    elsif reg_etat = s5 then
        en <= '1';
    else
        en <= '0';
    end if;
end process;
end archi_machine;

```

```

-- description de la memoire de transcodage

```

```

library ieee;
use ieee.std_logic_1164.all;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity mux is port (
    sel: in std_logic;
    e0,e1 : in std_logic_vector(3 downto 0);
    s : out std_logic_vector(3 downto 0)
);
end entity;
architecture behavior of mux is
begin
    with sel select
        s <= e0 when '0',
            e1 when others;

```

```

end behavior;

-- description de la memoire de transcodage
library ieee;
use ieee.std_logic_1164.all;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity mem is port (
    clk : in std_logic;
    en : in std_logic;
    addr : in std_logic_vector(3 downto 0);
    data : out std_logic_vector(7 downto 0)
);
end entity;
architecture behavior of mem is
    type rom_type is array (0 to 15) of std_logic_vector(7 downto 0);
    -- MSB
    signal rom : rom_type := (
        x"7E",x"30",x"6D",x"79",x"33",x"5B",x"5F",x"70",
        x"7F",x"7B",x"77",x"1F",x"0D",x"3D",x"4F",x"47");
    begin
        process(clk)
        begin
            if rising_edge(clk) then
                if en='1' then
                    data <= rom(conv_integer(addr));
                end if;
            end if;
        end process;
    end behavior;

-- description du premier composant : diviseur de frequence
library ieee;
use ieee.std_logic_1164.all;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity divfreq is port (
    clk : in std_logic;
    clk_out1,clk_out2 : out std_logic );
end entity;

architecture behavior of divfreq is
    signal n : std_logic_vector(23 downto 0);
begin
    -- division de la frequence de base de 50MHz vers 3Hz
    divfreq :process(clk) begin
        if clk'event and clk='1' then
            n <= n+1;
        end if;
    end process;
    clk_out1 <= n(23); -- visualisation de l'horloge
    clk_out2 <= n(17); -- gestion afficheurs 7 segments
end behavior;

-- description du composant compteur/decompteur cascadable
library ieee;
use ieee.std_logic_1164.all;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity compteurbcd is port (
    clk :in std_logic;

```

```

    en : in std_logic;
    init : in std_logic;
    ud : in std_logic;
    enout : out std_logic;
    s: out std_logic_vector(3 downto 0)
    );
end entity;

architecture behavior of compteurbcd is
signal n : std_logic_vector(3 downto 0);
begin
increment : process(clk) begin
    if clk'event and clk='1' then
        if init='1' then
            n <= (others => '0');
        elsif en='1' then
            if ud = '1' then --up
                if n<9 then
                    n <= n + 1 ;
                else
                    n <= (others => '0');
                end if;
            else -- down
                if n=0 then
                    n <= "1001";
                elsif n<10 then
                    n <= n - 1 ;
                else
                    n <= (others =>'0');
                end if;
            end if;
        end if;
    end if;
end process;
enableout: process(n,en,ud)
begin
    if en='1' then
        if ud='1' and n=9 then
            enout<= '1';
        elsif ud='0' and n=0 then
            enout<='1';
        else
            enout<='0';
        end if;
    else -- ajouté 4/5/2011 pour éviter comptage intempestif sur dizaine
        enout<='0';
    end if;
end process;
s <= n;
end behavior;

```