Règles de déroulement de l'épreuve

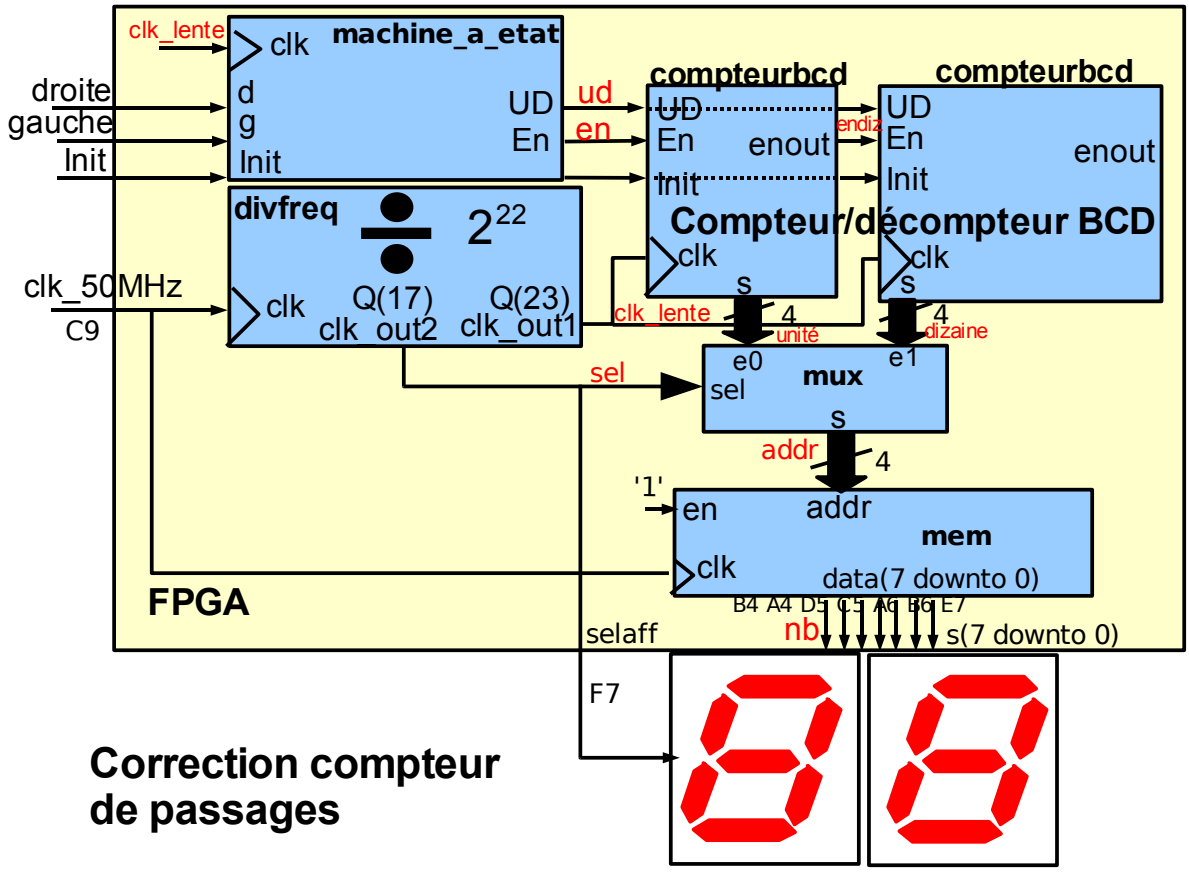
Il n'est pas interdit de communiquer verbalement entre binômes tant que cela ne perturbe pas le déroulement correct de cette séance.
Il vous est absolument interdit, par contre, de vous déplacer, d'échanger des feuilles de brouillons, des notes ainsi que des fichiers.

Introduction

Nous allons partir d'une version corrigée du compteur de passages du TP6 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.

I) Compteur de passages corrigé.

Pour simplifier la lecture du fichier VHDL , on vous propose un schéma complet de la version corrigée.



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

Travail à réaliser (3 pts)

On vous demande
- de prendre le fichier source (en annexe plus loin) ou disponible sur internet
- de le couper en deux (là où cela est indiqué)

- de construire le fichier ucf
- de réaliser un projet comprenant les trois fichiers (2 VHD + 1 UCF)
- de compiler en corrigeant les éventuelles petites erreurs, d'essayer l'ensemble et de faire valider.

Le travail qui suit consiste à modifier le compteur de passages en utilisant les composants tout faits (disponibles en schématique). C'est pour cela que l'on vous a fait couper le fichier en deux. On gardera le deuxième fichier (l'implantation des composants bleus) intact et vous n'aurez à modifier que le premier fichier petit à petit même si petit à petit vous allez abandonner certains composants de ce deuxième fichier.

Remarque : l'utilisation des composants de la schématique en VHDL nécessite absolument l'utilisation de la librairie unisim (comme indiqué ci-dessous) dans le fichier du haut de la hiérarchie (entité "**tp6_exo2**").

```
1             library IEEE;
2             use IEEE.STD_LOGIC_1164.ALL;
3             library unisim;
4             use unisim.vcomponents.all;
```

II) Modification du transcodeur

Nous avons déjà utilisé une mémoire mais maintenant nous allons le faire avec un composant tout fait de UNISIM.

1°) Présentation de la mémoire utilisée

Nous allons utiliser une **RAM16X8S**. Pour celle-ci, lisez attentivement le programme d'exemple ci-dessous. Vous en déduirez la façon de l'initialiser : comme des LUTs, sortie par sortie.

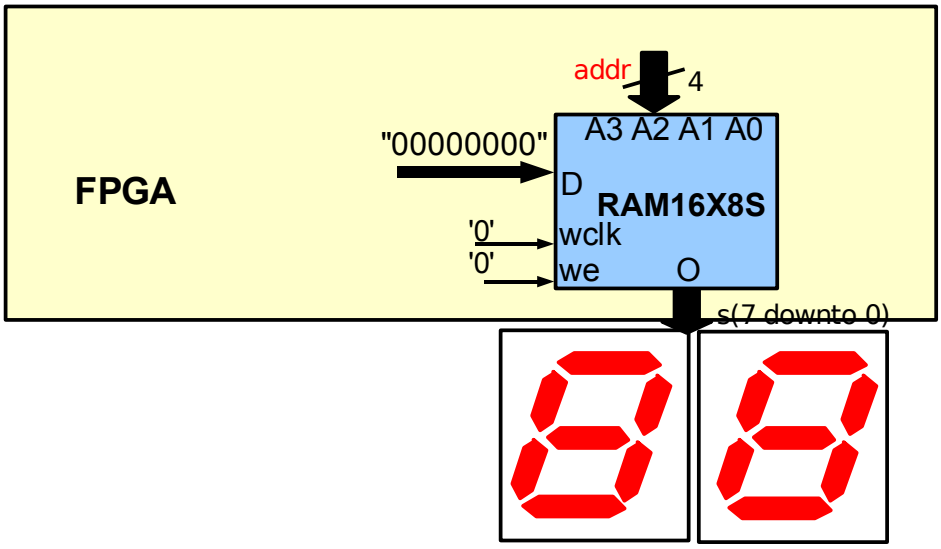
Indication : voici comment est instanciée la RAM16X8S en VHDL :

```
1             Library UNISIM;
2             use UNISIM.vcomponents.all;
3             -- RAM16X8S: 16 x 8 posedge write distributed  => LUT RAM
4             --           Virtex-II/II-Pro
5             -- Xilinx HDL Libraries Guide, version 10.1.2
6             RAM16X8S_inst : RAM16X8S
7             generic map (
8             INIT_00 => X"0000", --  INIT for bit 0 of RAM
9             INIT_01 => X"0000", --  INIT for bit 1 of RAM
10            INIT_02 => X"0000", --  INIT for bit 2 of RAM
11            INIT_03 => X"0000", --  INIT for bit 3 of RAM
12            INIT_04 => X"0000", --  INIT for bit 4 of RAM
13            INIT_05 => X"0000", --  INIT for bit 5 of RAM
14            INIT_06 => X"0000", --  INIT for bit 6 of RAM
15            INIT_07 => X"0000") --  INIT for bit 7 of RAM
16            port map (
17            O => O,         -- 8-bit RAM data output
18            A0 => A0,       -- RAM address[0] input
19            A1 => A1,       -- RAM address[1] input
20            A2 => A2,       -- RAM address[2] input
21            A3 => A3,       -- RAM address[3] input
22            D => D,         -- 8-bit RAM data input
23            WCLK => WCLK,   -- Write clock input
24            WE => WE        -- Write enable input
25            );
26            -- End of RAM16X8S_inst instantiation
```

La documentation des composants UNISIM que l'on va utiliser peut être facilement trouvée avec l'outil schématique. Chercher le composant en question dans les symboles et dessiner-le dans la partie schématique puis en marquant le composant :
"**click droit ->Object properties -> Symbol Info**"
L'information est en anglais mais elle a le mérite d'exister.

2°) Travail à réaliser (4 pts)

Vous allez remplacer le composant "**mem**" par une **RAM16X8S**. Celle-ci comprend des entrées et sorties supplémentaires. Elle est donc à câbler comme suit :



Donner sur votre feuille réponse toutes les valeurs d'initialisation de la **RAM16X8S** en prévoyant un affichage hexadécimal.
Modifier l'architecture de "tp6_exo2" conformément au schéma ci-dessus en supprimant le composant "**mem**".
Compiler et valider le fonctionnement du compteur de passages.

III) Remplacement du compteur décimal cascadé

Nous allons maintenant remplacer le compteur - décompteur décimal cascadé par un simple compteur décompteur binaire sur 8 bits. Nous nous intéresserons ensuite à la conversion binaire vers décimale.

Travail à réaliser (3 points)

Modifier l'ensemble pour ajouter le compteurs décompteur 8 bit à la place des deux compteurs BCD cascadés.

Indications : Voici l'entité à implanter et l'architecture à compléter.

```
1               library ieee;
2               use ieee.std_logic_1164.all;
3               use IEEE.STD_LOGIC_ARITH.ALL;
4               use IEEE.STD_LOGIC_UNSIGNED.ALL;
5               ENTITY compteurbin8 is port (
6                         clk :in std_logic;
7                         en : in std_logic;
8                         ud: in std_logic;
9                         init : in std_logic;
10                        s8: out std_logic_vector(7 downto 0)
11                        );
12              end compteurbin8;
13
```

```
14              ARCHITECTURE acmpt8 of compteurbin8 is
15              signal cmpt8 : std_logic_vector(7 downto 0);
16              begin
17                process(clk,init) begin
18                  if init='1' then cmpt8 <= x"00";
19                  elsif rising_edge(clk) then
20                   -- ???????????? a completer
21                  end if;
22                end process;
23                s8 <= cmpt8;
24              end acmpt8;
```

IV) Réalisation de la conversion binaire décimale

La conversion binaire vers le décimal est réalisée avec un décompteur binaire, un compteur BCD cascadé et un séquenceur.

1°) Travail de préparation à faire (3 pts)

A partir du morceau de programme ci-dessous qui réalise à proprement parler la conversion, on vous demande d'en faire un schéma complet :

```
1               entity convertisseur is port (
2                    clk, en, init : in std_logic;
3                    entreebin8 : in std_logic_vector(7 downto 0);
4                    unite, dizaine : out std_logic_vector(3 downto 0));
5               end convertisseur;
6
7               architecture convert of convertisseur is
8                 component CB8EDL is port (
9                         clk :in std_logic;
10                        en : in std_logic;
11                        load : in std_logic;
12                        e8 : in std_logic_vector(7 downto 0);
13                        done : out std_logic;
14                        s8: out std_logic_vector(7 downto 0)
15                        );
16                end component;
17                component cmpt_bcd10 IS -- Définition des entrées/sorties
18                    PORT(clk, en, Init : IN std_logic;
19                    rco : OUT std_logic;
20                    q : OUT std_logic_vector(3 downto 0));
21                END component;
22                component sequenceur is
23                  port (
24                    clk,en,done,init : in std_logic;
25                    endec,load,memorize : out std_logic);
26                end component;
27                signal en_seq,s_done,s_load,s_rco,s_memorize : std_logic;
28                signal regAff : std_logic_vector(7 downto 0);
29                signal s_dizaine, s_unite : std_logic_vector(3 downto 0);
30              begin
31                conv1:CB8EDL port map (
32                  clk => clk,en => en_seq,load=>s_load,e8 =>entreebin8,done => s_done,
33                  s8(3 downto 0) => open,s8(7 downto 4)=>open);
34                conv2:sequenceur port map (
35                  clk=>clk,en => en, endec => en_seq,load=>s_load,done => s_done,init =>
                init,
36                  memorize=>s_memorize);
37                conv3:cmpt_bcd10 port map (
38                  clk => clk,en => en_seq, Init => s_load,rco => s_rco, q => s_unite);
39                conv4:cmpt_bcd10 port map (
```

```
40                clk => clk,en => s_rco, Init => s_load,rco => open, q => s_dizaine);
41            --   registre d'affichage
42              process(clk) begin
43                if rising_edge(clk) then
44                  if Init = '1' then
45                       regAff <= x"00";
46                    elsif s_memorize='1' then
47                     regAff <= s_dizaine & s_unite;
48                  end if;
49                end if;
50              end process;
51              dizaine <= regAff(7 downto 4);
52              unite <= regAff(3 downto 0);
53            end convert;
```

Indication : vous disposez d'un outil automatique pour vous aider à faire cela :

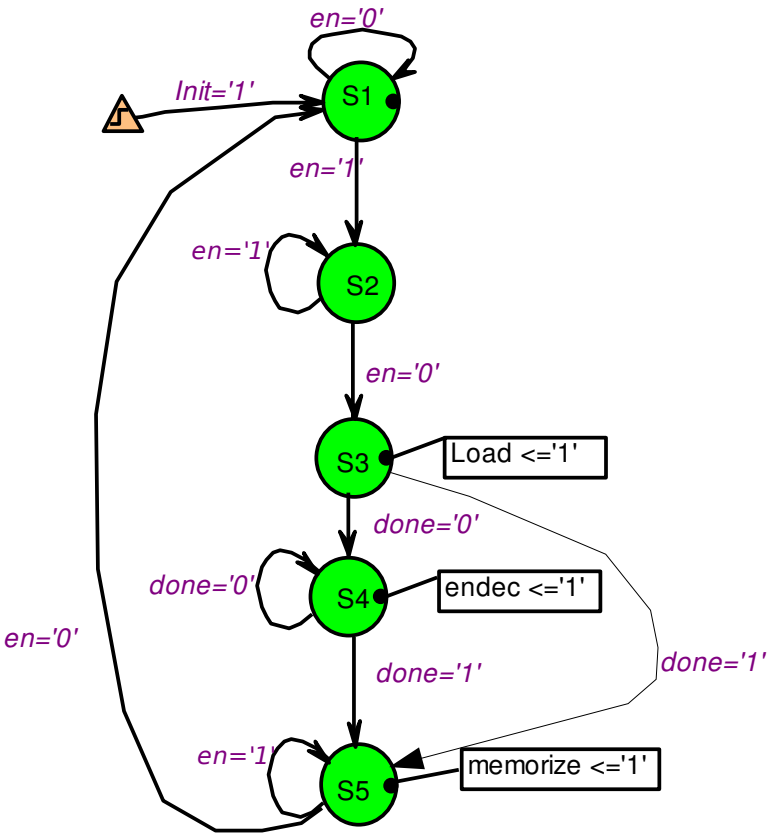
"synthese XST"->"View RTL Schematic"

mais il ne fonctionne que lorsque le projet est complet. Pensez donc à vérifier votre schéma après la question suivante.

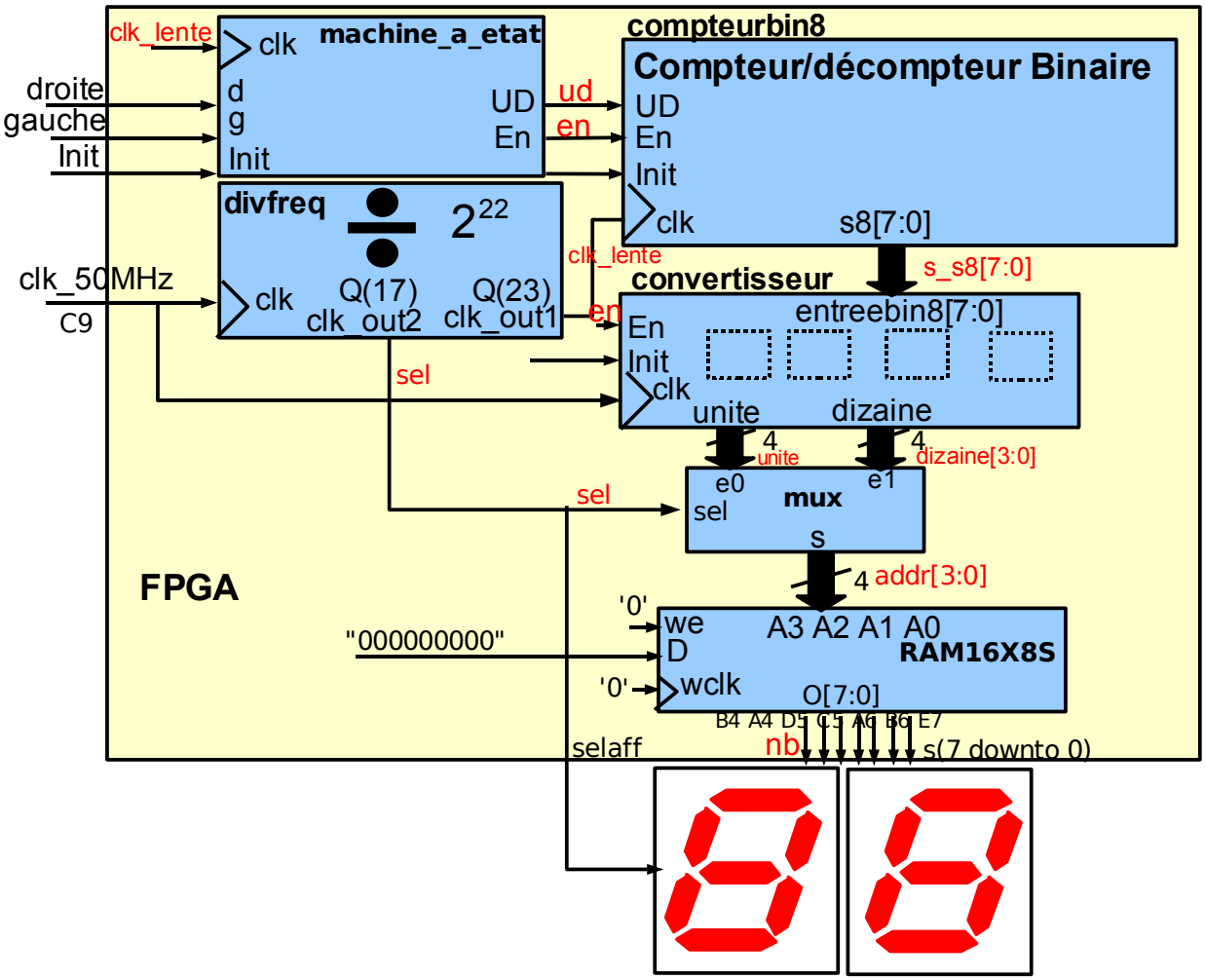
2°) Travail d'implantation du séquenceur (4 pts)

On vous demande de réaliser l'architecture du séquenceur en respectant le graphe d'évolution ci-dessous.

Attention : pour vos tests éviter de commencer par une décrémentation. La valeur 255 sera forcément mal convertie !!!! C'est le cas de toute valeur qui dépasse la valeur décimale "99". Si l'on voulait palier à cela il faudrait un troisième compteur décimal et un troisième afficheur.
Quand les actions ne sont pas à '1' elles sont supposées être à '0' !



Le convertisseur est naturellement à insérer dans le schéma de la manière suivante :



Indications

Pour vous aider à tester on vous donne le code du décompteur binaire (composant du convertisseur) :

```
1               library ieee;
2               use ieee.std_logic_1164.all;
3               use IEEE.STD_LOGIC_ARITH.ALL;
4               use IEEE.STD_LOGIC_UNSIGNED.ALL;
5               ENTITY CB8EDL is port (
6                         clk :in std_logic;
7                         en : in std_logic;
8                         load : in std_logic;
9                         e8 : in std_logic_vector(7 downto 0);
10                        done : out std_logic;
11                        s8: out std_logic_vector(7 downto 0)
12                        );
13              end CB8EDL;
14
15              ARCHITECTURE aCB8EDL of CB8EDL is
16              signal cmpt8 : std_logic_vector(7 downto 0);
17              begin
18                process(clk) begin
19                  if rising_edge(clk) then
20                    if load = '1' then
21                         cmpt8 <= e8;
22                    elsif en = '1' then cmpt8 <= cmpt8 - 1;
23                       end if;
24                  end if;
25                end process;
26                s8 <= cmpt8;
```

```
27              with cmpt8 select
28                done <= '1' when x"00",
29                        '0' when others;
30            end aCB8EDL;
```

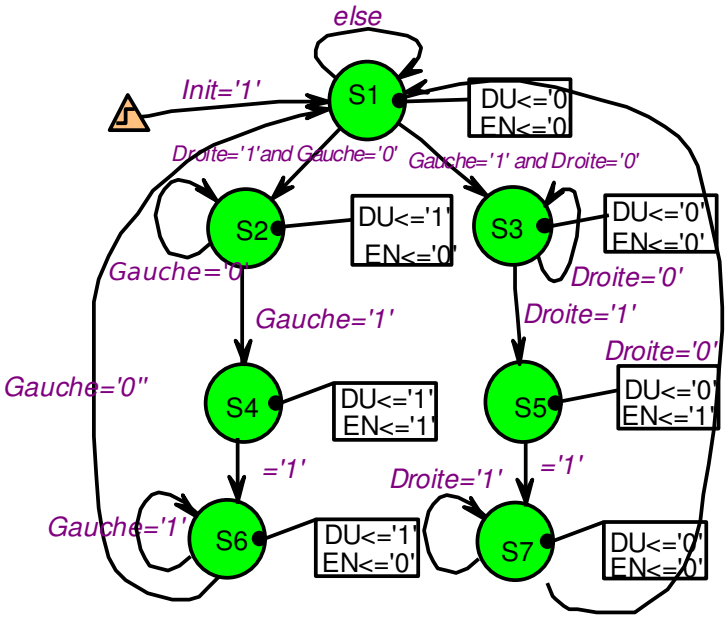
Vous devez avoir aussi quelque part le compteur décimal cascadable mais celui-ci est particulier, il s'initialise à 9. C'est un composant du convertisseur :

```
1             --******* Description : Compteur BCD de 0 a 9 initialisé à 9 *****
2             library IEEE;                -- La librairie IEEE
3             use IEEE.std_logic_1164.all;
4             use IEEE.STD_LOGIC_ARITH.ALL;
5             use IEEE.STD_LOGIC_UNSIGNED.ALL;
6             ENTITY cmpt_bcd10 IS -- Définition des entrées/sorties
7                 PORT(clk, en, Init : IN std_logic;
8                 rco : OUT std_logic;
9                 q : OUT std_logic_vector(3 downto 0));
10            END cmpt_bcd10;
11            ARCHITECTURE behav OF cmpt_bcd10 IS
12            SIGNAL cnt : std_logic_vector(3 downto 0); -- signal interne
13            BEGIN
14              q <= cnt; -- q, la sortie vaut la valeur du compte actuel en tout temps
15              PROCESS(clk, Init)       -- Process sensible à l'horloge et au "clear"
16              BEGIN
17                IF (Init='1') THEN                    -- "clear" asynchrone
18                  cnt <= x"9";
19                ELSIF (rising_edge(clk)) THEN -- au front montant
20                  IF (en='1') THEN -- si enable est à 1
21                    IF (cnt = 9) THEN -- Si on atteint 9 on fait un rco
22                      cnt <= x"0"; -- et on remet le compteur a 0
23                    ELSE -- Sinon on compte
24                      cnt <= cnt + 1;
25                    END IF;
26                  END IF;
27                END IF;
28              END PROCESS;
29              PROCESS(cnt,en) BEGIN
30                 IF cnt=9 and en='1' then
31                    rco <='1';
32                 ELSE
33                    rco <='0';
34                 END IF;
35              END PROCESS;
36            END behav;
```

3°) Travail final à faire (3 pts)

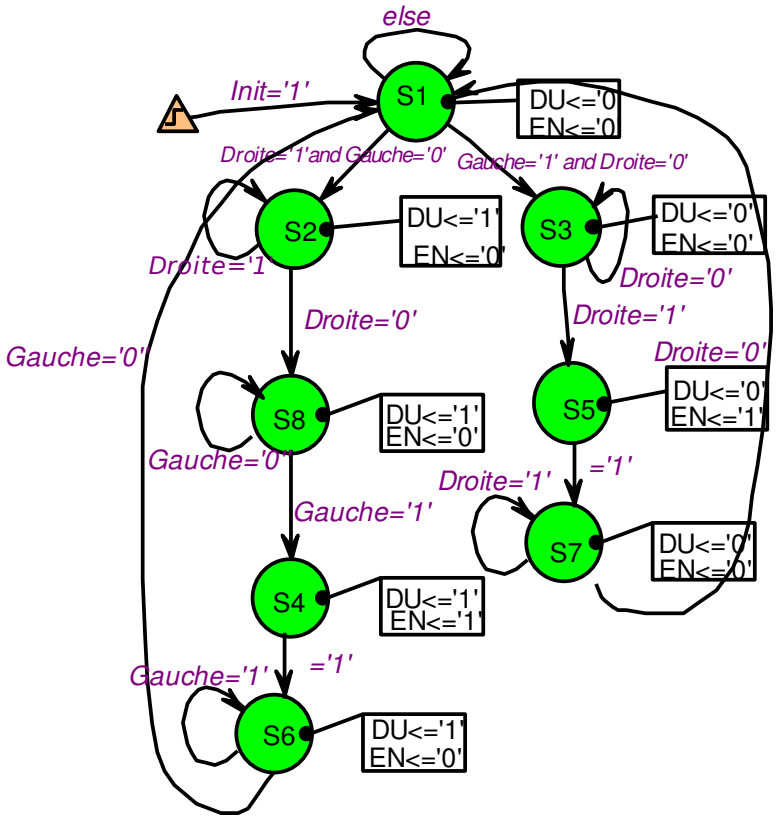
On rappelle que le séquenceur du compteur de passage est décrit par un graphe d'évolution rappelé ci-dessous. Ce graphe est réalisé par un composant appelé "**machine_a_etat**" dans la correction.
Le graphe d'évolution a été programmé avec une technique appelée "deux process" présentée en cours (voir polycopié LO11).

Le graphe d'évolution du compteur de passages est présenté ci-dessous. Dans ce graphe, le cas où les deux capteurs sont activés simultanément n'est pas géré (On reste dans S1 avec la boucle ELSE qui est une notation pas très standard !)



Modification du graphe d'évolution

Examinons la branche gauche du graphe d'évolution et particulièrement la suite S1 -> S2 -> S4.
Imaginons la situation :
Droite est activé (par une personne qui rentre) et peu après gauche est activé (par une personne qui sort) avant que droite soit désactivée ! Le tout sera considéré comme une personne qui entre et incrémentera le compteur ! C'est ce type de situation que l'on veut gérer en tout cas partiellement car notre solution suppose que les personnes soient polies et bien éduquées pour que la deuxième arrivée fasse demi-tour. Voici donc le nouveau graphe d'évolution :



Compléter le graphe d'évolution ci-dessus pour que la branche de droite soit symétrique de la branche de gauche.
Programmer ce nouveau graphe d'évolution en modifiant le composant "**machine_a_etat**".
Compiler et tester. Faire valider.

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 eviter comptage intempestif sur dizaine
              enout<='0';
            end if;
      end process;
      s <= n;
end behavior;
```