Règles de déroulement de l'épreuve

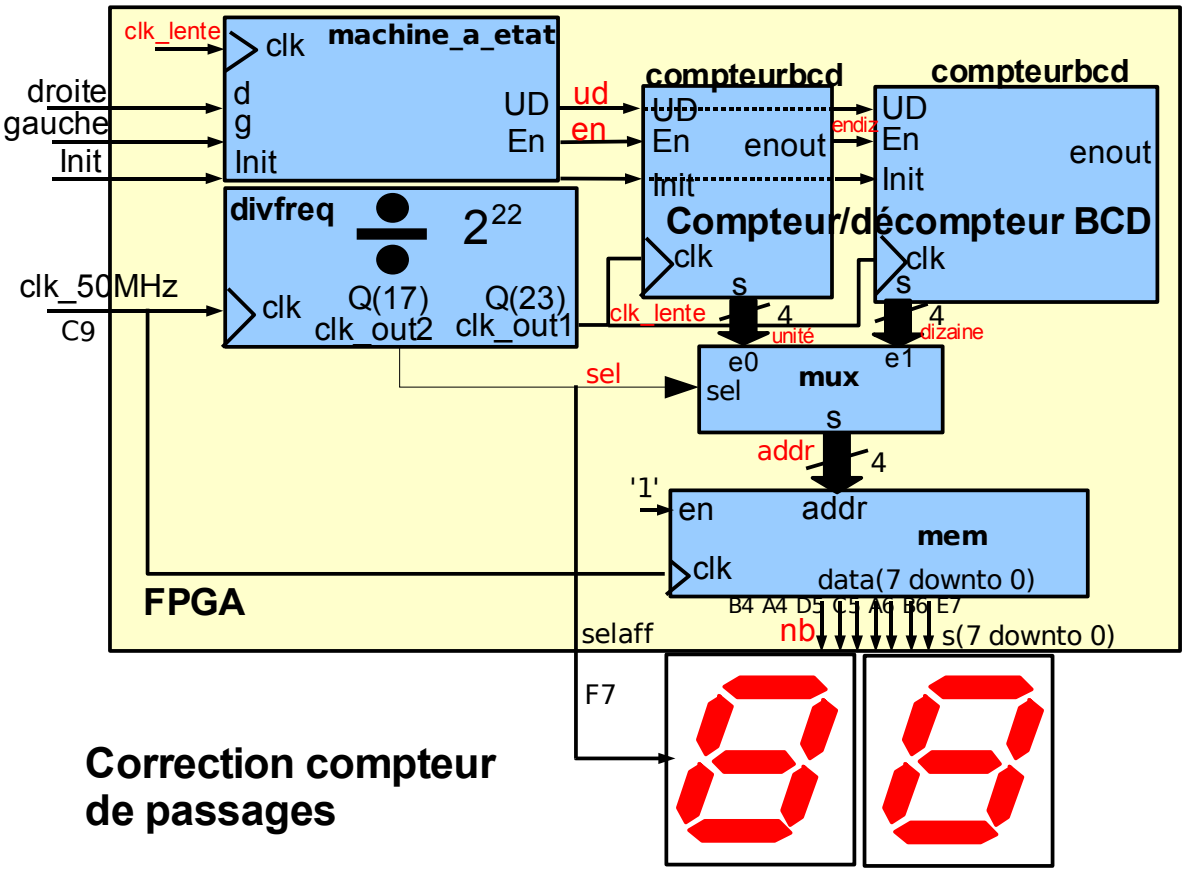
Il n'est pas interdit de communiquer verbalement entre binômes tant que cela ne perturbe pas le déroulement correct de cette séance.
Il vous est absolument interdit, par contre, de vous déplacer, d'échanger des feuilles de brouillons, des notes ainsi que des fichiers.

Introduction

Nous allons partir d'une version corrigée du compteur de passages du TP6 et en faire un certain nombre de modifications. Engagez-vous dans cette épreuve avec comme état d'esprit, de suivre les indications sans trop vous poser de questions sur le pourquoi 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.

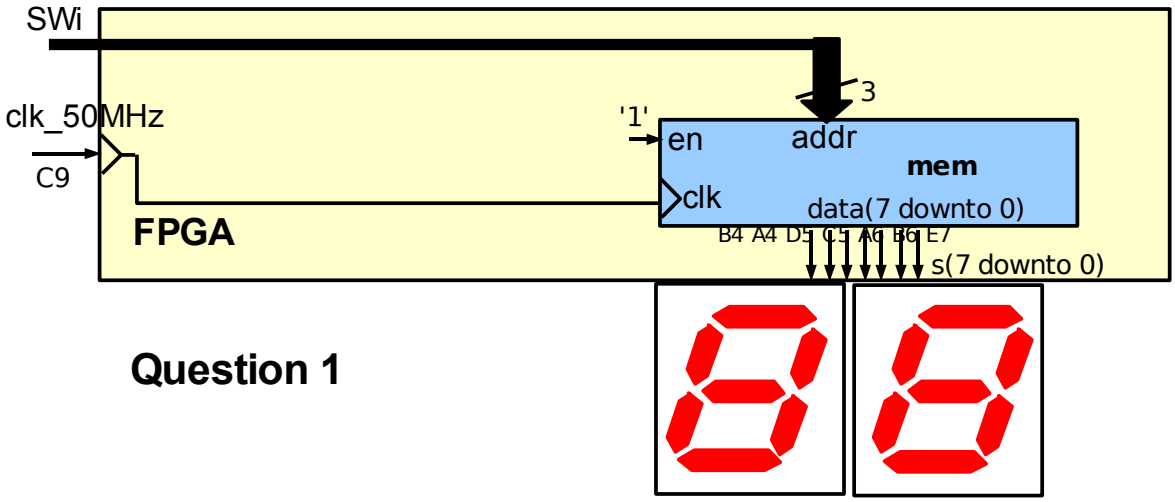


Prenez le temps de remarquer les conventions du dessin, le nom des composants (en gras), le nom des entrées et sortie (à l'intérieur des rectangles bleus et du jaune) et le nom des signaux (en rouge).
Nous allons modifier cet ensemble pour réaliser petit à petit un ensemble capable de simuler le lancer de un ou deux dés. Une des caractéristique d'un dé c'est qu'il affiche un résultat entre un et 6. Nous allons donc modifier petit à petit cet ensemble pour réaliser notre objectif. Votre travail consistera à piocher dans le fichier de correction et y puiser tout ce qui a déjà été réalisé pour une réutilisation et a y apporter toutes les modifications nécessaires.
Le fichier contenant la version corrigée peut être trouvée dans UTTLO11.zip (sur mon site) dans le répertoire

TP7 (fichier median.vhd) ou en fin de ce fichier pdf.

Affichage 1 à 6 (4 pts)

On désire afficher les nombres variant de 1 à 6. Dimensionner et modifier le contenu de la mémoire destinée à transcoder pour gérer une entrée sur 3 bits seulement. Si l'entrée binaire est 0 ou 7 ce transcodeur devra afficher un tiret sur le segment "g". Pour les tests les trois entrées seront reliées à des boutons poussoir (3 seulement)
On vous demande :
- de prendre le fichier source (en annexe plus loin) ou disponible sur internet et de chercher la mémoire et de la mettre dans son propre fichier
- de construire l'entité globale et son architecture dans un fichier séparé sans oublier de connecter l'ensemble en déclarant la mémoire comme composant.
- 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.

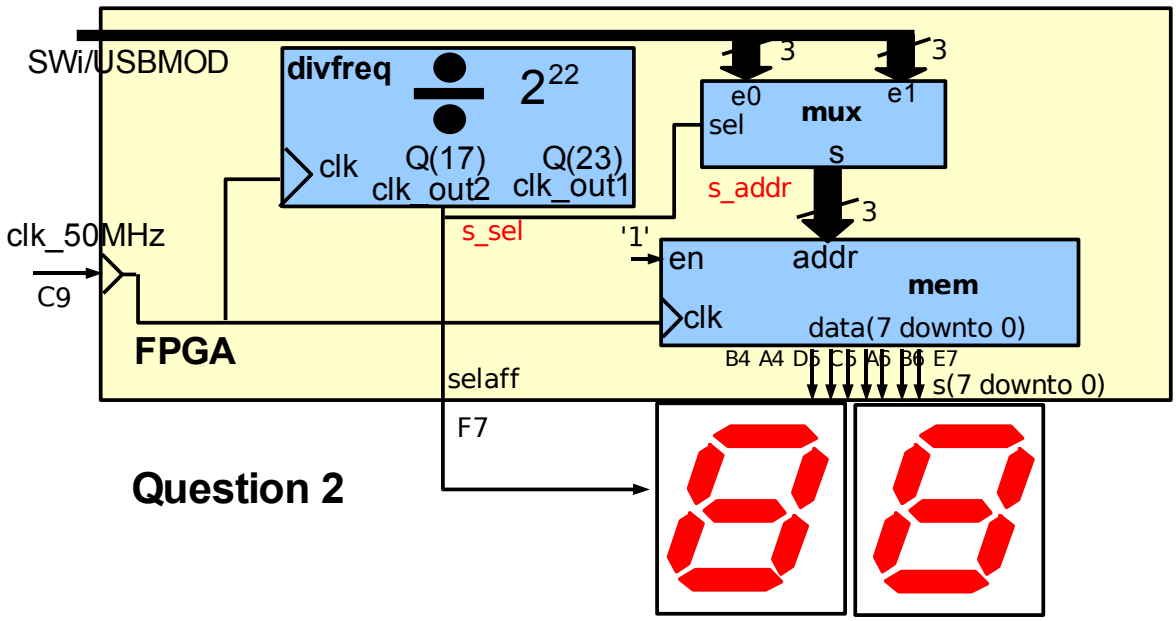


On ne gère pas les deux affichages pour le moment mais un seul.

II) Travail à réaliser (5 pts)

On ajoute maintenant le multiplexeur qui possède toujours deux entrées mais de trois bits seulement.
On vous demande
- de prendre le multiplexeur du fichier source (en annexe plus loin) ou disponible sur internet
- de le dimensionner pour trois bits et de l'ajouter au fichier qui contient la mémoire
- de prendre le diviseur du fichier source et de le mettre aussi dans le fichier qui contient la mémoire
- de modifier le fichier ucf pour prendre les 6 interrupteurs d'USBMOD
- 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.

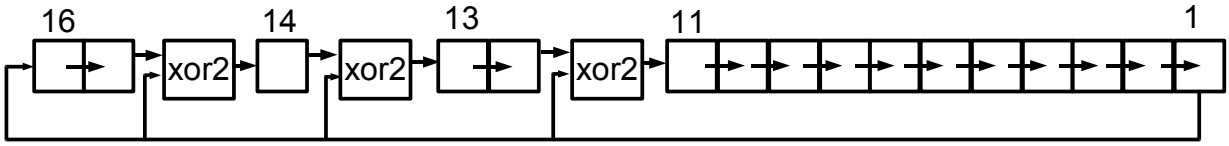
Le schéma de principe est en page suivante.



III) Réalisation d'un registre LFSR (5 pts)

Le graphe d'états est réalisé par un composant appelé "**machine_a_etat**" dans la correction. Lisez-le pour vous imprégnez du style deux process que l'on va utiliser dans cette question. Autrement référez vous à votre polycopié de cours p 119 et suivantes.

Les LFSR (registres à décalage à rétroaction linéaire) sont une famille de générateurs pseudo-aléatoires. La taille (n en bits) du registre détermine sa périodicité (2^n-1). Même si 6 bits nous suffisent nous allons gérer un générateur de Galois de 16 bits dont voici le schéma tiré de Wikipédia (http://en.wikipedia.org/wiki/Linear_feedback_shift_register):



C'est un grand registre à décalage séparé par des calculs simples avec des ou exclusifs. Si l'on prend 16 bits pour n'en utiliser que 6 c'est uniquement pour avoir une périodicité assez grande.

1°) Compléter sur la feuille réponse le schéma de calcul de l'état futur en fonction de l'état présent.

2°) En déduire les équations de récurrences bits à bits. Si vous ne voulez pas les écrire bit à bit vous utiliserez l'opérateur '&' VHDL qui définit une concaténation. Réaliser ce LFSR en utilisant deux process, un qui calcule de manière combinatoire l'état futur en fonction de l'état présent et un qui sur front d'horloge mettra à jour l'état présent.

Indications :

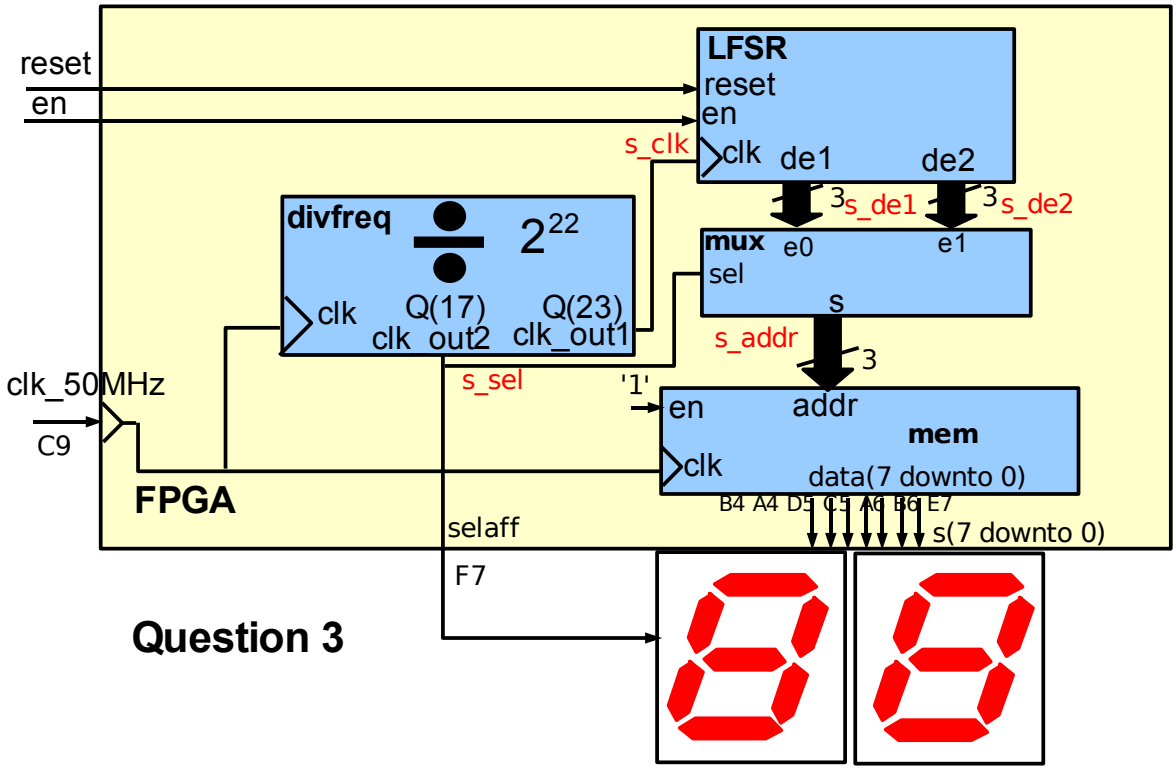
```
architecture aLFSR of LFSR is
signal etatpresent, etatfutur : std_logic_vector(16 downto 1);
signal s_xor1, s_xor2, s_xor3 : std_logic;
begin
  -- Calcul intermediaire des ou exclusifs
  s_xor1 <= etatpresent(15) xor etatpresent(1);
  s_xor2 <= etatpresent(14) xor etatpresent(1);
  s_xor3 <= etatpresent(12) xor etatpresent(1);
  -- Calcul de l'état futur en fonction de l'état présent et des ou exclusifs
  process(etatpresent) begin
    etatfutur(16) <= etatpresent(1);
```

```
    -- A compléter

  end process;
  process(clk,reset) begin
    -- A compléter : gestion du "reset" et du "en"
    -- Inspirez-vous du poly de TD/TP p28
  end process;
 -- cablage des deux sorties
  de1 <= etatpresent(16 downto 14);
  de2 <= etatpresent(3 downto 1);
end aLFSR;
```

Il faut gérer un reset qui initialise à une autre valeur que 0 !!! "0000000000000001" est une bonne valeur. Insérer ce LFSR comme indiqué dans le schéma ci-dessous et faire constater. Il vous faut aussi gérer une entrée "en" pour enable (en français autoriser) "l'incrémentation".

Faire constater.



IV) Ajout et modification du séquenceur (9 pts)

Pour que le joueur puisse déterminer quand il lance les dés, il nous faut ajouter un graphe d'états.

1°) Détection des mauvais lancers (2 points)

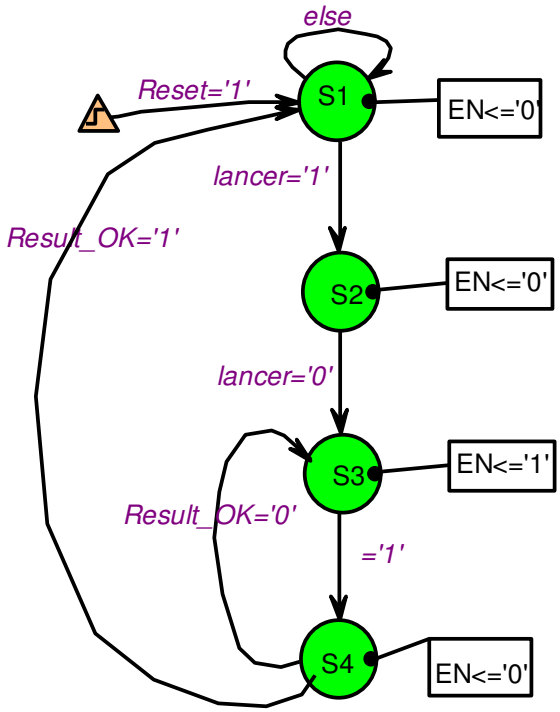
Sur le schéma ci-dessus les résultats de lancer de dés se trouvent dans deux signaux séparés appelés "de1" (pour dé1, les accents étant interdits) et "de2". Construire une fonction combinatoire capable de détecter que ni dé1 ni dé2 sont à 0 ou à 7.

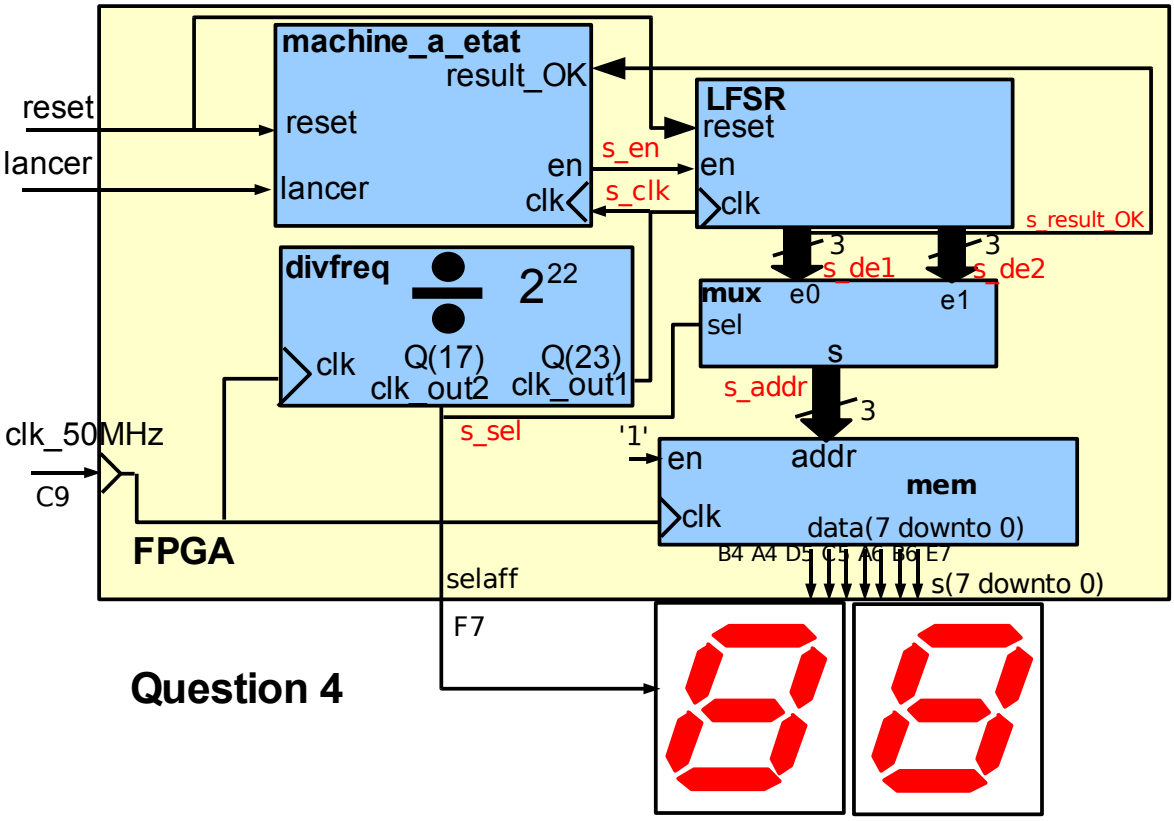
Compléter l'équation correspondante sur votre feuille réponse : Result_NOK et Result_OK. Il est plus facile de raisonner sur Result_NOK qui est à '1' si l'un des dés affiche un résultat incorrect (0 ou 7).

2°) Présentation du graphe d'évolution

Nous allons ajouter un graphe d'évolution qui a pour objectif de n'afficher les deux lancers que lorsque l'entrée "lancer" est positionnée à un. D'autre part ce graphe sera responsable d'un nouveau lancer au cas où le résultat est zéro ou sept.
Voici donc le nouveau graphe d'évolution ci-contre.

Si vous n'avez plus assez de temps (moins de 30 mn) aller directement en question 5°)





3°) Travail à réaliser (4 points)

Programmer ce nouveau graphe d'évolution en modifiant le composant "**machine_a_etat**". L'insérer ensuite dans le composant global et le câbler. La réalisation du signal "s_Result_OK" sera fait avec une simple

équation.
Compiler et tester. Faire valider.

4°) Travail final (3 points)

Si vous avez testé la question 3, vous vous êtes aperçu que les résultats intermédiaires non valides sont affichés avec un tiret. Si l'affichage est seulement mis à jour pour un lancé correct, modifier l'ensemble pour que les résultats intermédiaires non valides ne soient plus affichés.

Indications : Cela peut être réalisé de manière séquentielle en ajoutant un état S5 au graphe d'états qui gère l'autorisation d'écriture "enAff" dans un registre d'affichage placé entre le **LFSR** et le **mux**. Ce registre doit être initialisé à une valeur fixe par le "reset" pour affichage au démarrage, et son entrée "en" sera reliée à la sortie "enAff" du séquenceur.

Pouvez-vous dire pourquoi on ne peut pas mettre le registre d'affichage entre **mux** et afficheur ?

Le problème de cette solution est qu'il faut parfois attendre pour avoir l'affichage et que l'utilisateur peut s'impatienter ! Vous pouvez ajouter un chenillard tant que l'affichage n'est pas correct.

5°) Travail de substitution (2 points)

Vous êtes arrivé ici s'il ne vous reste que peu de temps pour finir l'épreuve. Plutôt que de tout décrire en VHDL, on vous demande de décrire en français le fonctionnement de l'ensemble de manière à me convaincre que vous avez bien compris le fonctionnement de l'ensemble.

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;
```