

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.

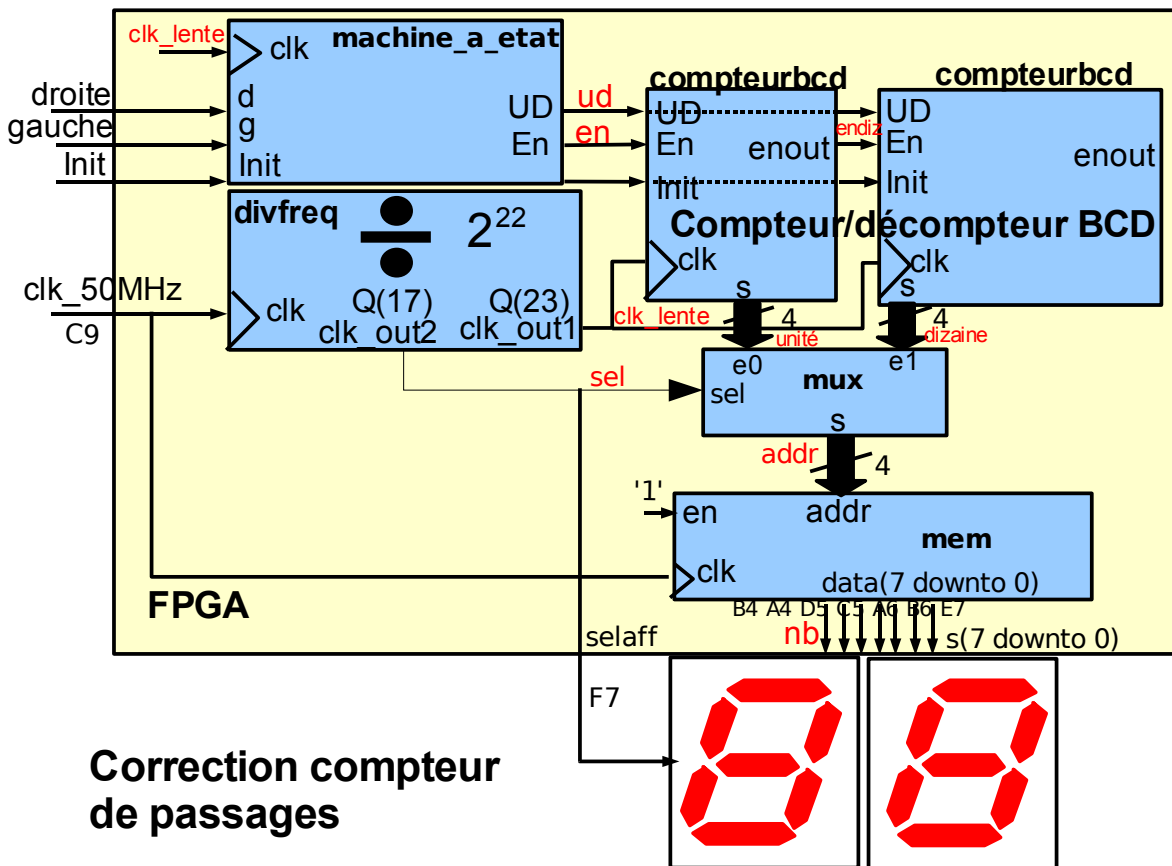
Introduction

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.

1) 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).

Travail à réaliser (4 pt)

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. 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.

II) Modification du transcodeur

Vous allez remplacer le composant "mem" par autre un composant appelé "transcodeur" par exemple. Son entité sera :

```

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

Sa particularité est qu'il sera réalisé avec les LUTs que l'on a utilisé en schématique mais que l'on va utiliser ici en VHDL.

1°) Rappel sur les LUTs

Une table de vérité de trois entrées peut être représentée par un nombre 8 bits que l'on convertit en hexadécimal. Soit donc la table de vérité suivante (trois entrées notées e0, e1 et e2, une sortie notée s) :

e2	e1	e0	s	
0	0	0	0	b0
0	0	1	1	b1
0	1	0	1	b2
0	1	1	0	b3
1	0	0	1	b4
1	0	1	0	b5
1	1	0	1	b6
1	1	1	0	b7

Vous pouvez synthétiser la table de vérité à l'aide d'un seul nombre sur 8 bit (poids faible en haut) :

- en binaire ce nombre vaut : 0b01010110

- en hexadécimal ce nombre vaut : 0x56 (noté X"56" en VHDL)

Aucune simplification à faire ici

La valeur hexadécimale 56 (notée X"56" en VHDL) est la valeur avec laquelle il faudra initialiser votre LUT avec un composant **lut3**.

Pour 4 entrées (ce qui est notre cas), on utilise une **lut4** avec 4 chiffres hexadécimaux.

2°) Utiliser des LUTs en VHDL

Un exemple est donné maintenant :

```

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

```

10         i1 : LUT4
11         --synthesis translate_off
12         generic map (INIT => X"EAAA")
13         --synthesis translate_on
14         port map( I0 => e(0),
15                  I1 => e(1),
16                  I2 => e(2),
17                  I3 => e(3),
18                  O  => s(0) );
19         .....
    
```

Cet exemple vous montre comment on câble une **LUT4** en VHDL (**port map**) et comment on l'initialise (**generic map**). Le câblage de ce composant est correct mais pas son initialisation puisqu'on vous demande de la calculer plus loin.

Les deux lignes **library ...** et **use ...** sont à ajouter avant toute entité qui utilise une LUT en plus bien sûr de **"library ieee;"**.

3°) Travail à réaliser (7 pt)

Réaliser le schéma sur votre feuille réponse correspondant au **"port map"** de l'exemple ci-dessus dans le composant **"transcodeur"** (deux rectangles, un appelé **"transcodeur"** et un appelé **"LUT4"**). Compléter ce schéma avec d'éventuels pointillés en montrant que sept LUTs seront nécessaires pour réaliser le transcodeur complet.

Réaliser sur votre feuille réponse une table de vérité complète du décodage demandé.

En déduire les 7 valeurs hexadécimales d'initialisation des LUTs.

Réaliser l'architecture complète de **"transcodeur"** et l'insérer en lieu et place de **"mem"**

Compiler et valider le fonctionnement du compteur de passages.

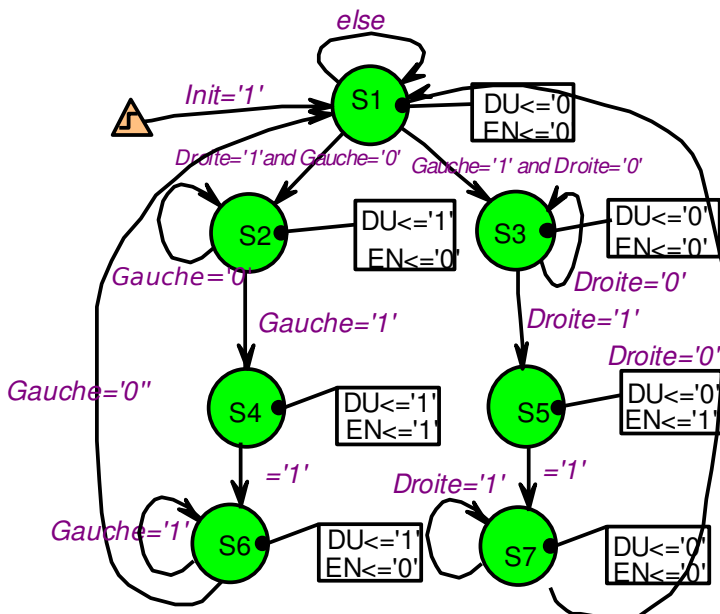
III) Modification du séquenceur

On rappelle que le séquenceur est décrit par un graphe d'évolution rappelé ci-dessous. Ce graphe est réalisé par un composant appelé **"machine_a_etat"** dans la correction.

1°) Graphe d'évolution initial

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 pour lequel 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 !)

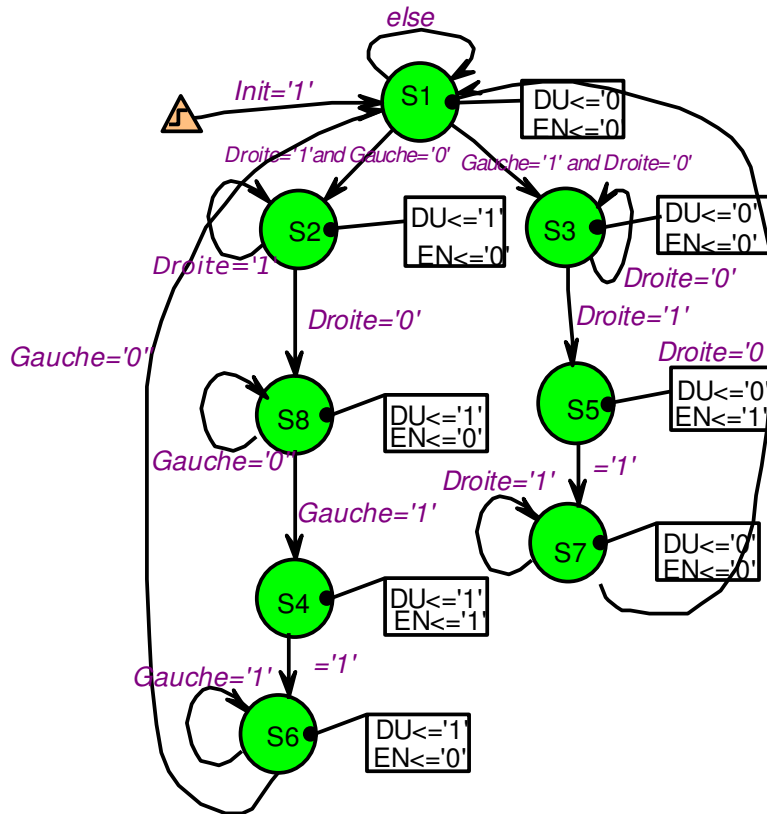


2°) Modification du graphe d'évolution

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 :



3°) Travail à réaliser (6 points)

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.

IV) Un petit complément

Travail à réaliser (3 points)

Modifier l'ensemble en ajoutant un comparateur pour que lorsque le compteur est supérieur à 12 (qui est le maximum) une alarme clignote (sur une led).

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;

```