MI41

Rapport de TP n°2

Petazzoni Thomas
Benkirane Zacharia

Partie 1 : Comptage synchrone

1) Compteur par 10

Le compteur par 10 est réalisé à l'aide de 4 bascules J-K. Voici la table des transitions :

X	*Sorties (t)*				*Basc.3*		*Basc. 2*		*Basc. 1*		*Basc. 0*		*Sorties (t+1)*			
	Q3	*Q2*	*Q1*	*Q0*	*J3*	*K 3*	*J2*	*K 2*	*J1*	*K 1*	*J0*	*K 0*	*Q3*	*Q2*	*Q1*	*Q0*
0	0	0	0	0	0	X	0	X	0	X	1	X	0	0	0	1
1	0	0	0	1	0	X	0	X	1	X	X	1	0	0	1	0
2	0	0	1	0	0	X	0	X	X	0	1	X	0	0	1	1
3	0	0	1	1	0	X	1	X	X	1	X	1	0	1	0	0
4	0	1	0	0	0	X	X	0	0	X	1	X	0	1	0	1
5	0	1	0	1	0	X	X	0	1	X	X	1	0	1	1	0
6	0	1	1	0	0	X	X	0	X	0	1	X	0	1	1	1
7	0	1	1	1	1	X	X	1	X	1	X	1	1	0	0	0
8	0	0	0	0	X	0	0	X	0	X	1	X	1	0	0	1
9	1	0	0	1	X	1	0	X	0	X	X	1	0	0	0	0
10																
11																
12																
13																
14																
15																

A partir de cette table, nous avons pu calculer les équations logiques correspondants aux entrées J et K des 4 bascules, en fonction des différentes sorties.

J0 = 1
K0 = 1
J1 = non(Q3).Q0
K1 = Q0
J2 = Q1.Q0
K2 = Q1.Q0
J3 = Q2.Q1.Q0
K3 = Q3.Q0

Le *Enable*, servant à activer/inhiber le comptage est simple à réaliser. Le comptage est actif quand *Enable* est à un niveau haut, le comptage est inhibé quand *Enable* est à un niveau bas. Les bascules J-K permettent très simplement de réaliser ce *Enable.* En effet, quand J=K=0, la valeur de la bascule est maintenue. Il suffit donc de faire un *ET* logique entre l'entrée *Enable* et les équations trouvées précédemment, et ceci pour chaque entrée J et chaque entrée K.

Enfin, la sortie *B* sert à signaler la fin de comptage. Ce signal doit donc être toujours à 0, sauf lorsqu'on arrive en fin de comptage, c'est à dire à 9. Il suffit donc de faire un *ET* logique entre la sortie de la première bascule et la sortie de la dernière bascule (*Q0 ET Q3).* En effet la valeur binaire *1001* correspond bien à la valeur décimale 9.

L'entrée *Reset* présente sur ce compteur permettra à la fin du TP de remettre à zéro le compteur suite à un appui long sur le bouton. Cette entrée, active niveau bas, a pour rôle de remettre à zéro toutes les bascules du compteur.

2) Compteur par 6

Le compteur par 6 est réalisé exactement comme le compteur par 10, sauf que 3 bascules J-K suffisent cette fois ci. Voici la table de transition :

X	*Sorties (t)*			*Basc. 2*		*Basc. 1*		*Basc. 0*		*Sorties (t+1)*		
	Q2	*Q1*	*Q0*	*J2*	*K2*	*J1*	*K1*	*J0*	*K0*	*Q2*	*Q1*	*Q0*
0	0	0	0	0	X	0	X	1	X	0	0	1
1	0	0	1	0	X	1	X	X	1	0	1	0
2	0	1	0	0	X	X	0	1	X	0	1	1
3	0	1	1	1	X	X	1	X	1	1	0	0
4	1	0	0	X	0	0	X	1	X	1	0	1
5	1	0	1	X	1	0	X	X	1	0	0	0
6												
7												

A partir de cette table, nous avons pu déterminer les équations logiques suivantes :

J0 = 1
K0 = 1
J1 = non(Q2).Q0
K1 = Q0
J2 = Q1.Q0
K2 = Q0

Le *Enable* fonctionne exactement comme le *Enable* du compteur par 10 (*ET* logique entre l'entrée *Enable* et les équations logiques trouvées grâce à la table de transition).

Le *B* fonctionne aussi sur le même principe, sauf qu'on s'arrête à 5. Il y a donc un *ET* logique entre la première et la dernière bascule, pour que *B* soit actif (niveau haut) lorsque le compteur est à *101* en binaire, soit 5 en décimal.

3) Compteur par 60

Le compteur par 60 est réalisé par l'association d'un compteur par 10 et d'un compteur par 6. Le compteur par 10 sert pour les unités, tandis que le compteur par 6 sert pour les dizaines. Les deux compteurs sont reliés à la même horloge (compteur par 60 synchrone). La sortie *B* du compteur par 10 est connectée à l'entrée *Enable* du compteur par 6, pour que celui-ci ne s'incrémente que tous les 10 impulsions d'horloge.

Partie 2 : VHDL

4) Affichage

L'affichage des secondes et des dixièmes de seconde s'effectue respectivement sur deux afficheurs BCD 7 segments. Nous avons donc besoin d'un composant permettant de convertir des valeurs numériques binaires, en code d'affichage pour les afficheurs BCD 7 segments. Le code VHDL suivant réalise très simplement cette opération :

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity transcodeur is
 port (-- On convertit 4 bits d'entrée (valeurs de 0 à 9)
       E: in std_logic_vector(3 downto 0);
       -- En 7 bits de sortie (BCD 7 segments)
       S: out std_logic_vector(6 downto 0)
      );
end transcodeur;

architecture a of transcodeur is
begin
  -- Un simple with … select permet de convertir les nombres
  -- en BCD
  with E select
            S <=  "0000001" when "0000", -- 0
                  "1001111" when "0001", -- 1
                  "0010010" when "0010", -- 2
                  "0000110" when "0011", -- 3
```

```
                    "1001100" when "0100", -- 4
                    "0100100" when "0101", -- 5
                    "0100000" when "0110", -- 6
                    "0001111" when "0111", -- 7
                    "0000000" when "1000", -- 8
                    "0000100" when "1001", -- 9
                      -- Si le nombre est supérieur à 9, on
                      affiche un E pour indiquer une erreur
                    "0110000" when others; -- ERREUR
end a;
```

5) Division d'horloge

L'ensemble de notre circuit est synchrone, mais sera connecté non pas à 25,175 Mhz, mais à une horloge de 100 Hz, que nous devons réaliser grâce à un diviseur. Il faut donc diviser la fréquence d'horloge pour obtenir une impulsion tous les centièmes de seconde.

Le code VHDL suivant permet de faire cela.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity diviseur is
port( clk : in std_logic;
       s : out std_logic
     );
end diviseur;

architecture a of diviseur is
begin
process(clk)
 variable cpt: integer range 0 to 251750;
 begin
  -- A chaque impulsion d'horloge sur front montant
  if ( clk'event and clk='1') then
     -- On incremente un compteur
     cpt:=cpt+1;
     -- Si on arrive a 251750 impulsions, on a attendu un
     centieme de seconde, on genere donc un '1' en sortie, et
     on repart a 0
     if cpt=251750 then
      s <= '1';
      cpt := 0;
     -- Le reste du temps, on a '0' en sortie
    else
      s <= '0';
     end if;
  end if;
 end process;
end a;
```

6) Gestion des interrupteurs

L'interrupteur présent sur la plaquette doit permettre de remettre le compteur à zéro (pression longue de plus d'une seconde) ou d'arrêter/remettre en marche le compteur (pression brève).

On utilise pour cela le code VHDL suivant, qui en fonction de l'entrée du bouton (entrée « but ») et de l'horloge (entrée « clk »), va s'il le faut arrêter/redémarrer l'ensemble des compteurs (sortie « startstop ») et/ou remettre l'ensemble des compteurs à zéro (sortie « reset »).

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity inter is
port(clk,button: in std_logic;
     arret,reset: out std_logic);
end inter;

architecture a of inter is
begin
process(clk)
type activated is (active, unactive);
type state is (on,off);
-- On peut appuyer jusqu'a 10 secondes sur le bouton. Au dela,
on repasse à zéro.
variable compt: integer range 0 to 1000;
variable butstate: activated;
variable comptstate: state;
begin
     if (clk'event and clk='1') then
          -- Si le bouton est actif, on compte
          if (button='0') then
               butstate:=active;
               compt:=compt+1;
               reset<='1';
          -- Le bouton n'est pas actif, doit-on resetter ?
          elsif (compt>=100) then
               compt:=0;
               arret<='0';
               reset<='0';
               comptstate:=on;
               butstate:=unactive;
          -- On ne doit pas resetter. C'était un appui bref, on
          change simplement l'état du compteur. Si il etait en
          route, on l'arrete, si il etait arrete, on le
          demarre.
          elsif (compt < 100 and butstate=active) then
               compt:=0;
               reset<='1';
```

```
                if (comptstate=on) then
                     arret<='0';
                     comptstate:=off;
                     butstate:=unactive;
                else
                     arret<='1';
                     comptstate:=on;
                     butstate:=unactive;
                end if;

           end if;
      end if;
 end process;
end a;
```

Assemblage des composants

L'horloge sur la plaque de test est à 25.175.000 Hz. Nous utilisons donc notre diviseur de fréquence pour obtenir des tics tous les centièmes de seconde, soit une fréquence de 100 Hz.

Tous nos composants sont ensuite reliés sur cette horloge de 100 Hz, le système est donc entièrement synchrone. Nous avons tout d'abord un compteur par 10, qui permet de diviser la fréquence par 10, pour obtenir les dixièmes de seconde que l'on souhaite avoir.

Ce premier compteur par 10 tourne toujours, son *Enable* est donc relié à *Vcc*. Le *Reset* est relié à la sortie *Reset* du gestionnaire d'interrupteur. Les sorties Q0, Q1, Q2 et Q3 de ce compteur ne sont pas utilisées.

Le deuxième compteur par 10 s'incrémente seulement lorsque le précédent compteur est arrivé en bout de course ET que le gestionnaire d'interrupteur indique que les compteurs doivent s'incrémenter. D'où le *ET* entre la sortie *B* du compteur précédent et la sortie *Arret* du gestionnaire d'interrupteur. L'entrée *Reset* est reliée à la sortie *Reset* du gestionnaire d'interrupteur. Les sorties de ce premier compteur sont reliées à un transcodeur BCD 7 segments.

Le troisième compteur par 10 (qui compte les secondes) s'incrémente lorsque les deux précédents compteurs sont en bout de course et que le gestionnaire d'interrupteur indique que les compteurs doivent s'incrémenter. Il est primordial que le *Enable* du dernier compteur ne soit à 1 que lorsque les sorties B des deux précédents compteurs soient à 1, sinon ce dernier compteur fait d'un seul coup le tour (0 à 9) lorsque le deuxième compteur arrive à 9. L'entrée *Reset* est reliée à la sortie *Reset* du gestionnaire d'interrupteurs, tandis que les sorties Q0, Q1, Q2 et Q3 sont reliées à un transcodeur BCD 7 segments.