

# MI41 – TP3

## Gestion des éléments de la platine DLP

Sébastien Simonin, Sébastien Huet, 27 novembre 2005

---

### Introduction

L'objectif de ce TP est de permettre la gestion des interrupteurs et des afficheurs de la platine DLP (*Digital Library Package*) en utilisant des descriptions VHDL. Dans un premier temps, nous allons réaliser la gestion de l'afficheur grâce à un transcodeur binaire/7 segments. Puis nous allons générer une horloge de 100 Hz que nous pourrons associer avec le transcodeur afin d'obtenir un chronomètre binaire. Enfin, en gérant les boutons poussoirs de la platine, nous pourrons mettre en œuvre l'arrêt-redémarrage du chronomètre créé précédemment.

---

### 1. Gestion de l'afficheur : transcodeur binaire/7 segments

Afin de visualiser les valeurs binaires, on crée un module pour effectuer le transcodage d'une valeur binaire 4 bits en un digit hexadécimal pour afficheur 7 segments. Pour ce faire, on génère en sortie un vecteur de 7 bits, chaque bit désignant un segment qui sera actif ou non. Le code VHDL est donné ci-dessous.

List. 1 – Transcodeur  
binaire/7 segments

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

entity transcodeur_seg is
port(
  bin: in std_logic_vector(3 downto 0);
  seg: out std_logic_vector(6 downto 0)
);
end transcodeur_seg;

architecture a of transcodeur_seg is
-- convertit une valeur binaire 4 bits en un digit
-- hexadecimal pour afficheur 7 segments
  signal nseg : std_logic_vector(6 downto 0);
begin
  with bin select
```

1

6

11

16

```

nseg <= "1111110" when "0000",
       "0110000" when "0001",
       "1101101" when "0010",
       "1111001" when "0011",
       "0110011" when "0100",
       "1011011" when "0101",
       "1011111" when "0110",
       "1110000" when "0111",
       "1111111" when "1000",
       "1111011" when "1001",
       "1110111" when "1010",
       "0011111" when "1011",
       "1001110" when "1100",
       "0111101" when "1101",
       "1001111" when "1110",
       "1000111" when "1111",
       "1001000" when others; -- affichage dans le cas d'un
       ↳ mot binaire invalide
seg <= not nseg; -- les segments sont actifs niveau bas
end a;

```

## 2. Gestion de l'horloge

### 2.1 Création de l'horloge

La platine DLP permet de générer une horloge à 25,175 Mhz à partir de laquelle on veut créer une horloge à 100 Hz. Nous allons donc diviser la fréquence de l'horloge d'origine par 251 750. Notre module générera un signal d'horloge en sortie qu'après réception de 251 750 signaux en entrée, grâce à l'utilisation d'un compteur.

List. 2 – Diviseur de fréquence d'horloge

```

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

entity div_clock_100 is
port(
  CLKin: in std_logic;
  CLKout: out std_logic
);
end div_clock_100;

architecture clock_25 of div_clock_100 is
-- divise la frequence d'une horloge de 25,175 MHz
-- pour obtenir une frequence de 100 Hz
begin

process(CLKin)
  variable compteur: integer range 1 to 251750;
begin
  if (CLKin'event and CLKin = '1') then
    if (compteur = 251750) then
      -- si on a reçu 251750 impulsions, envoie un signal
      -- actif en sortie et reinitialise le compteur

```



### 3. Gestion des boutons poussoirs

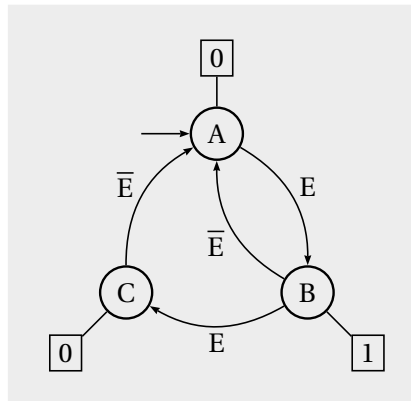
Un appui manuel sur un interrupteur génère un signal actif durant de nombreux cycles d'horloge. Pour qu'un signal s'étalant sur plusieurs cycles ne soit pas interprété comme une succession de signaux distincts, nous allons réaliser un module transformant le signal reçu en un signal actif durant un seul cycle.

#### 3.1 Graphe à états

Un graphe à états est utilisé pour servir de base au code VHDL. On peut distinguer 3 états :

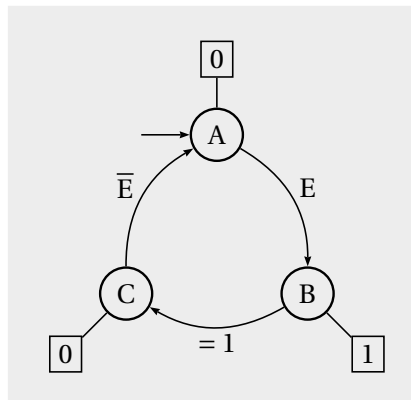
- (A) le gestionnaire est en attente d'un appui de la part de l'utilisateur,
- (B) l'utilisateur a appuyé sur le bouton,  
le gestionnaire renvoie un signal actif en sortie,
- (C) le gestionnaire attend que l'utilisateur relâche le bouton.

Fig. 1 – Graphe à états du gestionnaire du bouton poussoir



Le passage de l'état A vers l'état B se fait lors de la réception d'un signal d'activation E, sur front montant de CLK (on échantillonne pour éviter les rebonds mécaniques). Lorsque le système est en B, on obtient un signal actif en sortie du module. Pour que ce signal ne soit actif qu'un seul cycle d'horloge, il est impératif que l'on change d'état au cycle suivant. Si l'utilisateur n'a toujours pas relâché le bouton en B, on passe à l'état C et on y reste jusqu'à ce qu'il le relâche, sinon on revient à l'état A. Le graphe à états décrit est donné fig. 1.

Fig. 2 – Graphe à états "simplifié" du gestionnaire



La fréquence d'horloge étant suffisamment élevée (on utilisera l'horloge à 100 Hz, soit un front montant toutes les 10 ms), nous pouvons simplifier ce graphe sans que cela ait de conséquence pour l'utilisateur. Comme le montre la fig. 2, on passe directement de l'état B à l'état C, sans se préoccuper du signal E. On ne cherche pas à savoir si l'utilisateur relâche le bouton lorsque l'on est en B. Si ce cas se présente, alors cela provoquera un retour à l'état A avec un cycle de retard (par rapport à la fig. 1), soit 10 ms qui sont négligeables pour ce type d'application. De plus, comme nous allons le voir, le signal E

correspond à une moyenne des 3 derniers signaux reçus, les conséquences de la simplification sont donc peu importantes.

#### 3.2 Code VHDL

Pour limiter les effets des rebonds mécaniques, 2 mécanismes sont mis en place : l'échantillonnage grâce à l'horloge, et l'utilisation d'un moyennneur.

En VHDL, il n'est pas possible de faire une moyenne des dernières valeurs de E, on utilise donc un entier  $E_m$  qui est égal à la somme des 3 dernières valeurs de E. La conversion binaire  $\rightarrow$  entier se fait grâce à la fonction `CONV_INTEGER ( )`. Les 2 valeurs précédentes de E sont mémorisées dans  $E_p$ . Si sur les 3 dernières valeurs de E, on en trouve au moins

2 actives, alors on considère que l'utilisateur appuie sur le bouton, ce qui revient à écrire :

$E_m < 2$  : pas d'appui sur le bouton

$E_m > 1$  : appui sur le bouton

List. 4 – Gestion du bouton poussoir

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

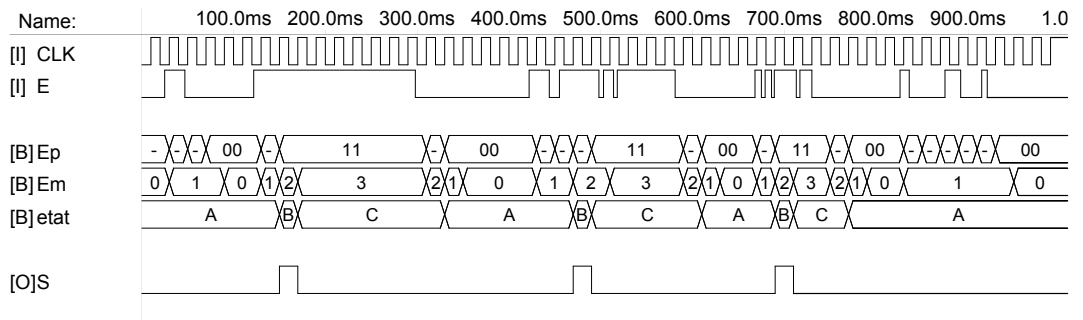
entity gestion_bouton is
port(
    CLK, E: in std_logic;
    S: out std_logic
);
end gestion_bouton;

architecture a of gestion_bouton is
-- transforme un appui de duree quelconque en une
-- commande active durant un seul cycle d'horloge
begin

process(CLK)
    type t_etat is (A, B, C);
    variable etat: t_etat;
    variable Ep: std_logic_vector(1 downto 0);
    variable Em: integer range 0 to 3;
begin
    if (CLK'event and CLK = '1') then
        Em := CONV_INTEGER(Ep(1)) + CONV_INTEGER(Ep(0)) +
        ↪ CONV_INTEGER(E);
        Ep := Ep(0) & E;
        -- calcul de la nouvelle somme et memorisation de la valeur
        ↪ actuelle de E
        if (etat = A and Em > 1) then
            -- on recoit une impulsion -> passage a l'etat B
            etat := B;
            S <= '1';
        elsif (etat = B) then
            -- passage a l'etat C
            etat := C;
            S <= '0';
        elsif (etat = C and Em < 2) then
            -- fin de l'impulsion -> passage a l'etat A
            etat := A;
        end if;
    end if;
end process;

end a;
```

Une simulation permet de vérifier le bon fonctionnement du module :



### 3.3 Chronomètre binaire v. 2

On souhaite maintenant gérer l'arrêt-redémarrage du chronomètre par l'intermédiaire du bouton poussoir. Un appui doit entraîner l'arrêt du chronomètre et un second le relancer, nous devons donc réécrire le code du compteur par 16, en utilisant une variable mode désignant l'état du chronomètre. À chaque fois que le compteur reçoit un signal d'activation, la valeur de mode est inversée.

List. 5 – Compteur par 16 avec commande d'activation

```

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

entity compteur_16_e is
port(
    CLK, E: in std_logic;
    Q: out std_logic_vector(3 downto 0)
);
end compteur_16_e;

architecture a of compteur_16_e is
-- compteur par 16 avec signal d'arret-redemarrage
begin

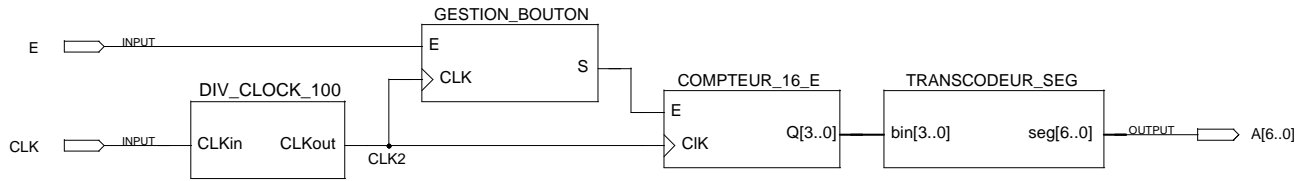
process(CLK)
    variable mem: std_logic_vector(3 downto 0);
    variable mode: std_logic;
begin
    if (CLK'event and CLK = '1') then
        if(E = '1') then
            -- reception d'un signal d'activation
            mode := not mode; -- inversion du mode du chronometre
        end if;

        if(mode = '1') then
            -- si le chronometre est en marche
            mem := mem + "0001";
            Q <= mem;
        end if;
    end if;
end process;

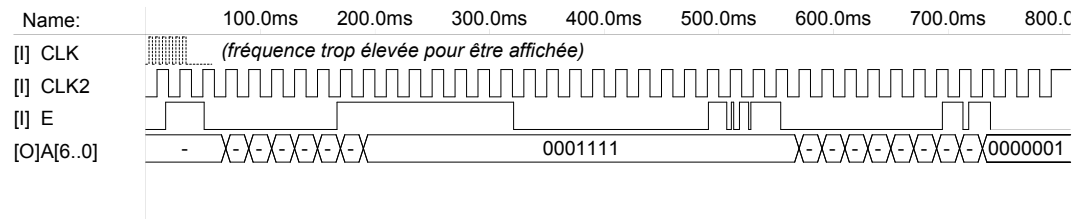
end a;

```

Nous disposons maintenant de tous les modules pour créer notre chronomètre :



La simulation donne bien le résultat attendu :



## Conclusion