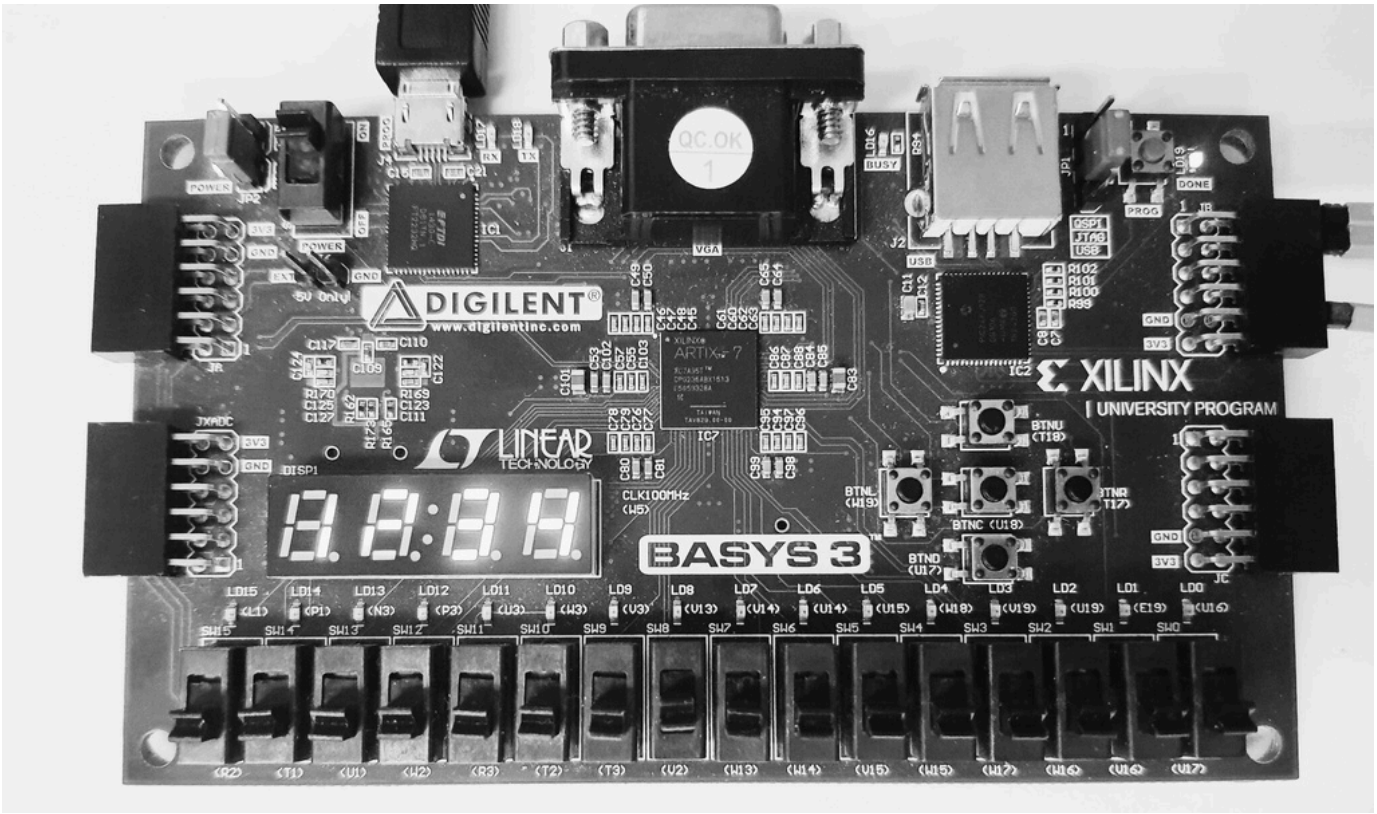
CHAUVELIERE MARTIN / NEHARI TAHA

APPLICATION VHDL SUR BASYS 3

RAPPORT CNSPS



SOMMAIRE

01. Introduction (p.3)

02. PWM (p.4)

03. Afficheur 7 Segments (p.10)

04. Gestion du Port VGA (p.17)

INTRODUCTION

Dans le but de traduire le code VHDL (Very High-Speed Integrated Circuit Hardware Description Language) en une configuration matérielle qui peut être chargée sur un dispositif FPGA (Field-Programmable Gate Array), nous avons dans le cadre de ce travail pratique (TP) exploré l'implémentation matérielle en utilisant le logiciel Vivado et la carte Basys 3 de Digilent.

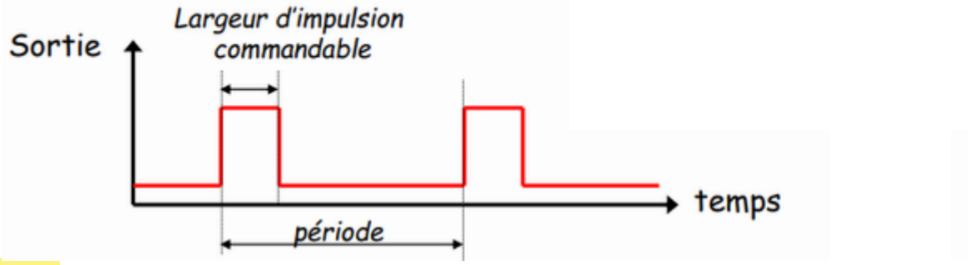
Ainsi, ces travaux pratiques ont pour objectif principal notre familiarisation avec le logiciel Vivado et plus particulièrement à l'implémentation de code VHDL dans une carte BASYS 3.

Dans ce rapport, nous allons décrire le processus d'implémentation des différents programmes que nous avons réalisés, allant de l'étude d'un Test Bench à la gestion d'un port VGA.

PWM

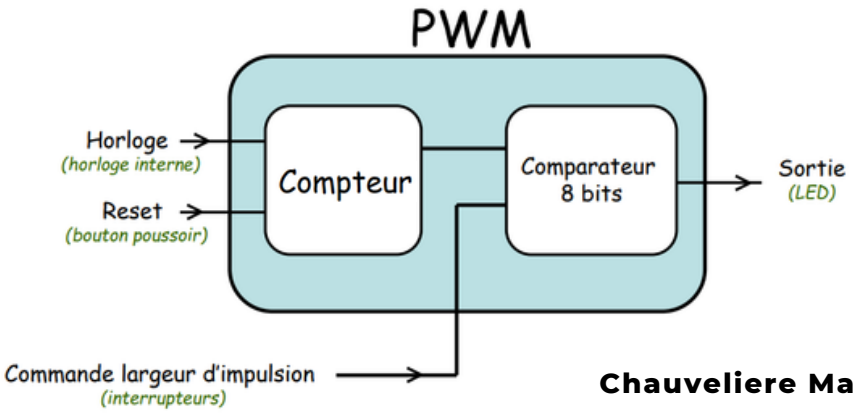
La PWM, ou “Pulse Width Modulation” est une technique couramment utilisée en électronique pour contrôler la quantité de puissance fournie à un dispositif, tel qu'une LED, en modulant la largeur des impulsions d'un signal électrique.

Le principe de base de la PWM consiste à générer un signal périodique, ici sous forme de signal carré, où il est possible de contrôler la période et la largeur d’impulsion à l’état haut.



OBJECTIF

Créer une PWM, de période 1 seconde, où la largeur d’impulsion à l’état haut est commandée à l’aide de plusieurs interrupteurs de notre Basys 3.
Pour cela, nous implémenterons un compteur, afin de créer notre horloge à partir de l’horloge interne de la BASYS 3 de 100 MHz.
Puis un comparateur 8 bits pour commander la largeur d’impulsion.



RÉALISATION

```
36  entity PWM is
37      Port ( clk_fpga : in STD_LOGIC;
38             reset : in STD_LOGIC;
39             sw : in STD_LOGIC_VECTOR (7 downto 0);
40             sortie : out STD_LOGIC);
41  end PWM;
42
43  architecture Behavioral of PWM is
44
45  SIGNAL count_int : std_logic_vector(7 downto 0):="00000000";
46  SIGNAL clk_250 : std_logic:='0';
47  SIGNAL COUNT_200K : integer range 0 to 200000:=0;
48  SIGNAL pwm_int : std_logic:='0';
49
50  begin
51
52  sortie <= pwm_int;
```

Nous créons d'abord la PWM qui possède en entrée la clock de la BASYS 3, un reset et un vecteur de 8 bits sw afin de contrôler la largeur d'impulsion.

Puis les signaux annexes de notre programme count_int qui est un compteur sur 8 bits, clk_250, notre clock de 250 Hz, count_200k le compteur permettant de créer notre clock et pwm_int notre sortie de pwm

```
58  Clock : process(clk_fpga)
59      begin
60      if clk_fpga'event and clk_fpga = '1' then
61          if COUNT_200K < 199999 then
62              COUNT_200K<=COUNT_200K+1;
63          else
64               clk_250 <= not(clk_250);
65               COUNT_200K <= 0;
66          end if;
67      end if;
68  end process;
```

Lorsque le compteur arrive à 200.000 alors notre clock change d'état et le compteur est réinitialiser.

100 MHZ / 2 * 200.000= 250 Hz

Ensuite vient le compteur count_int qui va venir s'incrémenter à chaque coup d'horloge de 250 Hz.

```
68  Compteur : process(clk_250,reset)
69      begin
70          if reset = '1' then
71              count_int <= "00000000";
72          elsif clk_250'event and clk_250 = '1' then
73                  count_int <= count_int+1;
74          end if;
75      end process;
```

L'objectif de ce compteur est de venir comparer sa valeur à celle de sw, notre entrée gérant la largeur de l'impulsion à l'état haut. Lorsque la valeur du compteur dépasse celle de sw, alors notre sortie passe à l'état haut pour le prochain coup de clock et count_int se réinitialise.

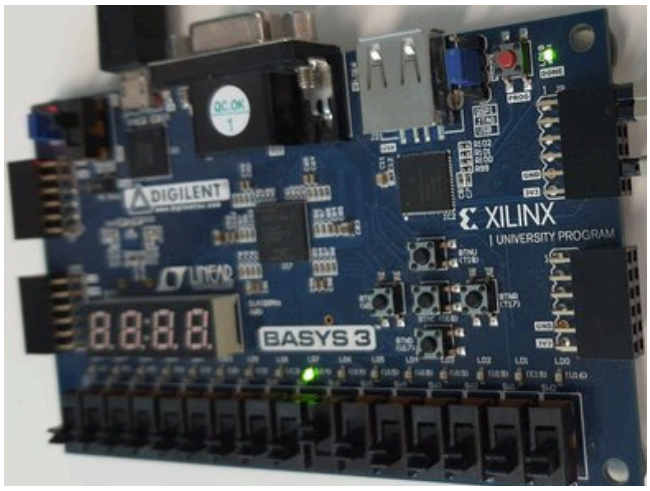
```
83  Comparateur : process(count_int)
84      begin
85          if count_int<sw then
86              pwm_int<='0';
87          else
88              pwm_int<='1';
89          end if;
90      end process;
```

On modifie maintenant, le fichier de contrainte de noter Basys 3 :

```
7   set_property -dict { PACKAGE_PIN W5   IOSTANDARD LVCMOS33 } [get_ports clk_fpga]

11  ## Switches
12  set_property -dict { PACKAGE_PIN V17   IOSTANDARD LVCMOS33 } [get_ports {sw[0]}]
13  set_property -dict { PACKAGE_PIN V16   IOSTANDARD LVCMOS33 } [get_ports {sw[1]}]
14  set_property -dict { PACKAGE_PIN W16   IOSTANDARD LVCMOS33 } [get_ports {sw[2]}]
15  set_property -dict { PACKAGE_PIN W17   IOSTANDARD LVCMOS33 } [get_ports {sw[3]}]
16  set_property -dict { PACKAGE_PIN W15   IOSTANDARD LVCMOS33 } [get_ports {sw[4]}]
17  set_property -dict { PACKAGE_PIN V15   IOSTANDARD LVCMOS33 } [get_ports {sw[5]}]
18  set_property -dict { PACKAGE_PIN W14   IOSTANDARD LVCMOS33 } [get_ports {sw[6]}]
19  set_property -dict { PACKAGE_PIN W13   IOSTANDARD LVCMOS33 } [get_ports {sw[7]}]

27  set_property -dict { PACKAGE_PIN R2    IOSTANDARD LVCMOS33 } [get_ports {reset}]
```



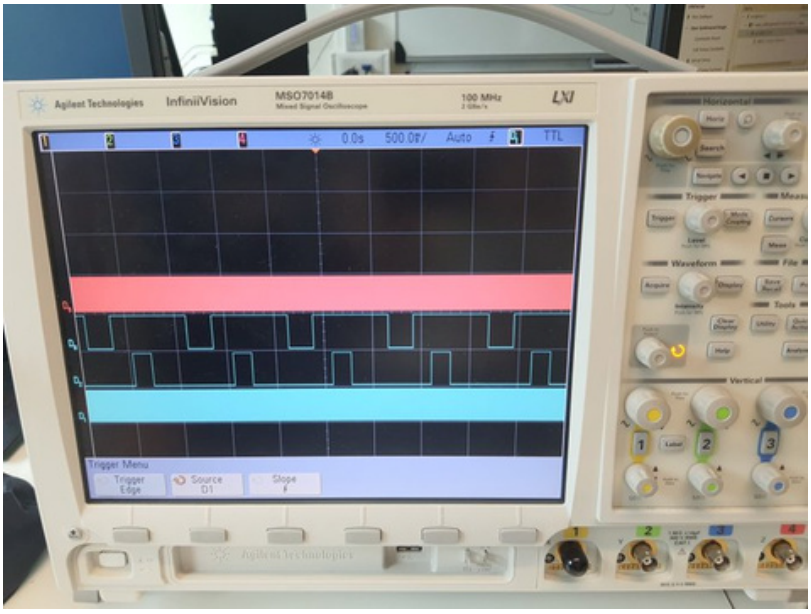
Nos interrupteurs sont fonctionnels et commandent la vitesse de clignotement de la led verte.

Ajout et Configuration de 2 nouvelles sorties à notre PWM :

```
36  entity PWM is
37      Port ( clk_250_visu : out std_logic;
38             sortie_visu : out std_logic;

54  clk_250_visu <= clk_250;
55  sortie_visu <= pwm_int;
```

Ces sorties vont nous être utile pour visualiser notre clock et sortie à l'aide d'un analyseur logique :



TEST BENCH

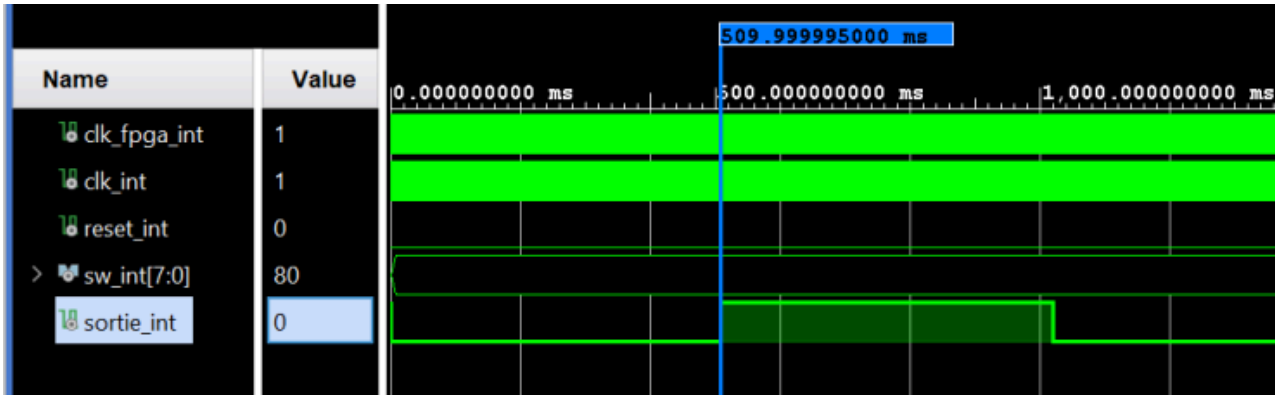
Création d'un Test Bench pour simuler notre programme.
Dans notre fichier de Test Bench, nous implémentons le composant PWM et à l'aide de signaux créons la Port Map du PWM.

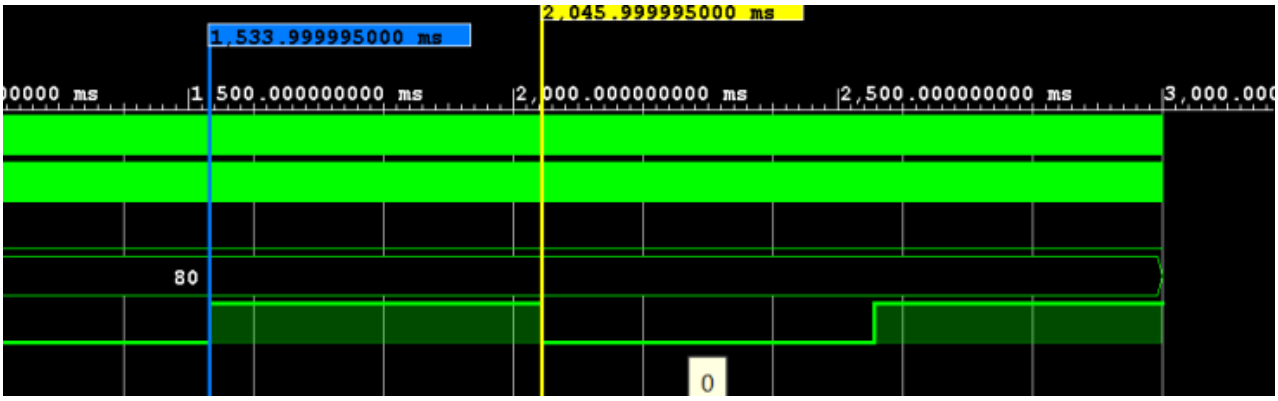
```
43  component PWM
44      Port ( clk_fpga : in STD_LOGIC;
45             reset : in STD_LOGIC;
46             sw : in STD_LOGIC_VECTOR (7 downto 0);
47             sortie : out STD_LOGIC);
48  end component;
49
50  signal clk_fpga_int : STD_LOGIC;
51  signal reset_int : STD_LOGIC;
52  signal sw_int : STD_LOGIC_VECTOR (7 downto 0);
53  signal sortie_int : STD_LOGIC;
54
55  begin
56
57  UUT : PWM port map(
58      clk_fpga => clk_fpga_int,
59      reset => reset_int,
60      sw => sw_int,
61      sortie => sortie_int);
```

Créations de situmulus pour simuler notre clock, reset et sw :

```
63  stimulus_horloge_100MHZ : process
64      begin
65          clk_fpga_int <= '0';
66          wait for 5ns;
67          clk_fpga_int <= '1';
68          wait for 5ns;
69  end process;
70
71  stimulus_reset: process
72      begin
73          reset_int<='0';
74          wait for 200us;
75          reset_int<='1';
76          wait for 20us;
77          reset_int<='0';
78          wait;
79  end process;
```

```
81  stimulus_sw: process
82      begin
83          sw_int<= conv_std_logic_vector(0,8);
84          wait for 50us;
85          sw_int<= conv_std_logic_vector(50,8);
86          wait for 50us;
87          sw_int<= conv_std_logic_vector(120,8);
88          wait for 50us;
89          sw_int<= conv_std_logic_vector(200,8);
90          wait;
91  end process;
```



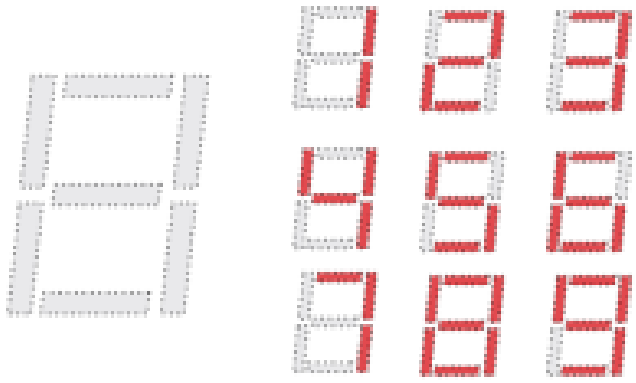


Le Test Bench et l'analyseur logique sont 2 excellents outils pour pouvoir debug un programme et trouver la cause de l'erreur.

AFFICHEUR 7 SEGEMNTS

OBJECTIF

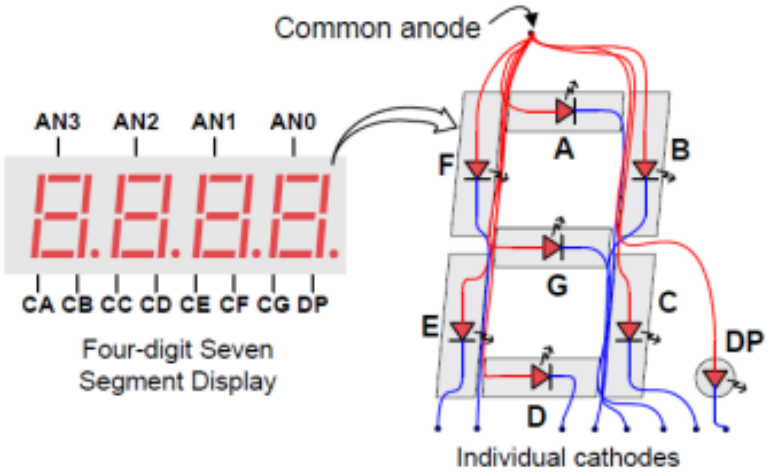
L'objectif ici est dans un premier temps d'afficher les chiffres 1234 sur l'afficheur 7 segments et dans un second, de faire défiler les lettres de POLYTECH sur l'afficheur.



RÉALISATION 1

Pour ce faire, voyons d'abord comment se contrôle notre afficheur: En plus des 8 bits de contrôle pour les segments et le point (seg), nous avons aussi 4 bit de contrôle des 4 caractères (an). En clair, nous avons qu'un seul octet de contrôle des segments (seg) qui agira sur l'un des caractères en fonction du byte de contrôle (an).

Remarque : 1 correspond à un segment éteint et 0 à un segment allumé.



Dès lors, pour pouvoir apercevoir les 4 caractères en même temps, il faut utiliser une fréquence adéquate, ici de 250Hz.

```
36  entity Seg7 is
37      Port ( clk_100mhz : in STD_LOGIC;
38             reset : in STD_LOGIC;
39             seg : out STD_LOGIC_VECTOR (7 downto 0):="11111111";
40             an : out STD_LOGIC_VECTOR (3 downto 0):="1111");
41  end Seg7;
42
43  architecture Behavioral of Seg7 is
44
45  signal compteur : integer:=0;
46  signal clk_250 : std_logic;
47  signal COUNT_400K : integer range 0 to 200000:=0;
```

Le process se présente comme suit :

```
63  process (clk_250,reset)
64      begin
65          if reset = '1' then
66              seg <= "11111111";
67          elsif clk_250'event and clk_250 = '1' then
68              if compteur = 0 then
69                  an(0) <= '1';
70                  an(3) <= '0';
71                  seg <= "11111001";
72                  compteur <= 1;
73              elsif compteur = 1 then
74                  an(3) <= '1';
75                  an(2) <= '0';
76                  seg <= "10100100";
77                  compteur <= 2;
78              elsif compteur = 2 then
79                  an(2) <= '1';
80                  an(1) <= '0';
81                  seg <= "10110000";
82                  compteur <= 3;
83              elsif compteur = 3 then
84                  an(1) <= '1';
85                  an(0) <= '0';
86                  seg <= "10011001";
87                  compteur <= 0;
88              end if;
89          end if;
90      end process;
91  end Behavioral;
```

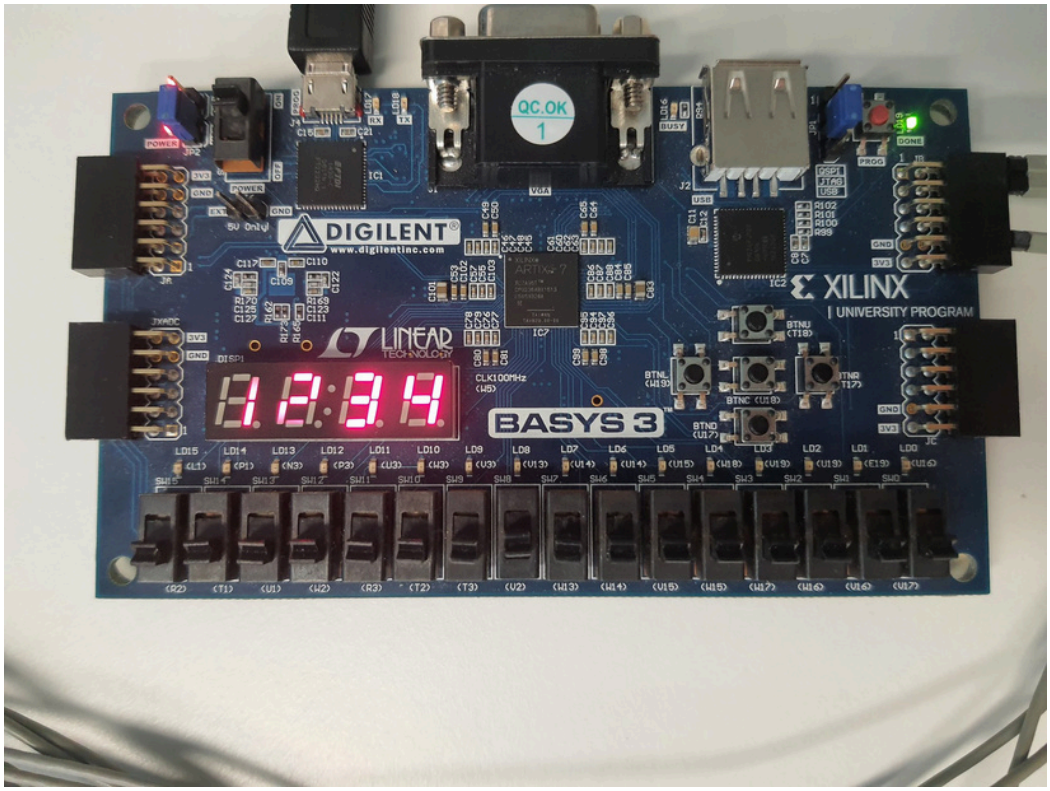
Comme on ne peut pas accéder à tous les caractères d'un seul coup, et qu'un octet de contrôle des segments est à disposition, il faut parcourir chaque caractère.

Un compteur est utilisé pour y parvenir et lorsque l'on a accès au caractère voulu, on allume et éteint les segments du chiffre souhaité.

Une fois le fichier de contrainte modifié :

```
50 set_property -dict { PACKAGE_PIN W7   IOSTANDARD LVCMOS33 } [get_ports {seg[0]}]
51 set_property -dict { PACKAGE_PIN W6   IOSTANDARD LVCMOS33 } [get_ports {seg[1]}]
52 set_property -dict { PACKAGE_PIN U8   IOSTANDARD LVCMOS33 } [get_ports {seg[2]}]
53 set_property -dict { PACKAGE_PIN V8   IOSTANDARD LVCMOS33 } [get_ports {seg[3]}]
54 set_property -dict { PACKAGE_PIN U5   IOSTANDARD LVCMOS33 } [get_ports {seg[4]}]
55 set_property -dict { PACKAGE_PIN V5   IOSTANDARD LVCMOS33 } [get_ports {seg[5]}]
56 set_property -dict { PACKAGE_PIN U7   IOSTANDARD LVCMOS33 } [get_ports {seg[6]}]
57
58 set_property -dict { PACKAGE_PIN V7   IOSTANDARD LVCMOS33 } [get_ports {seg[7]}]
59
60 set_property -dict { PACKAGE_PIN U2   IOSTANDARD LVCMOS33 } [get_ports {an[0]}]
61 set_property -dict { PACKAGE_PIN U4   IOSTANDARD LVCMOS33 } [get_ports {an[1]}]
62 set_property -dict { PACKAGE_PIN V4   IOSTANDARD LVCMOS33 } [get_ports {an[2]}]
63 set_property -dict { PACKAGE_PIN W4   IOSTANDARD LVCMOS33 } [get_ports {an[3]}]
64
```

On obtient cela :



RÉALISATION 2

L'objectif ici étant de faire défiler les lettres POLYTECH sur l'afficheur, une fréquence de déplacement des lettres doit être mise en place, de la même manière que précédemment, une clock de fréquence 10Hz est créée.

Pour se simplifier la tâche, on crée un tableau des lettres POLYTECH :

```
57  type Tab is array (0 to 7) of STD_LOGIC_VECTOR(7 downto 0);
58  signal polytech :  Tab;
59  begin
60
61  polytech(0) <= "10001100"; --P
62  polytech(1) <= "11000000"; --O
63  polytech(2) <= "11000111"; --L
64  polytech(3) <= "10010001"; --Y
65  polytech(4) <= "11111000"; --T
66  polytech(5) <= "10000110"; --E
67  polytech(6) <= "11000110"; --C
68  polytech(7) <= "10001001"; --H
```

Comme à chaque instant, une suite de 4 lettres est affichée à l'écran, on déclare 4 indices qui parcourront le tableau de lettres :

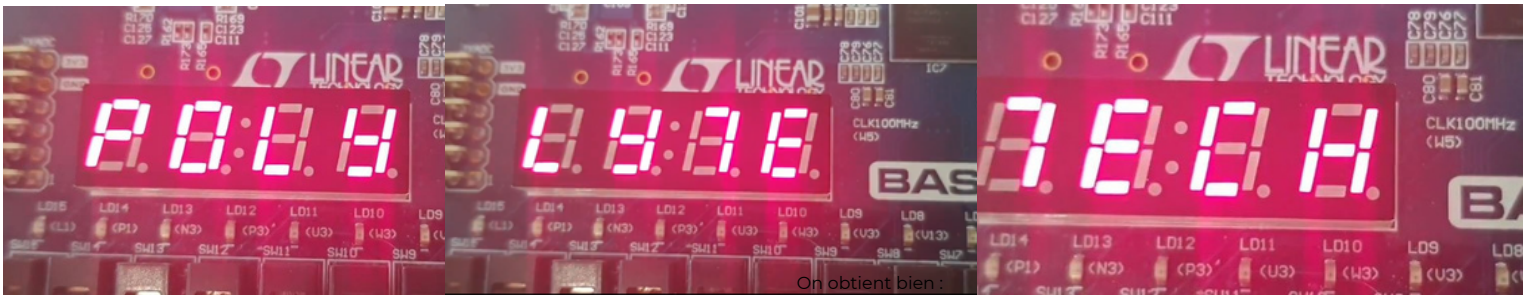
```
46  signal indice0 : integer :=0;
47  signal indice1 : integer :=1;
48  signal indice2 : integer :=2;
49  signal indice3 : integer :=3;
```

Ainsi, la logique reste la même que précédemment, on contrôle chaque caractère un par un et on affiche ce que l'on souhaite :

```
94  Affichage : process (clk_250,reset)
95      begin
96          if reset = '1' then
97              seg <= "11111111";
98          elsif clk_250'event and clk_250 = '1' then
99              if compteur = 0 then
100                 an(0) <= '1';
101                 an(3) <= '0';
102                 seg <= polytech(indice0);
103                 compteur <= 1;
104             elsif compteur = 1 then
105                 an(3) <= '1';
106                 an(2) <= '0';
107                 seg <= polytech(indice1);
108                 compteur <= 2;
109             elsif compteur = 2 then
110                 an(2) <= '1';
111                 an(1) <= '0';
112                 seg <= polytech(indice2);
113                 compteur <= 3;
114             elsif compteur = 3 then
115                 an(1) <= '1';
116                 an(0) <= '0';
117                 seg <= polytech(indice3);
118                 compteur <= 0;
119             end if;
120         end if;
121 end process;
```

Cependant, pour gérer le déplacement, un autre processus fonctionnera en parallèle pour modifier la valeur des indices, et déplacer les lettres. C'est alors ce processus qui utilisera la fréquence de déplacement :

```
123  Deplacement : process (clk_10)
124  begin
125      if clk_10'event and clk_10 = '1' then
126          if indice0 = 7 then indice0 <= 0;
127          else indice0 <= indice0 +1;
128          end if;
129          if indice1 = 7 then indice1 <= 0;
130          else indice1 <= indice1 +1;
131          end if;
132          if indice2 = 7 then indice2 <= 0;
133          else indice2 <= indice2 +1;
134          end if;
135          if indice3 = 7 then indice3 <= 0;
136          else indice3 <= indice3 +1;
137          end if;
138      end if;
139  end process;
140  end Behavioral;
```

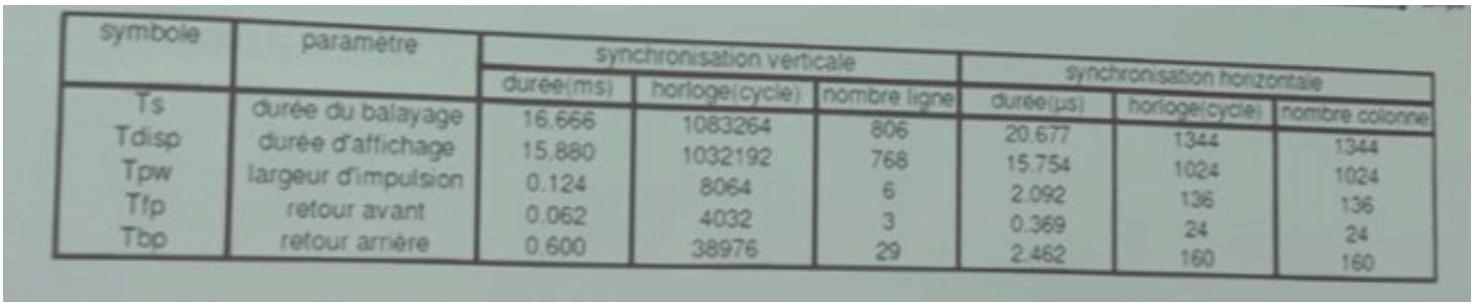


On obtient bien :

GESTION DU PORT VGA

OBJECTIF

Le but est d'obtenir une résolution de 1024x768, d'y afficher un carré en plein centre capable de changer de couleur et enfin de le faire rebondir sur les parois de l'écran.



symbole	paramètre	synchronisation verticale			synchronisation horizontale		
		durée(ms)	horloge(cycle)	nombre ligne	durée(µs)	horloge(cycle)	nombre colonne
Ts	durée du balayage	16.666	1083264	806	20.677	1344	1344
Tdisp	durée d'affichage	15.880	1032192	768	15.754	1024	1024
Tpw	largeur d'impulsion	0.124	8064	6	2.092	136	136
Tfp	retour avant	0.062	4032	3	0.369	24	24
Tbp	retour arrière	0.600	38976	29	2.462	160	160

La gestion des couleurs se fera en RGB via les ports vgaRed, vgaBlue et vgeGreen. La résolution est créée grâce au balayage de HS et VS qui respecteront une certaine forme d'onde (voir tableau).

```
34  entity VGA_Controller is
35  Port ( vgaRed : out std_logic_vector(0 to 3):="0000";
36         vgaGreen : out std_logic_vector(0 to 3):="0000";
37         vgaBlue : out std_logic_vector(0 to 3):="0000";
38         clk_fpga : in std_logic;
39
40         HS : out std_logic;
41         VS : out std_logic;
42         BoutonR, BoutonG, BoutonB : in std_logic);
43  end VGA_Controller;
```

Les signaux HS_sig et VS_sig sont présents pour nous permettre de faire des opérations (notamment des conditions) sur les valeurs de HS et VS impossible autrement (à cause de leur nature 'out').

maxH, minH, maxV et minV correspondent aux dimensions du carré.

sensH et sensV aux sens de déplacement du carré (horizontale ou vertical).

COUNT_60 pour obtenir un déplacement à 60pix/s.

```
54  signal clk_65mhz : std_logic;
55  signal VS_sig, HS_sig: std_logic;
56  signal compteurH : integer range 0 to 1344:=0;
57  signal compteurV : integer range 0 to 806:=0;
58  signal maxH : integer := 612;
59  signal minH : integer := 412;
60  signal maxV : integer := 284;
61  signal minV : integer := 484;
62  signal sensH : integer := 1;
63  signal sensV :integer := 1;
64  signal COUNT_60 : integer := 0;
```

Le processus d'obtention de la résolution se déroule comme suit :

```
process(clk_65mhz)
begin
    if clk_65mhz'event and clk_65mhz = '1' then
        if compteurH < 1343 then
            compteurH <= compteurH +1;
        else
            compteurH <= 0;
            if compteurV < 805 then
                compteurV <= compteurH +1;
            else
                compteurV <= 0;
                end if;
        end if;
        if compteurH > 1047 and compteurH < 1184 then
            HS <= '0';
            HS_sig <= '0';
        else
            HS <= '1';
            HS_sig <= '1';
        end if;
        if compteurV > 770 and compteurV < 777 then
                VS <= '0';
                VS_sig <= '0';
            else
                VS <= '1';
                VS_sig <= '1';
        end if;
```

Les compteurs compteurH et compteurV correspondent à la longueur totale de la trame du signal de synchronisation.
En fonction de la trame, on choisit les valeurs de chaque compteur pour lesquelles HS et VS sont nuls. Lorsque l'on atteint ces valeurs, on procède alors à l'affectation de '0' dans HS et VS.
On obtient bien une résolution de 1024x768.

Pour obtenir un carré, on joue sur la couleur des pixels qui nous intéressent.

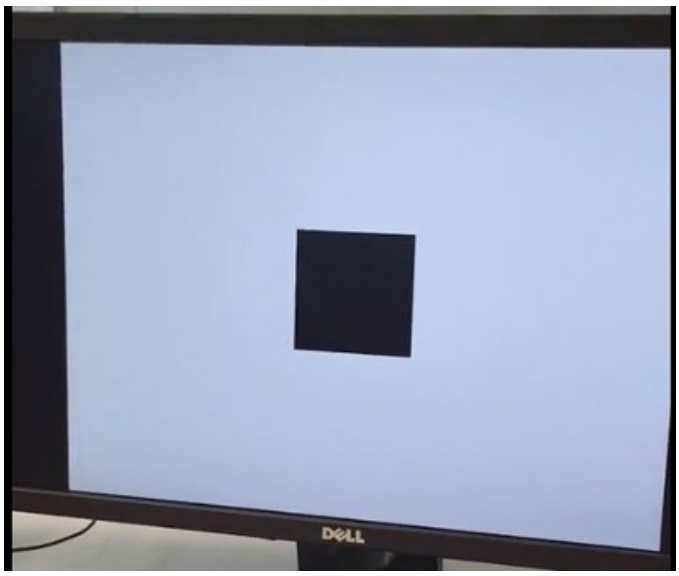
```
if compteurH > minH and compteurH < maxH and compteurV > minV and compteurV < maxV then
    if (BoutonB = '1') then vgaBlue <= "1111"; else vgaBlue <= "0000"; end if;
    if (BoutonG = '1') then vgaGreen <= "1111"; else vgaGreen <= "0000"; end if;
    if (BoutonR = '1') then vgaRed <= "1111"; else vgaRed <= "0000"; end if;
elsif HS_sig ='1' and VS_sig = '1' then
    vgaRed <= "0111";
    vgaGreen <= "0011";
    vgaBlue <= "0011";
else
    vgaRed <= "0000";
    vgaGreen <= "0000";
    vgaBlue <= "0000";
```

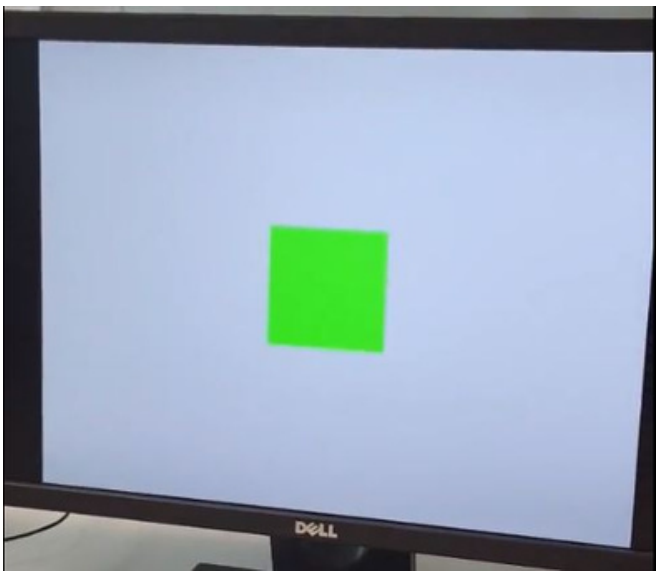
C'est ici que l'on utilise HS_sig et VS_sig, sans qui, on ne pourrait pas effectuer ces conditions.
Lorsqu'un des boutons est activé la couleur change mais seulement aux limites du carré établis précédemment.
Sinon on choisit une couleur qui correspondra au fond de l'écran (ici en blanc bleuté).
Dans tous les autres cas, vgaRed/Blue/Green doivent être nuls.

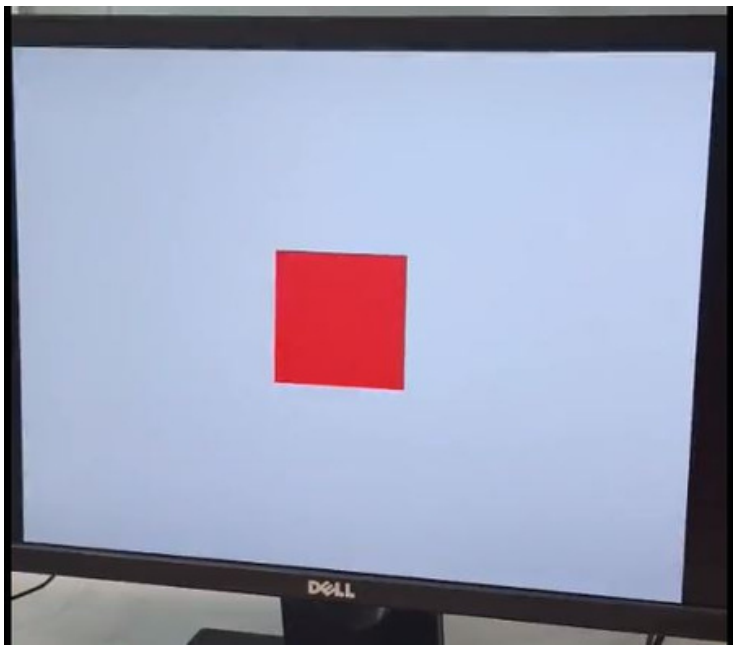
Une fois le fichier de contrainte modifié :

```
##VGA Connector
set_property -dict { PACKAGE_PIN G19   IOSTANDARD LVCMOS33 } [get_ports {vgaRed[0]}]
set_property -dict { PACKAGE_PIN H19   IOSTANDARD LVCMOS33 } [get_ports {vgaRed[1]}]
set_property -dict { PACKAGE_PIN J19   IOSTANDARD LVCMOS33 } [get_ports {vgaRed[2]}]
set_property -dict { PACKAGE_PIN N19   IOSTANDARD LVCMOS33 } [get_ports {vgaRed[3]}]
set_property -dict { PACKAGE_PIN N18   IOSTANDARD LVCMOS33 } [get_ports {vgaBlue[0]}]
set_property -dict { PACKAGE_PIN L18   IOSTANDARD LVCMOS33 } [get_ports {vgaBlue[1]}]
set_property -dict { PACKAGE_PIN K18   IOSTANDARD LVCMOS33 } [get_ports {vgaBlue[2]}]
set_property -dict { PACKAGE_PIN J18   IOSTANDARD LVCMOS33 } [get_ports {vgaBlue[3]}]
set_property -dict { PACKAGE_PIN J17   IOSTANDARD LVCMOS33 } [get_ports {vgaGreen[0]}]
set_property -dict { PACKAGE_PIN H17   IOSTANDARD LVCMOS33 } [get_ports {vgaGreen[1]}]
set_property -dict { PACKAGE_PIN G17   IOSTANDARD LVCMOS33 } [get_ports {vgaGreen[2]}]
set_property -dict { PACKAGE_PIN D17   IOSTANDARD LVCMOS33 } [get_ports {vgaGreen[3]}]
set_property -dict { PACKAGE_PIN P19   IOSTANDARD LVCMOS33 } [get_ports HS]
set_property -dict { PACKAGE_PIN R19   IOSTANDARD LVCMOS33 } [get_ports VS]
```

On obtient alors :







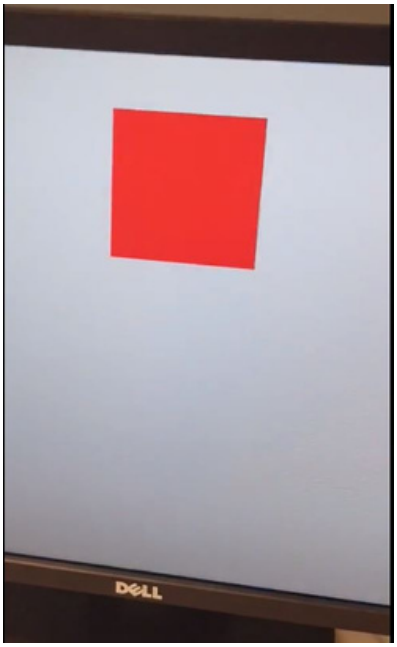
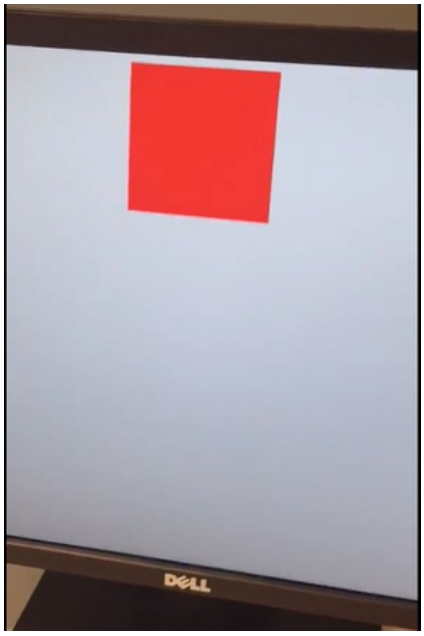
On voit bien que le carré change de couleur, lorsque l'on actionne les switchs

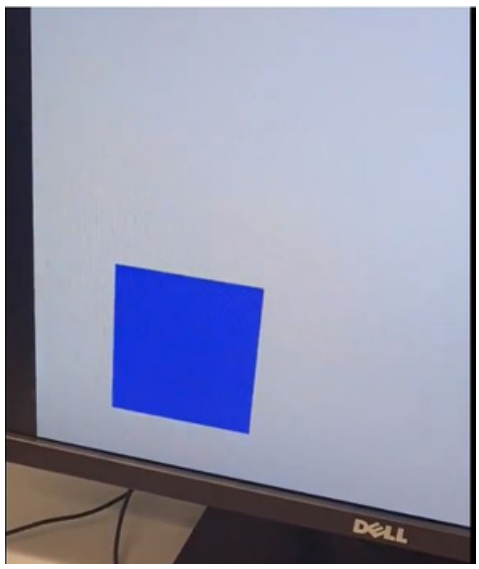
Afin d'obtenir un déplacement pas trop rapide, on utilisera le COUNT_60 qui correspond à un déplacement à 60Hz :

```
73  process(clk_65mhz)
74  begin
75
76      if clk_65mhz'event and clk_65mhz = '1' then
77          if COUNT_60 < 833332 then
78              COUNT_60 <= COUNT_60 + 1;
79          else
80              if maxH = 767 then sensH <= 1;
81              elsif minH = 0 then sensH <= -1;
82              elsif maxV = 1023 then sensH <= -1;
83              elsif minV = 0 then sensH <= 1;
84              end if;
85              maxH <= maxH + sensH;
86              minH <= minH + sensH;
87              maxV <= maxV + sensV;
88              minV <= minV + sensV;
89              COUNT_60 <= 0;
90          end if;
91      end if;
92  end process;
```

A chaque coup de compteur, on test si l'une des frontières du carré a atteint un bord de l'écran, auquel cas le sens de déplacement correspondant change et permet au carré de se déplacer dans le sens opposé.
Ensuite, on déplace le carré en ajoutant les valeurs des sensH et sensV aux frontières correspondantes.

Voici quelques images du carré pendant son déplacement :

Ça rebondit!

Merci de votre attention!