Université de Montpellier

MASTER ÉLECTRONIQUE, ÉNERGIE ÉLECTRIQUE & AUTOMATIQUE

Année Universitaire 2024-2025

Compte Rendu des travaux pratiques

CONCEPTION D'UN CONTRÔLEUR VGA EN VHDL

Réalisé par

Amine Adel BEKDOUCHE

Parcours

Capteurs, Électronique & Objets Connectés (CEO)

Encadrant : M. Podlecki

1 Introduction

1.1 Contexte et Objectif : Présentation du projet et des objectifs principaux.

Ce projet vise à concevoir un contrôleur VGA en utilisant le langage de description matériel VHDL. L'objectif principal est de générer les signaux nécessaires pour afficher des images sur un écran, en utilisant différentes techniques de stockage et de gestion des pixels.

1.2 Description Générale: Vue d'ensemble de la conception du contrôleur VGA.

La conception comprend plusieurs composants, chacun jouant un rôle spécifique dans la génération des signaux VGA et l'affichage des images. Nous avons utilisé des mémoires PROM et des blocs ROM pour stocker les données des images (sprites) et les afficher correctement sur l'écran.

2 Génération de l'horloge

Objectif : Diviser l'horloge de la carte (100 MHz) pour générer un signal clk25 à 25 MHz, nécessaire pour le fonctionnement du contrôleur VGA. De plus, une horloge à 1 Hz a été générée pour permettre le déplacement des images.

2.1 Description des Signaux

- rst : remise à zéro (entrée)
- clk : horloge maître à 100 MHz (entrée)
- clk25 : horloge à 25 MHz (sortie)
- clk_1hz : horloge à 1 Hz (sortie)

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_ARITH.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6  entity gen_pixel_clk is
7      Port ( rst : in STD_LOGIC;
8             clk : in STD_LOGIC;
9             clk25 : out STD_LOGIC;
10            clk_1hz : out STD_LOGIC);
11 end gen_pixel_clk;
12
13 architecture Behavioral of gen_pixel_clk is
14 signal compteur : STD_LOGIC_VECTOR(2 downto 0);
15 signal compteur2 : STD_LOGIC_VECTOR(24 downto 0);
```

```
16
17 begin
18
19 process(clk, rst) is
20 begin
21     if rst = '1' then
22         compteur <= "000";
23         compteur2 <= "00000000000000000000000000";
24     elsif clk'event and clk = '1' then
25         compteur <= compteur + 1;
26         compteur2 <= compteur2 + 1;
27
28         if compteur2 < "01011111010111100001000000" then
29             clk_1hz <= '1';
30
31         elsif compteur2 < "10111110101111000010000000" then
32             clk_1hz <= '0';
33         else
34             clk_1hz <= '1';
35             compteur2 <= "00000000000000000000000000";
36         end if;
37
38         if compteur < "001" then
39             clk25 <= '1';
40         elsif compteur < "011" then
41             clk25 <= '0';
42         else
43             clk25 <= '1';
44             compteur <= "000";
45         end if;
46
47     end if;
48 end process;
49
50 end Behavioral;
```

Listing 1: Code VHDL pour le composant gen_pixel_clk

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity gen_pixel_clk_tb is
5  end gen_pixel_clk_tb;
6
7  architecture Behavioral of gen_pixel_clk_tb is
8
9  signal clk, rst, clk25, clk_1hz: STD_LOGIC;
10 component gen_pixel_clk is port (rst : in std_logic; clk : in
       std_logic; clk25 : out std_logic; clk_1hz : out std_logic); end
        component;
11
12 begin
```

```
13
14 clk_test : gen_pixel_clk port map(rst => rst, clk => clk, clk25 =>
       clk25, clk_1hz => clk_1hz);
15
16 clock : process is
17 begin
18     clk <= '0';
19     wait for 10ns;
20     clk <= '1';
21     wait for 10ns;
22 end process;
23
24 reset : process is
25 begin
26     rst <= '1';
27     wait for 2ns;
28     rst <= '0';
29     wait;
30 end process;
31
32 end Behavioral;
```

Listing 2: Code VHDL pour le composant gen_pixel_clk_tb

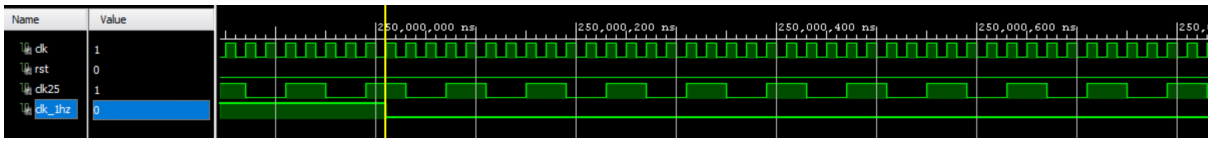

Figure 1: Résultat du test bench de générateur d'horloge

Interprétation :

Le générateur d'horloge a réussi à diviser l'horloge principale de 100 MHz pour produire une horloge à 25 MHz nécessaire pour le contrôleur VGA, ainsi qu'une horloge à 1 Hz pour le déplacement des images, voir la figure 1. Cela montre que le système peut fonctionner avec les fréquences requises pour synchroniser les signaux VGA et gérer le déplacement des images, assurant ainsi une base solide pour le fonctionnement du contrôleur.

3 Composant VGA_640x480

3.1 Objectif : Générer les signaux de synchronisation horizontale et verticale.

3.2 Constantes et Paramètres

- hpixels : Nombre de pixels sur une ligne = 800 ("1100100000")
- vlines : Nombre total de lignes horizontales = 521 ("1000001001")

- hbp : Horizontal back porch = 144 ("0010010000")
- hfp : Horizontal front porch = 784 ("1100010000")
- vbp : Vertical back porch = 31 ("0000011111")
- vfp : Vertical front porch = 511 ("0111111111")

3.3 Fonctionnement

3.3.1 Compteur Horizontal

Le compteur horizontal compte les pixels sur chaque ligne. Il génère les signaux de synchronisation horizontale (hsync) et détermine la zone active d'affichage.

3.3.2 Compteur Vertical

Le compteur vertical compte les lignes. Il génère les signaux de synchronisation verticale (vsync) et détermine la zone active d'affichage.

3.3.3 Sorties

- hsync : Signal de synchronisation horizontale
- vsync : Signal de synchronisation verticale
- vidon : Indicateur de la zone active d'affichage

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity VGA_640x480 is
    Port ( clk25 : in STD_LOGIC;
           rst : in STD_LOGIC;
           hsync : out STD_LOGIC;
           vsync : out STD_LOGIC;
           hc : out STD_LOGIC_VECTOR (9 downto 0);
           vc : out STD_LOGIC_VECTOR (9 downto 0);
           vidon : out STD_LOGIC);
end VGA_640x480;

architecture Behavioral of VGA_640x480 is
-- definition des constantes
constant hpixels: std_logic_vector(9 downto 0) :="1100100000"; --
   nombre de pixels sur une ligne = 800
constant vlines: std_logic_vector(9 downto 0) :="1000001001"; --
   nombre total de lignes horizontales = 521
constant hbp: std_logic_vector(9 downto 0) :="0010010000"; --
   horizontal back porch = 128 + 16 = 144 (ou 96 + 48)
```

```
21 constant hfp: std_logic_vector(9 downto 0) :="1100010000"; --
   horizontal front porch = 128 + 16 + 640 = 784
22 constant vbp: std_logic_vector(9 downto 0) :="0000011111"; --
   vertical back porch = 2 + 29 = 31
23 constant vfp: std_logic_vector(9 downto 0) :="0111111111"; --
   vertical front porch = 2 + 29 + 480 + 10 = 511
24 -- definition des signaux
25 signal hcs : std_logic_vector(9 downto 0); -- compteur horizontal
   / vertical
26 signal vcs: std_logic_vector(9 downto 0); -- compteur vertical
27 signal vsenable: std_logic := '0'; -- enable pour le compteur
   vertical
28
29 -- debut du programme
30 begin
31     process(clk25, rst)
32     begin
33         if rst='1' then
34             hcs <= "0000000000"; -- remise    zero du compteur
                horizontal
35         elsif rising_edge(clk25) then
36             if hcs = (hpixels - 1) then
37                 hcs <= "0000000000";
38                 vsenable <='1';
39             else
40                 hcs <= (hcs + 1);
41                 vsenable <='0';
42             end if;
43         end if;
44     end process;
45
46     -- Compteur pour le signal de synchronisation verticale vcs
47     process(clk25, rst)
48     begin
49         if rst='1' then
50             vcs <= "0000000000"; -- remise    zero du compteur
                vertical
51         elsif rising_edge(clk25) then
52             if vsenable='1' then
53                 if vcs =(vlines - 1) then
54                     vcs <= "0000000000"; -- remise a zero du
                        compteur vertical
55                 else
56                     vcs <= vcs + 1;
57                 end if;
58             end if;
59         end if;
60     end process;
61
62     -- Generation des signaux de sortie
63     hsync <= '0' when hcs < 96 else '1'; -- synchronisation
```

```
        horizontale
64      vsync <= '0' when vcs < 2 else '1';  -- synchronisation
        verticale
65
66      vidon <= '1' when ((hcs >= hbp) and (hcs < hfp) and (vcs >=
        vbp) and (vcs < vfp)) else '0'; -- signal video active
67      hc <= hcs;
68      vc <= vcs;
69  end Behavioral;
```

Listing 3: Code VHDL pour le composant VGA_640x480

```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3
4   entity VGA_640x480_tb is
5   end VGA_640x480_tb;
6
7   architecture Behavioral of VGA_640x480_tb is
8
9       signal clk25,rst,hsync,vsync,vidon : STD_LOGIC;
10      signal hc,vc : STD_LOGIC_VECTOR (9 downto 0);
11      component VGA_640x480 is
12          Port ( clk25 : in STD_LOGIC;
13                 rst : in STD_LOGIC;
14                 hsync : out STD_LOGIC;
15                 vsync : out STD_LOGIC;
16                 hc : out STD_LOGIC_VECTOR (9 downto 0);
17                 vc : out STD_LOGIC_VECTOR (9 downto 0);
18                 vidon : out STD_LOGIC);
19      end component;
20
21  begin
22
23      VGA_640x480_test : VGA_640x480 port map(clk25 => clk25, rst =>
            rst, hsync => hsync, vsync => vsync, hc => hc, vc => vc,
           vidon => vidon);
24      clock : process is
25      begin
26          clk25 <= '0';
27          wait for 20ns;
28          clk25 <= '1';
29          wait for 20ns;
30      end process;
31      reset : process is
32      begin
33          rst <= '0';
34          wait for 21ns;
35          rst <= '1';
36          wait for 2ns;
37          rst <= '0';
38          wait;
```

```
39      end process;
40
41  end Behavioral;
```

Listing 4: Code VHDL pour le composant VGA_640x480_tb

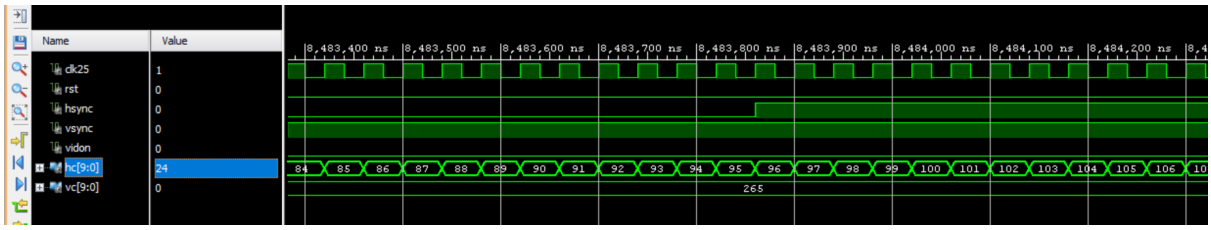

Figure 2: Résultat du test bench de VGA_640x480

Interprétation :

Les signaux de synchronisation horizontale (hsync) et verticale (vsync) ont été générés correctement, permettant un affichage stable sur l'écran, voir la figure 2. La génération correcte de ces signaux est cruciale pour un affichage VGA stable, indiquant que le contrôleur peut gérer les timings nécessaires pour afficher des images sans distorsion.

4 Composant VGA_STRIPES

4.1 Objectif : Affecter les valeurs des pixels en fonction de HC, VC et VIDON.

4.2 Fonctionnement : Conditions d'affichage

Les valeurs des couleurs rouge (red), vert (green) et bleu (blue) sont déterminées en fonction des valeurs des compteurs horizontaux (HC) et verticaux (VC) ainsi que du signal VIDON.

- Red : La valeur de la composante rouge est déterminée par le bit 4 de VC. Si VIDON est actif, la composante rouge est définie par la répétition de ce bit.
- Green : La valeur de la composante verte est l'inverse de la composante rouge.
- Blue : La composante bleue est fixée à zéro.

5 Composant TOP_VGA_STRIPES

Objectif : Instancier les composants nécessaires pour l'architecture initiale afin d'afficher des lignes rouges et vertes sur tout l'écran. L'architecture est décrite dans la figure 3.

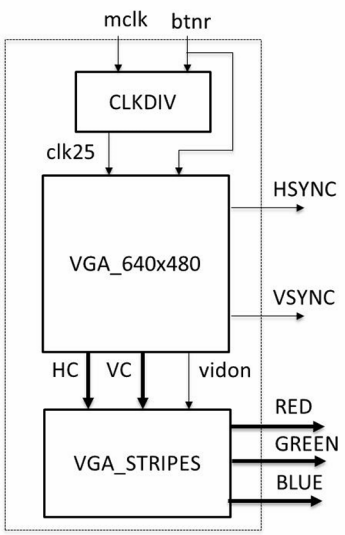

Figure 3: Interface et architecture du composant TOP_VGA_SPRITES

5.1 Explication des Composants et Signaux

5.1.1 TOP_VGA_STRIPES

Entité principale qui regroupe tous les composants nécessaires pour générer les signaux VGA et afficher des lignes rouges et vertes sur l'écran.

5.1.2 Ports :

- clk : Entrée de l'horloge principale.
- rst : Entrée de réinitialisation.
- hsync : Sortie du signal de synchronisation horizontale.
- vsync : Sortie du signal de synchronisation verticale.
- RGB : Sortie des valeurs RGB pour les pixels.

5.1.3 Signaux Internes :

- clk25 : Signal d'horloge à 25 MHz généré par le composant gen_pixel_clk.
- vidon : Signal d'activation vidéo, indiquant si les pixels doivent être affichés.
- vc et hc : Compteurs verticaux et horizontaux utilisés pour générer les signaux de synchronisation et déterminer les coordonnées des pixels.

5.1.4 Composants :

- VGA_640x480 : Composant responsable de générer les signaux de synchronisation horizontale et verticale, ainsi que les compteurs de pixels.
- gen_pixel_clk : Composant générant l'horloge à 25 MHz à partir de l'horloge principale.
- VGA_STRIPES : Composant responsable de générer les signaux RGB.

5.1.5 Instanciations :

- **gen_pixel_clk_c :** Instanciation du générateur de l'horloge des pixels.
- **VGA_640x480_c :** Instanciation du composant VGA_640x480.
- **VGA_STRIPES_c :** Instanciation du composant VGA_STRIPES.

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity TOP_VGA_SPRITES is
5      Port ( clk : in STD_LOGIC;
6             rst : in STD_LOGIC;
7             hsync : out STD_LOGIC;
8             vsync : out STD_LOGIC;
9             RGB : out STD_LOGIC_VECTOR (11 downto 0));
10 end TOP_VGA_SPRITES;
11
12 architecture Behavioral of TOP_VGA_SPRITES is
13     signal clk25, vidon : STD_LOGIC;
14     signal vc, hc: STD_LOGIC_VECTOR(9 downto 0);
15     component VGA_640x480 is
16         Port ( clk25 : in STD_LOGIC;
17                rst : in STD_LOGIC;
18                hsync : out STD_LOGIC;
19                vsync : out STD_LOGIC;
20                hc : out STD_LOGIC_VECTOR (9 downto 0);
21                vc : out STD_LOGIC_VECTOR (9 downto 0);
22                vidon : out STD_LOGIC);
23     end component;
24
25     component VGA_STRIPES is
26         Port ( HC : in STD_LOGIC_VECTOR (9 downto 0);
27                VC : in STD_LOGIC_VECTOR (9 downto 0);
28                vidon : in STD_LOGIC;
29                RGB : out STD_LOGIC_VECTOR (11 downto 0));
30     end component;
31
32     component gen_pixel_clk is
33         Port ( rst : in STD_LOGIC;
34                clk : in STD_LOGIC;
35                clk25 : out STD_LOGIC);
36     end component;
37
38 begin
39
40     gen_pixel_clk_c : gen_pixel_clk port map(rst => rst, clk =>
       clk, clk25 => clk25);
41     VGA_640x480_c : VGA_640x480 port map(clk25 => clk25, rst =>
       rst, hsync => hsync, vsync => vsync, hc => hc, vc => vc,
       vidon => vidon);
```

```
42      VGA_STRIPES_c : VGA_STRIPES port map(hc => hc, vc => vc, vidon
            => vidon, RGB => RGB);
43
44  end Behavioral;
```

Listing 5: Code VHDL pour le composant TOP_VGA_SPRITES

```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3
4   entity TOP_VGA_SPRITES_tb is
5   end TOP_VGA_SPRITES_tb;
6
7   architecture Behavioral of TOP_VGA_SPRITES_tb is
8
9   signal clk, rst, hsync, vsync: STD_LOGIC;
10  signal RGB : STD_LOGIC_VECTOR (11 downto 0);
11
12  component TOP_VGA_SPRITES is
13      Port ( clk : in STD_LOGIC;
14             rst : in STD_LOGIC;
15             hsync : out STD_LOGIC;
16             vsync : out STD_LOGIC;
17             RGB : out STD_LOGIC_VECTOR (11 downto 0));
18  end component;
19  begin
20
21  TOP_VGA_SPRITES_test : TOP_VGA_SPRITES port map (clk => clk, rst
       => rst, hsync => hsync, vsync => vsync, RGB => RGB);
22
23  clock : process is
24  begin
25      clk <= '0';
26      wait for 10ns;
27      clk <= '1';
28      wait for 10ns;
29  end process;
30
31  reset : process is
32  begin
33      rst <= '0';
34      wait for 9ns;
35      rst <= '1';
36      wait for 2ns;
37      rst <= '0';
38      wait;
39  end process;
40
41  end Behavioral;
```

Listing 6: Code VHDL pour le composant TOP_VGA_SPRITES_tb

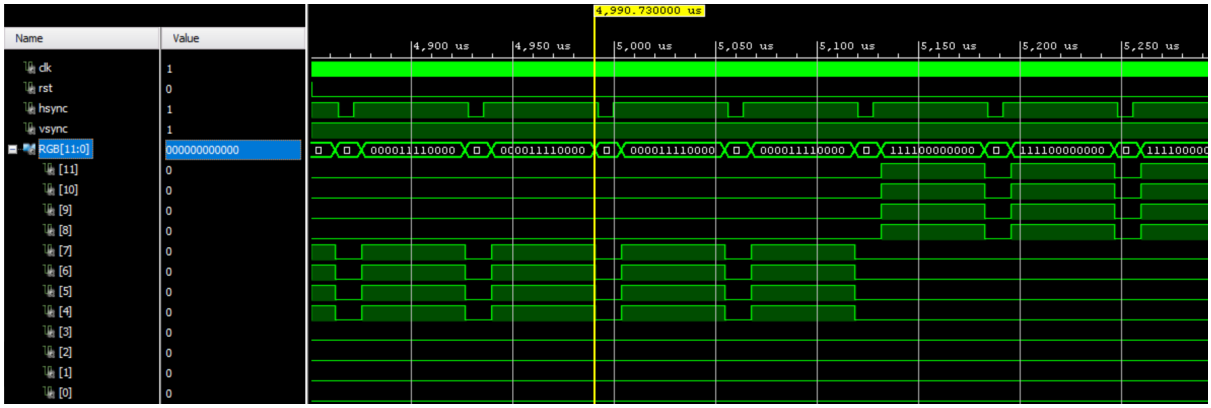

Figure 4: Résultat du test bench de Top VGA Stripes

Interprétation :

Les valeurs RGB ont été générées correctement, comme le montre la figure 4, permettant l'affichage des lignes rouges et vertes. Cela indique que le système est capable de gérer efficacement les signaux de couleur et de les synchroniser avec les signaux de synchronisation horizontale et verticale, assurant ainsi un affichage précis et stable des motifs de base.

5.2 Simulation et Vérification

Les tests effectués incluent la vérification des signaux de synchronisation horizontale et verticale, ainsi que l'affichage correct des lignes rouges et vertes en fonction des coordonnées et du signal VIDON.

Figure 5: Validation du programme Top_VGA_STRIPES par l'affichage des lignes rouges et vertes)

Interprétation :

Les lignes rouges et vertes ont été affichées correctement sur tout l'écran, voir la figure 5. Cela démontre que le système est capable de gérer des motifs simples et que les signaux

RGB sont générés et synchronisés de manière adéquate avec les signaux de synchronisation, prouvant ainsi la capacité du système à afficher des motifs de base.

6 Composant VGA_PROM

Objectif : Afficher un sprite en haut à gauche de l'écran en utilisant une PROM.

6.1 Constantes et Paramètres

- **hbp :** Horizontal back porch = 144 ("0010010000")
- **vbp :** Vertical back porch = 31 ("0000011111")
- **w :** Largeur du sprite = 32 pixels ("0000100000")
- **h :** Hauteur du sprite = 16 lignes ("0000010000")

6.2 Fonctionnement :

6.2.1 Calcul des Adresses

Les adresses pour accéder à la PROM sont calculées en fonction des compteurs horizontaux (HC) et verticaux (VC), ainsi que des positions du sprite.

- rom_addr : Adresse de la ligne du sprite, calculée à partir de VC et vbp.
- rom_pix : Adresse de la colonne du sprite, calculée à partir de HC et hbp.

6.2.2 Conditions d'affichage

Le signal spriteon détermine si le pixel actuel se trouve dans la zone d'affichage du sprite.

- Affectation des Couleurs : Les couleurs des pixels (RED, GREEN, BLUE) sont affectées en fonction des données lues dans la PROM et du signal spriteon.
- Code VHDL : Extrait du code pour le composant VGA_PROM.

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_UNSIGNED.ALL;
4  use ieee.std_logic_1164.all;
5  use ieee.numeric_std.all;
6
7  entity VGA_PROM is
8      Port ( clk_1hz : in STD_LOGIC;
9             HC : in STD_LOGIC_VECTOR (9 downto 0);
10            VC : in STD_LOGIC_VECTOR (9 downto 0);
11            VIDON : in STD_LOGIC;
12            M : in STD_LOGIC_VECTOR (0 to 31);
13            ROM_ADDR4 : out STD_LOGIC_VECTOR (3 downto 0);
14            RGB : out STD_LOGIC_VECTOR (11 downto 0);
```

```
15              sw : in STD_LOGIC_VECTOR (3 downto 0));
16 end VGA_PROM;
17
18 architecture Behavioral of VGA_PROM is
19
20 constant hbp: std_logic_vector(9 downto 0) :="0010010000"; --
      horizontal back porch = 128 + 16 = 144 (ou 96 + 48)
21 constant vbp: std_logic_vector(9 downto 0) :="0000011111"; --
      vertical back porch = 2 + 29 = 31
22 constant height: std_logic_vector(9 downto 0) :="0000010000"; --
      16 lines
23 constant width: std_logic_vector(9 downto 0) := "0000100000"; --
      32 pixels
24
25 constant dx: std_logic_vector(9 downto 0) :="0000001000"; -- 64
26 constant dy: std_logic_vector(8 downto 0) :="000001000"; -- 48
27
28 signal rom_addr : STD_LOGIC_VECTOR (9 downto 0);
29 signal rom_pix : STD_LOGIC_VECTOR (9 downto 0);
30 signal spriteon : STD_LOGIC;
31 signal rgb_sig : STD_LOGIC_VECTOR (2 downto 0);
32
33 signal x : STD_LOGIC_VECTOR (9 downto 0):= "0000000000";
34 signal y : STD_LOGIC_VECTOR (8 downto 0):= "000000000";
35
36 begin
37    game : process(clk_1hz,sw)
38    begin
39         if clk_1hz'event and clk_1hz = '1' then
40             case sw is
41                 when "0001" => x <= x+dx; y <= y;
42                 when "0010" => x <= x-dx; y <= y;
43                 when "0100" => x <= x; y <= y-dy;
44                 when "1000" => x <= x; y <= y+dy;
45                 when "0101" => x <= x+dx; y <= y-dy;
46                 when "1001" => x <= x+dx; y <= y+dy;
47                 when "0110" => x <= x-dx; y <= y-dy;
48                 when "1010" => x <= x-dx; y <= y+dy;
49                 when others => x <= x; y <= y;
50             end case;
51         end if;
52    end process;
53
54    calculate_addr_pix : process(VC, HC, rom_addr, VIDON, spriteon,
          rom_pix)
55     variable j: integer;
56     begin
57         j:= conv_integer(rom_pix);
58         rgb_sig <= M(j)&M(j)&M(j);
59
60         if spriteon = '1' and VIDON = '1' then
```

```
61              RGB <= rgb_sig(0)&rgb_sig(0)&rgb_sig(0)&rgb_sig(0)&
                   rgb_sig(1)&rgb_sig(1)&rgb_sig(1)&rgb_sig(1)&rgb_sig
                   (2)&rgb_sig(2)&rgb_sig(2)&rgb_sig(2);
62          else
63              RGB <= "000000000000";
64          end if;
65      end process;
66
67      rom_addr <= VC-vbp-y;
68      rom_pix <= HC-hbp-x;
69
70      ROM_ADDR4 <= rom_addr(3 downto 0);
71
72      spriteon <= '1' when ((HC > hbp+x)
73      and (HC < hbp+width+x)
74      and (VC > vbp+y)
75      and (VC < vbp+height+y)) else '0';
76
77  end Behavioral;
```

Listing 7: Code VHDL pour le composant VGA_PROM

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_UNSIGNED.ALL;
4
5  entity PROM is
6      Port ( addr : in STD_LOGIC_VECTOR (3 downto 0);
7             m : out STD_LOGIC_VECTOR (0 to 31));
8  end PROM;
9
10 architecture Behavioral of PROM is
11 type rom_array is array (NATURAL range <>) of std_logic_vector(0
       to 31);
12 constant rom:rom_array:= (
13                              "0000000000000000",
14                              "0111100000101010",
15                              "0111101110101010",
16                              "0111101010101010",
17                              "0111101111111010",
18                              "0111100000000000",
19                              "0111111101111010",
20                              "0111110001000010",
21                              "0110000101111010",
22                              "0110100100001010",
23                              "0110111111111010",
24                              "0000100000000000",
25                              "0011100100111110",
26                              "0000000000111110",
27                              "0111111111111110",
28                              "0000000000000000"
29                            );
```

```
30 begin
31
32 process(addr)
33     variable j: integer;
34     begin
35         j:= conv_integer(addr);
36         m <= rom(j);
37     end process;
38
39 end Behavioral;
```

Listing 8: Code VHDL pour le composant PROM

7 Composant TOP_VGA_PROM_1

Objectif : Instancier les composants nécessaires pour l'architecture finale afin d'afficher un sprite en utilisant une PROM. L'architecture est décrite dans la figure 6.

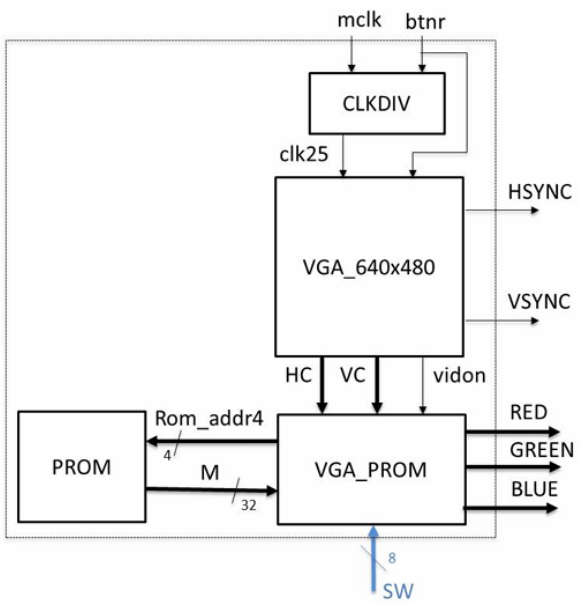

Figure 6: Interface du composant TOP_VGA_PROM

7.1 Explication des Composants et Signaux

7.1.1 TOP_VGA_PROM

Entité principale qui regroupe tous les composants nécessaires pour générer les signaux VGA et afficher une icône en utilisant une PROM.

7.1.2 Ports

- clk : Entrée de l'horloge principale.
- rst : Entrée de réinitialisation.

- hsync : Sortie du signal de synchronisation horizontale.
- vsync : Sortie du signal de synchronisation verticale.
- RGB : Sortie des valeurs RGB pour les pixels.
- sw : Entrée des switchs pour contrôler l'icône.

7.1.3 Signaux Internes :

- clk25 : Signal d'horloge à 25 MHz généré par le composant gen_pixel_clk.
- clk_1hz : Signal d'horloge à 1 Hz généré par le composant gen_pixel_clk.
- vidon : Signal d'activation vidéo, indiquant si les pixels doivent être affichés.
- vc et hc : Compteurs verticaux et horizontaux utilisés pour générer les signaux de synchronisation et déterminer les coordonnées des pixels.
- M : Données lues de la PROM.
- ROM_ADDR4 : Adresse de la PROM.

7.1.4 Composants :

- **VGA_640x480 :** Composant responsable de générer les signaux de synchronisation horizontale et verticale, ainsi que les compteurs de pixels.
- **VGA_PROM :** Composant responsable de l'affichage de l'icône en utilisant les données de la PROM.
- **gen_pixel_clk :** Composant générant l'horloge à 25 MHz et l'horloge à 1 Hz à partir de l'horloge principale.
- **PROM :** Composant représentant la mémoire PROM contenant les données de l'icône.

7.1.5 Instanciations :

- **gen_pixel_clk_c :** Instanciation du générateur de l'horloge des pixels.
- **VGA_640x480_c :** Instanciation du composant VGA_640x480.
- **VGA_PROM_c :** Instanciation du composant VGA_PROM.
- **PROM_c :** Instanciation du composant PROM.

```
1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3
4 entity TOP_VGA_PROM is
5     Port ( clk : in STD_LOGIC;
6            rst : in STD_LOGIC;
7            hsync : out STD_LOGIC;
```

```
8             vsync : out STD_LOGIC;
9             RGB : out STD_LOGIC_VECTOR (11 downto 0);
10            sw : in STD_LOGIC_VECTOR (3 downto 0));
11 end TOP_VGA_PROM;
12
13 architecture Behavioral of TOP_VGA_PROM is
14     signal clk25, vidon, clk_1hz : STD_LOGIC;
15     signal vc, hc: STD_LOGIC_VECTOR(9 downto 0);
16     signal M : STD_LOGIC_VECTOR (31 downto 0);
17     signal ROM_ADDR4 : STD_LOGIC_VECTOR (3 downto 0);
18
19     component VGA_640x480 is
20         Port ( clk25 : in STD_LOGIC;
21                rst : in STD_LOGIC;
22                hsync : out STD_LOGIC;
23                vsync : out STD_LOGIC;
24                hc : out STD_LOGIC_VECTOR (9 downto 0);
25                vc : out STD_LOGIC_VECTOR (9 downto 0);
26                vidon : out STD_LOGIC);
27     end component;
28
29     component VGA_PROM is
30         Port ( clk_1hz : in STD_LOGIC;
31                HC : in STD_LOGIC_VECTOR (9 downto 0);
32                VC : in STD_LOGIC_VECTOR (9 downto 0);
33                VIDON : in STD_LOGIC;
34                M : in STD_LOGIC_VECTOR (31 downto 0);
35                ROM_ADDR4 : out STD_LOGIC_VECTOR (3 downto 0);
36                RGB : out STD_LOGIC_VECTOR (11 downto 0);
37                sw : in STD_LOGIC_VECTOR (3 downto 0));
38     end component;
39
40     component gen_pixel_clk is
41         Port ( rst : in STD_LOGIC;
42                clk : in STD_LOGIC;
43                clk25 : out STD_LOGIC;
44                clk_1hz : out STD_LOGIC);
45     end component;
46
47     component PROM is
48         Port ( addr : in STD_LOGIC_VECTOR (3 downto 0);
49                m : out STD_LOGIC_VECTOR (31 downto 0));
50     end component;
51
52 begin
53
54     gen_pixel_clk_c : gen_pixel_clk port map(rst => rst, clk =>
          clk, clk25 => clk25, clk_1hz => clk_1hz);
55     VGA_640x480_c : VGA_640x480 port map(clk25 => clk25, rst =>
          rst, hsync => hsync, vsync => vsync, hc => hc, vc => vc,
          vidon => vidon);
```

```
56      VGA_PROM_c : VGA_PROM port map(clk_1hz => clk_1hz, hc => hc,
           vc => vc, vidon => vidon, M => M, ROM_ADDR4 => ROM_ADDR4,
           RGB => RGB, sw => sw);
57      PROM_c : PROM port map(addr => ROM_ADDR4, m => M);
58
59  end Behavioral;
```

Listing 9: Code VHDL pour le composant TOP_VGA_PROM

7.2 Simulation et Vérification

Les tests effectués incluent la vérification des signaux de synchronisation horizontale et verticale, ainsi que l'affichage correct de l'icône en fonction des coordonnées et du signal VIDON.

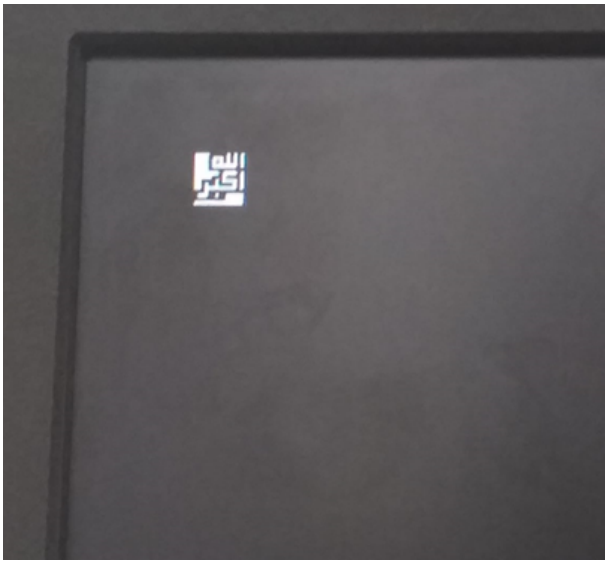

Figure 7: Validation de l'affichage et du déplacement de l'icône générée à partir du PROM

Interprétation :

Une icône a été affichée en haut à gauche de l'écran en utilisant une PROM, voir la figure 7. L'utilisation de la PROM pour stocker et afficher des sprites montre que le système peut gérer des données graphiques stockées en mémoire et les afficher correctement à l'écran, démontrant ainsi l'efficacité de la mémoire PROM pour le stockage des sprites.

8 Utilisation des Block ROM

Objectif : Utiliser des blocs mémoires internes pour stocker et afficher des sprites complexes.

8.1 Génération de la ROM

- **Configuration :** Utilisation du Block Memory Generator pour configurer la mémoire en "Single Port ROM" avec les caractéristiques nécessaires pour stocker les sprites.
- **Initialisation :** Chargement du fichier COE pour initialiser la mémoire ROM avec les données du sprite.

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

LIBRARY blk_mem_gen_v8_3_1;
USE blk_mem_gen_v8_3_1.blk_mem_gen_v8_3_1;

ENTITY blk_mem_gen_2 IS
  PORT (
    clka : IN STD_LOGIC;
    addra : IN STD_LOGIC_VECTOR(16 DOWNTO 0);
    douta : OUT STD_LOGIC_VECTOR(7 DOWNTO 0)
  );
END blk_mem_gen_2;
```

Listing 10: Code VHDL pour le composant blk_mem_gen_2

8.2 Composant VGA_PROM_2

8.2.1 Paramètres :

- **hbp :** Horizontal back porch = 144 ("0010010000")
- **vbp :** Vertical back porch = 31 ("0000011111")
- **height :** Hauteur de l'image = 240 lignes ("0100101000")
- **width :** Largeur de l'image = 160 pixels ("0011110000")
- **Adressage de la ROM :** Calcul de l'adresse de la ROM (rom_addr16) en fonction des coordonnées des pixels (xpix, ypix).
- **Affectation des Couleurs :** Les couleurs des pixels (RED, GREEN, BLUE) sont affectées en fonction des données lues dans la ROM et du signal spriteon.

9 Composant TOP_VGA_PROM_2

Objectif : Instancier les composants nécessaires pour l'architecture finale afin d'afficher des images en utilisant une Block ROM. L'architecture est décrite dans la figure 8.

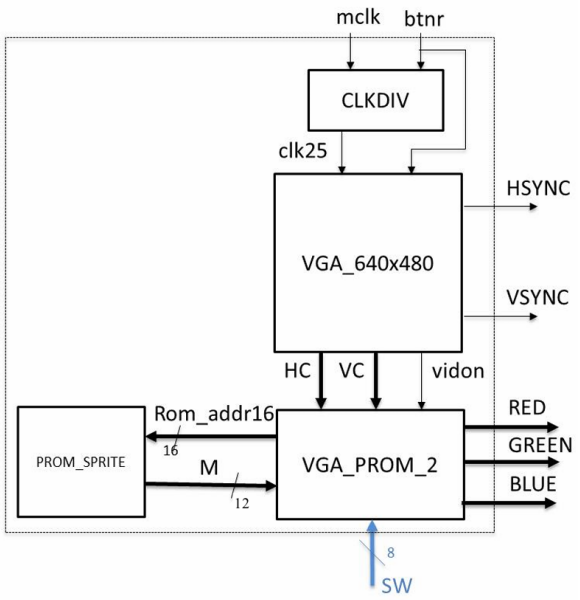

Figure 8: Architecture TOP_VGA_PROM_2

9.1 Explication des Composants et Signaux

9.1.1 TOP_VGA_PROM_2

Entité principale qui regroupe tous les composants nécessaires pour générer les signaux VGA et afficher des sprites complexes en utilisant une Block ROM.

9.1.2 Ports :

- **clk :** Entrée de l'horloge principale.
- **rst :** Entrée de réinitialisation.
- **hsync :** Sortie du signal de synchronisation horizontale.
- **vsync :** Sortie du signal de synchronisation verticale.
- **RGB :** Sortie des valeurs RGB pour les pixels.
- **sw :** Entrée des switchs pour contrôler les sprites.

9.1.3 Signaux Internes :

- **clk25 :** Signal d'horloge à 25 MHz généré par le composant gen_pixel_clk.
- **clk_1hz :** Signal d'horloge à 1 Hz généré par le composant gen_pixel_clk.
- **vidon :** Signal d'activation vidéo, indiquant si les pixels doivent être affichés.

- **vc et hc :** Compteurs verticaux et horizontaux utilisés pour générer les signaux de synchronisation et déterminer les coordonnées des pixels.
- **M :** Données lues de la Block ROM.
- **ROM_ADDR17 :** Adresse de la Block ROM.

9.1.4 Composants :

- **VGA_640x480 :** Composant responsable de générer les signaux de synchronisation horizontale et verticale, ainsi que les compteurs de pixels.
- **VGA_PROM_2 :** Composant responsable de l'affichage des sprites en utilisant les données de la Block ROM.
- **gen_pixel_clk :** Composant générant l'horloge à 25 MHz et l'horloge à 1 Hz à partir de l'horloge principale.
- **blk_mem_gen_1 :** Composant représentant la Block ROM contenant les données des sprites.

9.1.5 Instanciations:

- **gen_pixel_clk_c :** Instanciation du générateur de l'horloge des pixels.
- **VGA_640x480_c :** Instanciation du composant VGA_640x480.
- **VGA_PROM_c :** Instanciation du composant VGA_PROM_2.
- **PROM_c :** Instanciation du composant blk_mem_gen_1.

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity TOP_VGA_PROM_2 is
5      Port ( clk : in STD_LOGIC;
6             rst : in STD_LOGIC;
7             hsync : out STD_LOGIC;
8             vsync : out STD_LOGIC;
9             RGB : out STD_LOGIC_VECTOR (11 downto 0);
10            sw : in STD_LOGIC_VECTOR (3 downto 0));
11 end TOP_VGA_PROM_2;
12
13 architecture Behavioral of TOP_VGA_PROM_2 is
14     signal clk25, vidon, clk_1hz : STD_LOGIC;
15     signal vc, hc: STD_LOGIC_VECTOR(9 downto 0);
16     signal M : STD_LOGIC_VECTOR (7 downto 0);
17     signal ROM_ADDR17 : STD_LOGIC_VECTOR (16 downto 0);
18
19     component VGA_640x480 is
20         Port ( clk25 : in STD_LOGIC;
21                rst : in STD_LOGIC;
```

```
22                  hsync : out STD_LOGIC;
23                  vsync : out STD_LOGIC;
24                  hc : out STD_LOGIC_VECTOR (9 downto 0);
25                  vc : out STD_LOGIC_VECTOR (9 downto 0);
26                  vidon : out STD_LOGIC);
27      end component;
28
29      component VGA_PROM_2 is
30          Port ( clk_1hz : in STD_LOGIC;
31                 HC : in STD_LOGIC_VECTOR (9 downto 0);
32                 VC : in STD_LOGIC_VECTOR (9 downto 0);
33                 VIDON : in STD_LOGIC;
34                 M : in STD_LOGIC_VECTOR (7 downto 0);
35                 ROM_ADDR : out STD_LOGIC_VECTOR (16 downto 0);
36                 RGB : out STD_LOGIC_VECTOR (11 downto 0);
37                 sw : in STD_LOGIC_VECTOR (3 downto 0));
38      end component;
39
40      component gen_pixel_clk is
41          Port ( rst : in STD_LOGIC;
42                 clk : in STD_LOGIC;
43                 clk25 : out STD_LOGIC;
44                 clk_1hz : out STD_LOGIC);
45      end component;
46
47      component blk_mem_gen_2 is
48        PORT (
49          clka : IN STD_LOGIC;
50          addra : IN STD_LOGIC_VECTOR(16 DOWNTO 0);
51          douta : OUT STD_LOGIC_VECTOR(7 DOWNTO 0)
52        );
53      end component;
54
55  begin
56
57      gen_pixel_clk_c : gen_pixel_clk port map(rst => rst, clk =>
           clk, clk25 => clk25, clk_1hz => clk_1hz);
58      VGA_640x480_c : VGA_640x480 port map(clk25 => clk25, rst =>
           rst, hsync => hsync, vsync => vsync, hc => hc, vc => vc,
           vidon => vidon);
59      VGA_PROM_c : VGA_PROM_2 port map(clk_1hz => clk_1hz, hc => hc,
            vc => vc, vidon => vidon, M => M, ROM_ADDR => ROM_ADDR17,
           RGB => RGB, sw => sw);
60      PROM_c : blk_mem_gen_2 port map(clka => clk25, addra =>
           ROM_ADDR17, douta => M);
61
62  end Behavioral;
```

Listing 11: Code VHDL pour le composant TOP_VGA_PROM_2

9.2 Simulation et Vérification

Les tests effectués incluent la vérification des signaux de synchronisation horizontale et verticale, ainsi que l'affichage correct des sprites en fonction des coordonnées et du signal VIDON.

Figure 9: Validation de l'affichage des images (au format .coe (Coefficient) générées par le code MATLAB fourni en annexe) à partir du bloc ROM

Interprétation :

Les tests ont confirmé que les signaux de synchronisation et l'affichage des images étaient corrects, voir la figure 9. La capacité à afficher des images complexes à partir de la Block ROM indique que le système est capable de gérer des graphiques avancés et de les afficher de manière stable et précise, démontrant ainsi la robustesse et la précision du système.

10 Implémentation

10.1 Intégration du fichier XDC

- Le fichier XDC (Xilinx Design Constraints) est utilisé pour définir les contraintes de placement des broches et les propriétés des signaux pour la carte Basys 3.
- Voici un exemple de fichier XDC pour la carte Basys 3 :

```
# Clock signal
set_property PACKAGE_PIN W5 [get_ports clk]

    set_property IOSTANDARD LVCMOS33 [get_ports clk]
    create_clock -add -name sys_clk_pin -period 10.00 -
       waveform {0 5} [get_ports clk]

# Buttons
set_property PACKAGE_PIN U18 [get_ports rst]
    set_property IOSTANDARD LVCMOS33 [get_ports rst]
set_property PACKAGE_PIN T18 [get_ports {sw[2]}]
```

```
    set_property IOSTANDARD LVCMOS33 [get_ports {sw[2]}]
set_property PACKAGE_PIN W19 [get_ports {sw[1]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {sw[1]}]
set_property PACKAGE_PIN T17 [get_ports {sw[0]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {sw[0]}]
set_property PACKAGE_PIN U17 [get_ports {sw[3]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {sw[3]}]

# VGA Connector
set_property PACKAGE_PIN G19 [get_ports {RGB[8]}]

    set_property IOSTANDARD LVCMOS33 [get_ports {RGB[8]}]
set_property PACKAGE_PIN H19 [get_ports {RGB[9]}]

    set_property IOSTANDARD LVCMOS33 [get_ports {RGB[9]}]
set_property PACKAGE_PIN J19 [get_ports {RGB[10]}]

    set_property IOSTANDARD LVCMOS33 [get_ports {RGB[10]}]
set_property PACKAGE_PIN N19 [get_ports {RGB[11]}]

    set_property IOSTANDARD LVCMOS33 [get_ports {RGB[11]}]
set_property PACKAGE_PIN N18 [get_ports {RGB[0]}]

    set_property IOSTANDARD LVCMOS33 [get_ports {RGB[0]}]
set_property PACKAGE_PIN L18 [get_ports {RGB[1]}]

    set_property IOSTANDARD LVCMOS33 [get_ports {RGB[1]}]
set_property PACKAGE_PIN K18 [get_ports {RGB[2]}]

    set_property IOSTANDARD LVCMOS33 [get_ports {RGB[2]}]
set_property PACKAGE_PIN J18 [get_ports {RGB[3]}]

    set_property IOSTANDARD LVCMOS33 [get_ports {RGB[3]}]
set_property PACKAGE_PIN J17 [get_ports {RGB[4]}]

    set_property IOSTANDARD LVCMOS33 [get_ports {RGB[4]}]
set_property PACKAGE_PIN H17 [get_ports {RGB[5]}]

    set_property IOSTANDARD LVCMOS33 [get_ports {RGB[5]}]
set_property PACKAGE_PIN G17 [get_ports {RGB[6]}]

    set_property IOSTANDARD LVCMOS33 [get_ports {RGB[6]}]
set_property PACKAGE_PIN D17 [get_ports {RGB[7]}]

    set_property IOSTANDARD LVCMOS33 [get_ports {RGB[7]}]
set_property PACKAGE_PIN P19 [get_ports hsync]
    set_property IOSTANDARD LVCMOS33 [get_ports hsync]
set_property PACKAGE_PIN R19 [get_ports vsync]
```

```
46    set_property IOSTANDARD LVCMOS33 [get_ports vsync]
```

Listing 12: Code exemple de fichier XDC pour la carte Basys 3caption

10.2 Génération du Bitstream

Après avoir intégré le fichier XDC et vérifié que toutes les connexions sont correctes, vous pouvez générer le bitstream.

- Utilisez l'outil de synthèse et d'implémentation de votre environnement de développement (par exemple, Vivado) pour générer le fichier bitstream (.bit).
- Assurez-vous que toutes les étapes de synthèse, d'implémentation et de génération du bitstream se terminent sans erreurs.

10.3 Programmation de la carte Basys 3

Connectez la carte Basys 3 à votre ordinateur via un câble USB.

- Ouvrez l'outil de programmation (par exemple, Vivado Hardware Manager).
- Importez le fichier bitstream généré (.bit) dans l'outil de programmation.
- Sélectionnez la carte Basys 3 comme cible de programmation.
- Programmez la carte en téléchargeant le bitstream sur le FPGA.
- Une fois la programmation terminée, vérifiez que les images ou les sprites s'affichent correctement sur l'écran connecté au port VGA de la carte Basys 3.

11 Conclusion

Au cours de ce projet, nous avons conçu et implémenté plusieurs architectures pour afficher des graphiques sur un écran VGA en utilisant un FPGA. Voici un récapitulatif des étapes et des résultats obtenus :

Le composant VGA_640x480 avait pour objectif de générer les signaux de synchronisation horizontale et verticale nécessaires pour un affichage VGA standard. Les signaux ont été correctement générés, permettant un affichage stable sur l'écran.

Le composant VGA_STRIPES visait à afficher des lignes rouges et vertes sur tout l'écran. Ces lignes ont été affichées correctement, démontrant la capacité du système à gérer des motifs simples.

Le composant VGA_PROM avait pour objectif d'afficher une icône en haut à gauche de l'écran en utilisant une PROM. L'icône a été affichée avec succès, montrant l'utilisation efficace de la mémoire PROM pour le stockage des sprites.

Le composant VGA_PROM_2 visait à utiliser des blocs mémoires internes (Block ROM) pour stocker et afficher des sprites complexes. Les sprites complexes ont été affichés

correctement, démontrant la capacité du système à gérer des graphiques plus élaborés.

Pour l'implémentation, les étapes ont inclus l'intégration du fichier XDC, la génération du bitstream, et la programmation de la carte Basys 3. Le système a été programmé avec succès sur la carte Basys 3, et les graphiques ont été affichés comme prévu.

12 Perspectives

Les travaux réalisés ouvrent la voie à plusieurs améliorations et développements futurs :

Pour les améliorations de la résolution, il est proposé d'augmenter la résolution de l'affichage pour des graphiques plus détaillés et précis, ainsi que d'optimiser les timings et les ressources pour supporter des résolutions plus élevées.

Concernant l'animation et le mouvement, il s'agit d'intégrer des fonctionnalités pour animer les sprites et les déplacer de manière fluide à l'écran, en utilisant des algorithmes de gestion de mouvement pour des animations plus complexes.

En termes d'interactivité, il est suggéré d'ajouter des contrôles interactifs pour permettre à l'utilisateur de manipuler les graphiques en temps réel, et d'intégrer des capteurs ou des interfaces utilisateur pour une interaction plus dynamique.

Pour l'optimisation des ressources, il est recommandé d'optimiser l'utilisation des ressources FPGA pour améliorer les performances et réduire la consommation d'énergie, ainsi que d'explorer des techniques de compression pour stocker plus de données graphiques dans la mémoire limitée.

Enfin, pour les applications avancées, il est envisagé de développer des applications spécifiques telles que des jeux vidéo, des interfaces utilisateur graphiques, ou des systèmes de visualisation de données, et d'intégrer des fonctionnalités avancées de traitement d'image pour des applications plus sophistiquées.

Annexe

```
function img2 = IMG2coe8(imgfile, outfile)
% Create .coe file from .bmp .jpg image
% .coe file contains 8-bit words (bytes)
% each byte contains one 8-bit pixel
% color byte: [R2,R1,R0,G2,G1,G0,B1,B0]
% img2 = IMG2coe8(imgfile, outfile)
% img2 is 256-bit color image
% imgfile = input .bmp file / .jpg
% outfile = output .coe file
% Example:
% img2 = IMG2coe8('image.jpg', 'image.coe');

img = imread(imgfile);
height = size(img, 1);
width = size(img, 2);

s = fopen(outfile,'wb');  %opens the output file

fprintf(s,'%s\n','; VGA Memory Map ');
fprintf(s,'%s\n','; .COE file with hex coefficients ');
fprintf(s,'; Height: %d, Width: %d\n\n', height, width);
fprintf(s,'%s\n','memory_initialization_radix=16;');
fprintf(s,'%s\n','memory_initialization_vector=');

cnt = 0;
img2 = img;
for r=1:height
    for c=1:width
        cnt = cnt + 1;
        R = img(r,c,1);
        G = img(r,c,2);
        B = img(r,c,3);
        Rb = dec2bin(double(R),8);
        Gb = dec2bin(double(G),8);
        Bb = dec2bin(double(B),8);
        img2(r,c,1) = bin2dec([Rb(1:3) '00000']);
        img2(r,c,2) = bin2dec([Gb(1:3) '00000']);
        img2(r,c,3) = bin2dec([Bb(1:2) '000000']);
        Outbyte = [ Rb(1:3) Gb(1:3) Bb(1:2) ];
        if (Outbyte(1:4) == '0000')
            fprintf(s,'0%X',bin2dec(Outbyte));
        else
            fprintf(s,'%X',bin2dec(Outbyte));
        end
        if ((c == width) && (r == height))
            fprintf(s,'%c',';');
        else
            if (mod(cnt,32) == 0)
                fprintf(s,'%c\n',',');
```

```
50                else
51                    fprintf(s,'%c',',');
52                end
53            end
54        end
55    end
56
57    fclose(s);
```

Listing 13: Code MATLAB pour générer les images en format COE