

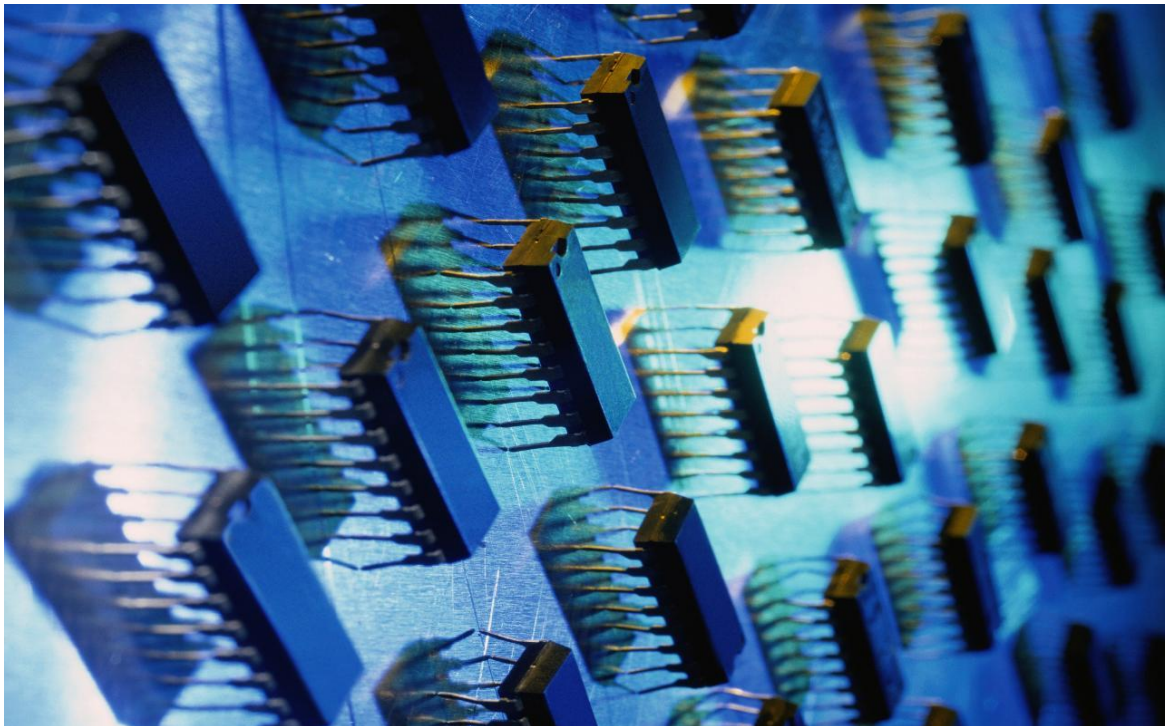
« DESSINE-MOI UN MOUTON »

Rapport de projet

Fred-Alain Codjia | Erwan Keraval | Lilia-Sara Kesouar | Germain Le

2008-2009

ING2 – GROUPE 3



Sommaire

1. Présentation du projet	3
1.1 Principe.....	3
1.2 Schéma fonctionnel de la fonction principale	3
2. Etude des fonctions de service	4
2.1 Fonction de service 1.....	4
2.2 Fonction de service 2.....	9
2.3 Fonction de service 3.....	13
2.4 Fonction de service 4.....	15
3. Etude de la fonction complète	18
3.1 Simulation de la fonction complète sans les améliorations	18
3.2 Etude de la fonction complète avec les améliorations	19
3.2.1 Simulation	19
3.2.2. Implémentation sur une carte Basys.....	23
4. Conclusion	23

1. Présentation du projet

1.1 Principe

Dans ce projet, nous avons pour objectif de synthétiser en langage VHDL une fonction (à implémenter sur un circuit FPGA) permettant d'afficher sur un écran VGA de résolution 640 × 480 une image, un logo ou un rectangle. Nous avons choisi d'afficher un rectangle de taille et de couleurs contrôlables grâce aux interrupteurs présents sur la carte Basys. L'objet pourra également être déplacé horizontalement et verticalement à l'aide des boutons poussoirs. Dans un premiers temps, nous analyserons chaque fonction de service, puis nous ferons une analyse de la fonction complète.

1.2 Schéma fonctionnel de la fonction principale

Voici le schéma fonctionnel de la fonction principale :

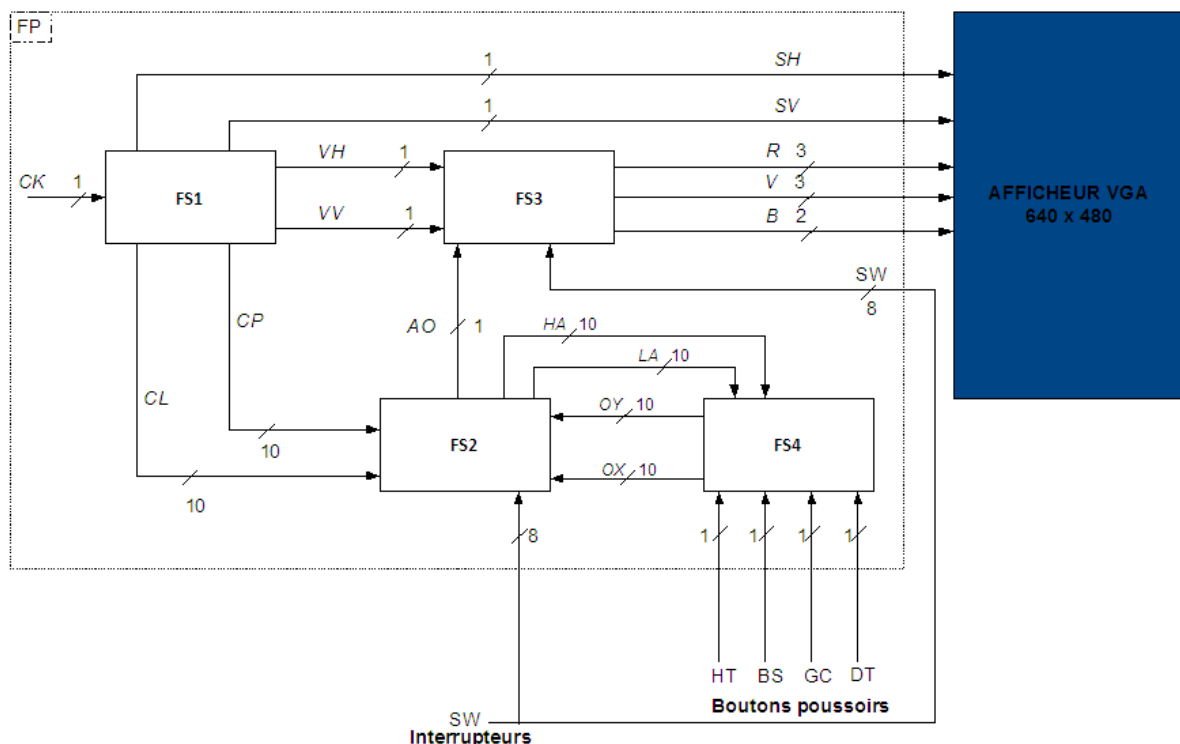


Figure 1 - Schéma fonctionnel de FP

CK : horloge 25 Mhz

SH, SV : synchro_V, synchro_H / Signaux de synchronisation VGA

VH, VV : valide_H, valide_V / Zones d'écriture sur l'écran

CL, CP : compteur_lignes, compteur_pixel / Compteurs horizontal et vertical

OX, OY : origine_X, origine_Y / Coordonnées de l'objet

AO : affiche_objet / Définition de l'objet

R, V, B : rouge, vert, bleu / Couleurs RGB

HT, BS, GC, DT : haut, bas, gauche, droite / Variables de déplacement

SW : switch / Variables de changement de taille et de couleur de l'objet

2. Etude des fonctions de service

Lors de l'étude des fonctions de services, nous n'analyserons que les courbes de simulations voulues par le cahier des charges, c'est-à-dire sans l'utilisation des interrupteurs. Sur les schémas fonctionnels et dans les codes VHDL, les parties en fonction de *switch* ne sont donc pas à prendre en compte. Nous avons choisi de simuler l'affichage d'un carré blanc de dimensions 50 × 50, dont le coin supérieur gauche correspond au point de coordonnées (10, 10).

2.1 Fonction de service 1

La fonction FS1 permet de générer les signaux de synchronisation *synchro_V* et *synchro_H*, nécessaires à l'afficheur VGA. Elle définit également les zones où on peut « écrire », ainsi que les compteurs de lignes et de pixels, utiles aux autres fonctions de service. Le schéma fonctionnel de FS1 est le suivant :

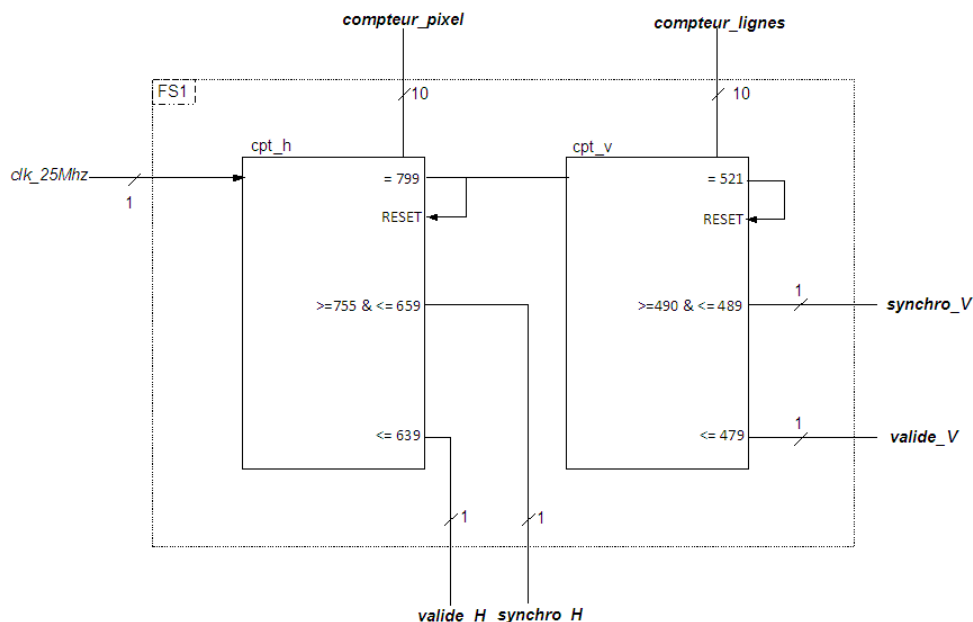


Figure 2 - Schéma fonctionnel de FS1

D'après la documentation de la carte Basys, ainsi que nos tests, nous souhaitons reproduire les signaux *SH* et *SV* tels que :

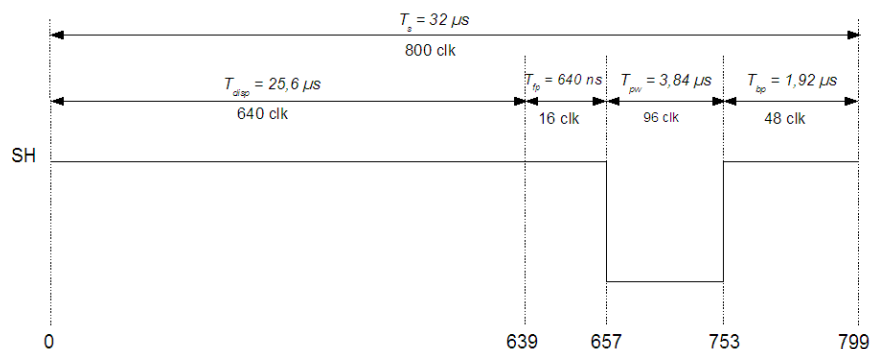


Figure 3 - Signal de synchronisation horizontale

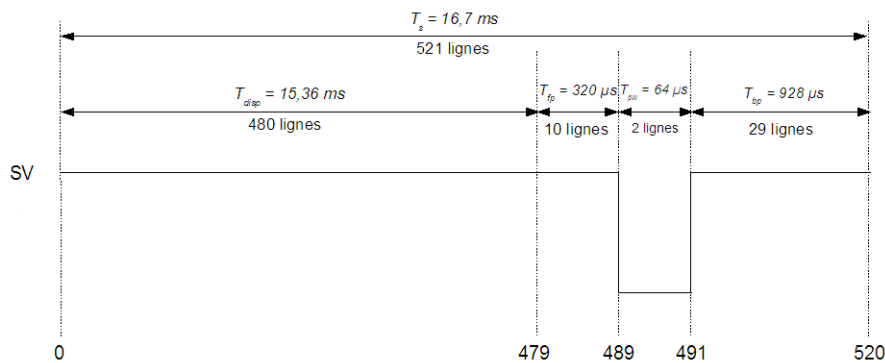


Figure 4 - Signal de synchronisation verticale

En effet, l'afficheur VGA utilise ces deux signaux de synchronisation pour afficher l'objet à l'écran. Comme nous utilisons un écran de résolution 640×480 , la trame horizontale est constituée de 521 lignes, et la trame verticale de 800 pixels. On peut alors représenter l'écran VGA comme ci-dessous :

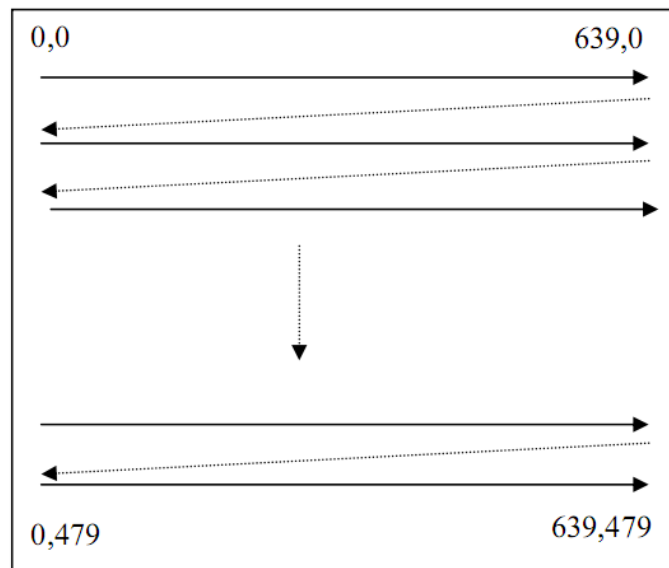


Figure 5 - Schéma d'une image 640 x 480

Le curseur parcourt l'écran de gauche à droite et de haut en bas. Les traits en pointillés représentent le temps que le curseur met pour revenir à la ligne. Comme l'écran commence au point de coordonnées (0,0), le dernier point est le point de coordonnées (639,479).

La zone d'« écriture » correspond à T_{disp} sur les figures 4 et 5 (640 coups d'horloge pour la synchronisation horizontale et 480 lignes pour la synchronisation verticale). Viennent ensuite un « front porch » de durée T_{fp} pendant lequel on n'écrit pas, un temps de synchronisation T_{pw} , et enfin un « back porch » de durée T_{bp} , qui, additionnés, correspondent au temps que met le curseur pour passer à la ligne ou au pixel suivant.

Ainsi, avec des variables temporaires, il est possible de synthétiser cette fonction en VHDL, comme suit :

```

entity FS1 is
  Port ( CLK : in  STD_LOGIC;
        SH, SV : out STD_LOGIC;
        CL : out  STD_LOGIC_VECTOR (9 downto 0);
        CP : out  STD_LOGIC_VECTOR (9 downto 0);
        VV, VH : out STD_LOGIC
        );
end FS1;

architecture Behavioral of FS1 is

  SIGNAL h_count, v_count :STD_LOGIC_VECTOR(9 DOWNT0 0):=(others=>'0');
  SIGNAL hori_sync, vert_sync : STD_LOGIC;

begin
  process (CLK)
    begin
      --A chaque coup d'horloge
      if (CLK'event) and (CLK='1') then
        --Reset du compteur horizontal si fin de ligne
        if (h_count = 799) then h_count <= (others => '0');
        --Sinon, incrémenter le compteur horizontal
        else h_count <= h_count + 1;
        end if;
        --Création du signal de synchro horizontale
        if (h_count <= 753) and (h_count >= 657) then hori_sync <= '0';
        else hori_sync <= '1';
        end if;

        --Reset du compteur vertical si fin d'écran
        if (v_count = 524) and (h_count = 799) then v_count <= (others=>'0');
        --Sinon, incrémenter le compteur vertical
        elsif (h_count = 799) then v_count <= v_count + 1;
        end if;
        --Création du signal de synchro verticale
        if (v_count <= 491) and (v_count >= 490) then vert_sync <= '0';
        else vert_sync <= '1';
        end if;

        --Si on est dans la zone d'écriture horizontale
        if (h_count <= 639) then
          --valide_H est à 1, et on met à jour le compteur de pixels
          --(pour qu'il ne compte qu'entre 0 et 639)
          VH <= '1'; CP <= h_count;
        else
          --sinon, valide_H est à 0
          VH <= '0';
        end if;

        --Si on est dans la zone d'écriture verticale
        if (v_count <= 479) then
          --valide_V est à 1, et on met à jour le compteur de lignes
          --(pour qu'il ne compte qu'entre 0 et 479)
          VV <= '1'; CL <= v_count;
        else
          --sinon, valide_V est à 0
          VV <= '0';
        end if;

        --Mise à jour des signaux de synchro
        SV <= vert_sync;
        SH <= hori_sync;
      end if;
    end process;
end Behavioral;

```

Après simulation, nous obtenons les résultats suivants :

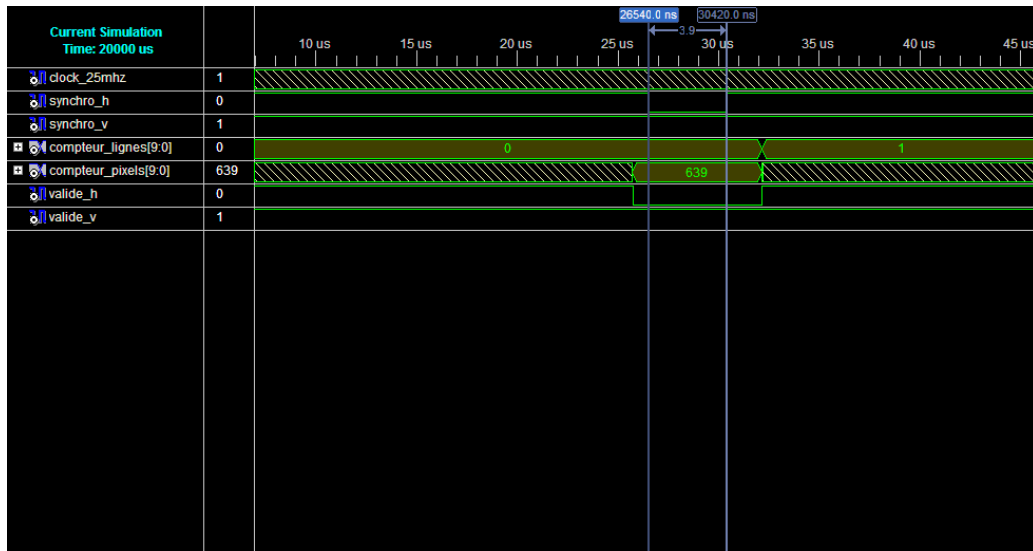


Figure 6 - Chronogramme de Synchro_H

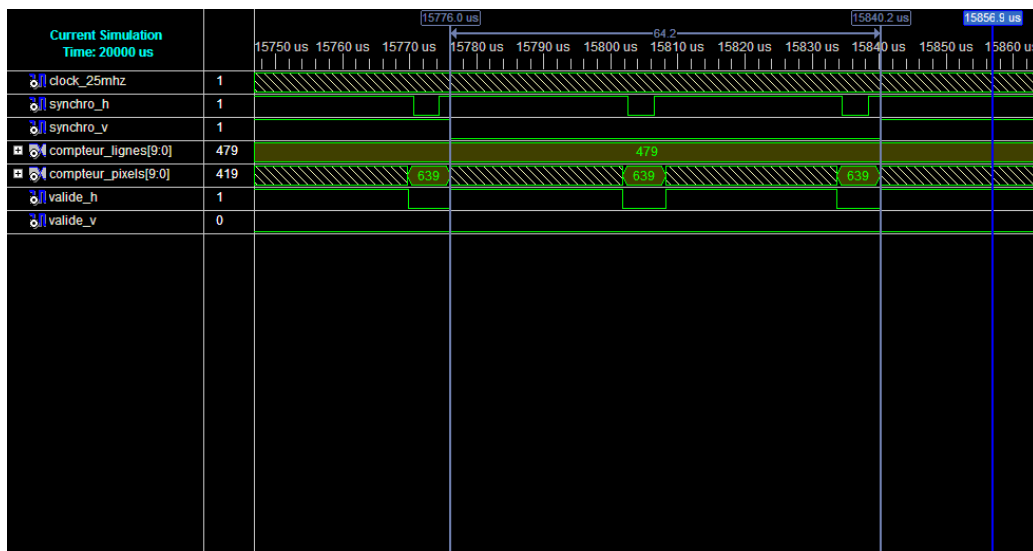


Figure 7 - Chronogramme de Synchro_V

Sur ces deux figures, on doit bien apparaître les signaux de synchronisation que l'on souhaitait obtenir. En effet, d'après la documentation, le temps de synchronisation (le « creux ») pour *SH* devrait durer $T_{pw} = 3,84 \mu s$. Par simulation, on trouve environ $T = 3,9 \mu s$.

De même, pour la synchronisation verticale, on trouve expérimentalement une durée de $T = 64,2 \mu s$, contre une durée théorique égale à $T_{pw} = 64 \mu s$.

Les signaux *synchro_V* et *synchro_H* ont donc bien été générés.

On peut aussi remarquer que la synchronisation horizontale se fait bien en fin de ligne ($CP = 639$), et que la synchronisation verticale se fait en fin de l'écran ($CP = 639$ et $CL = 479$).

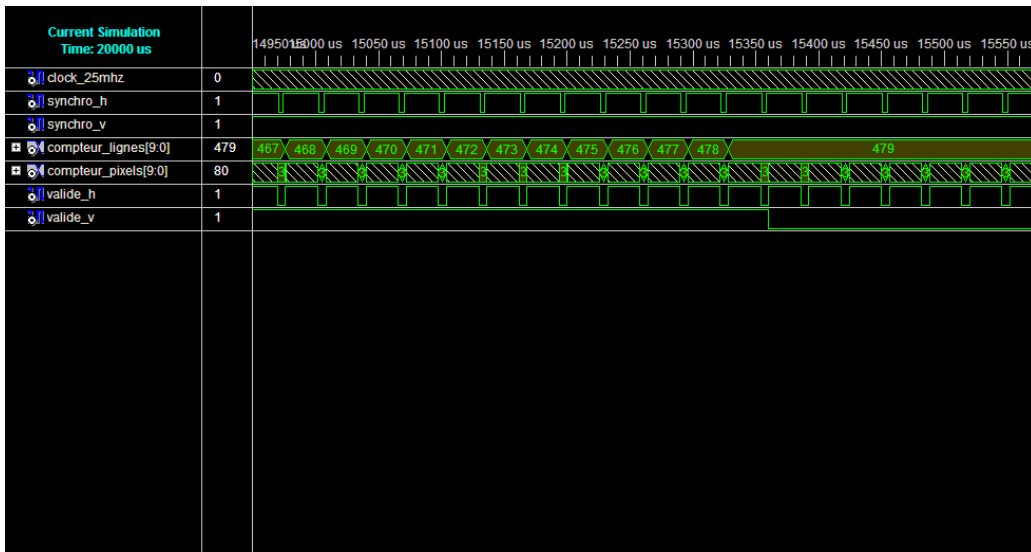


Figure 8 - Chronogramme de Valide_V

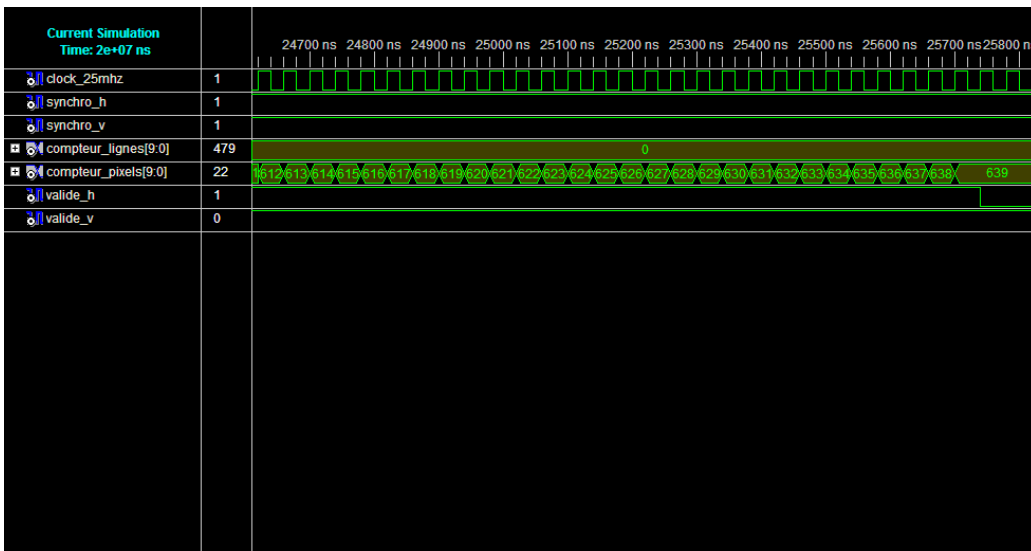


Figure 9 - Chronogramme de Valide_H

Sur ces deux chronogrammes, on voit bien que les signaux *valide_V* et *valide_H* sont bien à un niveau logique haut lorsque les compteurs de lignes et de pixels s'incrémentent. Lorsque l'on a atteint les deux limites ($CP = 639$ et $CL = 479$), on n'est plus dans la zone d'écriture, donc les deux signaux *valide_V* et *valide_H* passent à un niveau logique bas.

On remarque aussi que les compteurs s'arrêtent bien de compter lorsqu'ils ont atteint leur limite respective, ce qui permet de ne compter que la zone d'écriture.

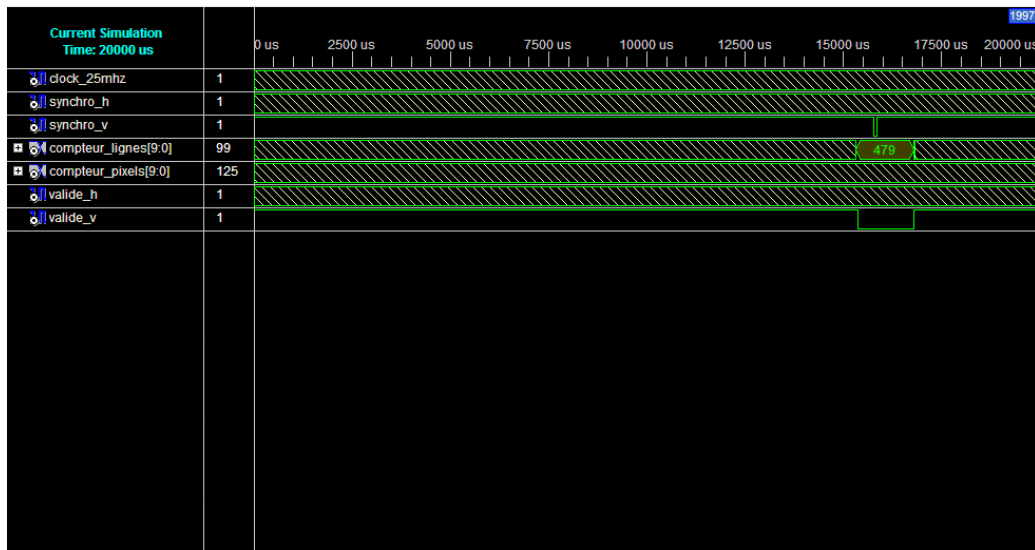


Figure 10 - Vue d'ensemble des chronogrammes de FS1

La vue d'ensemble nous permet de confirmer que *valide_V* n'est actif que lorsque l'on est à l'intérieur de la zone d'écriture.

2.2 Fonction de service 2

Le but de FS2 est de définir l'objet à afficher. Pour tester la fonction, nous avons simulé l'affichage d'un carré de dimensions 50 × 50, dont l'origine est située au point (10 ; 10). Comme FS2 a besoin de l'horloge pour fonctionner, il s'agit d'une fonction séquentielle. Voici son schéma fonctionnel :

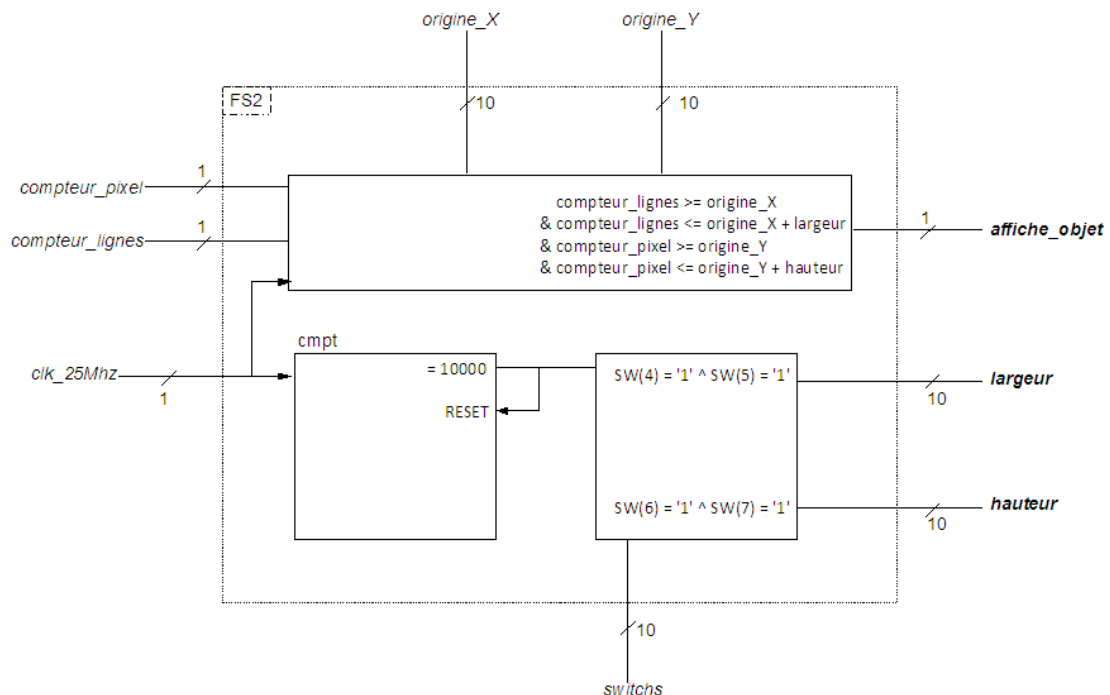


Figure 11 - Schéma fonctionnel de FS2

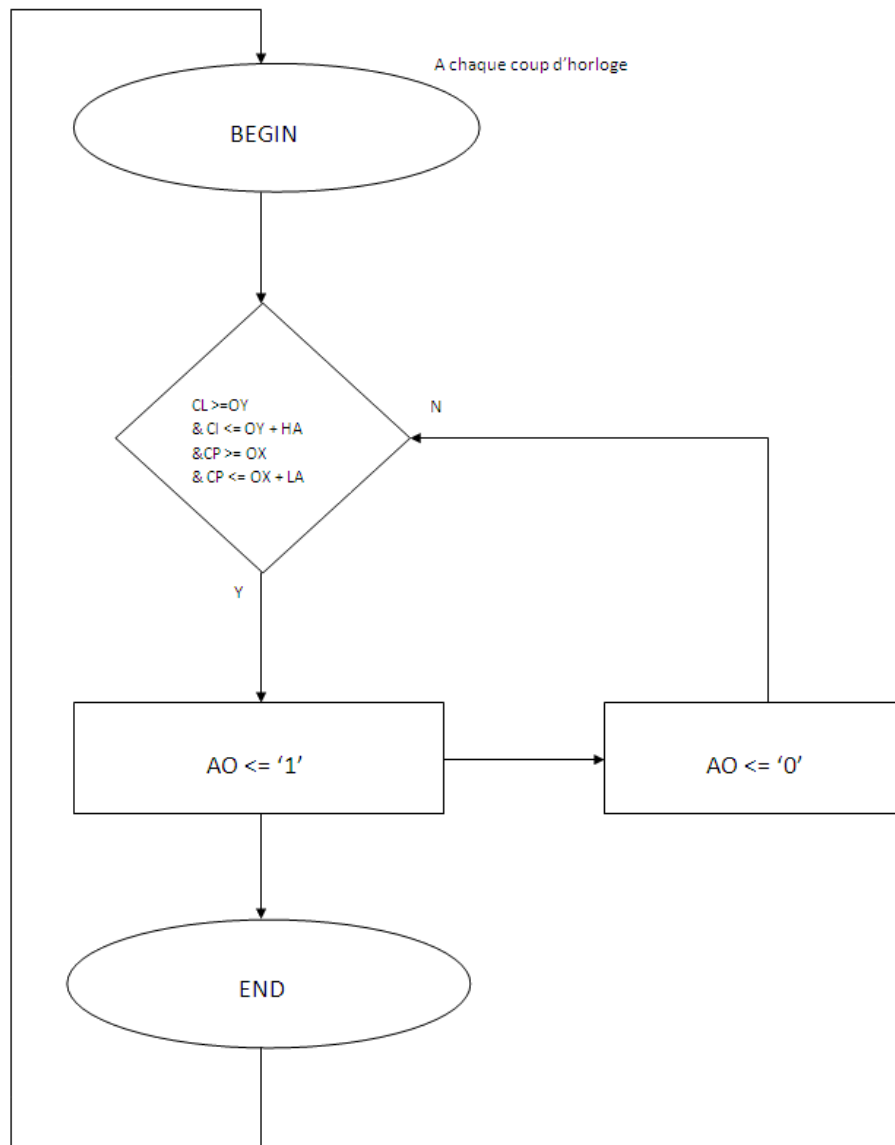


Figure 12 - Diagramme de FS2

FS2 fonctionne à chaque coup d'horloge. Ensuite, après détection de la zone à dessiner (ici, il s'agit donc d'un carré de côté $HA = LA = 50$ pixels, et d'origine $OX = OY = 10$ pixels). Ainsi, si les compteurs de lignes et les compteurs de pixels se situent dans la zone décrite par l'objet, la variable *affiche_objet* passe à un niveau logique haut. Sinon, elle reste à un niveau logique bas. Ainsi, *affiche_objet* définit l'objet que l'on souhaite avoir sur l'écran.

Le code VHDL correspondant à FS2 est le suivant :

```
entity FS2 is
Port ( CLK : in STD_LOGIC;
      CL : in STD_LOGIC_VECTOR (9 downto 0);
      CP : in STD_LOGIC_VECTOR (9 downto 0);
      OX : in STD_LOGIC_VECTOR (9 downto 0);
      OY : in STD_LOGIC_VECTOR (9 downto 0);
      AO : out STD_LOGIC;
      SW : in STD_LOGIC_VECTOR (7 downto 0);
      LA, HA : out STD_LOGIC_VECTOR (9 downto 0)
);

end FS2;

architecture Behavioral of FS2 is

SIGNAL hauteur : STD_LOGIC_VECTOR(9 downto 0):="0010010110";
SIGNAL largeur : STD_LOGIC_VECTOR(9 downto 0):="0010010110";
SIGNAL cmpt : STD_LOGIC_VECTOR(25 downto 0):=(others=>'0');

begin
  process(CLK, hauteur, largeur)
  begin
    --A chaque coup d'horloge
    if(CLK'EVENT) and (CLK='1') then
      cmpt <= cmpt + 1;
      --Si le compteur de lignes et le compteur de pixels sont dans la zone à écrire
      if(CL >= OY)
        and (CL <= OY + hauteur)
        and (CP >= OX)
        and (CP <= OX + largeur) then
        --affiche_objet passe à 1
        AO <= '1';
      else
        --sinon, il est à 0
        AO <= '0';
      end if;
      --Détection des interrupteurs pour modifier la taille
      if(cmpt = 100000) then
        cmpt <= (others=>'0');
        if(SW(4) = '1') and (largeur < 150) then largeur <= largeur + 1;
        elsif(SW(5) = '1') and (largeur > 20) then largeur <= largeur - 1;
        end if;
        if(SW(6) = '1') and (hauteur < 150) then hauteur <= hauteur + 1;
        elsif(SW(7) = '1') and (hauteur > 20) then hauteur <= hauteur - 1;
        end if;
      end if;
    end if;
    --Mise à jour de la hauteur et de la largeur
    HA <= hauteur;
    LA <= largeur;
  end process;
end;
```

Après simulation, on obtient les courbes suivantes :

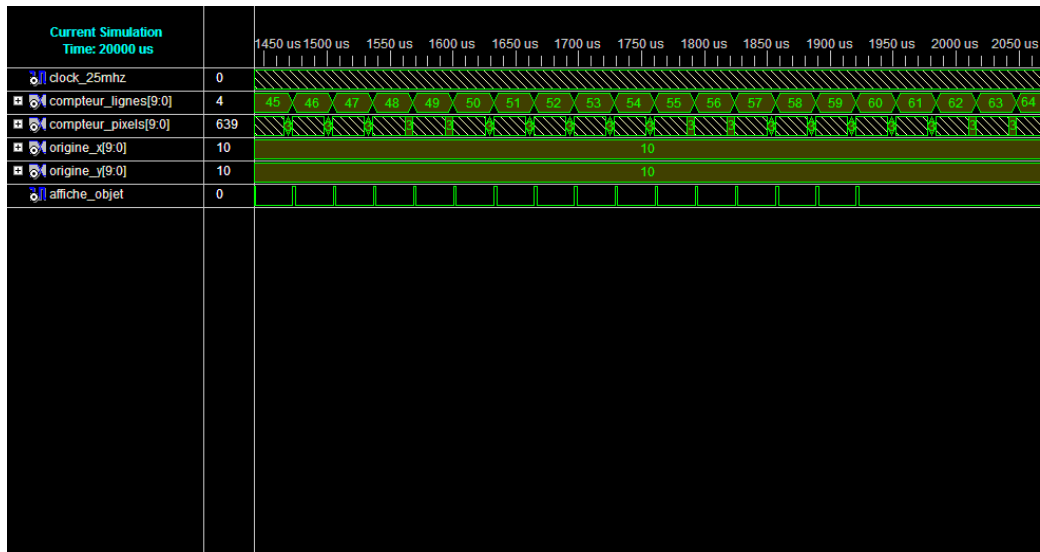


Figure 13 - Chronogrammes de FS2

Sur la figure ci-dessus, on peut voir l'affichage de l'objet. Chaque « pic » de *affiche_objet* correspond à l'affichage d'une ligne de longueur 50 pixels. On voit bien que l'affichage a lieu au début de chaque ligne, car les coordonnées à l'origine sont ($X = 10 ; Y = 10$).

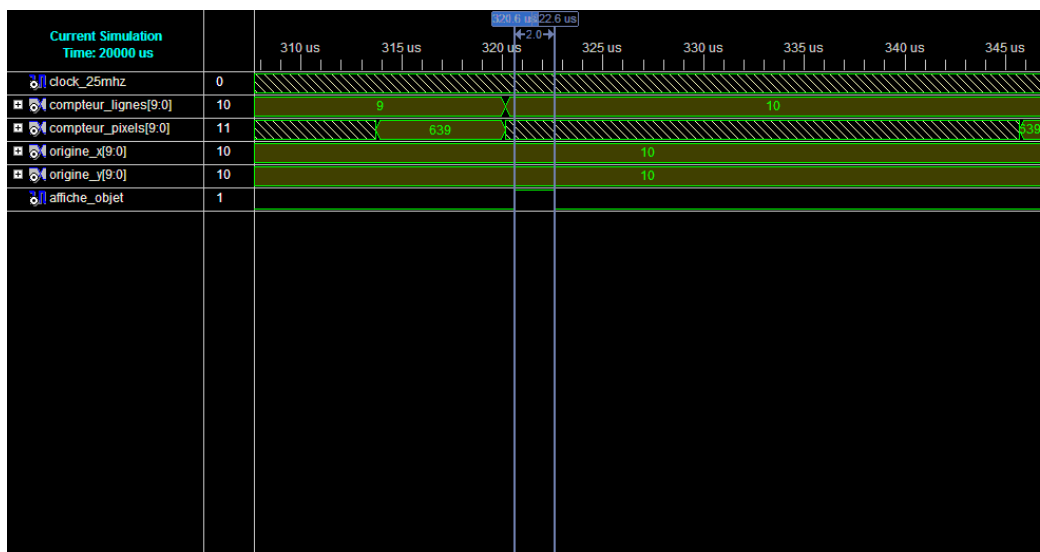


Figure 14 - Chronogrammes de FS2 (agrandi)

Nous voyons ici plus en détail l'affichage d'une seule ligne, qui commence au pixel 10 de la ligne 10.

2.3 Fonction de service 3

La fonction FS3 a pour but de générer les signaux RGB (Red Green Blue) qui permettront l'affichage des couleurs de notre objet. Sur un afficheur VGA, le rouge et le vert sont codés sur trois bits, le bleu sur deux. Pour le test de cette fonction, nous avons décidé d'afficher un carré blanc (le même que pour FS2). FS3 est aussi une fonction séquentielle, dans la mesure où il faut également une horloge pour pouvoir générer les signaux de couleur. Voici son schéma :

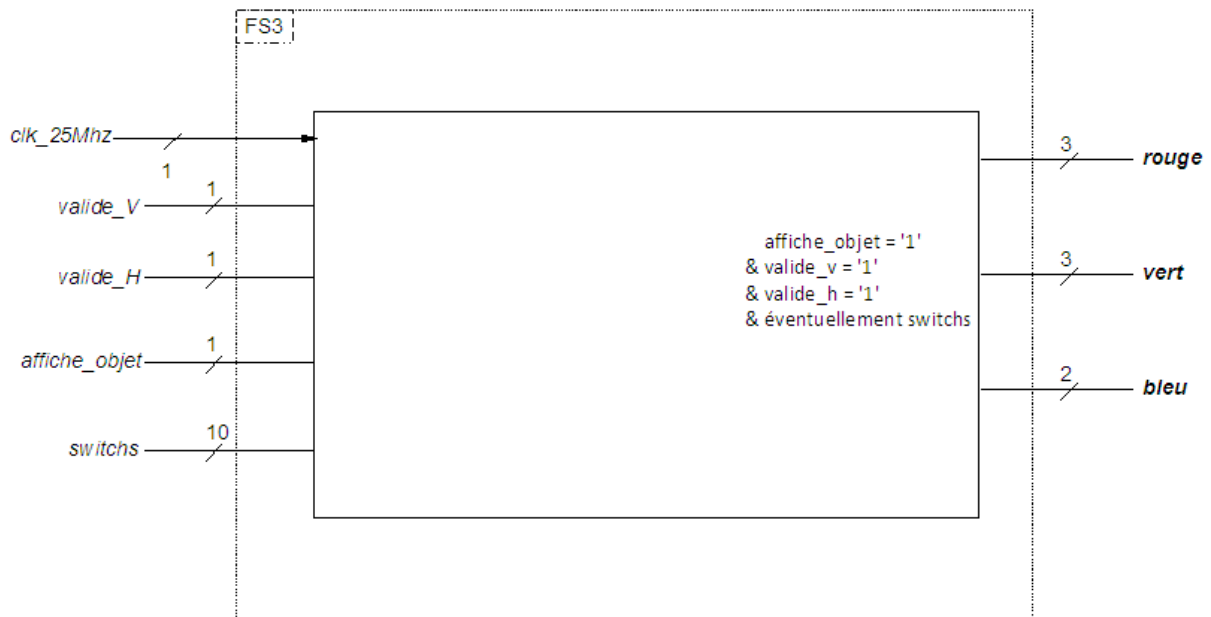


Figure 15 - Schéma fonctionnel de FS3

La table de vérité de FS3 est la suivante :

valide_v	valide_h	affiche_objet	clk_25Mhz	rouge	vert	bleu
0	0	0	X	000	000	00
0	0	1	X	000	000	00
0	1	0	X	000	000	00
0	1	1	X	000	000	00
1	0	0	X	000	000	00
1	0	1	X	000	000	00
1	1	0	X	000	000	00
1	1	1	↗	111	111	11

Figure 16 - Schéma fonctionnel de FS3

N.B. : X désigne ici un état indéterminé, qui n'a pas d'utilité pour le résultat en sortie.

Pour que les signaux de couleur soient actifs, il faut que l'on soit dans la zone d'écriture, soit lorsque *valide_v* et *valide_h* sont actifs. De plus, *affiche_objet* doit être sur un niveau haut pour que l'on affiche autre chose que du noir. Il faut donc que ces trois variables soient actives afin d'afficher du blanc.

Voici le code VHDL de la fonction FS3 :

```
entity FS3 is
Port ( CLK: in STD_LOGIC;
      R, V : out STD_LOGIC_VECTOR (2 downto 0);
      B : out STD_LOGIC_VECTOR (1 downto 0);
      AO : in STD_LOGIC;
      VV, VH : in STD_LOGIC;
      SW : in STD_LOGIC_VECTOR (7 downto 0)
);
end FS3;

architecture Behavioral of FS3 is

begin
  process(CLK)
  begin
    --A chaque coup d'horloge
    if (CLK'event) and (CLK='1') then
      --Si affiche_objet, valide_v et valide_h sont à 1
      if(AO = '1') and (VV = '1') and (VH = '1') then
        --Couleur blanche
        R <= "111";
        V <= "111";
        B <= "11";
        --Détection des switches
        if (SW(0) = '1') then
          R <= "101";
          V <= "010";
          B <= "10";
        end if;
        if (SW(1) = '1') then
          R <= "000";
          V <= "100";
          B <= "00";
        end if;
        if (SW(2) = '1') then
          R <= "000";
          V <= "110";
          B <= "10";
        end if;
        if (SW(3) = '1') then
          R <= "111";
          V <= "100";
          B <= "00";
        end if;
        --Sinon, couleur noire
        else R <= "000"; V <= "000"; B <= "00";
      end if;
    end if;
  end process;
end Behavioral;
```

Après simulation, nous obtenons les courbes suivantes :

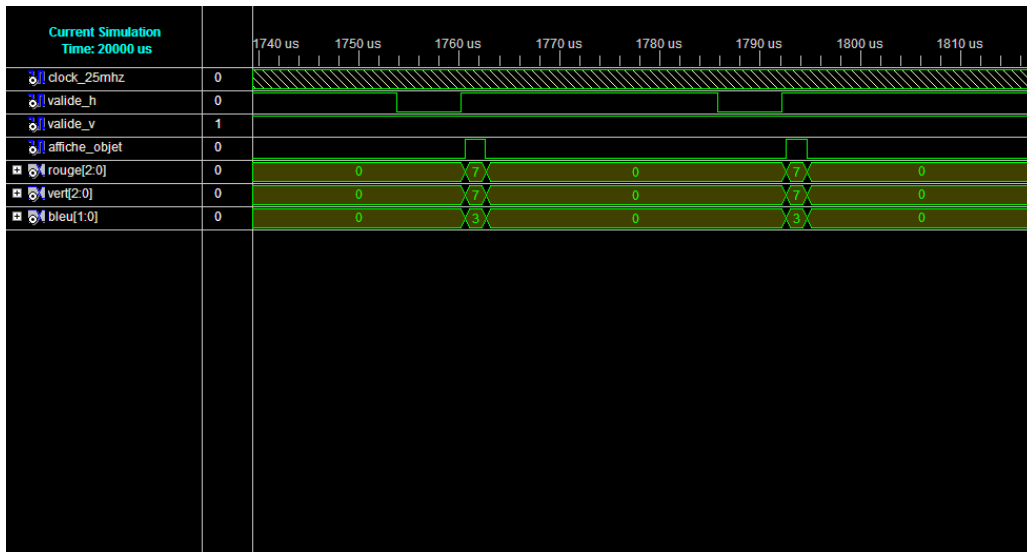


Figure 17 - Chronogrammes de FS3

Nous voyons bien ici que lorsque les trois variables *valide_h*, *valide_v* et *affiche_objet* sont à '1', nous obtenons une couleur blanche.

2.4 Fonction de service 4

La fonction FS4 permet de déplacer horizontalement et verticalement notre carré à l'aide des boutons poussoirs de la carte Basys. Cette fonction est également séquentielle, car elle nécessite elle aussi une horloge. Voici son schéma fonctionnel :

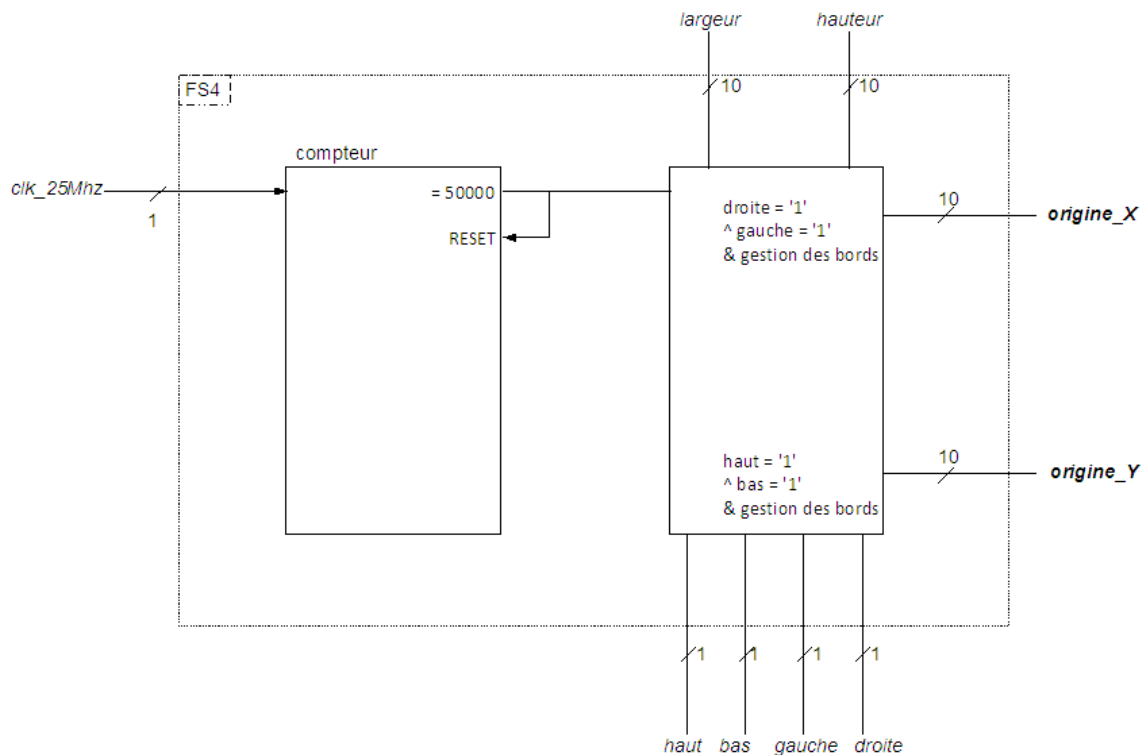


Figure 18 - Schéma fonctionnel de FS4

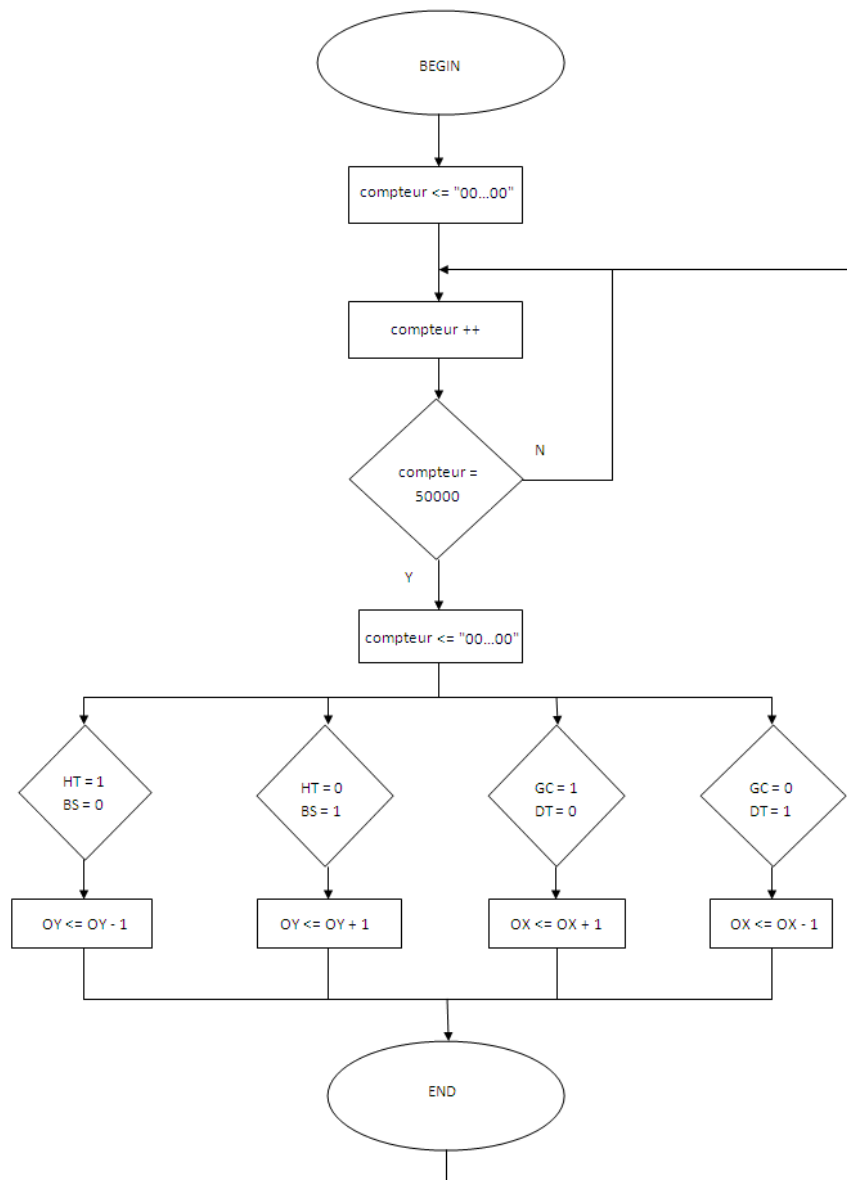


Figure 19 - Diagramme de FS4

Deux boutons poussoirs sont assignés à chaque direction. Lorsque l'un d'entre eux est enfoncé, la coordonnée correspondante à celui-ci est modifiée. Nous obtenons en sortie *origine_X* et *origine_Y*, les coordonnées de l'objet qui sont transmises à FS2 qui va redéfinir l'objet à sa nouvelle position. Dans notre projet, nous avons décidé de ralentir cette fonction (le phénomène étant trop rapide et faisant passer l'objet d'un bout à l'autre de l'écran trop rapidement) à l'aide d'un « ralentisseur ». Le ralentisseur est en fait un compteur qui tous les 50000 coups d'horloge, prend en compte les appuis sur les boutons et se remet à 0.

Voici le code VHDL de la fonction FS4 :

```

entity FS4 is
Port ( CLK: in STD_LOGIC;
      OX : out STD_LOGIC_VECTOR (9 downto 0);
      OY : out STD_LOGIC_VECTOR (9 downto 0);
      HT, BS, GC, DT : in STD_LOGIC;
      LA, HA : in STD_LOGIC_VECTOR (9 downto 0)
      );
end FS4;

architecture Behavioral of FS4 is

SIGNAL compteur : STD_LOGIC_VECTOR(25 downto 0):=(others=>'0');
SIGNAL hauteur : STD_LOGIC_VECTOR(9 downto 0):=(others=>'0');
SIGNAL largeur : STD_LOGIC_VECTOR(9 downto 0):=(others=>'0');
SIGNAL ox_tmp : STD_LOGIC_VECTOR(9 downto 0):=(others=>'0');
SIGNAL oy_tmp : STD_LOGIC_VECTOR(9 downto 0):=(others=>'0');

begin
  process(CLK, LA, HA)
  begin
    hauteur <= HA;
    largeur <= LA;
    --A chaque coup d'horloge
    if (CLK'event) and (CLK='1') then
      --incrémenter le compteur
      compteur <= compteur +1;
      --Une fois la valeur 50000 atteinte, remettre le compteur à 0
      if (compteur = 50000) then compteur <= (others => '0');
        --Gestion des bords et des déplacements (boutons)
        if(HT = '1') and (BS = '0') and (oy_tmp > 1) then
          oy_tmp <= oy_tmp - 1;
        end if;
        if(BS = '1') and (HT = '0') and(oy_tmp + hauteur < 479) then
          oy_tmp <= oy_tmp + 1;
        end if;
        if(GC = '1') and (DT = '0') and (ox_tmp > 1) then
          ox_tmp <= ox_tmp - 1;
        end if;
        if(DT = '1') and (GC = '0') and (ox_tmp + largeur < 639) then
          ox_tmp <= ox_tmp + 1;
        end if;
      end if;
      --Mise à jour des coordonnées
      OX <= ox_tmp;
      OY <= oy_tmp;
    end if;
  end process;
end Behavioral;

```

Voici les courbes obtenues par simulation :

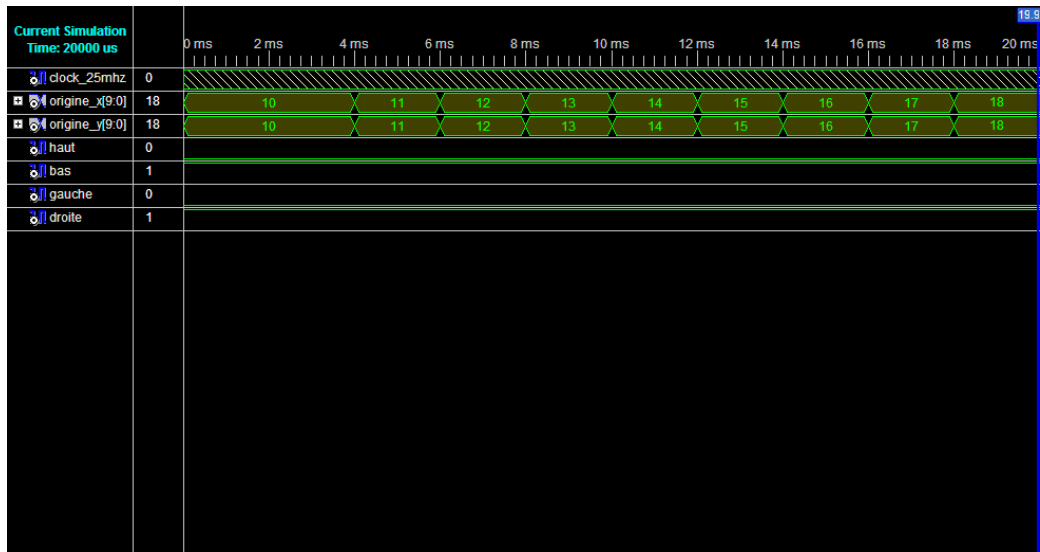


Figure 20 - Chronogrammes de FS4

Nous voyons bien ici que lorsque l'on met les variables *bas* et *droite* à un niveau logique haut, les coordonnées de l'origine s'incrémentent.

3. Etude de la fonction complète

3.1 Simulation de la fonction complète sans les améliorations

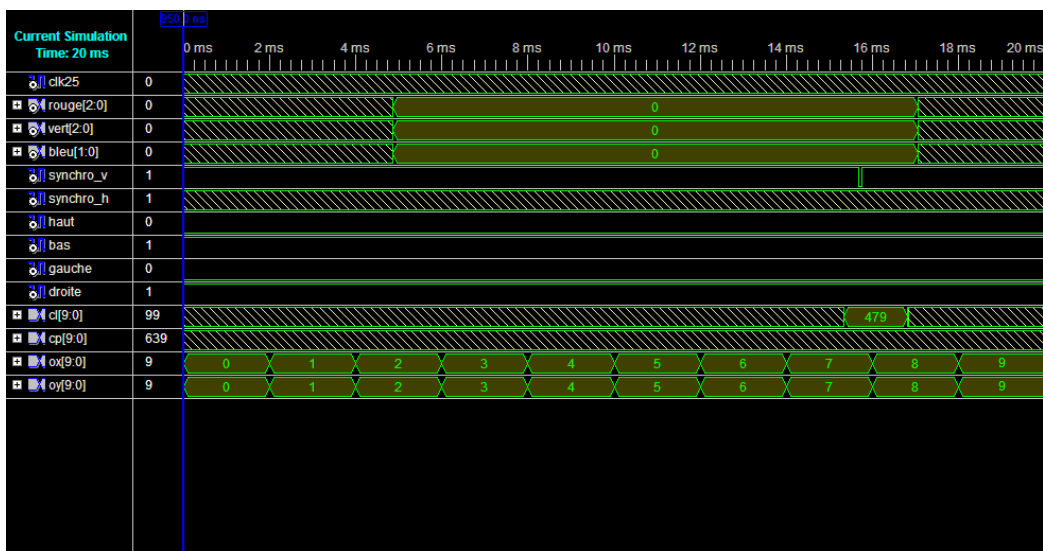


Figure 21 - Chronogrammes de FP

On peut voir sur cette figure que la fonction principale résulte bien des quatre fonctions secondaires.

3.2 Etude de la fonction complète avec les améliorations

3.2.1 Simulation

Les deux fonctionnalités principales que nous avons ajoutées sont respectivement dans FS3 et dans FS2. Il s'agit d'un changement de couleur lorsque l'on active ou on désactive les interrupteurs 0 à 3 (la couleur par défaut restant le blanc), ainsi qu'un changement de taille lorsque l'on active ou on désactive les interrupteurs 4 à 7. Ils permettent d'agrandir et de rétrécir le rectangle, et de figer sa taille. Nous nous sommes servis d'un compteur, car sinon, le phénomène était trop rapide pour être vu correctement.

Les interrupteurs ont été définis comme suit (pour plus de précisions, voir le code dans la partie précédente) :

numéro	valeur de <i>switch</i>	variable(s) modifiée(s)	valeur(s) prise(s)
0	$(00000001)_2 = (1)_{10}$	rouge, vert, bleu	$(101 010 10)_2 = (5 2 2)_{10}$
1	$(00000010)_2 = (2)_{10}$	rouge, vert, bleu	$(000 100 00)_2 = (0 4 0)_{10}$
2	$(00000100)_2 = (4)_{10}$	rouge, vert, bleu	$(000 110 10)_2 = (0 6 2)_{10}$
3	$(00001000)_2 = (8)_{10}$	rouge, vert, bleu	$(111 100 00)_2 = (7 4 0)_{10}$
4	$(00010000)_2 = (16)_{10}$	largeur	largeur \leq largeur + 1
5	$(00100000)_2 = (32)_{10}$	largeur	largeur \leq largeur - 1
6	$(01000000)_2 = (64)_{10}$	hauteur	hauteur \leq hauteur + 1
7	$(10000000)_2 = (128)_{10}$	hauteur	hauteur \leq hauteur - 1

Figure 22 - Chronogrammes de FS4

N.B. :

- La valeur de *switch* inscrite dans le tableau pour un interrupteur sous-entend que seul ce dernier est actif parmi les sept.
- Les couleurs ont été choisies arbitrairement.

Pour relier les quatre fichiers .vhd, nous sommes passés par un fichier « général » : **main.vhd**. Chaque fonction secondaire était représentée par un composant (**component**). Il suffit alors de relier les ports des composants aux signaux d'entrée, de sortie ou aux signaux temporaires : c'est l'instanciation. Ci-dessous, le code VHDL du « main » :

```

entity main is
  Port (
    clk25 : in STD_LOGIC;
    rouge, vert : out STD_LOGIC_VECTOR(2 downto 0);
    bleu : out STD_LOGIC_VECTOR (1 downto 0);
    synchro_V, synchro_H : out STD_LOGIC;
    haut, bas, gauche, droite : in STD_LOGIC;
    switch : in STD_LOGIC_VECTOR(7 downto 0)
  );
end main;

architecture Behavioral of main is
--déclaration
component FS1 is
  Port ( CLK: in  STD_LOGIC;
    SH, SV : out STD_LOGIC;
    CL : out  STD_LOGIC_VECTOR (9 downto 0);
    CP : out  STD_LOGIC_VECTOR (9 downto 0);
    VV, VH : out STD_LOGIC
  );
end component;

component FS2 is
Port ( CLK : in STD_LOGIC;
  CL : in STD_LOGIC_VECTOR (9 downto 0);
  CP : in STD_LOGIC_VECTOR (9 downto 0);
  OX : in STD_LOGIC_VECTOR (9 downto 0);
  OY : in STD_LOGIC_VECTOR (9 downto 0);
  AO : out STD_LOGIC;
  SW : in STD_LOGIC_VECTOR (7 downto 0);
  LA, HA : out STD_LOGIC_VECTOR (9 downto 0)
  );
end component;

component FS3 is
Port ( CLK: in STD_LOGIC;
  R, V : out STD_LOGIC_VECTOR (2 downto 0);
  B : out STD_LOGIC_VECTOR (1 downto 0);
  AO : in STD_LOGIC;
  VV, VH : in STD_LOGIC;
  SW : in STD_LOGIC_VECTOR (7 downto 0)
  );
end component;

component FS4 is
Port ( CLK: in STD_LOGIC;
  OX : out STD_LOGIC_VECTOR (9 downto 0);
  OY : out STD_LOGIC_VECTOR (9 downto 0);
  HT, BS, GC, DT : in STD_LOGIC;
  LA, HA : in STD_LOGIC_VECTOR (9 downto 0)
  );
end component;

[...]
```

```
[...]

SIGNAL origine_x, origine_y : STD_LOGIC_VECTOR (9 downto 0):=(others=>'0');
SIGNAL compteur_lignes, compteur_pixels : STD_LOGIC_VECTOR (9 downto 0):=(others=>'0');
SIGNAL valide_v, valide_h : STD_LOGIC:='0';
SIGNAL affiche_objet : STD_LOGIC;
SIGNAL hauteur : STD_LOGIC_VECTOR(9 downto 0):="0010010110";
SIGNAL largeur : STD_LOGIC_VECTOR(9 downto 0):="0010010110";

begin
--instanciation
synchro : FS1
port map(
    clk25,--in
    synchro_H,--out
    synchro_V,--out
    compteur_lignes,--out
    compteur_pixels,--out
    valide_v,--out
    valide_h--out
);

objet : FS2
port map(
    clk25,--in
    compteur_lignes,--in
    compteur_pixels,--in
    origine_x,--in
    origine_y,--in
    affiche_objet,--out
    switch,--in
    largeur,--out
    hauteur--out
);

couleur : FS3
port map(
    clk25,--in
    rouge,--out
    vert,--out
    bleu,--out
    affiche_objet,--in
    valide_v,--in
    valide_h,--in
    switch
);

position : FS4
port map(
    clk25,--in
    origine_x,--out
    origine_y,--out
    haut,--in
    bas,--in
    gauche,--in
    droite,--in
    largeur, hauteur--in
);

end Behavioral;
```

Nous avons simulé cette fonction principale, ce qui nous a donné les chronogrammes suivants :

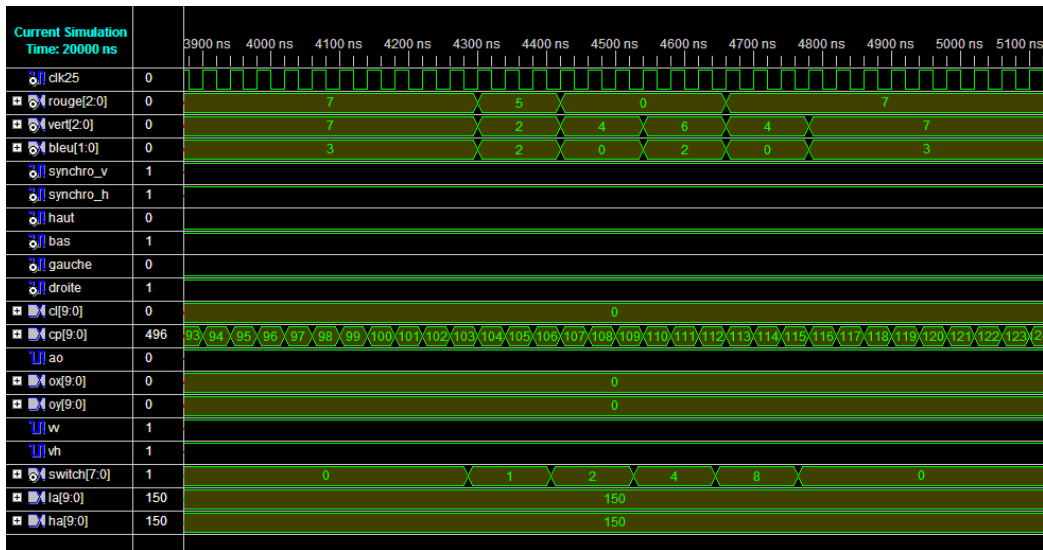


Figure 23 - Chronogrammes de FP (agrandi)

Sur cette figure, on observe les changements de couleur, avec les quatre interrupteurs. En effet, nous avons simulé l'appui successif sur chacun des interrupteurs (en relâchant à chaque fois le précédent). On voit bien que *switch* prend les valeurs 1, 2, 4, puis 8. Les couleurs changent ainsi en fonction (voir tableau au-dessus pour les valeurs).

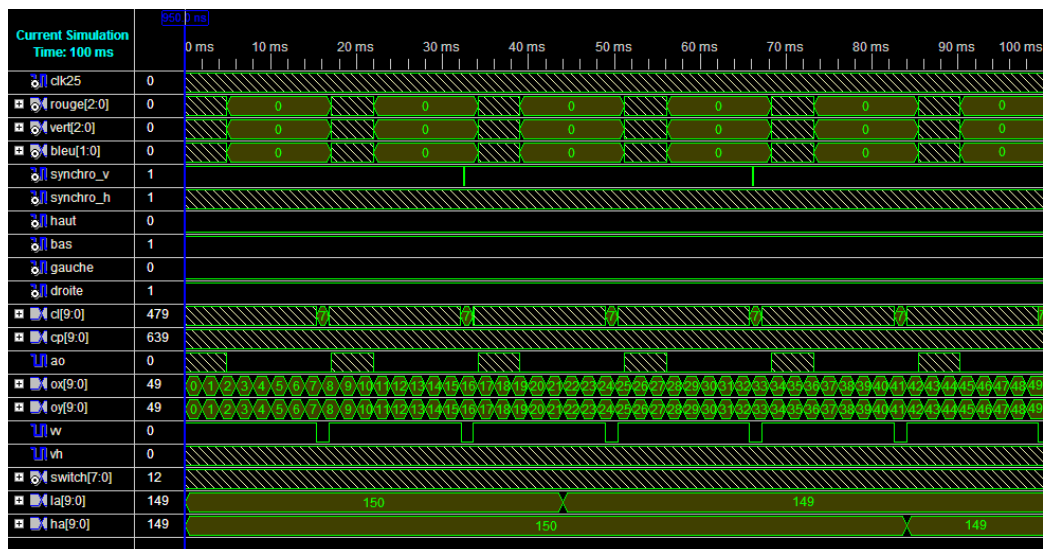


Figure 24 - Chronogrammes de FP

On observe ici la simulation de l'appui sur les interrupteurs qui permettent de réduire en largeur et en hauteur. On voit bien que les deux grandeurs diminuent.

3.2.2. Implémentation sur une carte Basys

Pour pouvoir observer la fonction principale sur un écran, nous l'avons implémentée sur une carte Basys. Pour cela, nous avons d'abord créé un fichier **.ucf**, avec l'affectation des variables sur broches de la carte.

Voici l'affectation des pins sur la carte :

variable	type	pin
clk25	entrée	p54
synchro_H	sortie	p39
synchro_V	sortie	p35
rouge<0>	sortie	p70*
rouge<1>	sortie	p68
rouge<2>	sortie	p67
vert<0>	sortie	p52
vert<1>	sortie	p51
vert<2>	sortie	p50
bleu<0>	sortie	p44
bleu<1>	sortie	p43
haut	entrée	p48
bas	entrée	p69
gauche	entrée	p41
droite	entrée	p47
switch<0>	entrée	p38
switch<1>	entrée	p36
switch<2>	entrée	p29
switch<3>	entrée	p24
switch<4>	entrée	p18
switch<5>	entrée	p12
switch<6>	entrée	p10
switch<7>	entrée	p6

*erreur sur la documentation, qui indique la broche 78 au lieu de 70

Figure 25 – Affectation des pins

Après génération du fichier **.bit**, on observe bien un rectangle dont la couleur et la taille change en fonction des boutons enclenchés, ainsi qu'un déplacement horizontal et vertical.

4. Conclusion

Ce projet nous a permis de voir plus en détail les différentes notions abordées en cours, et de manipuler des éléments non traités en séance de TP, comme les *components* ou encore l'affichage VGA. Comme nous avons acheté une carte Basys, il était intéressant de pouvoir tester notre programme directement sur un écran, ce qui est beaucoup plus « parlant » que les courbes de simulation.