Contrôleur VGA

Un contrôleur VGA[1] est un composant qui contrôle 5 signaux de base pour afficher de la vidéo :

- un signal de synchronisation horizontale HS
- un signal de synchronisation verticale VS
- les 3 signaux R (Rouge), G (Vert), B (Bleu)

L'énergie émise dans le spectre R (G, B) est proportionnelle à la tension appliquée sur la broche R (G, B) – analogique – en entrée de l'écran. Chaque point coloré sur l'écran est appelé pixel (picture element). Le procédé d'émission des couleurs dépend de la technologie (écran cathodique ou LCD). Dans chaque cas, l'écran affiche les pixels en partant du coin en haut à gauche en se déplaçant vers la droite sur la même ligne, ligne après ligne, jusqu'au bas de l'écran. Une impulsion de synchronisation horizontale synchronise chaque nouvelle ligne. A chaque fois que le bas de l'écran est atteint, une impulsion de synchronisation verticale est générée pour recommencer le processus en partant du coin en haut à gauche, et ainsi de suite.

La figure ci-dessous issue de la datasheet de la carte Nexys-3 montre les connexions entre les broches du FPGA Spartan 6 et la sortie VGA de la carte. Les signaux logiques R, G, B en sortie des broches du FPGA sont convertis en analogique à l'aide d'un circuit composé de plusieurs résistances (2 ou 3 suivant la couleur, 3 pour le rouge et le vert, 2 pour le bleu), permettant de générer une tension proportionnelle à la valeur numérique appliquée.

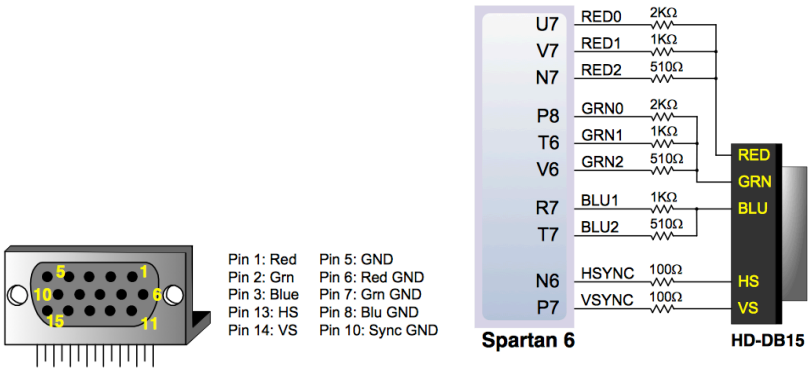


Figure 1. Le port VGA sur la carte Nexys-3

Timing des signaux VGA

Un moniteur standard fonctionne à 60Hz, ce qui signifie que l'écran est rafraichi 60 fois par seconde, c'est-à-dire chaque 1/60ème de seconde. Donc pour une résolution donnée (supportée par le moniteur), le timing de chaque pixel, de chaque ligne horizontale et le tracé vertical peut être calculé.

Nous allons dans cette série de TP concevoir un contrôleur pour la une basse résolution VGA : 640x480. Néanmoins les principes développés restent les mêmes pour les résolutions plus élevées. Nous utiliserons une horloge à 25MHz (notée pixel clock) pour notre contrôleur, ce qui est suffisant pour atteindre la résolution souhaitée.

[1] VGA: Video Graphics Array

Comme nous l'avons vu plus haut, en plus des signaux analogiques RGB, 2 signaux de synchronisation sont nécessaires. Le premier, HS, est la synchronisation horizontale : cette synchronisation repose sur 4 paramètres temporels :

- le *sync pulse* SP, qui est la durée de l'impulsion de synchronisation, et qui annonce le début d'une nouvelle ligne
- le *back porch* BP, qui correspond à une durée qui précède l'affichage de la ligne
- le *horizontal video* HV, qui correspond à l'affichage effectif des pixels de la ligne
- le *front porch* FP, délai après l'affichage de la ligne avant le prochain *sync pulse*

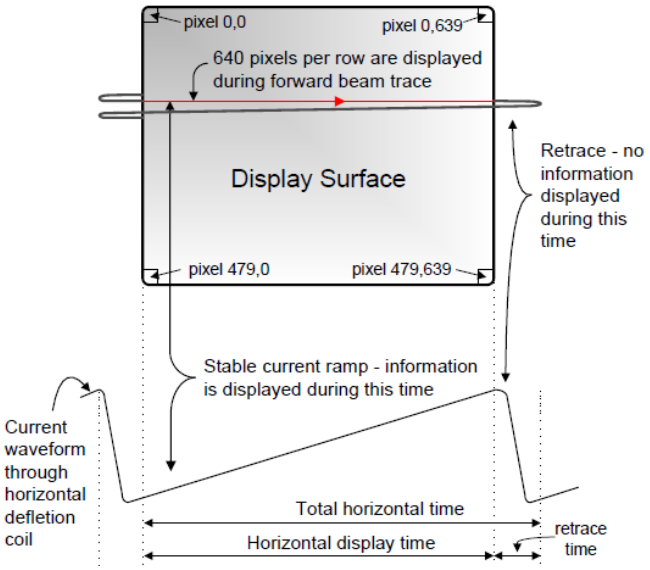


Figure 2. Balayage de l'écran et synchronisation horizontale

De la même manière, VS, la synchronisation verticale, est composée de 4 paramètres temporels :

- le *sync pulse* SP, qui est la durée de l'impulsion de synchronisation, et qui annonce le début d'une nouvelle image
- le *back porch* BP, qui correspond à une durée qui précède l'affichage
- le *vertical video* VV, qui correspond à l'affichage effectif
- le *front porch* FP, délai après l'affichage de l'image avant le prochain *sync pulse*

Calcul du timing horizontal

Pixel clock = 25 MHz → Pixel time = 0,04 µs
SP[2] = 96 pixels x 0,04 µs = 3,84 µs
BP = 48 pixels x 0,04 µs = 1,92 µs
HV = 640 pixels x 0,04 µs = 25,6 µs
FP = 16 pixels x 0,04 µs = 0,64 µs

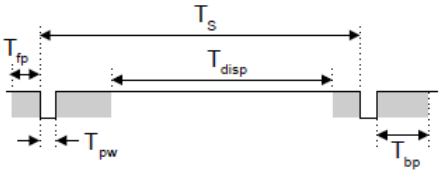
1 ligne horizontale = SP + BP + HV + FP = 128 + 16 + 640 + 16 = 800 x 0,04 µs = 32 µs
Avec un rafraichissement à 60Hz (16,67 ms) on a 16,67/0,032 = 521 lignes par image.

Calcul du timing vertical

[2] On peut aussi prendre SP = 128 pixels x 0,04 µs = 5,12 µs et BP = 16 pixels x 0,04 µs = 0,64 µs

Pixel clock = 25 MHz → durée de la ligne horizontale = 32 µs
SP = 2 lignes x 32 µs = 64 µs
BP = 29 lignes x 32 µs = 928 µs
VV = 480 lignes x 32 µs = 15,36 ms
FP = 10 lignes x 32 µs = 320 µs

Affichage vertical = SP + BP + VV + FP = 521 x 32µs = 16,67ms = 1/60Hz



Symbol	Parameter	Vertical Sync			Horiz. Sync	
		Time	Clocks	Lines	Time	Clks
T_S	Sync pulse	16.7ms	416,800	521	32 us	800
T_{disp}	Display time	15.36ms	384,000	480	25.6 us	640
T_{pw}	Pulse width	64 us	1,600	2	3.84 us	96
T_{fp}	Front porch	320 us	8,000	10	640 ns	16
T_{bp}	Back porch	928 us	23,200	29	1.92 us	48

Figure 2. Résumé des paramètres temporels de synchronisation

Pour les autres résolutions VGA, il existe des normes industrielles permettant de fixer les différents paramètres temporels. Par exemple pour une résolution de 1024x768 une pixel clock de 64 MHz serait nécessaire pour respecter les délais.

Conception d'un premier contrôleur VGA

Nous allons dans ce premier exercice faire simplement un contrôleur qui affiche des bandes de couleur en alternance. L'objectif principal est de mettre en œuvre un composant qui génère les différents signaux de synchronisation respectant les timings précédemment définis et de vérifier son fonctionnement en simulation, et sur la carte.

1) Génération de la « pixel clock »

Histoire de se remettre doucement dans le bain, on va commencer par un classique du genre : la division d'horloge. A partir de l'horloge de la carte à 100 MHz, il faut donc générer un signal clk25 à 25 MHz :

- rst : remise à zéro (entrée)
- mclk : horloge maître à 100MHz (entrée)
- clk25 (sortie)

A priori pour celui-ci, il ne devrait pas être nécessaire de faire un testbench et simuler…

2) Composant VGA_640x480

Là nous entrons dans le vif du sujet… L'objectif est de générer les signaux de sortie suivants, simplement à partir de clk25 et d'un reset :

- hsync : le signal de synchronisation horizontale
- vsync : le signal de synchronisation verticale
- hc : signal permettant de connaître le numéro du pixel courant sur une ligne
- vc : signal permettant de connaître le numéro de ligne courant
- vidon : signal permettant d'indiquer si la position courante est dans une zone d'affichage ou non

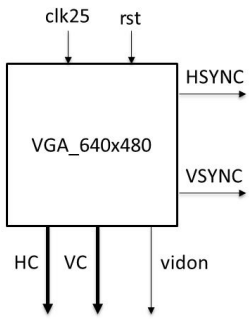


Figure 1. Interface du composant VGA_640x480

a. Définir les valeurs des différentes constantes pour représenter les paramètres temporels du contrôleur

On déclarera les constantes suivantes :

- hbp : back porch horizontal
- hfp : front porch horizontal
- hpixels : nombre de pixels (total) sur une ligne
- vbp : back porch vertical
- vfp : front porch vertical
- vlines : nombre de lignes (total) par frame

Rappel *: la syntaxe pour déclarer une constante est la suivante :*
constant n : std_logic_vector(1 downto 0) := « 00 » ;

b. *Définir le compteur horizontal*

Conformément à ce que nous avons vu l'an passé, écrire un process synchronisé sur l'horloge clk25 avec reset asynchrone :

- si la valeur courante du compteur est égale à hpixels-1, il est remis à 0 ; en même temps, on réalisera l'affectation à 1 du signal d'activation du compteur vertical
- sinon, le compteur est incrémenté et le signal d'activation du compteur vertical est mis à 0

c. *Définir le compteur vertical*

De la même manière pour le compteur vertical :

- si le signal d'activation est à 1 alors :
    - si la valeur courante du compteur est égale à vlines-1, il est remis à 0
    - sinon, le compteur est incrémenté

d. *Affecter les valeurs des sorties HSYNC, VSYNC et VIDON*

- la sortie HSYNC est à 0 quand la valeur du compteur horizontal est inférieure à 96[3], 1 sinon
- la sortie VSYNC est à 0 quand la valeur du compteur vertical est inférieure à 2, 1 sinon
- la sortie VIDON est à 1 quand :
    - la valeur du compteur horizontal est comprise entre hbp et hfp
    - et que la valeur du compteur vertical est comprise entre vbp et vfp
    - 0 sinon

e. *Ecrire un testbench permettant de simuler le composant VGA_640x480 et vérifiez que les chronogrammes respectent la spécification temporelle du VGA*

3) Composant VGA_STRIPES

Ce composant dont l'interface est représentée ci-dessous, va réaliser l'affectation des valeurs des pixels en fonction des valeurs de HC, VC et VIDON :

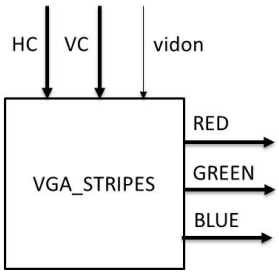


Figure 2. Interface du composant VGA_STRIPES

[3] Ou 128, cf. spécifications

Ce composant est basé sur un processus purement combinatoire qui affecte aux sorties RED, GREEN, BLUE une valeur telle que :

- si VIDON est à 1, alors
    - red <= vc(4) & vc(4) & vc(4)
    - green <= not (vc(4) & vc(4) & vc(4))
    - blue <= « 00 »
- sinon
    - red<= « 000 »
    - green <= « 000 »
    - blue <= « 00 »

4) Composant TOP_VGA_SPRITES

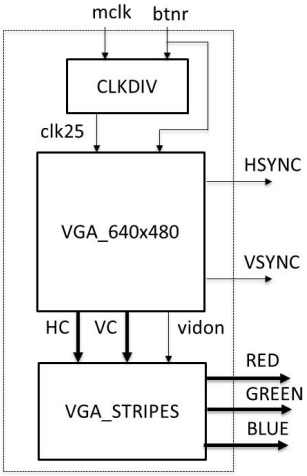


Figure 3. Interface et architecture du composant TOP_VGA_SPRITES

Réalisez l'architecture de l'entité conformément au schéma décrit ci-dessus, en faisant l'instanciation des composants conçus précédemment et en utilisant les signaux nécessaires à leur interconnexion.

Faîtes un testbench et simulez ce composant pour comprendre et vérifier son fonctionnement.

5) Implémentation

Après avoir intégré le fichier UCF, modifié en conséquence, dans votre projet, faîtes l'implémentation, la génération du bitstream et la programmation de la carte Nexys 3 à l'aide du logiciel Adept.

Contrôleur VGA – PROM

Dans l'exemple précédent, nous nous sommes servis des valeurs du compteur vertical pour définir la valeur des pixels à afficher. Dans la pratique, on utilisera généralement des objets graphiques qui peuvent être décrits à l'aide d'algorithmes, ou bien de manière statique en étant stockés dans des mémoires. Ici nous allons nous pencher sur l'utilisation d'une ROM programmable ou PROM, dans laquelle nous allons stocker cet objet graphique (que l'on appelle communément « sprite » ou lutin).

1. Description d'une PROM

Il s'agit d'un composant simple à décrire en VHDL, mais qui requiert quelques éléments de syntaxe. L'exemple ci-dessous décrit une PROM dont l'adresse est codée sur 2 bits, et les mots sur 8 bits : il s'agit donc d'une ROM 4 octets. Elle est programmable dans le sens où son contenu peut-être modifié directement à partir du code VHDL.

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

entity PROM is
    Port ( addr : in  STD_LOGIC_VECTOR (1 downto 0);
           M : out  STD_LOGIC_VECTOR (0 to 7));
end PROM;

architecture Behavioral of PROM is

type rom_array is array (NATURAL range <>) of std_logic_vector(0 to 7);

constant rom:rom_array:= (
       "11111111", --0
       "11000011", --1
       "11000011", --2
       "11111111", --3
       );


begin

process(addr)
variable j: integer;
begin
       j:= conv_integer(addr);
       M <= rom(j);
end process;
end Behavioral;
```

Modifiez ce code afin d'avoir une PROM :

- avec un bus d'adresse sur 4 bits
- des données sur 32 bits

➔ Vous choisirez vous même le contenu de la PROM (cercle, rectangle, lettres, initiales, etc.).

2. Composant VGA_PROM

Il faut concevoir ici une entité dont l'interface est représentée dans le schéma ci-dessous. On remarque qu'en plus des signaux d'entrée/sortie de VGA_STRIPES (HC, VC, vidon, RED, GREEN, BLUE) on a ajouté une entrée et une sortie (pour l'instant on laisse de côté l'entrée SW) :

- Sortie ROM_ADDR4 : bus d'adresse pour la PROM

- Entrée M : bus de données pour la PROM

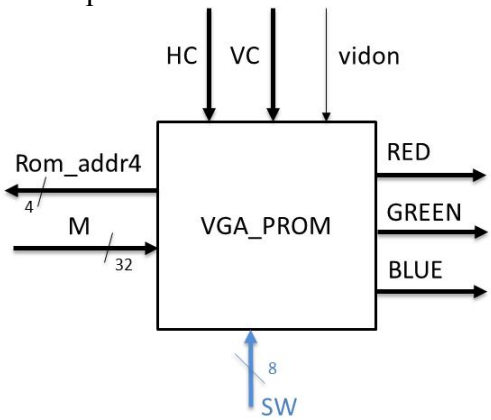


Figure 1. Interface du composant VGA_PROM

L'objectif dans un premier temps est d'afficher le sprite en haut à gauche de l'écran, ce qui correspond donc au premier pixel dans l'affichage visible après les back porch horizontaux et verticaux.

- Vous définirez les 2 constantes relatives aux valeurs de hbp et vbp
- Vous définirez 2 constantes relatives aux dimensions du sprite (w pour la largeur, h pour la hauteur)
- On déclarera 2 signaux rom_addr et rom_pix de même dimension que VC et HC pour le calcul de l'adresse de ligne et de numéro de colonne du pixel
- On déclarera un signal spriteon de type std_logic, utilisé pour calculer si l'on se trouve dans la zone d'affichage du sprite ou non
- On déclarera 3 signaux R, G, B de type std_logic qui seront directement positionnés à 0 ou 1 en fonction de la valeur de data dans la PROM
- Dans l'architecture de ce composant, il faudra :
    - Calculer la valeur de rom_addr à partir de VC et vbp
    - Calculer la valeur de rom_pix à partir de HC et hbp
    - Affecter la bonne valeur à rom_addr4 à partir de rom_addr
    - Calculer la valeur de spriteon :
        - Lorsque HC est compris entre hbp + 1 et hbp + w, et que VC est compris entre vbp + 1 et vbp + h, spriteon est positionné à 1
        - 0 sinon
    - Dans un processus purement combinatoire, affecter les valeurs à RED, GREEN et BLUE :
        - Si spriteon et vidon sont à 1
            - On récupère la valeur de l'index du pixel correspondant (j := conv_integer(rom_pix)
            - On affecte à R, G et B la valeur de M(j)
            - On affecte les sorties :
                - RED <= R&R&R,
                - GREEN <= G&G&G
                - BLUE <= B&B
        - Sinon, on affiche du noir, i.e. :
                - RED <= « 000 »,
                - GREEN <= « 000 »
                - BLUE <= « 00 »

3. Composant TOP_VGA_PROM

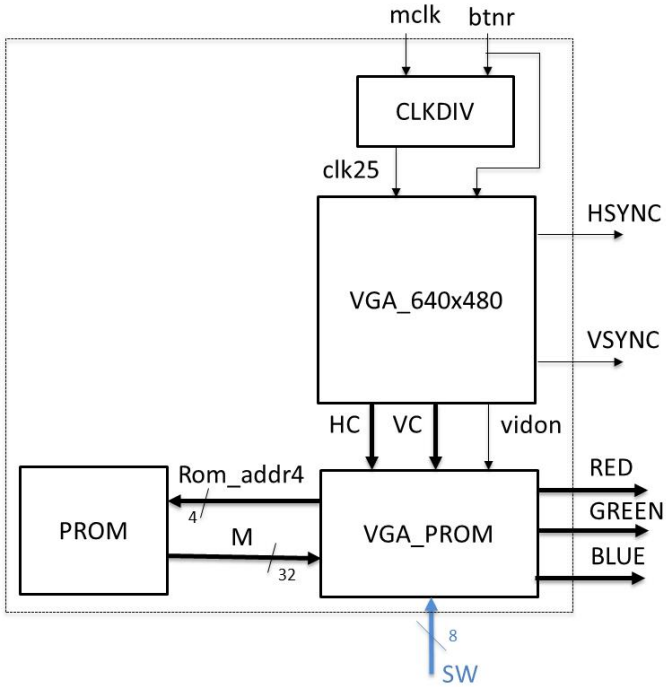


Figure 2. Interface du composant TOP_VGA_PROM

Dans une nouvelle entité, instanciez les différents composants nécessaires à la réalisation de l'architecture décrite dans la figure 2.

Par simulation, vérifiez d'abord le fonctionnement de TOP_VGA_PROM.

4. Implémentation

Ajoutez le fichier UCF, faîtes l'implémentation et la génération du Bitstream. Une fois validé, vous pouvez essayer de modifier certains paramètres pour bien comprendre le mécanisme d'affichage des sprites.

5. Contrôle de la position d'affichage

L'idée de ce dernier exercice est de contrôler la position d'affichage du sprite à l'aide des 8 switch disponibles sur la carte. On pourra par exemple définir dans le fichier VGA_PROM 2 signaux supplémentaires correspondant aux coordonnées du premier pixel, situé en haut à gauche, et que ces signaux seront modifiés en fonction de la valeur positionnée sur les switch (on pourra par exemple prendre 4 switch pour repérer la ligne, 4 switch pour repérer la colonne).
Modifiez le top en conséquence et générez le bitstream correspondant pour valider le fonctionnement du contrôle de position d'affichage.