Travaux pratiques : commande d'un écran VGA par un circuit programmé en VHDL

Nous allons au cours de cette séance programmer un hybride CPLD/FPGA en VHDL afin de lui permettre de commander un écran VGA pour obtenir un affichage en 640 par 480 pixels non entrelacé.
Nous supposons une familiarisation avec l'utilisation du logiciel Max+plus II.

Dans un premier temps nous nous contenterons d'afficher un écran rouge, puis une mire à barres verticales, puis un carré au centre de l'écran et enfin nous ferons rebondir ce carré sur les bords de l'écran.

La carte de développement que nous utiliserons contient un CPLD et un "FPGA", le circuit EPF10K20 (20 000 portes) ayant ses sorties directement reliées à un connecteur pour écran VGA.
On se référencera aux annexes, ainsi qu'à la documentation de la carte, pour connaître les bornes du circuit reliées à l'écran et à l'horloge 25,175 MHz de la carte. On pensera d'autre part à configurer correctement les cavaliers de la carte afin de pouvoir programmer le bon circuit. Ces quatre cavaliers se trouvent entre le circuit EPM7128SLC et le connecteur JTAG_IN du câble de programmation : les deux premiers TDI et TDO doivent être en position basse, tandis que les deux derniers DEVICE et BOARD doivent être en position haute.

On trouvera également en annexe les chronogrammes des signaux de commande attendus par un écran VGA : synchronisation horizontale et verticale (ligne et trame) ainsi que le moment d'application des signaux RVB (rouge, vert et bleu).

Pour ces derniers, nous nous limiterons à des signaux binaires, et seules huit couleurs seront possibles à l'affichage. A la sortie du CPLD un simple circuit résistance diode (se trouvant sur la carte) permet de convertir le signal compatible TTL (0/ 5 V) en un signal compatible RVB (0/ 700 mV).

L'écran commandé doit être un moniteur VGA classique, sans commande de mode d'économie d'énergie.

1. Affichage d'une image fixe

Le programme principal, écrit en VHDL, est fourni sur une disquette ou sur un CDROM et donné en annexe 3. Le copier sur le disque dur, ouvrir le logiciel MAX+plus II et sauvegarder le programme sous un nom différent afin de toujours disposer d'une source intacte. La sauvegarde se fera dans un nouveau répertoire afin de ne pas mélanger avec d'autres fichiers.

Ce programme génère :

- les différents signaux de synchronisation verticale (**vert_sync_out**) et horizontale (**hor_sync_out**)
- l'affichage RVB de la couleur (**r_out, v_out et b_out**)
- les signaux des compteurs de ligne (**pixel_ln**) et de colonne (**pixel_col**) permettant de déterminer quel pixel est à un instant donné rafraîchi sur l'écran.

Les entrées sont :

- l'horloge **ck** à 25,175 MHz de la carte
- les entrées couleurs (**r_in**, **v_in** et **b_in**).

Etudier ce programme et en prenant comme référence le premier point affiché (en haut à gauche de l'écran), calculer la valeur des différentes constantes.

Déterminer alors la fréquence de rafraîchissement de l'écran.

Initialiser ces constantes, compiler (compilation fonctionnelle seulement) le programme.

La vérification du bon fonctionnement nous impose de préciser la valeur des entrées couleur.
Pour cela, ouvrir un nouveau projet à l'aide de l'entrée graphique.
Importer le symbole créé lors de la compilation du programme précédent (clic droit puis **enter symbol**).
Etablir les connexions d'entrée et sortie (symboles nécessaires dans la bibliothèque **c:\maxplus2\maxlib\prim** accessible par clic droit puis **enter symbol**) de manière à obtenir un écran rouge.
Sauvegarder et vérifier le projet (**ctrl K**) puis assigner le circuit cible ainsi que le broches d'entrées sorties (menu **Assign / Device** puis **Assign / Pin**). On veillera à entrer la référence exacte du circuit cible.
Compiler ensuite le projet.

Remarque : si le fichier graphique et la description VHDL associée au symbole entré ne sont pas dans le même répertoire, il faut préciser à l'éditeur graphique l'utilisation de ressources externes par le menu **Option / User Librairies**.

Les connexions du circuit cible étant en technologie SRAM, (contrairement à l'autre circuit de la carte dont les connexions sont en technologie EEPROM) le terme utilisé pour transférer les données de l'ordinateur vers la cible est "**configur**" et non "**program**". Cette opération se fait avec un fichier **.sof** -SRAM Object File- généré lors de la compilation et non un fichier **.pof** -Programmer Object File- (qui pourrait servir à programmer éventuellement l'EEPROM associée à un circuit SRAM).

Lancer le programmateur (**Maxplus2 / Programmer**), puis déclarer la cible et le fichier de configuration (**JTAG / Multi-Device JTAG Chain Setup** puis le nom du circuit cible dans **Device Name** et celui du fichier de configuration **.sof** dans **Select Programming File Name** puis **Add**).
Vérifier la cohérence des données par **Detect JTAG Chain Info** puis **OK**.
Lancer la configuration.

Connecter un écran VGA à la carte et vérifier le bon fonctionnement

Eteindre l'alimentation de la carte pendant quelques secondes puis rallumer. Justifier le comportement du circuit.

2. Affichage d'une mire

2.1. Mire à l'aide d'un fichier graphique

Créer un nouveau projet graphique (.gdf) dans un nouveau répertoire.

Placer le symbole correspondant au générateur de signaux de synchronisation étudié au chapitre précédent (clic droit puis **enter symbol**).
On n'oubliera pas de déclarer l'utilisation de ressources externes (menu **Option / User Librairies**).

Connecter les sorties du comptage de pixels lignes de la synchronisation sur les entrées couleur de manière à obtenir au moins une fois chaque couleur dans une bande suffisamment large.

Sauvegarder et vérifier le projet, assigner le circuit et les connexions, configurer le composant cible et vérifier le fonctionnement.

2.2. Mire à l'aide d'un fichier VHDL

On se propose maintenant d'afficher sur l'écran huit barres verticales de largeurs égales correspondant aux huit couleurs possibles.
Pour cela on utilise le programme VHDL fourni sur la disquette ou sur le CDROM et donné en annexe 4.
Ce programme comporte une erreur qu'il faudra corriger.

Avant correction, compiler ce programme, puis faire l'association au programme générant la synchronisation, à l'aide de l'éditeur graphique.
Compiler l'ensemble, configurer le circuit cible et observer le fonctionnement.
Expliquer puis corriger l'erreur et vérifier cette fois le bon fonctionnement.

Refaire l'association des deux programmes en utilisant uniquement l'éditeur VHDL. On se servira de l'instruction **component**.

Vérifier.

3. Affichage d'un carré fixe

Le programme donné en annexe 5 (et présent sur la disquette ou le CDROM) permet, associé au générateur de synchronisation, d'afficher un carré blanc au centre de l'écran sur fond rouge.
Tester le bon fonctionnement de l'ensemble.

4. Affichage d'un carré "ludion"

On donne en annexe 6 une description VHDL, que l'on trouvera sur la disquette ou sur le CDROM, permettant de faire rebondir le carré sur les bords haut et bas de l'écran.
Vérifier le bon fonctionnement de ce programme.

5. Affichage d'un carré bondissant

Modifier le programme pour que le carré rebondisse également sur les bords droit et gauche de l'écran.
Vérifier.

Modifier maintenant le programme pour donner l'illusion d'un rebondissement sur l'avant et l'arrière de l'écran. Pour cela, on modifiera de manière dynamique la taille du carré.

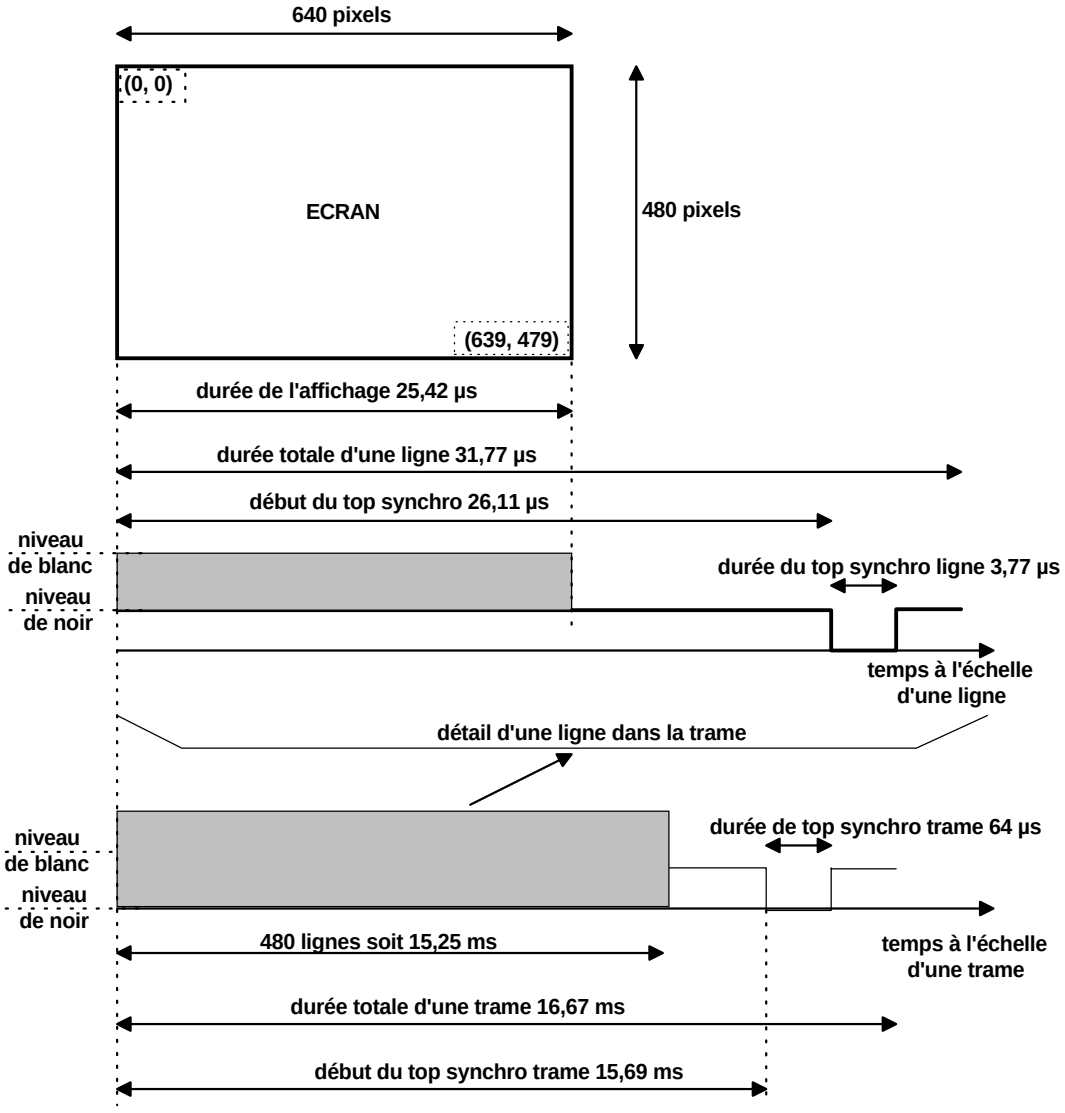
Annexe 1 : connectique

Connexion des signaux sur la carte

signaux de la carte	connecteur VGA				
horloge 25,175 MHz	synchro horizontale	synchro verticale de l'écran	rouge	vert	bleu
borne 91	borne 240	borne 239	borne 236	borne 237	borne 238
bornes du circuit EPF10K20 RC240-4					

Annexe 2 : signal vidéo composite

Représentation de l'écran et du signal vidéo composite



Annexe 3 : génération des signaux de synchronisation

```
---------------------------------------------------------------------------------------------------------------------------

library IEEE;
use  IEEE.STD_LOGIC_1164.all;
use  IEEE.STD_LOGIC_ARITH.all;
use  IEEE.STD_LOGIC_UNSIGNED.all;

ENTITY aff_vga0 IS
        PORT
(ck, r_in, v_in, b_in                           : IN      STD_LOGIC;
r_out, v_out, b_out, hor_sync, vert_sync: OUT   STD_LOGIC;
pixel_ln, pixel_col                     : OUT STD_LOGIC_VECTOR(9 DOWNTO 0));
        END aff_vga0;

-- la position des pixels affichés est donnée à chaque instant par pixel_ln et pixel_col
-- les sorties couleurs r_out, v_out et b_out donnent la valeur (1 ou 0) du pixel pour la position.
-- les entrées de couleur r_in, v_in et b_in devront être synchronisées avec la position des pixels.

ARCHITECTURE arch OF aff_vga0 IS

-- le comptage commençant à 0,
-- les constantes sont diminuées d'une unité par rapport à la grandeur représentée

        CONSTANT NPA            : INTEGER :=xxx;--nombre de pixels affichés par lignes
        CONSTANT DSH            : INTEGER :=xxx;--début du top synchro horizontale
        CONSTANT FSH            : INTEGER :=xxx;--fin du top synchro horizontale
        CONSTANT NPT            : INTEGER :=xxx;--nombre de pixels total par lignes

        CONSTANT NLA            : INTEGER :=xxx;--nombre de lignes affichées
        CONSTANT DSV            : INTEGER :=xxx;--début du top synchro verticale
        CONSTANT FSV            : INTEGER :=xxx;--fin du top synchro horizontale
        CONSTANT NLT            : INTEGER :=xxx;--nombre de lignes total


SIGNAL video_on, video_on_v, video_on_h     : STD_LOGIC;
SIGNAL h_cmpt                   : STD_LOGIC_VECTOR(9 DOWNTO 0);
SIGNAL v_cmpt                   : STD_LOGIC_VECTOR(9 DOWNTO 0);


BEGIN

-- autorisation d'affichage uniquement dans la zone image verticale et horizontale
video_on <= video_on_H AND video_on_V;
```

```
PROCESS
BEGIN
        WAIT UNTIL(ck 'EVENT) AND (ck ='1');

-- génération de la synchronisation et de l'affichage horizontal

-- h_cmpt compte (NPA+1) pixels d'affichage plus la synchronisation horizontale
-- le comptage commence au début de l'affichage d'une ligne

-- pixel_col donne la position des pixel affichés
-- pixel_col reste à NPA lors de signaux de synchronisation

-- video_on définit la zone d'affichage sur la ligne

--  hor_sync  ------------------------------------------------_____________---------
--  h_cmpt    0               NPA               DSH        FSH    NPT
--  pixel_col  0              NPA               NPA        NPA    NPA
--  video_on_h ----------------------_____________________________


        IF (h_cmpt >= NPT)              THEN  h_cmpt <= "0000000000";
                                              pixel_col <= "0000000000";
                                              hor_sync<='1';
                                              video_on_h <= '1';

        ELSIF (h_cmpt < NPA)            THEN  h_cmpt <= h_cmpt + 1;
                                              pixel_col <= h_cmpt +1;
                                              hor_sync<='1';
                                              video_on_h <= '1';

        ELSIF (h_cmpt >= NPA AND h_cmpt < DSH)
                                        THEN  h_cmpt <= h_cmpt + 1;
                                              hor_sync<='1';
                                              video_on_h <= '0';

        ELSIF (h_cmpt >= DSH AND h_cmpt < FSH)
                                        THEN  h_cmpt <= h_cmpt + 1;
                                              hor_sync<='0';
                                              video_on_h <= '0';

        ELSE                                  h_cmpt <= h_cmpt + 1;
                                              hor_sync<='1';
                                              video_on_h <= '0';

END IF;
```

```
-- v_cmpt compte (NLA+1) lignes d'affichage plus la synchronisation verticale
-- l'incrémentation du compteur ne doit se faire qu'en fin de ligne (h_cmpt=NPT)
-- pixel_ln donne la position des pixels affichés
-- video_on autorise ou non l'affichage

--  vert_sync      ------------------------------------_______------------
--  v_cmpt         0                    NLA     DSV     FSV       NLT
--  pixel_ln       0                    NLA     NLA     NLA       NLA
--  video_on_v     -------------------------_____________________


        IF (v_cmpt >= NLT) AND (h_cmpt >= NPT)       THEN   v_cmpt <= "0000000000";
                                                            pixel_ln <="0000000000";
                                                            vert_sync <= '1';
                                                            video_on_v <= '1';

        ELSIF (v_cmpt < NLA) AND (h_cmpt >= NPT) THEN       v_cmpt<= v_cmpt + 1;
                                                            pixel_ln <= v_cmpt +1;
                                                            vert_sync<='1';
                                                            video_on_v <= '1';

        ELSIF (v_cmpt >= NLA AND v_cmpt < DSV) AND (h_cmpt >= NPT)
                                                      THEN  v_cmpt<= v_cmpt + 1;
                                                            vert_sync<='1';
                                                            video_on_v <= '0';

        ELSIF (v_cmpt >= DSV AND v_cmpt < FSV) AND (h_cmpt >= NPT)
                                                      THEN  v_cmpt<= v_cmpt + 1;
                                                            vert_sync<='0';
                                                            video_on_v <= '0';

        ELSIF (h_cmpt >= NPT)                         THEN  v_cmpt<= v_cmpt + 1;
                                                            vert_sync<='1';
                                                            video_on_v <= '0';

                END IF;

-- l'affichage n'est pas autorisé pendant les temps de synchronisation
                r_out <= r_in AND video_on;
                v_out <= v_in AND video_on;
                b_out <= b_in AND video_on;

        END PROCESS;

END arch;
-------------------------------------------------------------------------------------------------------------------------
```

Annexe 4 : mire à huit barres verticales

Les trois sorties couleur sont considérées comme un seul bus afin de simplifier l'écriture. Ces sorties seront reliées aux entrées couleur correspondantes du circuit de synchronisation.

ATTENTION : CE PROGRAMME COMPORTE UNE ERREUR

```
---------------------------------------------------------------------------------------------------------
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.all;
USE IEEE.STD_LOGIC_ARITH.all;
USE IEEE.STD_LOGIC_SIGNED.all;

ENTITY mire0 IS
 PORT(          couleur                         : OUT std_logic_vector (2 downto 0);
                colonne                         : IN std_logic_vector (9 downto 0));
END mire0;

ARCHITECTURE archi OF mire0 IS

BEGIN

couleur <=      "000" WHEN   (          0 <= conv_integer (colonne(9 downto 0))
                                         and     conv_integer (colonne(9 downto 0)) < 80  )  ELSE

                "001" WHEN   (          80 <= conv_integer (colonne(9 downto 0))
                                         and     conv_integer (colonne(9 downto 0)) < 160  )  ELSE

                "010" WHEN   (          160 <= conv_integer (colonne(9 downto 0))
                                         and     conv_integer (colonne(9 downto 0)) < 240  )  ELSE

                "011" WHEN   (          240 <= conv_integer (colonne(9 downto 0))
                                         and     conv_integer (colonne(9 downto 0)) < 320  )  ELSE

                "100" WHEN   (          320 <= conv_integer (colonne(9 downto 0))
                                         and     conv_integer (colonne(9 downto 0)) < 400  )  ELSE

                "101" WHEN   (          400 <= conv_integer (colonne(9 downto 0))
                                         and     conv_integer (colonne(9 downto 0)) < 480  )  ELSE

                "110" WHEN   (          480 <= conv_integer (colonne(9 downto 0))
                                         and     conv_integer (colonne(9 downto 0)) < 560  )  ELSE

                "111" ;
END archi;
---------------------------------------------------------------------------------------------------------
```

Annexe 5 : génération d'un carré fixe

```
---------------------------------------------------------------------------------------------------------------

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.all;
USE IEEE.STD_LOGIC_ARITH.all;
USE IEEE.STD_LOGIC_UNSIGNED.all;

ENTITY ball_fixe_1 IS
PORT(  rouge, vert, bleu                    : OUT std_logic;
          pixel_colone, pixel_ligne         : IN std_logic_vector (9 downto 0));
END ball_fixe_1;

ARCHITECTURE archi OF ball_fixe_1 IS
--ce programme affiche une "balle carrée" fixe blanche sur fond rouge

--taille représente (en pixel) un demi-côté du carré
--ball_Y_pos et ball_X_pos donne la position du centre du carré

CONSTANT taille                             :INTEGER :=10;
CONSTANT ball_Y_pos                         :INTEGER :=239;
CONSTANT ball_X_pos                         :INTEGER :=319;

--balle est au NL1 lorsque le carré se trouve sur les pixels rafraîchi
SIGNAL balle                                : std_logic;

BEGIN

-- le fond d'écran est rouge
rouge <=  '1';
-- le carré est blanc, la position du carré correspondant aux pixels où balle =1
vert <=  balle;
bleu <=  balle;

--en fonction du pixel rafraîchi, on détermine la couleur
balle <= '1'       when          (Ball_X_pos  <= pixel_colone + taille )
                                 AND     (Ball_X_pos  >= pixel_colone - taille)
                                 AND     (Ball_Y_pos  <= pixel_ligne + taille)
                                 AND     (Ball_Y_pos  >= pixel_ligne - taille )

                   else      '0';

END archi;
---------------------------------------------------------------------------------------------------------------
```

Annexe 6 : génération d'un carré ludion

```
---------------------------------------------------------------------------------------------------------------------------

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.all;
USE IEEE.STD_LOGIC_ARITH.all;
USE IEEE.STD_LOGIC_UNSIGNED.all;

ENTITY ludion IS
        PORT( rouge, vert, bleu                         : OUT std_logic;
                sync_verti                              : IN std_logic;
                pixel_colone, pixel_ligne               : IN std_logic_vector (9 downto 0));
END ludion;

ARCHITECTURE archi OF ludion IS

--ce programme fait rebondir une "balle carrée"
--sur les bords inférieur et supérieur de l'écran
--le déplacement du carré sera recalculé à chaque top synchro verticale
--le fond de l'écran est rouge, la "balle" est blanche

-- taille représente (en pixel) un demi-côté du carré
CONSTANT taille                                 :INTEGER :=10;

--balle est au NL1 lorsque le carré se trouve sur les pixels rafraîchi
--ball_Y_pos et ball_X_pos donne la position du centre du carré
--mouv_Y donne le déplacement du carré
--mouv_Y pouvant être négatif, le plus simple est de prendre un entier
SIGNAL balle                                    : std_logic;
SIGNAL ball_Y_pos, ball_X_pos                   : std_logic_vector(9 DOWNTO 0);
SIGNAL mouv_                                    : INTEGER RANGE -7 TO 7 :=2;


BEGIN

--la position du centre du carré sur l'axe x reste au milieu de l'écran
--conversion sur 10 bits de (640/2-1)=319
ball_X_pos <= CONV_STD_LOGIC_VECTOR(319,10);

-- le fond d'écran est rouge
rouge <=  '1';
-- le carré est blanc, la position du carré correspondant aux pixels où balle =1
vert <=  balle;
bleu <=  balle;

--en fonction du pixel rafraîchi, on détermine la couleur
balle <= '1' when       (Ball_X_pos - taille <= pixel_colone )
                        AND     (Ball_X_pos + taille >= pixel_colone)
                        AND     (Ball_Y_pos - taille <= pixel_ligne )
                        AND     (Ball_Y_pos + taille >= pixel_ligne )

                        else            '0';
```

```
--calcul de la position de la "balle"
--qui se déplace de +/-2 pixels à chaque top synchro verticale
mouvement_balle: process
	BEGIN
		WAIT UNTIL sync_verti'event and sync_verti = '1';

--sens du déplacement suivant que nous sommes en haut ou en bas de l'écran
		IF ball_Y_pos >= 479 - taille THEN	mouv_Y <=-2;
		ELSIF ball_Y_pos <= taille THEN		mouv_Y <=2;
		END IF;

-- Calcul de la prochaine position de Y
		ball_Y_pos <= ball_Y_pos + mouv_Y;

	END process mouvement_balle;

END archi;

-------------------------------------------------------------------------------------------------------------
```