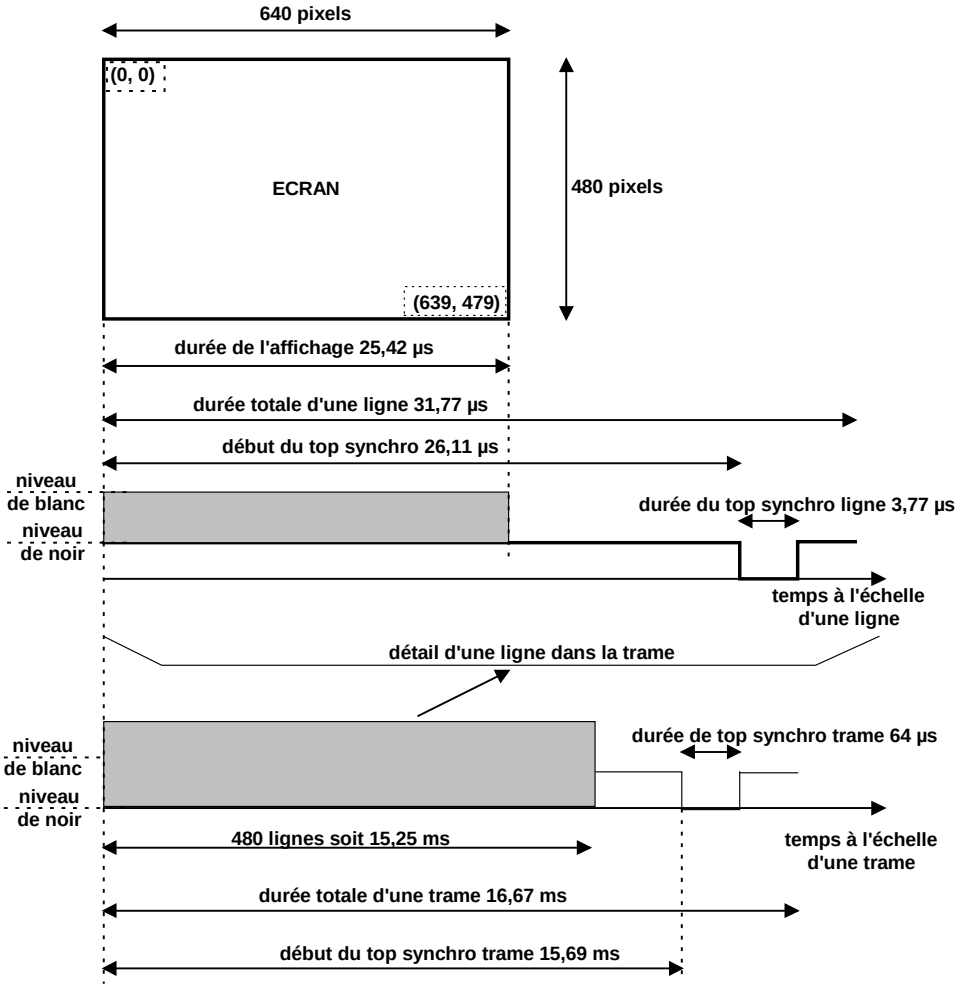
Commande d'un écran VGA par un circuit programmé en VHDL

Nous allons au cours de cette séance programmer un FPGA en VHDL afin de lui permettre de commander un écran VGA (Video Graphic Array) pour obtenir un affichage en 640 par 480 pixels non entrelacé.
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 UP1 ou UP2 que nous utiliserons contient un FPGA, le circuit EPF10K20 (20 000 portes) dont les sorties sont directement reliées à un connecteur pour écran VGA.

1 Rappel sur l'affichage VGA

Afficher une image sur un écran consiste à imposer une couleur particulière sur chaque point élémentaire (ou pixel) de l'écran, tandis qu'afficher une séquence vidéo consiste à enchaîner des images rapidement.
Pour une image, plutôt que d'afficher tous les points en même temps, il est plus simple de faire un balayage de l'écran à vitesse importante, la persistance des impressions lumineuses sur la rétine de l'œil donnant l'impression d'une image stable.



Le balayage de l'écran sera composé d'un balayage ligne (donc horizontal) associé à un balayage colonne (donc vertical). Ainsi, après avoir balayé toutes les colonnes, de gauche à droite, d'une ligne, on passe à la ligne suivant jusqu'à la dernière ligne en bas, puis on remonte à la première en haut.

Pour permettre la synchronisation, après le balayage de chaque ligne il existe un temps mort, et le balayage dure réellement 31,77 µs soit 800 pixels sur une ligne.
De même, le balayage complet de l'écran dura 16,67 µs soit 525 lignes au total. On aura donc un rafraîchissement de l'écran à la fréquence de 60 Hz (1/16,67 ms).

Dans notre application, une ligne comprendra 640 pixels visibles à l'écran et sera balayée en 24,43 µs. Il y aura 480 lignes (donc 480 pixels) affichées à l'écran, d'où la dénomination 640x480.

La couleur de chaque pixel sera la somme pondérée de trois couleurs élémentaires : rouge, vert et bleu (il s'agit ici de trichromie additive, différente de celle de la palette du peintre qui est soustractive avec comme couleurs primaires magenta, cyan et jaune).
Pour afficher un point blanc, les trois couleurs auront l'intensité maximale, et l'intensité minimale pour afficher un point noir (ou éteindre le pixel).

Dans notre application, nous ne pourrons pas moduler les couleurs élémentaires (il faudrait 3 CNA sur la carte), nous nous contenterons d'afficher « binairement » chaque couleur. A la sortie du FPGA 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).
Nous ne pourrons donc afficher que 8 couleurs à l'écran.

On se référencera aux annexes, 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.(voir annexe du TP de prise en main de Quartus)

Affichage d'un fond couleur unie

Ouvrir un nouveau projet dans Quartus II (VGA.qdf par exemple), en y associant le fichier « synchroX.vhd » du répertoire ressource. On imposera le circuit cible de la carte UP1, le FLEX 10K20RC240-4 ou le le FLEX 10K100RC240-4 de la carte UP2.
Ouvrir le programme VHDL et le sauvegarder dans votre répertoire, supprimer l'ancien de votre projet.

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 :

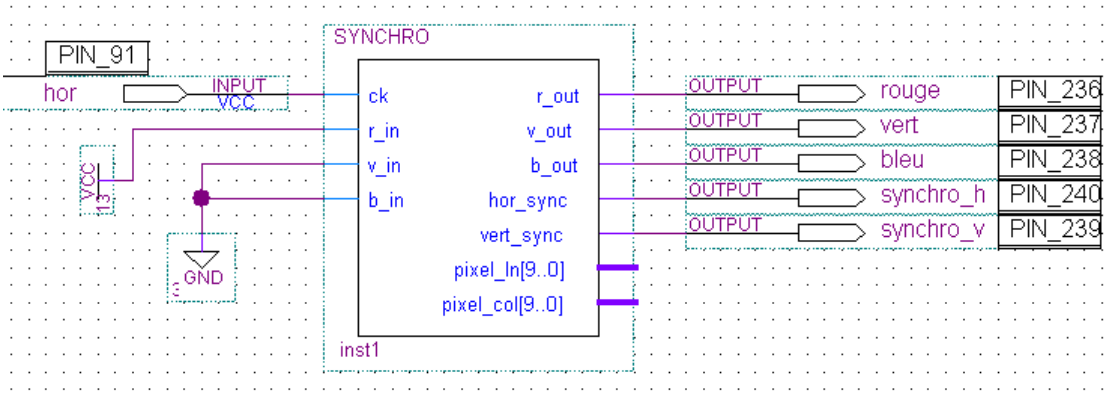
- l'horloge **ck** à 25,175 MHz de la carte
- les entrées couleurs (**r_in**, **v_in** et **b_in**).

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 (notées XXX dans le programme).

Déterminer alors la fréquence de rafraîchissement de l'écran.

Initialiser ces constantes, faire une « analyse et synthèse » du programme. Créer un symbole graphique.

La vérification du bon fonctionnement nous impose de préciser la valeur des entrées couleur. Ouvrir pour cela un feuille graphique, y placer le symbole du fichier VHDL, relier les entrées sorties comme indiqué ci-après :



Compiler ensuite le projet.

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 transfert des données vers la carte se fait avec un fichier **.sof** -SRAM Object File- généré lors de la compilation et non un fichier **.pof** -Programmer Object File-.

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

Affichage d'une mire

Mire à l'aide d'un fichier graphique

Sauvegarder la description graphique sous un nom différent (Mire.bdf par exemple) et la modifier en connectant 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.

Tester votre solution.

Mire à l'aide d'un fichier VHDL

On se propose d'afficher la mire à l'aide de l'association du programme de synchronisation et du programme VHDL fourni dans le répertoire ressource et donnée en annexe 3 .

Ce programme comporte une erreur qu'il faudra corriger par la suite.

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.

Affichage d'un carré fixe

Le programme donné en annexe 4 (et disponible dans le dossier ressource) permet, associé au générateur de synchronisation, d'afficher un carré blanc au centre de l'écran sur fond rouge.

Etudier le programme.
Tester le bon fonctionnement de l'ensemble.

Affichage d'un carré "ludion"

On donne en annexe 5 une description VHDL, que l'on trouvera dans le répertoire ressource, permettant de faire rebondir le carré sur les bords haut et bas de l'écran.

Etudier le programme.
Vérifier le bon fonctionnement de ce programme.

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 : 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 SYNCHROX 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 SYNCHROX;

-- 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 SYNCHROX 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 : 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 4 : 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 5 : 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;

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

Annexes : exemples de solutions

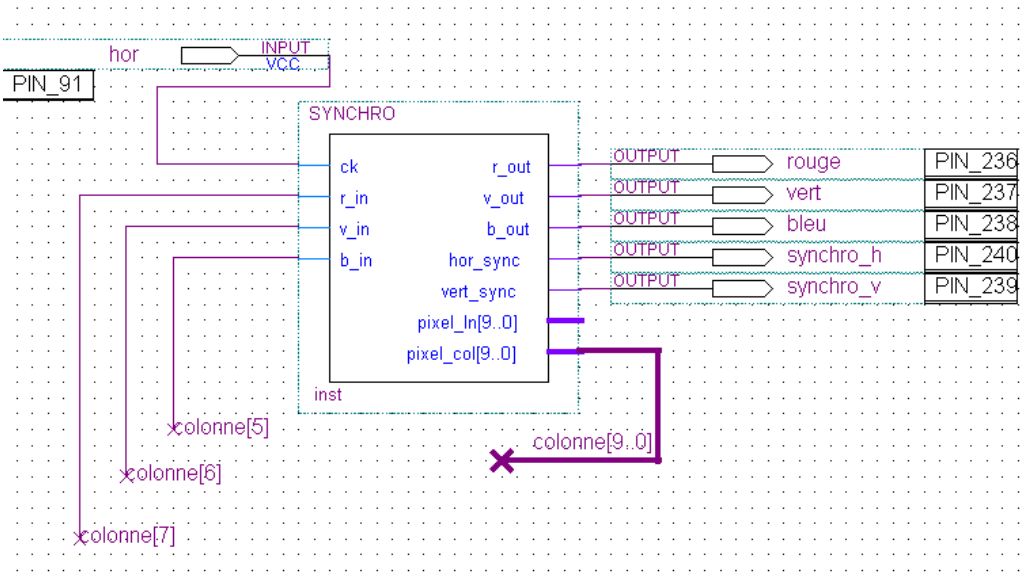
Constantes du programme SYNCHRO.vhd

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

	CONSTANT NPA		: INTEGER :=639;--nombre de pixels affichés par lignes
	CONSTANT DSH		: INTEGER :=656;--début du top synchro horizontale
	CONSTANT FSH		: INTEGER :=751;--fin du top synchro horizontale
	CONSTANT NPT		: INTEGER :=799;--nombre de pixels total par lignes

	CONSTANT NLA		: INTEGER :=479;--nombre de ligne affichées
	CONSTANT DSV		: INTEGER :=493;--début du top synchro verticale
	CONSTANT FSV		: INTEGER :=494;--fin du top synchro horizontale
	CONSTANT NLT		: INTEGER :=524;--nombre de lignes total
```

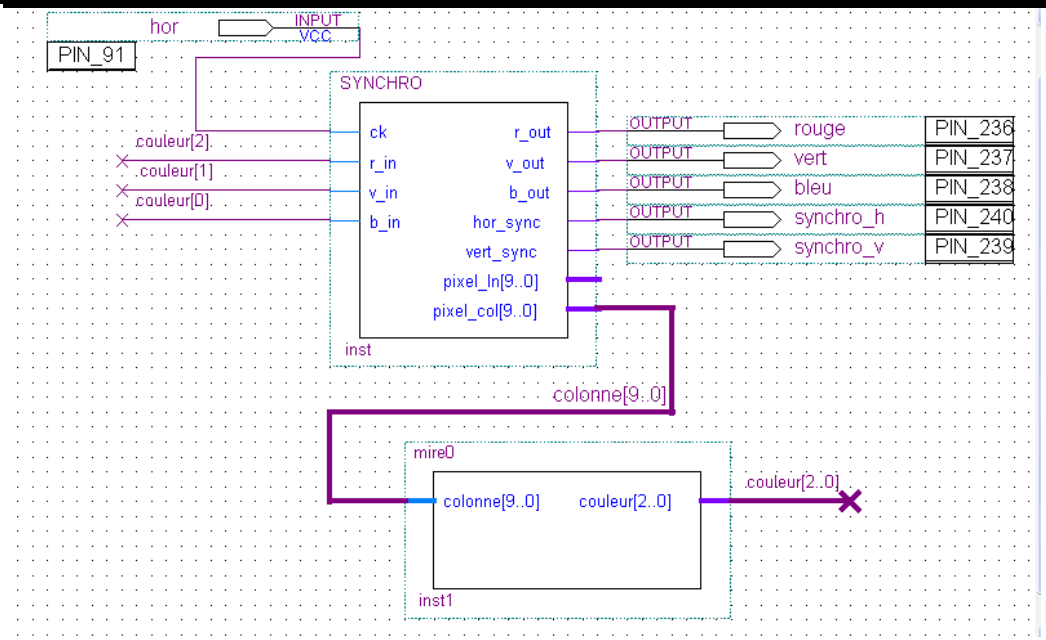
Affichage d'une mire (description graphique)



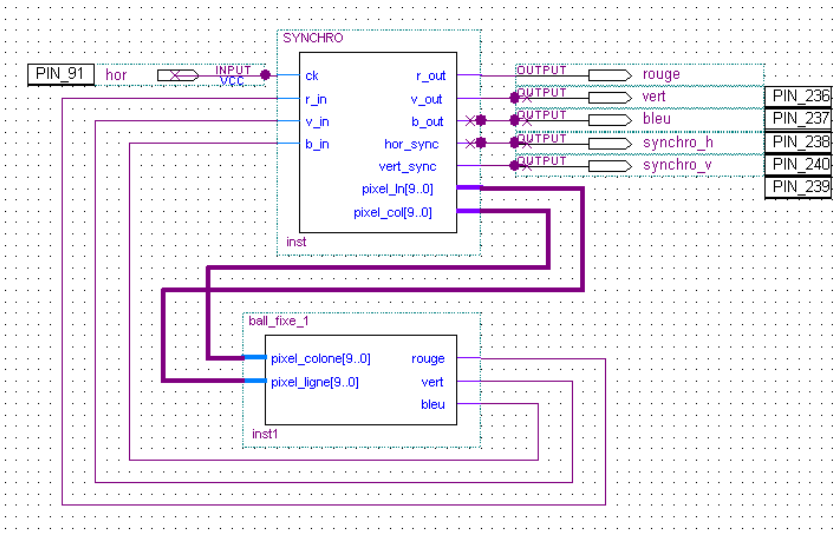
Affichage d'une mire (description vhdl)

On remarque que si on ne modifie pas le fichier mire0.vhd proposé, les bande de la mire sont irrégulière ; en effet dans ce fichier c'est le paquetage .STD_LOGIC_SIGNED.all qui est utilisé pour effecteur les opération arithmétiques ; les opération étant signée, le code utilisé est le complément à 2 et non le binaire naturel.
Il suffit d'effectuer la modification suivant pour revenir en binaire naturel :

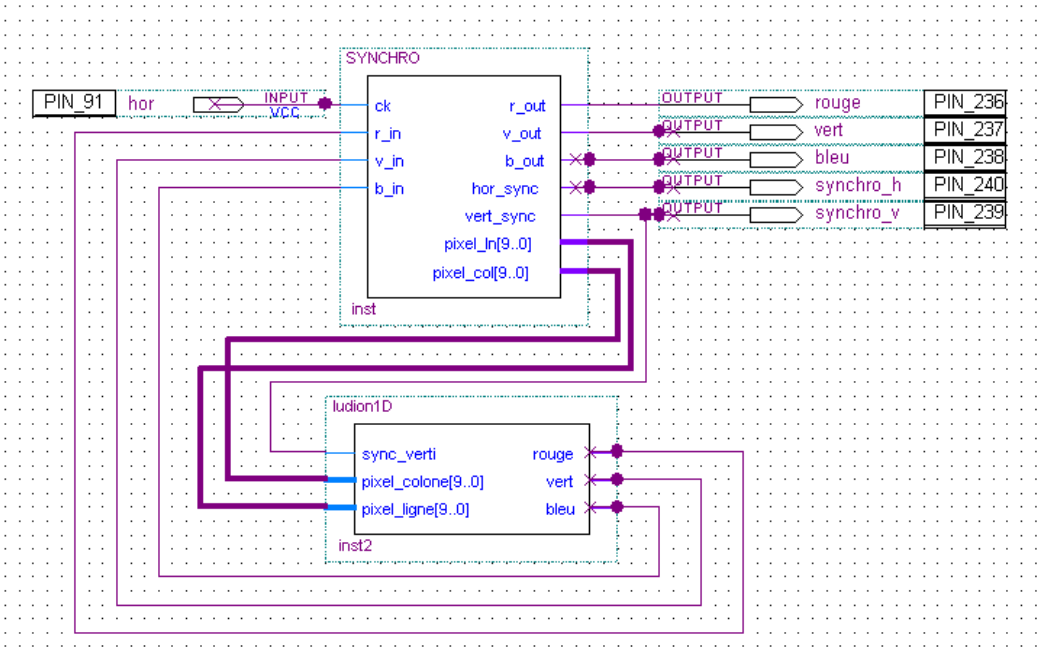
```
USE IEEE. LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.all;
USE IEEE.STD_LOGIC_ARITH.all;
USE IEEE.STD_LOGIC_UNSIGNED.all;;
```



Affichage d'un carré fixe



Affichage d'un carré rebondissant verticalement



Affichage d'un carré rebondissant sur les bords

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

ENTITY ludion_2d 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_2d;
```

```
architecture archi of ludion_2d is

--ce programme fait rebondir une "balle carrée"
--sur les bords 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 et mouv_X donne le déplacement du carré
--mouv_Y et mouv_X pouvant être négatifs, le plus simple est de prendre des entiers

SIGNAL balle                            : std_logic;
SIGNAL ball_Y_pos, ball_X_pos           : std_logic_vector(9 DOWNTO 0);
SIGNAL mouv_Y                           : INTEGER RANGE -7 TO 7 :=2;
SIGNAL mouv_X                           : INTEGER RANGE -7 TO 7 :=2;


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 - 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" sur l'axe Y
--qui se déplace de +/-2 pixels à chaque top synchro verticale
mouvement_balle_Y: process
        BEGIN
                WAIT UNTIL sync_verti'event and sync_verti = '1';
--sens du déplacement suivant que nous sommmes 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_Y;
```

```
--calcul de la position de la "balle" sur l'axe X
--qui se déplace de +/-2 pixels à chaque top synchro verticale
mouvement_balle_X: process
        BEGIN
                WAIT UNTIL sync_verti'event and sync_verti = '1';
--sens du déplacement suivant que nous sommmes à gauche ou à droite de l'écran
                        IF ball_X_pos >= 639 - taille THEN       mouv_X <=-2;
                        ELSIF ball_X_pos <= taille THE mouv_X <=2;
                        END IF;

-- Calcul de la prochaine position de X
                ball_X_pos <= ball_X_pos + mouv_X;
END process mouvement_balle_X;

END archi;
```

Affichage d'un carré rebondissant en 3D

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




ENTITY ludion_3d 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_3d;

architecture archi of ludion_3d is

--ce programme fait rebomdir une "balle carrée" en 3d
--le déplacement du carré sera recalculé à chaque top synchro verticale
--le fond de l'écran est rouge, la "balle" est blanche

--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 et mouv_X donne le déplacement du carré
--mouv_Y et mouv_X pouvant être négatifs, le plus simple est de prendre des entiers
--taille représente (en pixel) un demi côté du carré
--pour diminuer la vitesse de déplacement sur l'axe Z, on divise sync_verti par 4
--DIV_Z est la sortie du compteur et HOR_Z le signal divisé par 4
SIGNAL balle                            : std_logic;
SIGNAL ball_Y_pos, ball_X_pos           : std_logic_vector(9 DOWNTO 0);
SIGNAL mouv_Y                           : INTEGER RANGE -7 TO 7 :=2;
SIGNAL mouv_X                           : INTEGER RANGE -7 TO 7 :=2;
SIGNAL mouv_Z                           : INTEGER RANGE -7 TO 7 :=1;
SIGNAL taille                           : INTEGER RANGE 0 TO 63 :=10;
SIGNAL DIV_Z                            : std_logic_vector(1 DOWNTO 0);
SIGNAL HOR_Z                            : 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 - 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" sur l'axe Y
--qui se déplace de +/-2 pixels à chaque top synchro verticale
mouvement_balle_Y: process
          BEGIN
          WAIT UNTIL sync_verti'event and sync_verti = '1';

--sens du déplacement suivant que nous sommmes 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_Y;

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

--sens du déplacement suivant que nous sommmes à droite ou à gauche de l'écran
                  IF ball_X_pos >= 639 - taille THEN        mouv_X <=-2;
                  ELSIF ball_X_pos <= taille THEN           mouv_X <=2;
                  END IF;

-- Calcul de la prochaine position de X
                  ball_X_pos <= ball_X_pos + mouv_X;

END process mouvement_balle_X;

--obtention de HOR_Z à partir de sync_verti
horloge_axe_Z: process
                  BEGIN
                  WAIT UNTIL sync_verti'event and sync_verti = '1';
                  DIV_Z <= DIV_Z +1;
                           if DIV_Z="11" then HOR_Z <='1';
                           else HOR_Z <='0';
                           end if;
END process horloge_axe_Z;
```

```
--calcul de l'illusion de déplacement suivant l'axe z
mouvement_balle_Z: process
                BEGIN
                WAIT UNTIL HOR_Z'event and HOR_Z = '1';
                        IF taille >= 29 THEN            mouv_Z <= -1;
                        ELSIF taille <= 9 THEN          mouv_Z <= 1;
                        END IF;

                taille <= taille + mouv_Z;

END process mouvement_balle_Z;


END archi;
```

Remarque : ce programme et le précédent, mis au point avec Maxplus+ II conduisent avec la synthèse de Quartus a des temps de propagation des signaux parfois trop importants ; il serait nécessaire de les modifier de manière à resynchroniser les signaux