Universidad Tecnológica Nacional

Técnicas Digitales IV

Trabajo Práctico 2: Controlador VGA en VHDL

Autores:
Ignacio Rodríguez Grandi
Matías Iván Bucca
Jonás Gabriel Ramírez Torres
Federico Elizondo

Supervisor:
Pablo Cayuela
Sergio Olmedo

June 15, 2021

Contents

1 Introduccion

Un monitor tradicional de computador se compone de un tubo de rayos catódicos como elemento fundamental. La pantalla contiene puntos de fósforo rojo, verde y azul que emiten luz cuando son bombardeados por un haz de electrones. La parte trasera contiene un cátodo que se calienta a una temperatura de alrededor de 800°, de forma que emiten tres haces de electrones (uno por cada color) que inciden sobre la pantalla de fósforo. Delante del cátodo se sitúa una rejilla de control que actúa sobre la intensidad de los haces. Estos deben orientarse para que lleguen a cualquier punto de la pantalla. Esta orientación se logra mediante planos de deflexión horizontal y vertical. Adicionalmente una rejilla denominada máscara de sombras garantiza que cada rayo incida sobre los puntos de fósforo de su color. El rápido refresco de la pantalla consigue hacer creer al ojo humano siempre que supere los 30 refrescos por segundos (30Hz), produciendo la impresión de una imagen fija. Si se pretenden representar imágenes en movimiento es recomendable aumentar esta frecuencia hasta los 50 ó 60Hz. En los monitores de computadores esta frecuencia de refresco no es inferior a 60Hz y generalmente superior.

Resumiendo, una pantalla de tubo de rayos catodicos esta formada por:

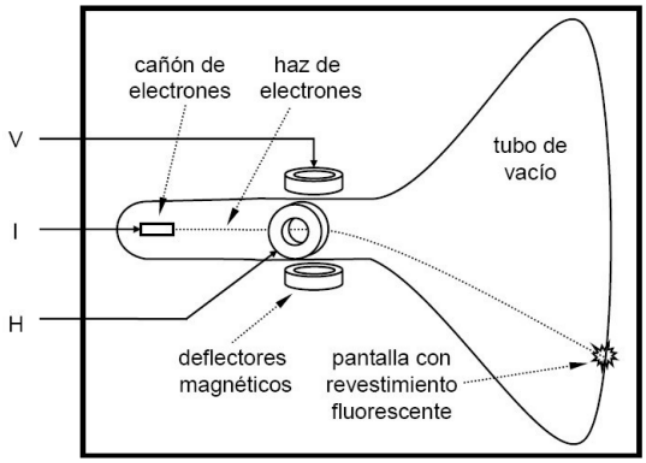

Figure 1: Tubo de rayos catodicos.

- Un tubo de vacío de forma piramidal cuya base está recubierta de un material fluorescente.
- Un filamento, cátodo, que genera el haz de electrones uno por cada color.
- Un par de bobinas deflectoras perpendiculares que permiten modificar la trayectoria de los electrones.

El choque del haz de electrones con el material fluorescente hace que este se ilumine, de manera que dependiendo del tipo de material con el que choca, determina el color que se visualiza. La densidad del haz de electrones determina la intensidad de la luz, y

además la desviación provocada por la bobinas determina el lugar de choque del haz, siendo el tamaño del punto de impacto el que determina la resolución de la pantalla.

2 Señales y temporización de la VGA

Si visualizamos un punto en color en un monitor, no proporciona mucha información para visualizar, sin embargo una línea horizontal de pixeles proporciona más información, pero quizá todavía no sea suficiente. Un frame compuesto por multitud de líneas puede representar una imagen en una pantalla. Un frame de VGA posee normalmente 480 líneas y cada una de ellas 680 pixeles, para conseguir pintar un frame, los circuitos deflectores se encargan de desviar el haz de electrones desde la izquierda a la derecha y de arriba abajo a través de toda la pantalla. Los circuitos de deflexión necesitan dos señales de sincronización para conseguir controlar el inicio y la parada de dichos circuitos en el momento adecuado, y de esta manera permitir que los pixeles se vayan dibujando en la pantalla del monitor.

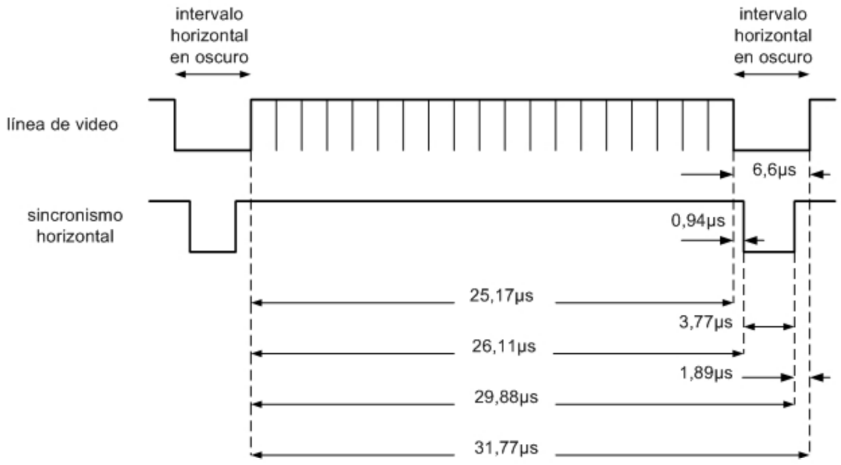

Figure 2: Cronograma de la VGA.

El pulso negativo de la señal sincronismo horizontal (*hsync*), marca el inicio y el final de una línea y asegura que el monitor muestre los pixeles entre los bodes izquierdo y derecho de la parte visible del monitor. Los pixeles reales se envían al monitor en una ventana de 25,17μs.

La señal de sincronismo horizontal (*hsync*) se pone a cero como mínimo 0,94μs después del último pixel y se mantiene a cero 3,77μs. Una nueva línea de pixeles se puede iniciar 1,89μs después de que la señal (*hsync*) ha terminado, es decir, se ha puesto a uno. Como una línea ocupa 25,17μs de los 31,77μs que dura la línea de video, esto significa que en los restantes 6,6μs la pantalla está oscura, siendo este tiempo el que se conoce como horizontal *blanking interval*.

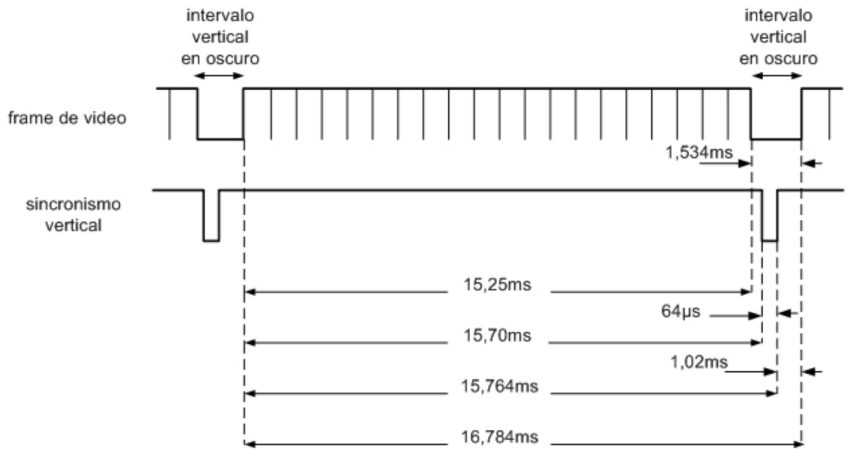

Figure 3: Cronograma de la VGA.

El pulso negativo de la señal sincronismo vertical (*vsync*), marca el inicio y el final de un frame y asegura que se visualice entre el borde superior e inferior y dentro del marco visible del monitor, todas las líneas que lo componen. Las líneas se envían al monitor en una ventana de 15,25ms.

La señal de sincronismo vertical (*vsync*) se pone a cero como mínimo 0,45ms después de la última línea y se mantiene a cero 64μs. La primera línea del siguiente frame se puede iniciar 1,02ms después de que la señal *vsync* ha terminado, es decir, se ha puesto a uno. Como un frame ocupa 15,25ms de los 16,784ms, esto hace que durante 1,534ms la pantalla esté oscura, siendo este tiempo el que se conoce como vertical *blanking interval*.

Para realizar el control de una VGA, necesitamos generar, hsync, es decir el tiempo necesario para generar todos los pixeles de una línea horizontal entera y *vsync*, que es el tiempo necesario para generar todas las líneas de una pantalla entera (frame). Por lo tanto se debe tener en cuenta que los valores se deberán calcular en función de la pantalla (640x480) y la frecuencia (25MHz), lo que significa que 25 Mpixeles se procesan en un segundo.

2.1 Sincronismo Horizontal

Un periodo de una línea horizontal contiene 800 pixeles que se pueden dividir en cuatro regiones:

- **Display:** región en la que se visualizan los pixeles, su longitud es de 640 pixeles.
- **Retrazado:** cuando el pixel vuelve al borde izquierdo, de manera que la señal de video esté deshabilitada (96 pixeles).
- **Borde derecho:** Final de la zona de visualización, la señal de video debe ser deshabilitada (16 pixeles), se suele conocer como *front porch*.
- **Borde izquierdo:** Inicio de la región de visualización, la señal de video debe estar deshabilitada (48pixeles), se suele conocer como *back porch*.

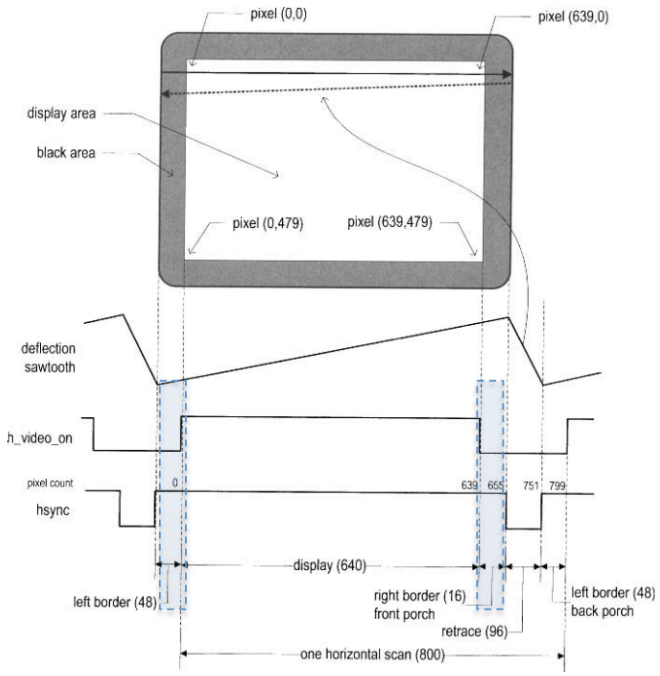

Figure 4: Sincronismo horizontal.

Hay que tener en cuenta que los bordes derecho e izquierdo pueden variar para cada marca de monitor. La señal *hsync* se puede obtener con un contador módulo 800, en la Fig. 4 se observa que se empezará la cuenta en el inicio de la región de visualización, lo que nos va a permitir utilizar el contador como coordenada x, de manera que así conseguiremos la salida *pixel_x*. La señal *hsync* se pone a cero entre los pixel 656 y 751. Igualmente se observa que se utiliza la señal *h_video_on* para indicar que se esta en la region visible (display).

2.2 Sincronismo Vertical

Durante el recorrido vertical se va pasando de una línea a otra y de arriba abajo toda la pantalla, esto se corresponde con el tiempo necesario para refrescar la pantalla entera. Un periodo completo contiene 525 líneas, que se pueden dividir en cuatro regiones:

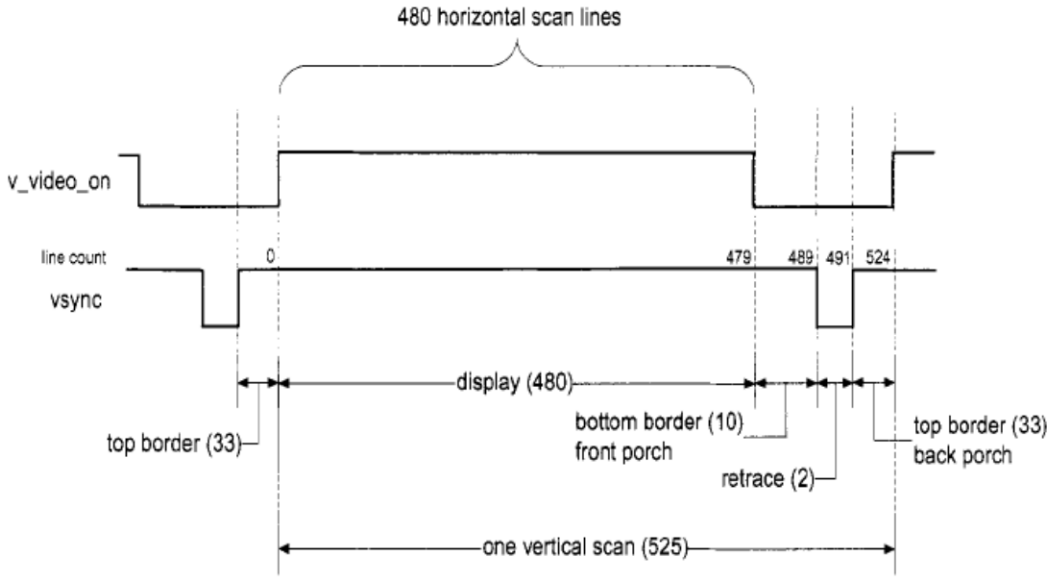

Figure 5: Sincronismo vertizal.

- **Display:** Zona en la que las líneas horizontales se ven, suele tener 480 lineas.
- **Retrazado:** Zona en la cual el pixel debe regresar al inicio de la pantalla, la señal de video debe ser deshabilitada (2 lineas).
- **Borde superior:** Región superior de la pantalla, conocida como back porch, la señal de video debe estar deshabilitada y debe tener 33 lineas.
- **Borde inferior:** : Región inferior de la pantalla, conocida como front porch, la señal de video debe estar deshabilitada y debe tener 10 lineas.

2.3 Cálculo del tiempo de las señales de sincronización

Asumimos que la frecuencia de pixel es de 25MHz, que viene determinado or los parámetros:

- **(pixel) p**: El número de pixeles en una línea horizontal, para una resolución de 640x480, es de 800 pixeles/línea.
- **(lineas) l**: El número de líneas verticales, para una resolución de 640x480, es de 525 líneas/pantalla.
- **(screen) s**: El número de pantallas por segundo, para un funcionamiento sin parpadeo, es de 60 pantallas/s.

  Este parámetro, s, nos indica cuanta veces se debe refrescar la pantalla. Para el ojo humano una sensación de continuidad se consigue a partir de 30 pantallas.

  La frecuencia de los pixeles se puede calcular utilizando estos tres parámetros de la siguiente forma:

$$f = p.l.s \approx 25Mpixel/s$$

3 Controlador de VGA

El esquema mas elemental de un controlador VGA:

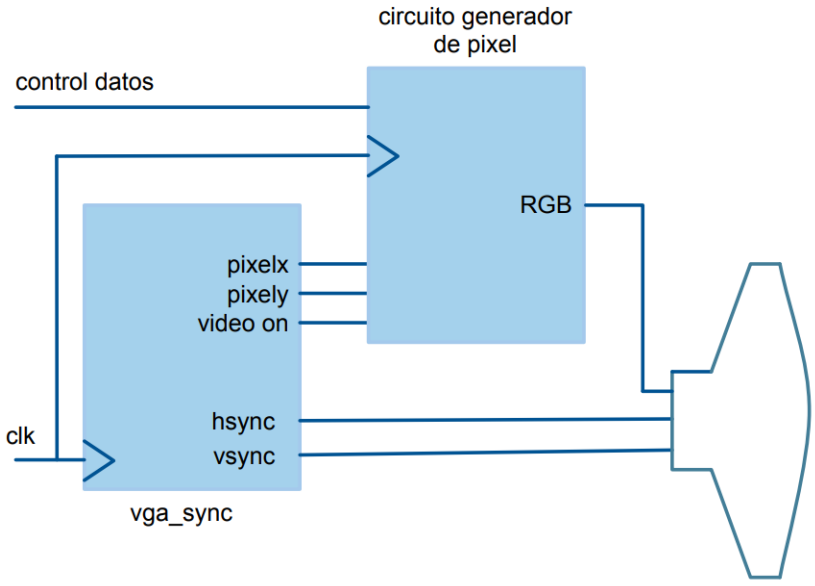

Figure 6: Esquema basico de un controlador VGA.

El bloque llamado *vga_sync* se encarga de generar los tiempos y señales de sincronización, las señales *hsync* y *vsync* se conectan van al conector de salida VGA, las

señales *pixelx* y *pixely* indican la posición relativa en la que se debe pintar el pixel actual, la señal *video on* habilita la visualización.

4 Diseño Top Down

Conocidas las señales, tiempos y pixeles, que componen una VGA, en la Fig. 7 se muestra una propuesta de controlador.

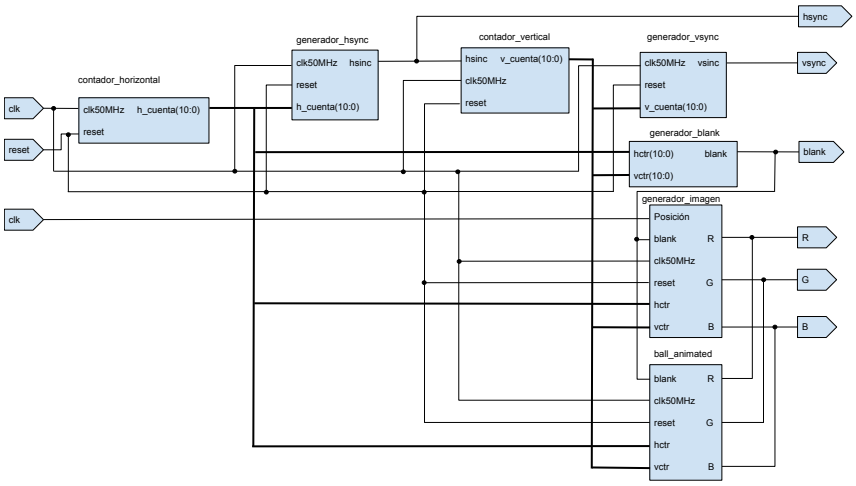

Figure 7: Bloques de un controlador VGA.

En este ejemplo se implementa el control VGA para generar una imagen sencilla, de manera que observando el diseño resulta fácil crear el controlador que se ajuste a nuestras necesidades.

Los bloques que componen el diseño son:

1. **Contador horizontal:** Cuenta los puntos de cada linea horizontal.
2. **Generador *hsync*:** Genera el sincronismo horizontal.
3. **Contador Vertical:** Cuenta las lineas de cada imagen.
4. **Generador *vsync*:** Genera el sincronismo vertical.
5. **Generador *blank*:** Genera los oscurecimientos de pantalla para lineas horizontales y verticales.
6. **Generador de imagen:** Genera la imagen a mostrar, para atacar a las patillas de salida (R, G, B).

4.1 Controlador Horizontal

La frecuencia del reloj principal es de 50MHz, es decir, T_{clk}=20ns, si tenemos en cuenta que según la Fig. 2 una línea horizontal dura 31,77 μs ($T = 1/31,5KHz = 31,746\mu s$). Partiendo del periodo del reloj y del periodo de la línea obtenemos que es necesario un contador de 11bits ($31,746\mu s/20ns = 1587,3$) los bits que son necesarios para poder contar hasta 1587 son 11, ya que 2^{11}=2048.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity contador_horizontal is
 Port ( clk50MHz : in std_logic; -- reloj principal
                  reset : in std_logic; -- reset global
                  h_cuenta : out std_logic_vector (10 downto 0)
 );
end contador_horizontal;
architecture comportamiento of contador_horizontal is
signal h_cuenta_int : integer range 1586 downto 0;
begin
-- convierte h_cuenta_int en entero y se asigna a la salida
h_cuenta <= CONV_STD_LOGIC_VECTOR (h_cuenta_int, 11);
process (clk50MHz,reset,h_cuenta_int)
begin
 if reset = '1' then
        h_cuenta_int <= 0;
 elsif clk50MHz='1' and clk50MHz'event then
-- para saber que se ha llegado a la cuenta 1587
 if h_cuenta_int = 1586 then --ajustar bien la cuenta
        h_cuenta_int <= 0;
 else
        h_cuenta_int <= h_cuenta_int + 1;
 end if;
 end if;
end process;
end comportamiento;
```

4.2 Generador de sincronizmo horizontal

El pulso negativo de la señal sincronismo horizontal (*hsync*), marca el inicio y el final de una línea y asegura que el monitor muestra los pixeles entre los bordes izquierdo y derecho de la parte visible del monitor. Los pixeles reales se envían al monitor en una ventana de 25,17μs.

La señal de sincronismo horizontal (*hsync*) se pone a cero como mínimo 0,94s después del último pixel y se mantiene a cero 3,77μs. Una nueva línea de pixeles se puede iniciar 1,89μs después de que la señal (*hsync*) ha terminado, es decir, se ha puesto a uno. Como una línea ocupa 25,17μs de los 31,77μs que dura la línea de video,

esto significa que en los restantes 6,6μs la pantalla está oscura, siendo este tiempo el que se conoce como horizontal *blanking interval*.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity generador_hsync is
 Port ( h_cuenta : in std_logic_vector (10 downto 0);
                  clk50MHz : in std_logic;
                  reset : in std_logic;
                  hsync : out std_logic);
end generador_hsync;
architecture comportamiento of generador_hsync is
signal h_cuenta_int : integer range 1586 downto 0;
signal hsync_aux, change_h: std_logic;
begin
-- conversión de tipo para trabajar con enteros
h_cuenta_int <= CONV_INTEGER (h_cuenta);
-- asignación a la salida, trabajamos con una señal de salida
-- auxiliar
hsync <= hsync_aux;
-- Biestable de generación de hsync, que se inicializa a nivel uno
process (clk50MHz,reset,change_h)
begin
 if reset = '1' then
        hsync_aux <= '1';
 elsif clk50MHz='1' and clk50MHz'event then
        if change_h = '1' then
                hsync_aux <= not hsync_aux;
        else
                hsync_aux <= hsync_aux;
 end if;
 end if;
end process;
-- Proceso que genera los instantes de cambios de estado de hsync
process (h_cuenta_int)
begin
-- La señal hsync debe cambiar de estado en la cuenta 1.327 y 1.515
-- del contador horizontal. Para ello, la señal change_h debe activarse
-- una valor antes en cada caso.
 if (h_cuenta_int = 1326) or (h_cuenta_int = 1514) then
 change_h <= '1';
 else
 change_h <= '0';
 end if;
end process;
end comportamiento;
```

4.3 Contador Vertical

De acuerdo con la Fig.2 el contador de vertical se debe incrementar con la señal procedente del contador horizontal, y teniendo en cuenta que el $T_{vertical}$=16,784ms, siendo el $T_{horizontal}$=31,746 μs. Podemos calcular la cantidad de bits que debe tener el contador vertical como $16,784ms/31,746\mu s = 528,69$, el contador debe llegar a 529, por lo tanto son necesarios 10bits. (2^{10}=1024).

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity contador_vertical is
 Port ( hsync : in std_logic; -- horizontal sync signal
                  clk50MHz: in std_logic; -- main clock
                  reset : in std_logic; -- global reset
                  v_cuenta : out std_logic_vector (9 downto 0)
);
end contador_vertical;
architecture comportamiento of contador_vertical is
signal v_cuenta_int : integer range 528 downto 0;
signal hsync_s, hsync_t_1, fa_hsync: std_logic;
begin
-- convierte v_cuenta_int en un entero y se asigna a la salida
v_cuenta <= CONV_STD_LOGIC_VECTOR (v_cuenta_int, 10);
-- detector del flanco de subida hsync
process (reset, clk50MHz, hsync_s, hsync_t_1)
begin
if reset = '1' then
        hsync_s <= '0';
        hsync_t_1 <= '0';
elsif clk50MHz = '1' and clk50MHz'event then
        hsync_t_1 <= hsync_s;
        hsync_s <= hsync;
end if;
fa_hsync <= hsync_s and not hsync_t_1;
end process;
-- Contador Vertical
process (fa_hsync,reset,v_cuenta_int,clk50MHz)
begin
 if reset = '1' then
        v_cuenta_int <= 0;
 elsif clk50MHz='1' and clk50MHz'event then
        if fa_hsync = '1' then
                if v_cuenta_int = 528 then
                        v_cuenta_int <= 0;
                else
                        v_cuenta_int <= v_cuenta_int + 1;
 end if;
 end if;
```

```
 end if;
end process;
end comportamiento;
```

4.4 Generador de sincronismo vertical

La señal de sincronismo vertical (*vsync*)se pone a cero como mínimo 0,45ms después de la última línea y se mantiene a cero 64μs. La primera línea del siguiente frame se puede iniciar 1,02ms después de que la señal *vsync* ha terminado, es decir, se ha puesto a uno. Como un frame ocupa 15,25ms de los 16,784ms, esto hace que durante 1,534ms la pantalla esté oscura, siendo este tiempo el que se conoce como vertical *blanking interval*.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity generador_vsync is
 Port ( v_cuenta : in std_logic_vector (9 downto 0);
                  clk50MHz : in std_logic;
                  reset : in std_logic;
                  vsync : out std_logic);
end generador_vsync;

architecture comportamiento of generador_vsync is
signal v_cuenta_int : integer range 524 downto 0;

begin
-- Convierte a entero el vector de entrada v_cuenta
v_cuenta_int <= CONV_INTEGER (v_cuenta);
-- vsync generation flip-flop, which is initialized to one state
process (clk50MHz,reset,v_cuenta_int)
begin
 if reset = '1' then
        vsync <= '1';
 elsif clk50MHz='1' and clk50MHz'event then
        if (v_cuenta_int >= 500) and (v_cuenta_int <= 502) then
                vsync <= '0';
        else
                vsync <= '1';
        end if;
 end if;
end process;
end comportamiento;
```

4.5 Generador de pantalla oscura

Para que la pantalla se quede en oscuro, es decir, para que no se pinte pixel en una línea ni una línea en un frame, son necesarios los ($BLANK$) tanto horizontal como vertical.

Una imagen de 640x480 tiene una frecuencia estándar de 31,5KHz que se corresponde con 31,746μs (periodo horizontal). Un tiempo máximo de 25,17μs corresponde a los puntos que hay que pintar en cada línea, se puede obtener el tiempo del *blank horizontal* $= 31,746 - 25,17 = 6,576\mu$s.

La frecuencia vertical estándar es de 60Hz, que equivale a un periodo de 16,666ms (periodo vertical). Como máximo una imagen (frame) tiene una tiempo de 15,25ms, se puede calcular el tiempo del *blank interval* $= 16,666 - 15,25 = 1,534$ms.

Los intérvalos de blank vertical y horizontal dependen de los contadores verticales y horizontales respectivamente, en concreto desde el valor 1.258 hasta 1586 del contador horizontal para el blank horizontal, y desde el valor 480 hasta el 524 del contador vertical para el *blank vertical*.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity generador_blank is
 Port ( hctr : in std_logic_vector (10 downto 0);
                  vctr : in std_logic_vector (9 downto 0);
                  blank : out std_logic
 );
end generador_blank;

architecture Behavioral of generador_blank is
signal hctr_int : integer range 1586 downto 0;
signal vctr_int : integer range 524 downto 0;
begin
-- conversión de tipos de los contadores a entero
hctr_int <= CONV_INTEGER (hctr);
vctr_int <= CONV_INTEGER (vctr);
-- Circuito combinacional que genera el blank horizontal y el vertical
-- El valor de blank se corresponde con un cero
process (hctr_int,vctr_int)
begin
if ((hctr_int >= 1258) and (hctr_int <= 1586)) or
((vctr_int >= 480) and (vctr_int <= 524)) then
 blank <= '0';
 else
 blank <= '1';
 end if;
end process;
end Behavioral;
```

4.6 Generador de imagen

Genera una imagen de 640x480 pixeles que consiste en una barras verticales u horizontales (según el nivel de la entrada "posición") de diferentes colores (figura 8). Los

colores dependen de los valores de R, G y B, y se utilizan 8 bits para representar esta salida. Se utilizan 3 bits para R, 3 bits para G y 2 bits para B, pudiendo generar así un máximo de 256 colores.

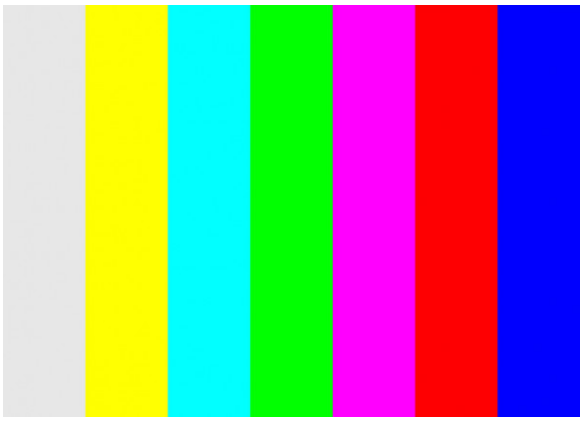

Figure 8: Imagen de barras a generar.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity image_generator is
 Port (hctr : in std_logic_vector (10 downto 0);
                 vctr : in std_logic_vector (9 downto 0);
                 blank : in std_logic; -- blank interval signal
                 posicion : in std_logic; --horizontal a vertical
                 clk50MHz : in std_logic; -- main clock
                 reset : in std_logic; -- global reset
                 RGB : out std_logic_vector (7 downto 0)); --Colour signal

end image_generator;
architecture Behavioral of image_generator is
signal hctr_int : integer range 1586 downto 0;
signal vctr_int : integer range 524 downto 0;
signal colorv: std_logic_vector (7 downto 0);
signal colorh: std_logic_vector (7 downto 0);
signal color: std_logic_vector (7 downto 0);
begin
hctr_int <= CONV_INTEGER (hctr);
vctr_int <= CONV_INTEGER (vctr);
-- utilizamos biestables de salida para evitar posibles Glitches
-- Iniiciaizamos los biestables a cero
process (clk50MHz,reset,color)
begin
 if reset = '1' then
 RGB <= "00000000";
 elsif clk50MHz='1' and clk50MHz'event then
 RGB <= color;
```

```
 end if;
end process;
-- Colores obtenidos en función de R G B (8 bits)
-- Circuito combinacional que genera los colores de cada franja
-- en función de la posición horizontal de cada punto
-- franja vertical blanca de la izda
colorv <= "11111111" when ((hctr_int >= 0) and (hctr_int < 158)
and (blank = '1') and (posicion='1'))--blanco
else
-- aquí comienza la imagen de 640 x 480
-- que consiste en 7 barras verticales de diferentes colores
"11111100" when ((hctr_int >= 158) and (hctr_int < 316) and
(blank = '1')and (posicion='1')) else --amarillo
"00011111" when ((hctr_int >= 316) and (hctr_int < 474) and
(blank = '1') and (posicion='1'))else --cyan
"00011100" when ((hctr_int >= 474) and (hctr_int < 632) and
(blank = '1')and (posicion='1')) else --verde
"11100011" when ((hctr_int >= 632) and (hctr_int < 790) and
(blank = '1')and (posicion='1')) else --rosa
"11100000" when ((hctr_int >= 790) and (hctr_int < 948) and
(blank = '1') and (posicion='1'))else --rojo
"00000011" when ((hctr_int >= 948) and (hctr_int < 1106) and
(blank = '1')and (posicion='1')) else --azul
"00000000" when ((hctr_int >= 1106) and (hctr_int <= 1257) and
(blank = '1')and (posicion='1')) else --negro
"00000000"; -- Intérvalos blank (blank = 0)
colorh <=
"11111111" when ((vctr_int >= 0) and (vctr_int < 60)
and (blank = '1') and (posicion='0')) else --blanco
"11111100" when ((vctr_int >= 60) and (vctr_int < 120)
and (blank = '1') and (posicion='0')) else --amarillo
"00011111" when ((vctr_int >= 120) and (vctr_int < 180)
and (blank = '1') and (posicion='0')) else --cyan
"00011100" when ((vctr_int >= 180) and (vctr_int < 240)
and (blank = '1') and (posicion='0')) else --verde
"11100011" when ((vctr_int >= 240) and (vctr_int < 300)
and (blank = '1') and (posicion='0')) else --rosa
"11100000" when ((vctr_int >= 300) and (vctr_int < 360)
and (blank = '1') and (posicion='0')) else --rojo
"00000011" when ((vctr_int >= 360) and (vctr_int < 420)
and (blank = '1') and (posicion='0')) else --azul
"00000000" when ((vctr_int >= 420) and (vctr_int <= 480)
and (blank = '1') and (posicion='0')) else --negro
"00000000"; -- Intérvalos blank (blank = 0)

color <= colorv or colorh;
end Behavioral;
```

4.7 Bola en movimiento

Para finalizar y ayudar en la realización del proyecto, se implementa un bloque de una pelota que deberá moverse por la pantalla y generar un rebote random cuando esta se encuentre con un borde. El color de la bola deberá ser el inverso de la barra por la que esté pasando.

```
library ieee;
use ieee.std_logic_1164.all ;
use ieee.numeric_std.all;

entity ball_animate is

port(
                clk, reset : std_logic;
                video_on: in std_logic;
                pixel_x : in std_logic_vector (10 downto 0) ;
                pixel_y : in std_logic_vector (9 downto 0) ;
                graph_rgb : out std_logic
);
end ball_animate;

architecture arch of ball_animate is
signal refr_tick : std_logic;
-- x, y coordinates (0,0) to (639,479)
signal pix_x: unsigned (10 downto 0);
signal pix_y: unsigned (9 downto 0);
constant MAX_Y: integer :=480;
constant MAX_X: integer :=1257;
-------------------------------------------------
-- square ball
-------------------------------------------------
constant BALL_SIZE: integer:=16; -- 8
-- ball left, right boundary
signal ball_x_l: unsigned (10 downto 0) ;
signal ball_x_r: unsigned (10 downto 0) ;
-- ball top , bottom boundary
signal ball_y_t , ball_y_b : unsigned (9 downto 0) ;
-- reg to track left , top boundary
signal ball_x_reg , ball_x_next : unsigned (10 downto 0) ;
signal ball_y_reg , ball_y_next : unsigned (9 downto 0) ;
-- reg to track ball speed
signal x_delta_reg , x_delta_next : unsigned(10 downto 0) ;
signal y_delta_reg , y_delta_next : unsigned(9 downto 0) ;

-- ball velocity can be pos or neg
constant BALL_V_P: unsigned (9 downto 0):=to_unsigned (2,10);
constant BALL_V_N: unsigned (9 downto 0):=unsigned (to_signed (-2,10));
constant BALL_V_P_X: unsigned (10 downto 0):=to_unsigned (2,11);
constant BALL_V_N_X: unsigned (10 downto 0):=unsigned (to_signed (-2,11));
-------------------------------------------------
```

```
-- round ball image ROM
------------------------------------------------
type rom_type is array (0 to 7) of std_logic_vector(0 to 15);
-- ROM definition
constant BALL_ROM: rom_type :=
(
        "0000111111110000", --    ********
        "0011111111111100", --   **********
        "1111111111111111", --  ************
        "1111111111111111", --  ************
        "1111111111111111", --  ************
        "1111111111111111", --  ************
        "0011111111111100", --   **********
        "0000111111110000"  --    ********
);
signal rom_addr , rom_col : unsigned (3 downto 0) ;
signal rom_data: std_logic_vector (15 downto 0) ;
signal rom_bit : std_logic;
------------------------------------------------
-- object output signals
------------------------------------------------
signal sq_ball_on, rd_ball_on: std_logic;
-- signals for random number
signal count_i      : std_logic_vector (9 downto 0);
signal feedback     : std_logic;

begin
------------------------------------------------
--random number
------------------------------------------------
feedback <= not(count_i(9) xor count_i(5));

process (reset, clk)
          begin
          if (reset = '1') then
                        count_i <= (others=>'0');
          elsif (rising_edge(clk)) then
                        count_i <= count_i(8 downto 0) & feedback;
          end if;
end process;

-- registers
process (clk,reset)
begin
        if reset='1' then
                ball_x_reg <= (others=>'0');
                ball_y_reg <= (others=>'0');
                x_delta_reg <= ("00000000100");
                y_delta_reg <= ("0000000100");
        elsif (clk'event and clk='1') then
                ball_x_reg <= ball_x_next ;
                ball_y_reg <= ball_y_next ;
```

```
                x_delta_reg <= x_delta_next ;
                y_delta_reg <= y_delta_next;
    end if ;
end process;
pix_x <= unsigned(pixel_x);
pix_y <= unsigned(pixel_y);
--refr_tick: 1-clock tick asserted at start of v-sync i.e.,
when the screen is refreshed (60 Hz)
refr_tick <= '1' when (pix_y = 481) and (pix_x = 0) else
'0';
-----------------------------------------------------
-- square ball
-----------------------------------------------------
-- boundary
ball_x_l <= ball_x_reg;
ball_y_t <= ball_y_reg;
ball_x_r <= ball_x_l + BALL_SIZE - 1;
ball_y_b <= ball_y_t + BALL_SIZE - 9;
-- pixel within ball
sq_ball_on <=
'1' when (ball_x_l <= pix_x) and (pix_x <= ball_x_r) and
(ball_y_t <= pix_y) and (pix_y <= ball_y_b) else
'0';
-- map current pixel location to ROM addr/col
rom_addr <= pix_y(3 downto 0) - ball_y_t(3 downto 0);
rom_col <= pix_x(3 downto 0) - ball_x_l(3 downto 0);
rom_data <= BALL_ROM(to_integer(rom_addr));
rom_bit <= rom_data(to_integer(rom_col));
-- pixel within ball
rd_ball_on <= '1' when (sq_ball_on='1') and (rom_bit='1') else
'0';

--ball rgb output
--ball_rgb <= "11100000";
--new ball position
ball_x_next <=  ball_x_reg + x_delta_reg when refr_tick= '1'
else ball_x_reg;
ball_y_next <=  ball_y_reg + y_delta_reg when refr_tick= '1'
else ball_y_reg;
--new ball velocity
process (x_delta_reg , y_delta_reg , ball_y_t ,
ball_x_l , ball_x_r ,ball_y_b,feedback)
begin
        x_delta_next <= x_delta_reg;
        y_delta_next <= y_delta_reg ;
        if ball_y_t < 1 then -- reach top
                y_delta_next <= BALL_V_P;
                if feedback='0' then
                        x_delta_next <= BALL_V_P_X;
                else
                        x_delta_next <= BALL_V_N_X;
                end if;
```

```
        elsif ball_y_b > (MAX_Y - 1) then -- reach bottom
                y_delta_next <= BALL_V_N;
                if feedback='0' then
                        x_delta_next <= BALL_V_P_X;
                else
                        x_delta_next <= BALL_V_N_X;
                end if;
        elsif ball_x_l < 1 then -- reach left
                x_delta_next <= BALL_V_P_X; -- bounce back
                if feedback='0' then
                        y_delta_next <= BALL_V_P;
                else
                        y_delta_next <= BALL_V_N;
                end if;
        elsif ball_x_r > (MAX_X - 1) then -- reach right
                x_delta_next <= BALL_V_N_X;
                if feedback='0' then
                        y_delta_next <= BALL_V_P;
                else
                        y_delta_next <= BALL_V_N;
                end if;
        end if;
end process;
-----------------------------------------------------------------
--rgb multiplexing circuit
-----------------------------------------------------------------
process(video_on,rd_ball_on)
begin
        if video_on= '0' then
                graph_rgb <= '0'; --blank
        else
                if rd_ball_on= '1' then
                        graph_rgb <= '1';
                else
                        graph_rgb <= '0';
                end if;
        end if;
end process;
end arch;
```

5 Videos de funcionamiento

Video 1: Cambio de posición de barras **Clic aquí para ver el video.**
Video 2: Rebotes aleatorios **Clic aquí para ver el video.**

References

[1] FPGA Prototyping by VHDL Examples (Xilinx Spartan 3 Version) - Pong P. Chu

[2] Diseño de un controlador VGA en VHDL Angel Grediaga