

UTN FRC
INGENIERÍA ELECTRÓNICA

TÉCNICAS DIGITALES I : PROYECTO
FINAL DE LA MATERIA

“Consola de Videojuegos : PONG”

INTEGRANTES:
CASTRO, CRISTIAN
LIPARI, SERGIO
ZALAZAR, FACUNDO

Curso 3R3

2 de diciembre de 2013

Índice general

0.1. Introducción:	2
0.2. El juego:	2
0.2.1. Un poco de historia:	2
0.3. Nuestras herramientas:	4
0.4. El código:	5
0.4.1. Diagrama de bloques:	5
0.4.2. Código n°1: Sincronismo VGA	7
0.4.3. Código n°2: Animacion	9
0.4.4. Código n°3 : Siete_segmentos.	14
0.4.5. Código n°4 y n°5 : Texto, font ROM.	14
0.4.6. Código n°6: Sonido	17
0.4.6.1. Teoria : PWM	17
0.4.6.2. Código.	17
0.4.7. Código n°7 : pckg	18
0.4.8. Código n°8 : Cabecera	19
0.5. Controles.	21
0.6. Conclusión	22

0.1. INTRODUCCIÓN:

0.1. Introducción:

En el presente informe se explicará el desarrollo del proyecto final de la materia Técnicas Digitales 1, correspondiente a Ingeniería Electrónica de la UTN Facultad Regional Córdoba. El mismo consiste en la implementación de un juego clásico, el primero a la venta en el año 1972 por la firma Atari, llamado “Pong”. Para ello se usará netamente programación en VHDL y luego la implementación en una FPGA de la firma Altera. Se verá en una pantalla mediante la salida VGA de la placa.

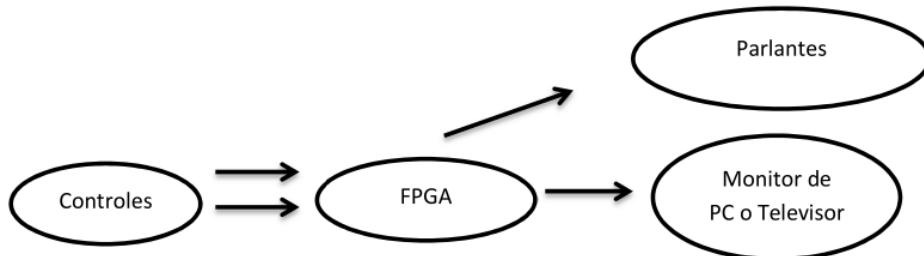


Figura 1: Diagrama simplificado del proyecto.

0.2. El juego:

0.2.1. Un poco de historia:

Pong (o Tele-Pong) fue un videojuego de la primera generación de videoconsolas publicado por Atari, creado por Nolan Bushnell y lanzado el 29 de noviembre de 1972. Pong está basado en el deporte de tenis de mesa (o ping pong). La palabra Pong es una marca registrada por Atari Interactive, mientras que la palabra genérica “pong” es usada para describir el género de videojuegos “paleta y pelota”. La popularidad de Pong dio lugar a una demanda de infracción de patentes y ganada por parte de los fabricantes de Magnavox Odyssey, que poseía un juego similar.

Pong es un juego de deportes en dos dimensiones que simula un tenis de mesa. El jugador controla en el juego una paleta moviéndola verticalmente en la parte izquierda de la pantalla, y puede competir tanto contra un oponente controlado por computadora, como con otro jugador humano que controla una segunda paleta en la parte opuesta. Los jugadores pueden usar las paletas para pegarle a la pelota hacia un lado u otro. El objetivo consiste en que uno de los jugadores consiga más puntos que el oponente al finalizar el juego. Estos puntos se obtienen cuando el jugador adversario falla al devolver la pelota.

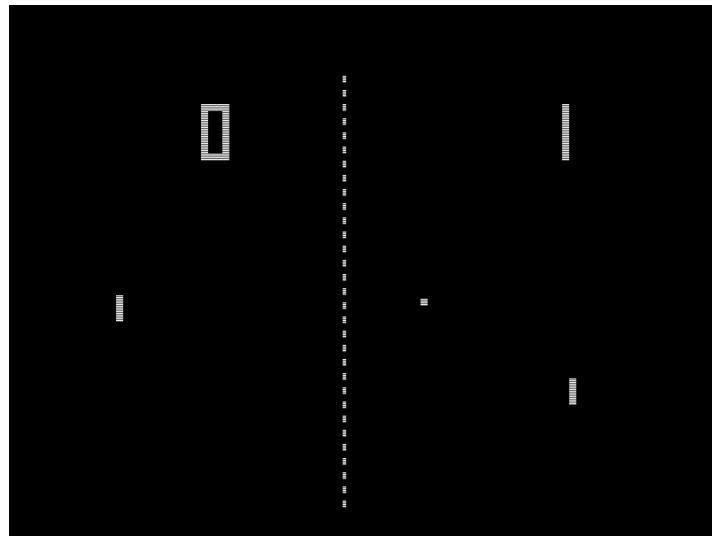


Figura 2: Pong “original”, Atari 1972

Nuestra versión:

Hemos realizado muchas modificaciones al juego, haciendolo mas interesante y emocionante. La versión realizada consiste en los siguiente:

Versión de 1 jugador:

- Versión en colores.
- 5 vidas, las cuales se van descontando cuando la bola sale de la pantalla, por detras de la paleta.
- Se va sumando de a 10 o 30 puntos, depende como toque la bola la paleta.
- Se tiene 2 contadores de puntaje, uno que muestra el puntaje maximo alcanzado en el modo un jugador, y otro que contiene el puntaje actual. Si el actual es mayor que el maximo, se ira actualizando.
- Al ir logrando mayor puntaje, el jugador se encuentra que la bola va aumentando su velocidad.
- La paleta va cambiando de color a medida que la bola impacta con la paleta.

Versión de 2 jugadores:

- Versión en colores.
- La modalidad consiste en ver quien llega primero a los 1000 puntos, los puntos se suman al tocar las barras.
- La paleta va cambiando de color a medida que la bola impacta con la paleta.
- Al ir logrando mayor puntaje, ambos jugadores se encuentran que la bola va aumentando su velocidad.

Además, cuenta con una pantalla de bienvenida en la cual aparecen los integrantes del proyecto, el logo de la facultad y el nombre del juego; y tambien sonido mediante PWM.

0.3. NUESTRAS HERRAMIENTAS:

Para realizar el videojuego, optamos por adquirir una FPGA propia, esta es la Altera DE1.

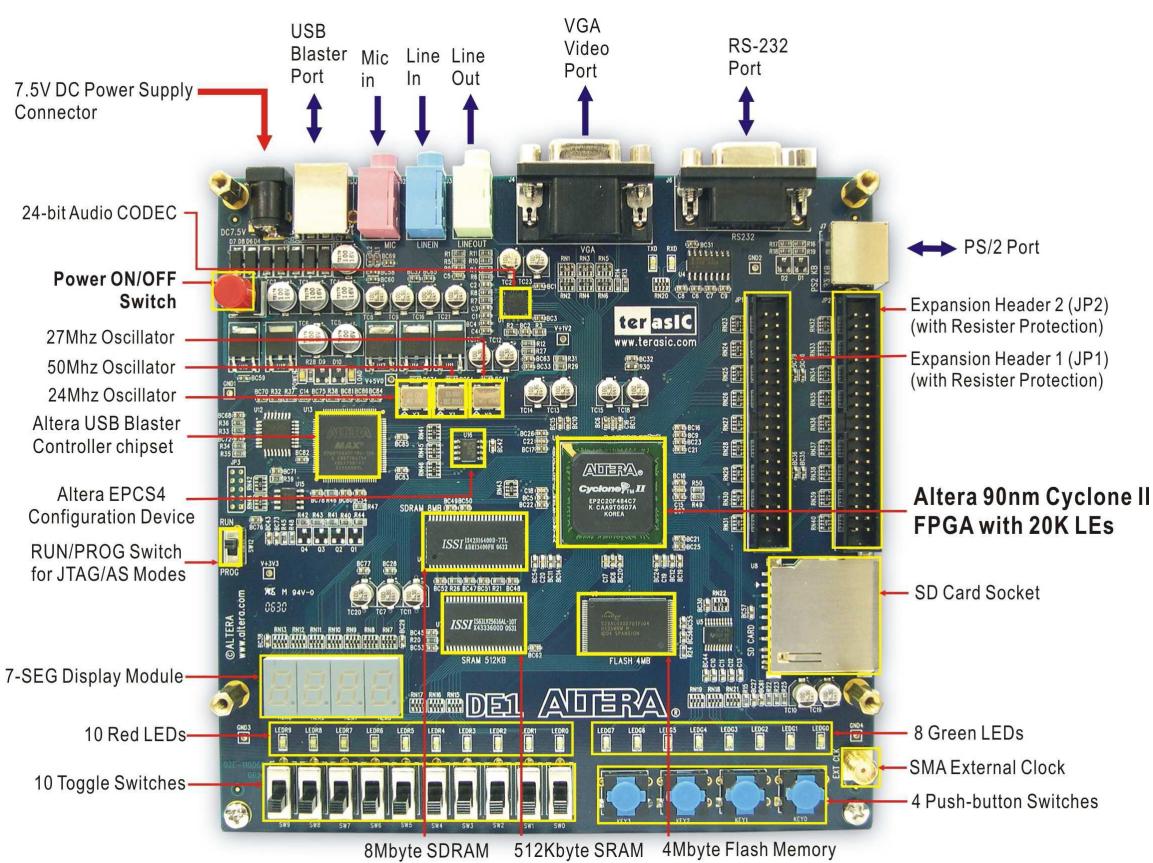


Figura 3: Altera DE1 y sus principales características.

Por lo tanto nuestro programa para programar sera el Quartus II version 13.1.

Los joysticks que usamos son los clásicos del Sega Génesis, solamente usamos sus carcasas, ya que el circuito interno lo realizamos nosotros.

0.4. EL CÓDIGO:



Figura 4: Control similar al empleado.

0.4. El código:

Se empezara esta sección mediante un diagrama que relaciona los codigos VHDL realizados. Luego, se explicaran las partes principales de cada uno de ellos. Al final del documento se encuentran todos los codigos completos.

0.4.1. Diagrama de bloques:

El proyecto no tiene un unico archivo VHDL que maneja ambos juegos.

La programación en Quartus se realizo con 7 archivos VHDL que van instanciados mediante otro archivo VHDL al que denominamos “cabecera”. En la figura 5 se muestra la interconección entre ellos, y que funciones realiza cada uno.

0.4. EL CÓDIGO:

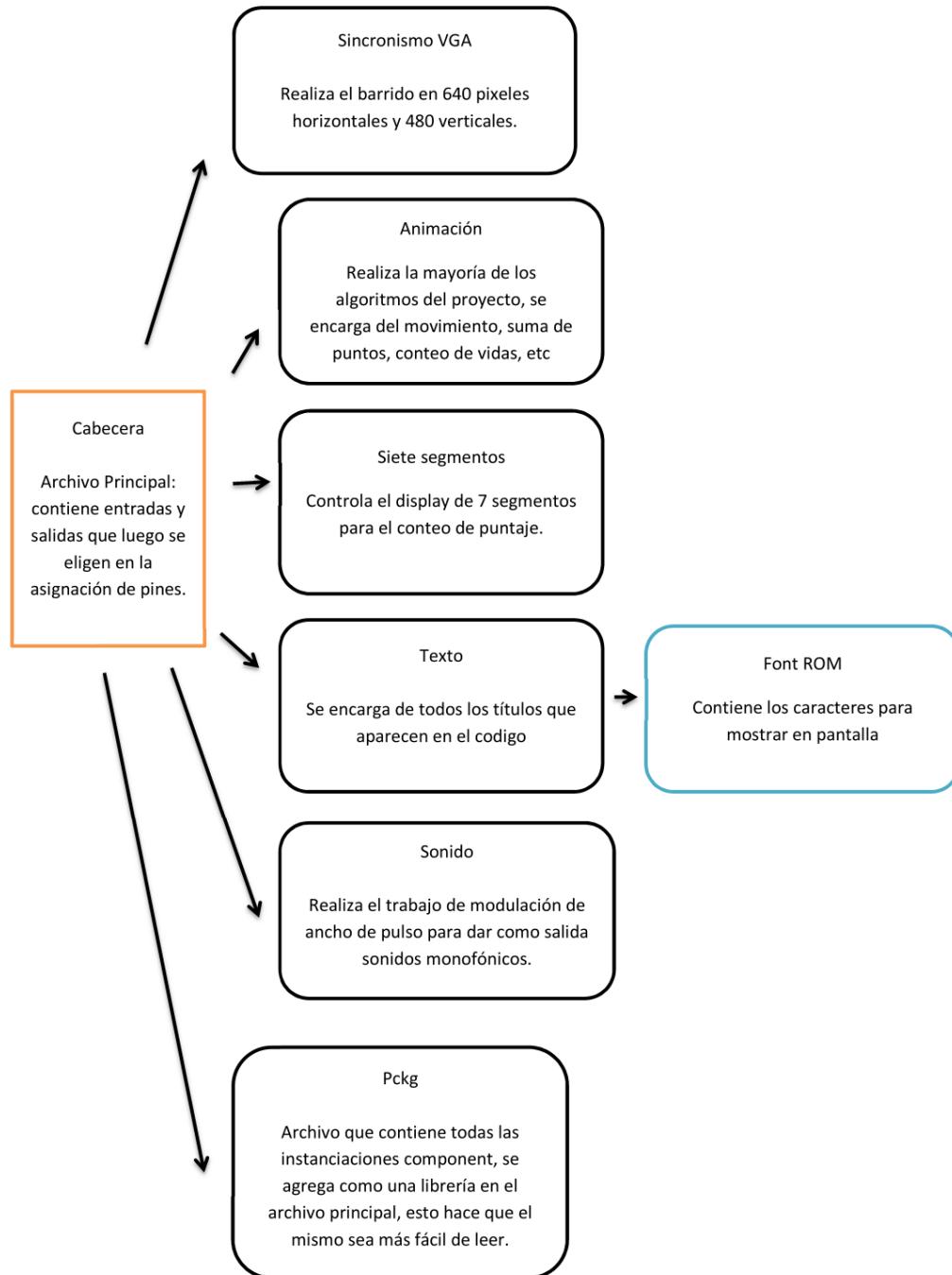


Figura 5: Estructura basica de los codigos.

0.4. EL CÓDIGO:

0.4.2. Código nº1: Sincronismo VGA

Este código contiene entradas:

- **clock**, reloj de 50 MHz.
- **reset**, reiniciar pantalla.

Sus salidas son :

- **en_tick**, que es el resultado de bajar la frecuencia del clock de la placa, de 50 Mhz, a solo 60 Hz, que es la tasa de refresco de la pantalla.
- **h_sync** es '1' cuando el contador que realiza el barrido horizontal esta efectivamente entre 0 y 640 (en realidad 2 y 638, para evitar errores).

De modo similar,

- **v_sync** es '1' cuando el contador que realiza el barrido vertical esta entre 0 y 480 (2 y 478).
- **pixel_x** y **pixel_y**, son vectores de 10 bits que dan la posicion actual en el eje x y en el eje y, respectivamente, gracias a esto, podemos realizar la animación.

```

entity sincronismo_vga is
  port(
    -- clock y reset
    fpga_50mhz_clk: in  std_logic;
    fpga_reset      : in  std_logic;
    -- Sync Horizontal y Vertical
    h_sync          : out std_logic;
    v_sync          : out std_logic;
    -- Habilitacion de Video
    video_on        : out std_logic;
    en_tick         : out std_logic;
    -- Pixels de Pantalla
    pixel_x         : out std_logic_vector (9 downto 0);
    pixel_y         : out std_logic_vector (9 downto 0)
  );
end sincronismo_vga;

```

0.4. EL CÓDIGO:

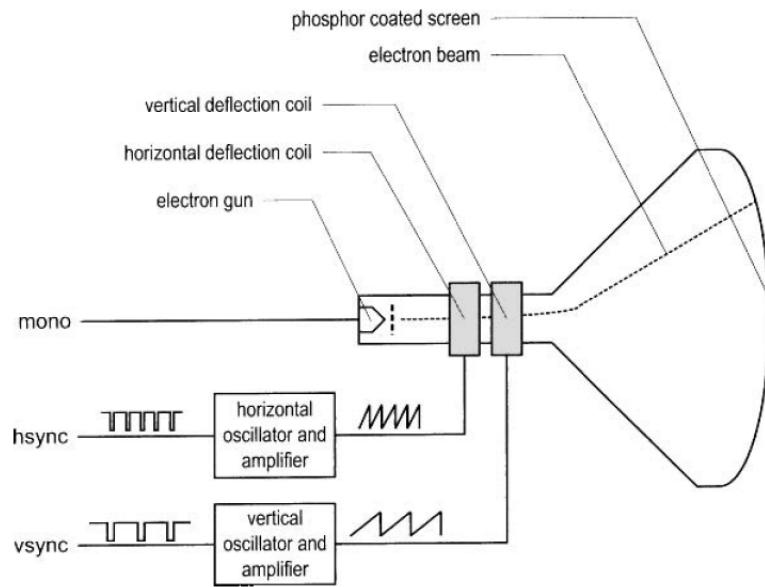


Figura 6: Funcionamiento de las salidas de sincronismo en un monitor CRT. Esto tambien se aplica a otro tipo de monitor (LED, LCD)

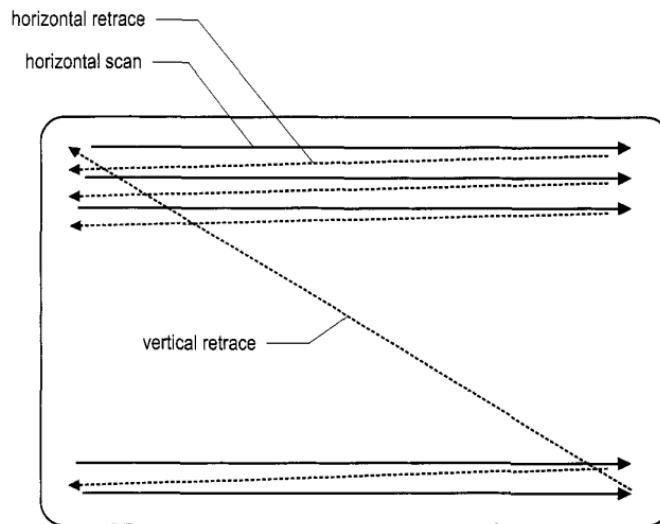


Figura 7: Patrón de barrido vertical y horizontal.

0.4. EL CÓDIGO:

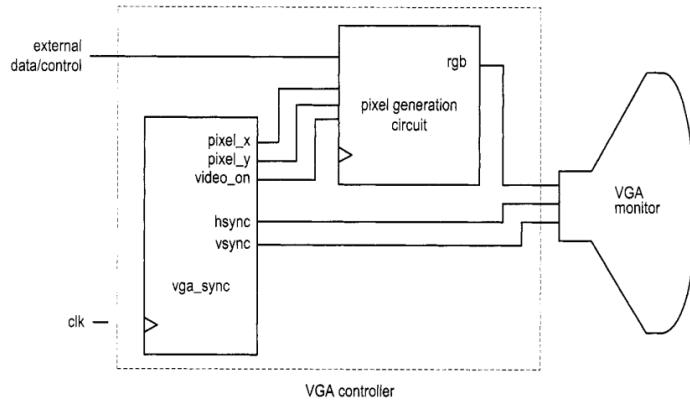


Figura 8: Diagrama de bloques simplificado para el controlador VGA.

0.4.3. Código n°2: Animacion

El código con las siguientes entradas:

- **clk** : clock de 50 MHz
- **on_off** : Enciende o apaga el videojuego.
- **reset** : Si esta en uno, se muestra la pantalla de bienvenida, ademas se reinician los contadores de puntajes (no el maximo) y las vidas.
- **multiplayer** : Si vale 0 el juego es para 1 jugador, si vale 1 es para 2 jugadores.
- **entrada1, entrada2** : Joystick 1 y Joystick 2 respectivamente.
- **pixel_x, pixel_y** : explicadas anteriormente, aqui son entradas que serviran para ubicar los elementos en pantalla.

Salidas:

- **graph_rgb** : Valores de salida para los colores en pantalla (R = ROJO, G = VERDE, B = AZUL)
ej: `graph_rgb = "100"` -> Rojo
- **bandera_1, bandera_2** : salidas que advierten determinadas situaciones (juego terminado, etc)
- **salida_un_jugador_x** : salidas para el contador, 4 de esos bits van para el contador que se ve en el display de 7 segmentos.
Los otros 4 van para el valor de puntaje maximo que aparece en pantalla.
- **salida_dos_jugadores_x** : salidas para ambos puntajes que se ven en pantalla, del bit 0 al 3 para jugador 1 y del 4 al 7 para el jugador 2.

0.4. EL CÓDIGO:

```

--Entidad
entity animacion is
  port(
    clk, on_off, reset, multiplayer : in std_logic;
    entrada1:  in std_logic_vector (1 downto 0);
    entrada2:  in std_logic_vector(3 downto 0);
    pixel_x:   in std_logic_vector(9 downto 0);
    pixel_y:   in std_logic_vector(9 downto 0);
    graph_rgb: out std_logic_vector(2 downto 0);
    bandera_1 : out std_logic;
    bandera_2 : out integer range 0 to 10;

    salida_un_jugador_0, salida_un_jugador_1, salida_un_jugador_2,
    salida_un_jugador_3, salida_un_jugador_4, salida_un_jugador_5,
    salida_un_jugador_6, salida_un_jugador_7,
    salida_dos_jugadores_0, salida_dos_jugadores_1, salida_dos_jugadores_2,
    salida_dos_jugadores_3, salida_dos_jugadores_4, salida_dos_jugadores_5,
    salida_dos_jugadores_6, salida_dos_jugadores_7 : out integer range 0 to 10; --salidas
  );
  --contador
  vidas : out std_logic_vector (4 downto 0);
  enable : out std_logic
);
end animacion;

```

Una vez declarada la arquitectura, se definen señales que utilizaremos en el programa. Luego, se definen los componentes principales que iran en la pantalla. Tenemos la barra para el jugador 1, la barra para el jugador 2, la bola y tambien la pared que esta disponible para la version de 1 jugador. Pero, sus caracteristicas son distintas, no todas se mueven del mismo modo, ni tampoco tienen la misma forma.

```

-- paleta derecha
-----
-- bar left, right boundary
constant BAR2_X_L: integer:=600;
constant BAR2_X_R: integer:=610;
-- bar top, bottom boundary
signal bar2_y_top, bar2_y_btn: unsigned(9 downto 0);
constant BAR2_Y_SIZE: integer:=72;
-- reg to track top boundary (x position is fixed)
signal bar2_y_position, bar2_y_next: unsigned(9 downto 0);
-- bar moving velocity when the button are pressed
shared variable BAR2_Move: integer:=4;

```

La paleta derecha se declara de modo “estatico” por asi decirlo, contiene una distancia fija x (**BAR2_X_R** - **BAR2_X_L** = 10) y un largo de 72 pixeles en vertical. **bar2_y_top** y **bar2_y_btn** son los valores superiores e inferiores de los pixeles de la barra. La señal **BAR2_Move** define la velocidad con la cual se mueve la misma.

El funcionamiento es el siguiente, cuando se da la orden en el joystick que la barra baje, el clock de 60 Hz valga uno, y el valor inferior sea menor a (480-velocidad de la barra) se movera hacia abajo, dado que no tiene que moverse saliendo de la pantalla, se impone esta condicion. Algo similar sucede para el movimiento hacia arriba.

0.4. EL CÓDIGO:

```

-----  

-- Posicion de la paleta 2 al tocar los controles  

-----  

process(clk, reset)
begin
  if (reset= '1') then
    bar2_y_position <= (others=>'0');
  elsif(rising_edge(clk)) then
    if(refr_tick = '1') then -- Mientras se refresca la pantalla a 60Hz
      if(entrada2 = "1110" and bar2_y_btn <(MAX_Y-BAR2_Move)) then
        bar2_y_position <= bar2_y_position + BAR2_Move; -- bar move down
      elsif(entrada2 = "1100" and bar2_y_top > BAR2_Move) then
        bar2_y_position <= bar2_y_position - BAR2_Move; -- bar move up
      end if;
    else
      bar2_y_position <= bar2_y_position; -- no move
    end if;
  end if;
end process;

```

De esta forma tambien se define la pared que existe en el modo 1 jugador. La diferencia es que esta no se mueve, por lo tanto se tiene la definicion y luego al calcular el movimiento de la bola se usa la definicion, como se vera mas adelante.

Para la bola, es un caso distinto. Algo tan simple como un objeto redondeado debe definirse y tratarse de un modo muy diferente. Para ello se emplea un “bitmap”, es decir, definir en este caso la bola, como un patron que defina la bola como tal, esto se realiza de este modo:

```

-----  

-- round ball image ROM  

-----  

type rom_type is array (0 to 7) of std_logic_vector (0 to 7);
-- ROM definition
constant BALL_ROM: rom_type :=
(
  "00111100", -- ****
  "01111110", -- ******
  "11111111", -- ******
  "11111111", -- ******
  "11111111", -- ******
  "11111111", -- ******
  "01111110", -- *****
  "00111100" -- ****
);
  

signal rom_addr, rom_col: unsigned(2 downto 0);
signal rom_data: std_logic_vector(7 downto 0);
signal rom_bit: std_logic;

```

Aqui se observa como ese arreglo de vectores, define mediante ceros y unos, la forma que tiene la bola; es decir que ya deja de ser un bloque, y pasa a ser una figura mas real. Como dato extra, al probar con diferentes formas, pudimos obtener una forma rapida de pasar una imagen a un mapa de

0.4. EL CÓDIGO:

bits como el que estamos hablando. Se puede usar una herramienta para pasar una imagen a caracteres ASCII, y definir que los caracteres sean los valores 1 y 0. Lo que resta es solamente agregarlo al código y definir los tamaños correspondientes. La figura 9, muestra el resultado de una de estas pruebas.

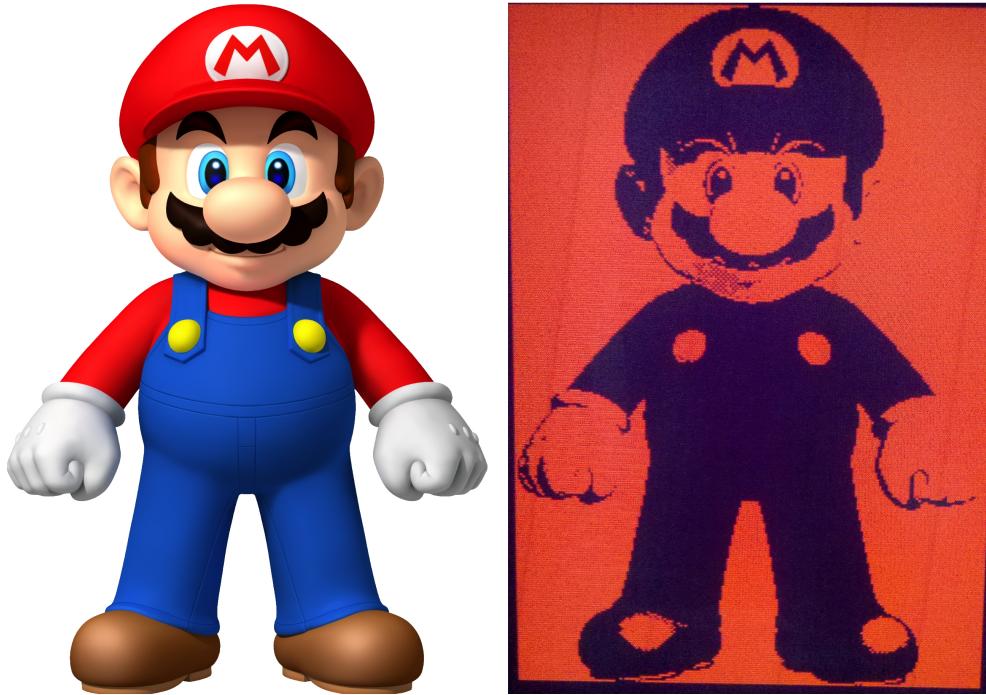


Figura 9: Imagen JPG en la izquierda. Imagen definida como mapa de bits, monocromática.

De este mismo modo se coloca en pantalla el logo de la UTN en la pantalla principal.

Volviendo al código, las señales **rom_addr**, **rom_col**, son para determinar la fila y la columna en la cual se va dibujar la bola, a medida que se va desplazando (esto se actualiza 60 veces por segundo). **rom_data** junto con **rom_bit** definen que cuando el barrido pasa por el valor de coordenadas en las cuales se encuentre la bola, ésta aparezca en pantalla.

Después de estas declaraciones, se encuentra el funcionamiento en si del juego, esta parte es bastante entendible con solo ver el código.

0.4. EL CÓDIGO:

```

----- paleta izquierda -----
  elsif ((BAR1_X_L <= ball_x_lft) and (ball_x_lft <= BAR1_X_R) and (band_2 = 0)
and mult = '1') then -- reach x of right bar

  if (bar1_y_top<=ball_y_btn) and (ball_y_top<=bar1_y_btn) then
    x_delta <= BALL_V_P + velocity_res; --hit, bounce back
    en <= '1';

  if (refr_tick = '1') then
    contador_player1 := contador_player1 + 10; --cuenta el puntaje

  if (bar1_rgb < "111") then
    bar1_rgb <= bar1_rgb + 1;
  else
    bar1_rgb <= "001";
  end if;

  end if;
end if;

```

Al cerrar el proceso que contiene la gran mayoria de los algoritmos del juego, estan las salidas que van a los displays. Luego de eso viene el ultimo bloque.

```

----- rgb multiplexing circuit
-----
process(game_on, mult, reset, bar1_on, rd_ball_on,
        bar1_rgb, ball1_rgb, bar2_on, bar2_rgb, wall_on, wall_rgb, logoutn_on, logoutn_rgb)
begin

  if game_on = '1' and reset = '1' and logoutn_on = '1' then
    graph_rgb <= logoutn_rgb;

  elsif game_on = '1' and reset = '0' then

    if wall_on='1' and mult = '0' then
      graph_rgb <= wall_rgb;
    elsif bar1_on='1' and mult = '1' then
      graph_rgb <= bar1_rgb;
    elsif bar2_on='1' then
      graph_rgb <= bar2_rgb;
    elsif rd_ball_on='1' then
      graph_rgb <= ball1_rgb;
    else
      graph_rgb <= "000"; -- black background
    end if;

  end if;
end process;

end arch;

```

Aqui, todas las señales por las que “pase” el “barrido”, y esten en 1, saldran en pantalla, con su determinado color y forma. Hay tambien algunas condiciones extra que tienen que cumplir, como que la pared este solo para **multiplayer** = 0, etc, pero el comportamiento general es ese.

0.4. EL CÓDIGO:

0.4.4. Código nº3 : Siete_segmentos.

Este código es bastante simple, acepta una entrada en forma de entero y sale un vector de números binarios, que corresponden al número que se mostrará en el display. Los mismos son de anodo común, por lo tanto un 0 (o masa) prende un segmento.

```

begin
    --0123456
    salida <= (NOT "1111110") when input = 0 else
    (NOT "0110000") when input = 1 else
    (NOT "1101101") when input = 2 else
    (NOT "1111001") when input = 3 else
    (NOT "0110011") when input = 4 else
    (NOT "1011011") when input = 5 else
    (NOT "1011111") when input = 6 else
    (NOT "1110000") when input = 7 else
    (NOT "1111111") when input = 8 else
    (NOT "1110011") when input = 9 else
    salida;

    seg_output <= salida;
  
```

0.4.5. Código nº4 y nº5 : Texto, font ROM.

font ROM define “el abecedario” empleado para que realmente aparezcan palabras en pantalla. No es ni más ni menos que un arreglo de mapas de bits, tal como lo vimos para definir la bola, o el logo de la UTN.

0.4. EL CÓDIGO:

```

-- code x41
"00000000", -- 0
"00000000", -- 1
"00010000", -- 2    *
"00111000", -- 3    ***
"01101100", -- 4    ** **
"11000110", -- 5    **  **
"11000110", -- 6    **  **
"11111110", -- 7    *****
"11000110", -- 8    **  **
"11000110", -- 9    **  **
"11000110", -- a    **  **
"11000110", -- b    **  **
"00000000", -- c
"00000000", -- d
"00000000", -- e
"00000000", -- f
-- code x42
"00000000", -- 0
"00000000", -- 1
"11111100", -- 2    *****
"01100110", -- 3    **  **
"01100110", -- 4    **  **
"01100110", -- 5    **  **
"01111100", -- 6    *****
"01100110", -- 7    **  **
"01100110", -- 8    **  **
"01100110", -- 9    **  **
"01100110", -- a    **  **
"11111100", -- b    *****
"00000000", -- c
"00000000", -- d
"00000000", -- e
"00000000", -- f

```

donde x41, x42, etc, define el valor en binario para referirse a esa letra.

A - 1000001 -> (41 en hexa)

B - 1000010 -> (42 en hexa)

C - 1000011 -> (43 en hexa)

etc.

En el código texto.vhd, se observan las siguientes entradas:

```

-- Entidad
entity texto is
  port (clk, reset, multiplayer : in std_logic;
        e0,e1,e2,e3,e4,e5,e6,e7 : in integer range 0 to 10; --salidas en entero del puntaje,
        ahora son entradas para mostrarlos en pantalla
        p_max0, p_max1, p_max2, p_max3 : in integer range 0 to 10;
        pixel_x, pixel_y : in std_logic_vector (9 downto 0);
        bandera_1 : in std_logic;                                --1 El juego ha terminado, 0 Juego
        (1 jugador)
        bandera_2 : in integer range 0 to 10;                  --0 Juego, 1 GANO P1, 2 GANO P2
        text_rgb : out std_logic_vector (2 downto 0));
  end texto;

```

La mayoría ya las hemos nombrado, a excepción de la salida **text_rgb**, que describe el color de la salida de texto.

El método para definir texto en pantalla es siempre el mismo, vamos a describir el segundo renglón del texto superior, que dice "Proyecto Final".

0.4. EL CÓDIGO:

```

----- TEXTO SUPERIOR, SEGUNDO RENGLON
-----

  titulo2 <=
  '1' when (pix_y (9 downto 4) = 1) and
  (pix_x (9 downto 4) >= 16) and (pix_x (9 downto 4) < 32) else
  '0';

  row_addr_titulo2 <= std_logic_vector(pix_y(3 downto 0)); -- escala : (4 downto 1), mas
grande (3 downto 0) mas chica
  bit_addr_titulo2 <= std_logic_vector(pix_x(2 downto 0));

  with pix_x (7 downto 3) select
  char_addr_titulo2 <=
  "1010000" when "00000",--P
  "1110010" when "00001",--r
  "1101111" when "00010",--o
  "1111001" when "00011",--y
  "1100101" when "00100",--e
  "1100011" when "00101",--c
  "1110100" when "00110",--t
  "1101111" when "00111",--o
  "0000000" when "01000",--n
  "1000110" when "01001",--F
  "1101001" when "01010",--i
  "1101110" when "01011",--n
  "1100001" when "01100",--a
  "1101100" when "01101",--l
  "0000000" when others;

```

Algo muy importante para aclarar, es que el texto se maneja distinto que los mapas de bits. Los mismos se los maneja directamente por los pixeles. Pero aqui tenemos bloques de texto, para este caso son bloques de 8x16 pixeles, en donde alli va la palabra. Puedo usar los bloques para hacer una palabra de 8x32, pero para que este centrada le debo poner espacios vacios al principio y al final. Si bien no es la forma mas comoda para trabajar con texto, funciona.

Para el resto de los titulos en pantalla, el sistema es el mismo.

Al final llegamos a un gran process, que basicamente repite lo mismo para distintas señales.

0.4. EL CÓDIGO:

```

begin
  text_rgb <= "000"; --Fondo negro

  if titulo = '1' and band_2 = 0 and band_1 = '0' then --Titulo 1
    char_addr <= char_addr_titulo1;
    row_addr <= row_addr_titulo1;
    bit_addr <= bit_addr_titulo1;
    if font_bit = '1' then
      text_rgb <= "111";
    end if;

  elsif titulo2 = '1' and band_2 = 0 and band_1 = '0' then --Titulo 2
    char_addr <= char_addr_titulo2;
    row_addr <= row_addr_titulo2;
    bit_addr <= bit_addr_titulo2;
    if font_bit = '1' then
      text_rgb <= "111";
    end if;

  (...)

  rom_addr <= char_addr & row_addr;
  font_bit <= font_word(to_integer(unsigned(not bit_addr)));
end arch;

```

En la imagen se lo muestra para titulo1 y titulo2. De vuelta, si la señal esta en uno, los valores de esa variable pasan a la señal de salida. Por lo tanto, si se cumplen las condiciones, van apareciendo en pantalla los textos. Dada la velocidad de refresco tan elevada, parece que estuviese estatico en pantalla.

0.4.6. Código nº6: Sonido

0.4.6.1. Teoria : PWM

La modulación por ancho de pulsos (también conocida como PWM, siglas en inglés de pulse-width modulation) de una señal o fuente de energía es una técnica en la que se modifica el ciclo de trabajo de una señal periódica (una senoidal o una cuadrada, por ejemplo), ya sea para transmitir información a través de un canal de comunicaciones o para controlar la cantidad de energía que se envía a una carga.

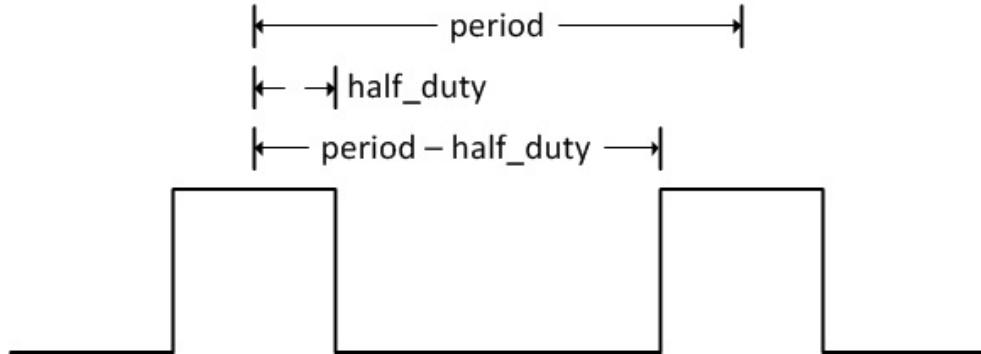


Figura 10: PWM

0.4.6.2. Código.

El PWM se puede utilizar para crear sonido monofónico, en este caso vamos a variar el ciclo de trabajo para lograr un ruido más envolvente.

Emplearemos un contador que lo varíe lentamente, para tal fin.

0.4. EL CÓDIGO:

```

ENTITY sonidos_pwm IS
  GENERIC(
    sys_clk          : INTEGER := 50000000;    --Frecuencia de clock del sistema en Hz
    pwm_freq         : INTEGER := 220;          --frecuencia de PWM en Hz
    bits_resolution : INTEGER := 10;            --Bits de resolucion
    phases          : INTEGER := 2);            --numero de salidas PWM
  PORT(
    clk      : IN STD_LOGIC;                  --Clock del sistema
    ena      : IN STD_LOGIC;                  --Enable
    pwm_out  : OUT STD_LOGIC_VECTOR(phases-1 DOWNTO 0); --Salidas
  );
END sonidos_pwm;

if (clk 'event and clk = '1') then
  if (contador > 0 and contador < 9999999) then
    contador := contador + 1;
  else
    contador := 1;
  end if;

  if (contador = 9999999) then
trabajo
    duty <= duty + 1;
  else
    duty <= duty;
  end if;
end if;

```

En el caso de que la bola toque una paleta, enviara una señal de enable por un leve instante para que se genere un “beep”. Durante la pantalla inicial, se reproduce por completo.

0.4.7. Código nº7 : pckg

Este código es simplemente un archivo incluido que contiene las instanciaciones mediante el comando “component”.

0.4. EL CÓDIGO:

```

  package pckg is

    component sincronismo_vga is
      port(
        fpga_50mhz_clk:      in std_logic;
        fpga_reset:          in std_logic;
        h_sync:              out std_logic;
        v_sync:              out std_logic;
        -- video_on:          out std_logic;
        en_tick:              out std_logic;
        pixel_x:             out std_logic_vector (9 downto 0);
        pixel_y:             out std_logic_vector (9 downto 0)
      );
    end component;

    component sonidos_pwm IS
      GENERIC(
        sys_clk          : INTEGER := 50000000;
        pwm_freq         : INTEGER := 140;
        bits_resolution : INTEGER := 8;
        phases           : INTEGER := 2);
      PORT(
        clk              : IN STD_LOGIC;
        -- reset_n        : IN STD_LOGIC;
        ena              : IN STD_LOGIC;
        -- duty            : IN STD_LOGIC_VECTOR(bits_resolution-1 DOWNTO 0);
        pwm_out         : OUT STD_LOGIC_VECTOR(phases-1 DOWNTO 0)
      );
      --pwm inverse outputs
    end component;

```

0.4.8. Código n°8 : Cabecera

Aquí se tienen las entradas y salidas de todo el proyecto. Además, comprende todos los “port map” que tiene el proyecto, es decir que es el nexo para que todos los códigos se puedan comunicar entre si, si fuese necesario.

0.4. EL CÓDIGO:

```

entity cabecera is
  port (
    -- clock & reset input signals
    clk:  in std_logic; -- 50MHz
    reset: in std_logic;
    multiplayer, on_off: in std_logic;
    -- Control VGA
    hsync: out std_logic; -- sincronismo horizontal --
    vsync: out std_logic; -- sincronismo vertical --
    -- control de paleta --
    entrada1:  in std_logic_vector (1 downto 0);
    entrada2:  in std_logic_vector(3 downto 0);
    -- control colores
    red, green, blue:  out std_logic_vector (3 downto 0);

    --salida del contador de score
    seg_output_d0, seg_output_d1 ,seg_output_d2,
    seg_output_d3 : out unsigned (6 downto 0); --Entrada del contador de score
    vidas : out std_logic_vector (4 downto 0);
    pwm_out : out std_logic_vector(1 DOWNTO 0)
  );
end cabecera;

-- Instanciar contador de puntaje (display HEX0)
contador_puntaje_d0 : entity work.siete_segmentos
  port map (
    input => input_signal_d0,
    seg_output => seg_output_d0);

sonidos : sonidos_pwm
  port map (clk => clk, --reset_n => reset,
            ena => enable, pwm_out => pwm_out);

```

Al final del archivo se tiene la salida global de lo que es el video en si, donde se decide entre mostrar los graficos, el texto o ambos, segun la situacion. Luego, las salidas RGB.

0.5. CONTROLES.

```

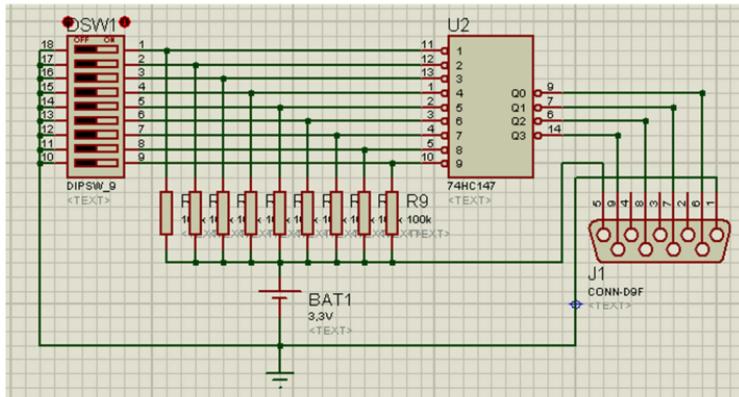
-- Buffer RGB
process (clk, pixel_tick, text_rgb, mult)
begin
  if (clk'event and clk='1' and pixel_tick = '1') then
    if (band_1 = '0' and mult = '0') then
      rgb_reg <= rgb_next or text_rgb;
    elsif (band_1 = '1' and mult = '0') then
      rgb_reg <= text_rgb;
    elsif (band_2 = 0 and mult = '1') then
      rgb_reg <= rgb_next or text_rgb;
    elsif (band_2 /= 0 and mult = '1') then
      rgb_reg <= text_rgb;
    end if;
  end if;
end process;

rgb <= rgb_reg;

red <= (rgb(2),rgb(2),rgb(2),rgb(2));
green <= (rgb(1),rgb(1),rgb(1),rgb(1));
blue <= (rgb(0),rgb(0),rgb(0),rgb(0));

```

0.5. Controles.



	A	B	C	D
Up	0	0	1	1
Down	0	1	1	1
Left	1	1	0	1
Right	1	0	1	1
START	0	1	1	0
A	1	1	1	0
B	0	0	0	1
C	0	1	0	1
Y	1	0	0	1
-	1	1	1	1

Figura 11: Joystick 1

0.6. CONCLUSIÓN

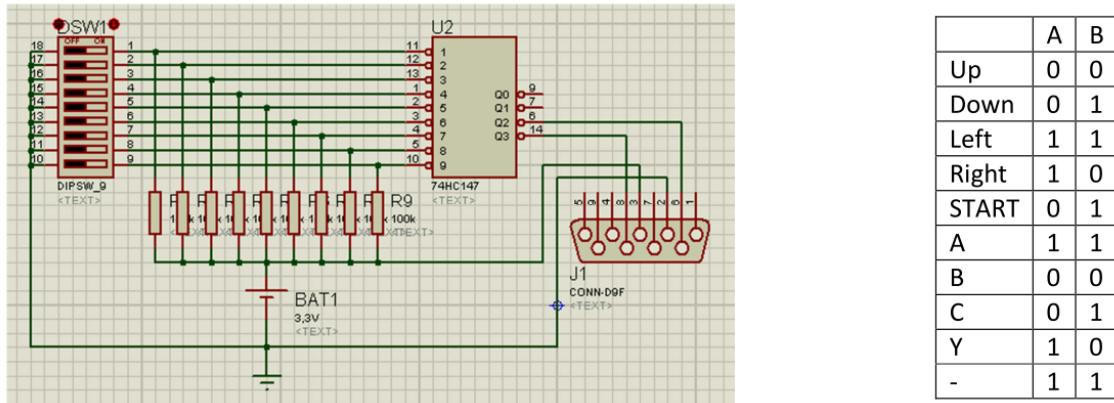


Figura 12: Joystick 2

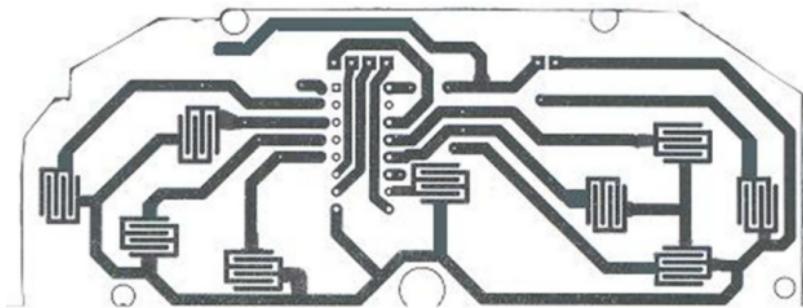


Figura 13: PCB de ambos joysticks

Partimos de 2 joysticks exactamente iguales, pero el cable que se utilizo en el joystick 2 solo poseía 5 conectores, de los cuales solamente 4 estaban disponibles, dos de ellos se utilizaron en la alimentación del integrado 74HC147, los otros dos, fueron utilizados para salida de datos.

0.6. Conclusión

Se logró con el objetivo propuesto. Con los temas vistos durante el año, las prácticas de laboratorio, más algo de investigación, logramos entender como se maneja el puerto VGA y en él, poder sacar las imágenes que queríamos, además de imágenes animadas.