



# DEEP TH!R? PONG

Juego estilo PONG en FPGA

Pablo Cáceres Gaitán

Asignatura:

Sistemas reconfigurables

Escuela Politécnica Superior de Córdoba

Enero-Febrero 2025

---

## Contents

<b>Abstract</b>	<b>ii</b>
<b>1 Introducción</b>	<b>iii</b>
<b>2 Materiales</b>	<b>iv</b>
2.1 Montaje . . . . .	v
2.2 Programación de la placa . . . . .	vi
<b>3 Manual de usuario</b>	<b>vii</b>
3.1 Elegir dificultad . . . . .	viii
3.2 Partida . . . . .	ix
3.3 Fin de la partida . . . . .	ix
<b>4 Descripción de los módulos</b>	<b>xi</b>
4.1 ClockController . . . . .	xi
4.1.1 Señales del módulo . . . . .	xi
4.2 BuzzerController . . . . .	xi
4.2.1 Señales del módulo . . . . .	xi
4.2.2 Código . . . . .	xii
4.3 GameController . . . . .	xiii
4.3.1 Señales del módulo . . . . .	xiii
4.3.2 Código . . . . .	xiv
4.4 BallController . . . . .	xvi
4.4.1 Señales del módulo . . . . .	xvi
4.4.2 Código . . . . .	xvii
4.5 Constantes . . . . .	xviii
<b>5 Conclusión</b>	<b>xix</b>

---

## Abstract

Las FPGAs han revolucionado la manera en la que entendemos el desarrollo de hardware. Gracias a ellas, somos capaces de "programar" componentes hardware que podemos utilizar como bloques de construcción para crear sistemas complejos, todo ello sin la necesidad de crear costosos prototipos en el proceso. Tan solo reconfigurando un mismo dispositivo. En este proyecto trataremos de programar el clásico PONG, probando la versatilidad y los beneficios del desarrollo de sistemas en FPGAs.

---

## 1 Introducción

Este es el trabajo final para la asignatura de Sistemas Reconfigurables. Con su realización se busca afianzar los conocimientos adquiridos durante la asignatura, sintetizándolos en un pequeño proyecto con una complejidad mayor.

En este caso, se realizará una implementación del clásico juego PONG [2]. Se realizará en una **placa FPGA Basys 3**, mediante la suite de Vivado y en lenguaje VHDL. Como añadido, se incorpora una pequeña melodía al inicio de la partida y sonido en los rebotes.

---

## 2 Materiales

Para la realización del trabajo utilizaremos la ***Basys 3***, una placa FPGA basada en la *Xilinx Artix-7*. Una placa de nivel de entrada, que cuenta con botones, *switches* y LEDs integrados, simplificando el montaje. Para el desarrollo y la programación de la placa utilizaremos la suite de ***Vivado*** en Windows.

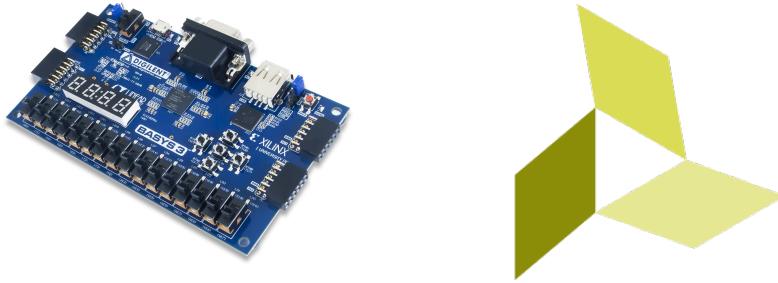


Figure 1: Basys3 con vivado

Además, utilizaremos un módulo de ***buzzer pasivo***, que nos permitirá convertir señales eléctricas en sonido. Respecto a su versión activa, tenemos la ventaja de variar las notas que se emiten.[1]



Figure 2: Módulo buzzer

Finalmente, necesitaremos un **monitor con entrada VGA** (y su respectivo cable). En este caso, utilizaremos un monitor de 640x480 píxeles a 60 Hz.

---

## 2.1 Montaje

Para el montaje, comenzaremos conectando el monitor a la Basys 3 mediante la interfaz VGA:



Figure 3: Conexión de la interfaz VGA

Posteriormente, conectaremos los pines de alimentación del buzzer (3v3 y GND) a los pines de 3v3 y GND de la placa, y el de señal al puerto **JB4**:

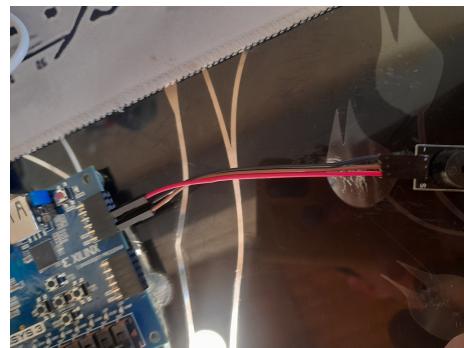


Figure 4: Conexión del buzzer

Deberíamos tener algo así:

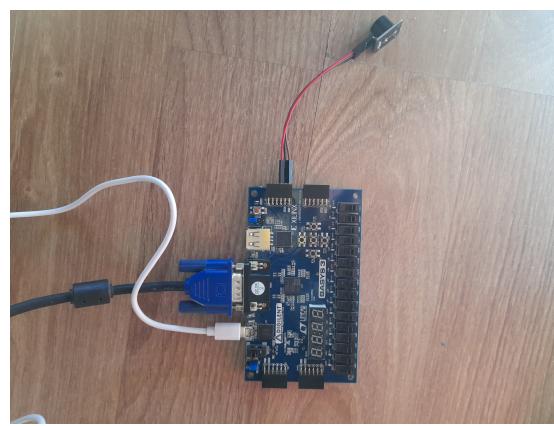


Figure 5: Conexión del buzzer

---

## 2.2 Programación de la placa

Una vez realizado el montaje, conectamos la placa al ordenador mediante el cable micro USB. Y en el menú ”*Program and Debug*” del ”*FLOW NAVIGATOR*” en Vivado, seleccionamos ”*Open Device*”, y posteriormente ”*Program Device*”:



Figure 6: Programación en Vivado

Una vez completado este paso, la FPGA debería tener cargado el programa y estar lista para utilizarse.

---

### 3 Manual de usuario

Nada más iniciar el programa, veremos la pantalla de título:

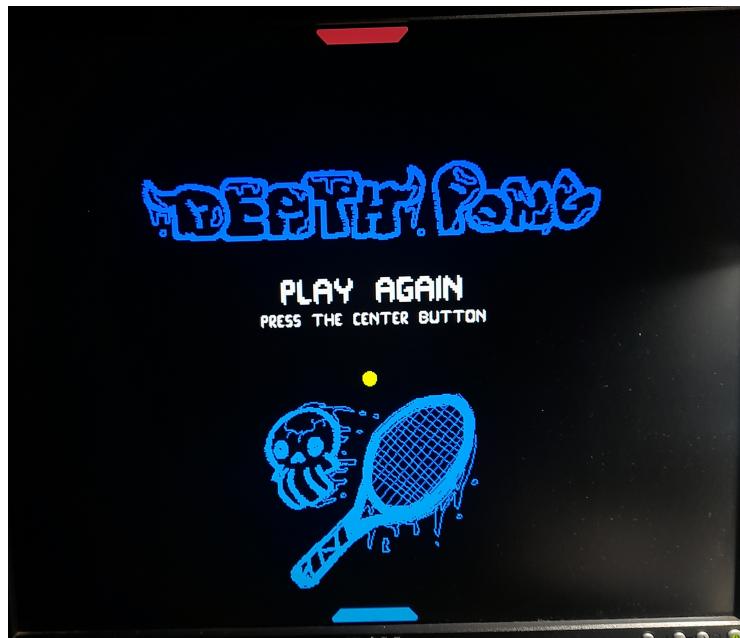


Figure 7: Pantalla de titulo.

---

### 3.1 Elegir dificultad

En este punto, podemos escoger la dificultad utilizando los interruptores: **SW15** para modo difícil y **SW14** para fácil (ninguno para normal).

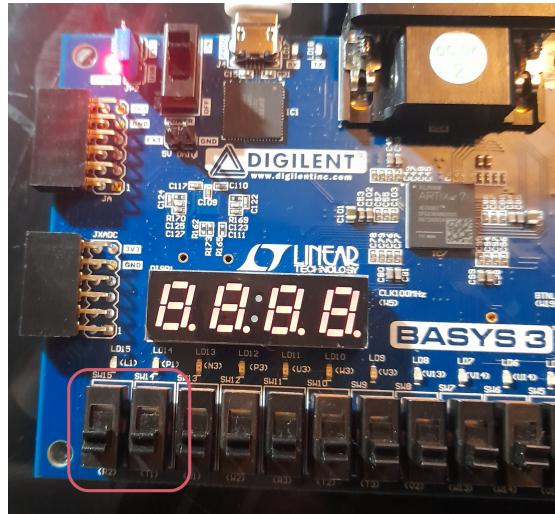


Figure 8: Interruptores para seleccionar la dificultad.

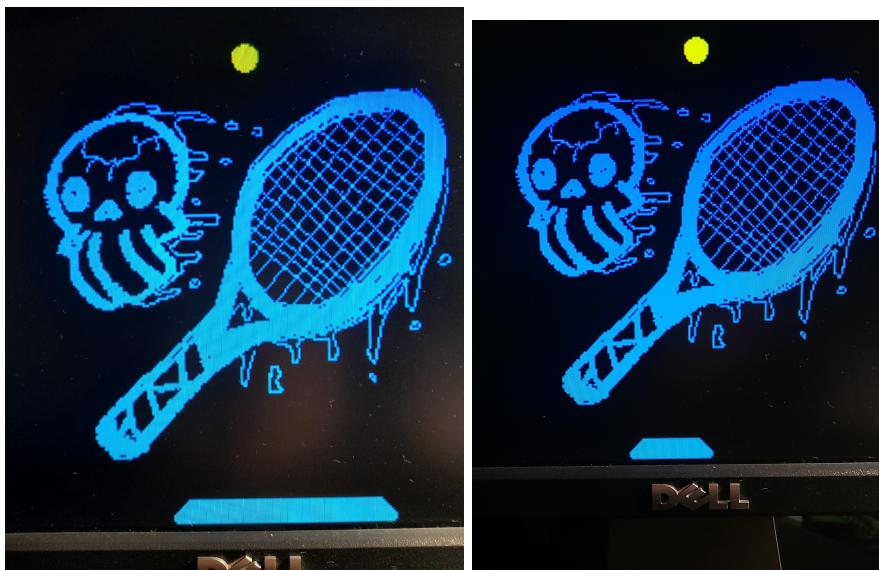


Figure 9: Modos fácil y difícil.

### 3.2 Partida

Si pulsamos el botón central, se empezará a oír la melodía de inicio y dará comienzo la partida:

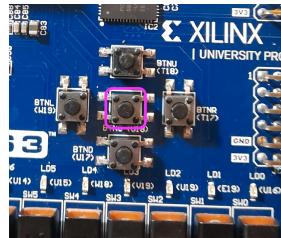


Figure 10: Botón central.

Veremos la pantalla del juego, y controlaremos la pala inferior (azul), mediante los botones izquierdo y derecho:

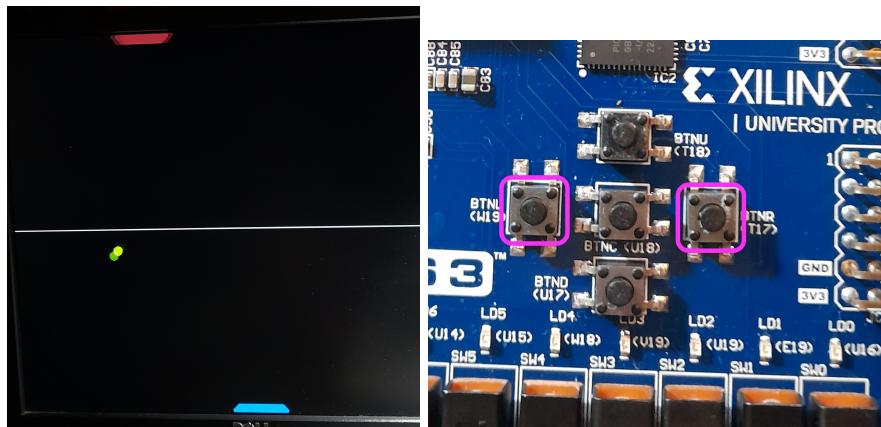


Figure 11: Pantalla del juego y botones para controlarlo.

### 3.3 Fin de la partida

En caso de que la pelota llegue a la parte inferior perderemos y en caso de tocar en la parte superior, ganaremos, llegando a la siguiente pantalla:

Volviendo a pulsar el botón central reiniciaremos el juego.



Figure 12: Pantallas de final del juego.

---

## 4 Descripción de los módulos

El proyecto se compone principalmente de 3 módulos, conectados en un módulo superior. Estos módulos serían:

1. *ClockController*: Divisor de reloj encargado de crear la señal de reloj para la VGA.
2. *GameController*: Encargado de gestionar el juego y enviar las señales gráficas a la interfaz VGA. Tiene como submódulo a *ballController*, que gestiona el movimiento y las colisiones.
3. *BuzzerController*: Controla la reproducción de sonido.

El diagrama de los módulos quedaría así:

### 4.1 ClockController

Este módulo es básicamente un divisor de frecuencia para adaptar la frecuencia del reloj de la Basys 3 (100MHz) a la frecuencia que necesita el monitor VGA (25MHz). Esta división se realiza mediante 2 *T-FlipFlops*, que dividen la señal original a una cuarta parte.

#### 4.1.1 Señales del módulo

El módulo cuenta con dos señales, una entrada y una salida:

- *clockIn*: entrada del reloj de la Basys 3 (100MHz).
- *clockOut*: salida de reloj a (25Mhz).

### 4.2 BuzzerController

Este módulo es el encargado de generar los sonidos del juego. Se compone de una máquina de estados que, en función del modo, enviará un pitido o una melodía al buzzer.

#### 4.2.1 Señales del módulo

Pese a que internamente es más complejo, su interfaz es muy sencilla:

- *clk*: Reloj de 100 MHz
- *reset*: Señal de reset.
- *mode*: '0' para melodía, '1' para pitido corto. Se pondrá a 0 cuando el juego esté parado y en 1 durante la partida.
- *button*: Señal que activa el sonido de cada modo.
- *buzzer (out)*: Salida para el buzzer

---

#### 4.2.2 Código

Las señales internas son las siguientes, vemos que es muy sencillo modificar los parámetros del sonido y componer melodías. Lo que sí se debe tomar en cuenta es representar las notas por su valor de frecuencia y su duración en milisegundos.

```
type state_type is (IDLE, BEEP, MELODY, DONE);
signal state : state_type := IDLE;

constant MAX_NOTE_INDEX : integer := 15;
type melody_array is array (0 to MAX_NOTE_INDEX) of integer;

-- Notas en Hz (Melodía de acción)
constant notes : melody_array := (196, 196, 262, 294,
                                    330, 294, 262, 196,
                                    220, 220, 294, 330,
                                    349, 392, 440, 494);

-- Duraciones en ms
constant durations : melody_array := (600, 200, 200, 200,
                                         400,
                                         200, 200, 400, 600,
                                         200,
                                         200, 200, 400, 200,
                                         200, 800);

signal note_index : integer range 0 to MAX_NOTE_INDEX := 0;
signal duration_counter : integer := 0;
signal toggle : std_logic := '0';
signal pwm_clk_div : integer := 0;

constant CLK_FREQ : integer := 100_000_000; -- 100 MHz
signal period : integer := 0;
signal half_period : integer := 0;

constant BEEP_FREQ : integer := 440; -- Nota LA
constant BEEP_DURATION : integer := 10_000_000;
signal beep_active : std_logic := '0';
```

---

Para controlar el sonido a generar se utiliza una máquina de estados (el código ha sido recortado para mejorar la legibilidad), que permite añadir fácilmente nuevos estados con nuevos sonidos:

```
case state is
when IDLE =>
    if mode = '1' then
        if button = '1' then
            beep_active <= '1';
            state <= BEEP;
            duration_counter <= 0;
        end if;
    elsif mode = '0' then
        if button = '1' then
            state <= MELODY;
            note_index <= 0;
            duration_counter <= 0;
        end if;
    end if;
when BEEP =>
    if beep_active = '1' then
        -- REDACTED - Reproduce el sonido durante el tiempo
        -- especificado
    else
        state <= IDLE;
        beep_active <= '0';
    end if;

when MELODY =>
    -- REDACTED - Reproduce el sonido durante el tiempo
    -- especificado

when DONE =>
    toggle <= '0';
    state <= IDLE;
```

### 4.3 GameController

Finalmente el módulo GameController, el más complejo y extenso. Es el encargado de generar las señales de sincronización VGA, así como enviar la imagen pixel por pixel.

#### 4.3.1 Señales del módulo

Al ser un módulo complejo, se compone de una gran cantidad de señales. Las de **entrada** son:

- *clock*: Señal de reloj.
- *left, right*: Controles de la pala del jugador.
- *start*: Señal que indica para iniciar el juego (se conecta a *BallController*).
- *difficultyControl*: Vector de 2 bits que ajusta la dificultad del juego.

---

Y las de **salida**:

- *hSync, vSync*: Señales de sincronización horizontal y vertical para la pantalla VGA.
- *r, g, b*: Señales de color para la pantalla VGA (4 bits por canal).
- *collision*: Señal de colisión (se propaga desde BallController hasta el top para activar el sonido de rebote).

#### 4.3.2 Código

En este módulo se definen los mapas de bits para dibujar los títulos y el logo del juego. Con un 1 definimos los píxeles en negro y con 0 los que componen la imagen:

```
--Definitions of the bitmaps for different images, they will
be stored in the ROM
type BITMAP1 is array (0 to 60) of std_logic_vector(0 to
299);
type BITMAP2 is array (0 to 46) of std_logic_vector(0 to
399);
type BITMAP3 is array (0 to 164) of std_logic_vector(0 to
199);
type BITMAP4 is array (0 to 63) of std_logic_vector(0 to
399);
-- REDACTED [...]
signal result1, result2: BITMAP1;
signal message: BITMAP2;
signal logo: BITMAP3;
signal gameLabel: BITMAP4;
signal paddleWidth: integer:= PADDLE_WIDTH;
-- REDACTED [...]

result1 <= ("111111...",
             "101101...",
             "110011..."
            );
```

Posteriormente se asignan las correspondientes señales VGA. Básicamente, los elementos se enmascarán mediante las señales *XVisible* y las flags que indican si la partida ha empezado o alguien ha ganado:

```
-- Controla si se debe mostrar cada elemento (pseudocódigo)
result1Visible <= inDisplayArea and AIWins and result1(
    vOffset)(hOffset) = '0';
result2Visible <= inDisplayArea and playerWins and result2(
    vOffset)(hOffset) = '0';
messageVisible <= inDisplayArea and newGame and message(
    vOffsetMsg)(hOffsetMsg) = '0';
logoVisible <= inDisplayArea and newGame and logo(
    vOffsetLogo)(hOffsetLogo) = '0';
gameLabelVisible <= inDisplayArea and newGame and not(AIWins
    or playerWins)
    and gameLabel(vOffsetLabel)(hOffsetLabel)
        ) = '0';

-- Paddle, Ball, and Borders (pseudocódigo)
paddleVisible <= inPaddleArea and inPaddleShape;
ballVisible <= inBallArea and inBallShape;
paddleAIVisible <= inAIArea and inAIShape;
borderVisible <= inBorder and not newGame;

-- Pixel Scanning - Barrido de la pantalla
hPosNext <= hPosCurrent + 1 when hPosCurrent < TOT_H else 1;
vPosNext <= vPosCurrent + 1 when hPosCurrent = TOT_H and
    vPosCurrent < TOT_V else
    1 when hPosCurrent = TOT_H and vPosCurrent =
        TOT_V else vPosCurrent;

-- Selección de colores de cada elemento
rgbNext <= "001001101110" when paddleVisible else
    "110111011101" when messageVisible else
    "101111110000" when ballVisible else
    "110111011101" when frameVisible or borderVisible
    else
    "100101000100" when paddleAIVisible or
        result1Visible else
    "000010101110" when result2Visible else
    "001001101110" when gameLabelVisible or
        logoVisible else
    "000000000000";

-- VGA Signal Outputs
hSync <= '0' when (hPosCurrent > FP_H) and (hPosCurrent <
    FP_H + SP_H + 1) else '1';
vSync <= '0' when (vPosCurrent > FP_V) and (vPosCurrent <
    FP_V + SP_V + 1) else '1';
r <= rgbCurrent(11 downto 8);
g <= rgbCurrent(7 downto 4);
b <= rgbCurrent(3 downto 0);
```

---

Además se implementa un proceso que controla los elementos móviles de la pantalla: genera una señal de reloj más lenta para el movimiento de la pelota y calcula la posición de las palas.

A la hora de mover la pala de la IA, se incluye un error pseudoaleatorio para que el juego no se vuelva injusto para el jugador. Esto se calcula de la siguiente forma:

```
-- En las dos primeras lineas se genera un valor
    pseudoaleatorio que depende de la posicion de la pelota,
    pero que mueve la pala en direccion a la pelota
( ballCursorY mod (ballCursorX mod 5) ) *
( ballCursorX mod (ballCursorY mod 5) )
-- se utiliza tambien la posicion del jugador humano para
    introducir aun mas variacion
+ ( paddleCursor * paddleAICursor ) mod 5
```

Esto introduce un error de entre 0 y 4 píxeles a la decisión de movimiento de la IA, lo que resulta en una experiencia de juego equilibrada.

Otra característica interesante es la suavización del movimiento de las palas, tanto para el jugador como para la IA.

```
if paddleRight = (PRESCALER_PADDLE - 5000)
    and paddleCursor < TOT_H - paddleWidth then

    paddleCursor <= paddleCursor + 1;

elsif paddleLeft = (PRESCALER_PADDLE - 5000)
    and paddleCursor > FP_H + SP_H + BP_H + 1 then

    paddleCursor <= paddleCursor - 1;
end if;
```

Con este código hacemos que la paleta se mueva a un ritmo determinado (sólo cuando el contador alcance PRESCALER\_PADDLE - 5000). Evitando así que al dejar pulsado la paleta se mueva demasiado rápido o de saltos, y manteniéndola dentro de los bordes de la pantalla.

## 4.4 BallController

Finalmente tenemos a *BallController*, el módulo encargado del movimiento de la pelota y el cálculo de las colisiones. Pese a tener una funcionalidad tan específica, posee gran complejidad y tiene bastantes señales de entrada y salida para comunicarse con *GameController*.

### 4.4.1 Señales del módulo

Las señales de **entrada** son:

- *start*: Indica el inicio del juego.
- *move*: Señal de reloj que actualiza la posición de la pelota. Es generada en *GameController*.

- 
- *paddleWidth*: Indica el ancho de las palas.
  - *paddlePos, paddleAIPos*: Posiciones de la pala del jugador y de la IA respectivamente.

Y las de **salida** son:

- *xPos, yPos*: Coordenadas actuales de la pelota.
- *newGame, play*: Indican si el juego ha comenzado o se ha reiniciado.
- *AIWon, PlayerWon*: Indican si la IA o el jugador ganaron.
- *ball\_collision*: Señal de colisión que se usa para sonido.

#### 4.4.2 Código

En este módulo se definen las condiciones de victoria (básicamente si la pelota colisiona con el borde inferior o superior):

```
if currentY <= FP_V + SP_V + BP_V + 1 then
    playerWins <= '1';
elsif start = '1' then
    playerWins <= '0';
end if;

if currentY >= TOT_V then
    AIWins <= '1';
elsif start = '1' then
    AIWins <= '0';
end if;
```

Otra parte importante es la detección de las colisiones, que se realiza calculando las posiciones relativas de la pelota con los bordes y las palas. Por ejemplo:

```
wallHorizontalBounce := currentX <= FP_H + SP_H + BP_H + 1
or currentX >= TOT_H - BALL_SIDE;
```

Las variables relacionadas con las dimensiones y márgenes de la señal VGA están definidas en el archivo de constantes 1.

---

## 4.5 Constantes

Hay un archivo con las constantes utilizadas en el proyecto. Se definen en él todas las variables con las dimensiones de la señal VGA:

Variable	Significado	Descripción
FP_H	<i>Front Porch Horizontal</i>	Margen izquierdo antes del área visible.
SP_H	<i>Sync Pulse Horizontal</i>	Pulso de sincronización horizontal.
BP_H	<i>Back Porch Horizontal</i>	Margen trasero horizontal después del área visible.
TOT_H	<i>Total Horizontal</i>	Ancho total de la pantalla, incluyendo <i>porches</i> y área visible.
VIS_H	<i>Visible Horizontal</i>	Ancho del área visible en la pantalla.
FP_V	<i>Front Porch Vertical</i>	Margen delantero vertical antes del área visible.
SP_V	<i>Sync Pulse Vertical</i>	Pulso de sincronización vertical.
BP_V	<i>Back Porch Vertical</i>	Margen trasero vertical después del área visible.
TOT_V	<i>Total Vertical</i>	Altura total de la pantalla, incluyendo márgenes y área visible.
VIS_V	<i>Visible Vertical</i>	Altura del área visible en la pantalla.

Table 1: Variables de sincronización VGA

Y las variables relativas a las palas y la pelota:

Constante	Descripción
PADDLE_WIDTH	Ancho estándar de la paleta
PADDLE_WIDTH.EASY	Ancho de la paleta en modo fácil
PADDLE_WIDTH.HARD	Ancho de la paleta en modo difícil
PADDLE_HEIGHT	Altura de la paleta
PRESCALER.PADDLE	Control de suavización del movimiento de la paleta (evita rebotes)
BALL_SIDE	Tamaño de la pelota (lado del cuadrado)
BALL_INITIAL_X	Posición inicial en X de la pelota
BALL_INITIAL_Y	Posición inicial en Y de la pelota
PRESCALER.BALL	Control de suavización del movimiento de la pelota

Table 2: Constantes de tamaño de la paleta y la pelota

---

## 5 Conclusión

A lo largo de este trabajo hemos puesto en práctica lo aprendido a lo largo de la asignatura: programación en VHDL, uso de módulos para crear un código limpio y reutilizable, buenas prácticas como el uso de máquinas de estados, a crear interfaces con distintos módulos... Al final obtenemos un sistema simple pero resultón, demostrando las virtudes del uso de sistemas reconfigurables en el desarrollo de sistemas hardware.

Como posibles mejoras para el proyecto, en caso de que alguien quisiera continuar el desarrollo, podrían implementarse las siguientes funcionalidades:

- Pausa: puede ser implementada en uno de los switches no utilizados.
- Reset: podría implementarse un botón de reset que nos devuelva al estado inicial en la pantalla del título (para por ejemplo, salir en caso de un *bug* que bloquee la pelota).
- Cambios de color: implementando distintas paletas al cambiar el estado de los switches ó en cada dificultad.
- Contador de puntos: aumentando la complejidad del juego, con reglas más similares a las de un deporte de raqueta.
- Más melodías: diversificando el juego, con melodías para cuando el jugador gane ó pierda.

---

## References

- [1] Murky Robot. Buzzer - ui displays guide, 2025. URL: <https://www.murkyrobot.com/guias/ui-displays/buzzer>.
- [2] Wikipedia contributors. Pong, 2025. URL: <https://es.wikipedia.org/wiki/Pong>.