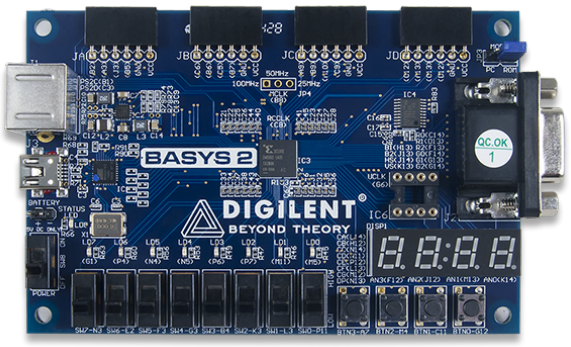
ESCUELA TECNICA SUPERIOR DE INGENIERIA

SEVILLA

FLOPPY GAME

Proyecto Sistemas electrónicos.

Programación en FPGA con VHDL.



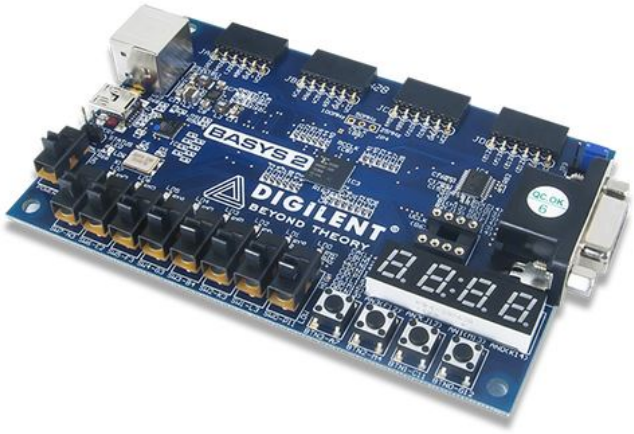
Integrantes del proyecto:
Fernandez Salcedo, Miguel
Lopez Gil, Miguel
Montes Grova, Marco Antonio

Indice

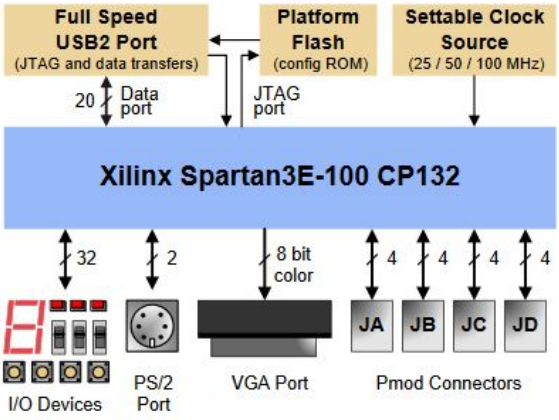
1. Introducción y objetivo del proyecto

1.1. Introducción al proyecto

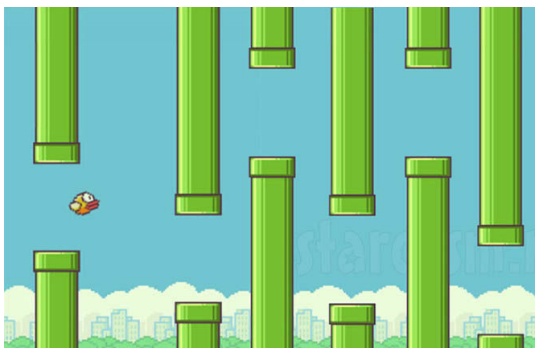
El proyecto que se muestra en éste documento será una modificación del videojuego *Flappy Bird* en VHDL e implementado en una FPGA. La FPGA que se utilizará será la BASYS 2 Spartan-3E del fabricante Xilinx:



La placa constituye un circuito programable de 100.000 puertas con las que podremos obtener casi cualquier circuito. Además de ello posee cómo perifericos externos 8 LEDs, 4 display de 7 segmentos y 8 switches. Un puerto P2/2 y un puerto VGA de color de 8 bits. A continuación, tenemos un diagrama de bloques de la placa, extraido de la documentación oficial, dónde podremos ver todos los componentes que posee.

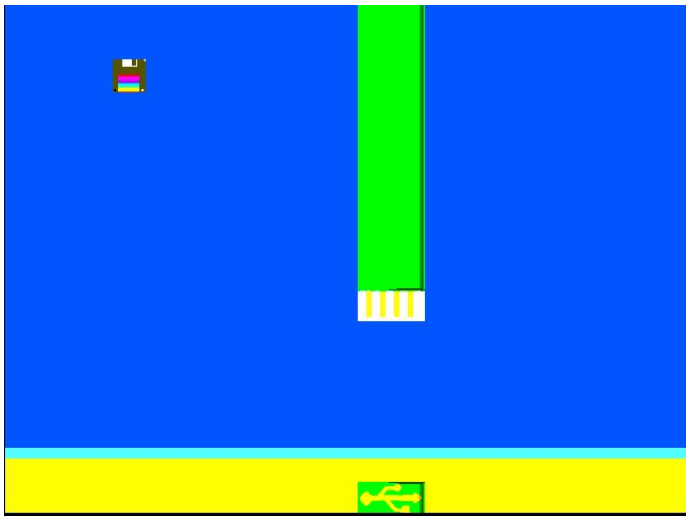


Por otro lado, realizaremos una breve introducción al videojuego:
El juego fue desarrollado para móviles en 2013 en Hanói por el desarrollador vietnamita Nguyen Hà Đông (Dong Nguyen) y publicado por *.GEARS Studios*. El juego, publicado el 24 de mayo de 2013, fue eliminado de App Store y Google Play por su creador el 9 de febrero de 2014.
El jugador controla un pájaro intentando volar entre filas de tuberías verdes sin tocarse con estas. La escena se va desplazando lateralmente. El desarrollador creó el juego en varios días utilizando como personaje central un pájaro que diseñó para otro juego cancelado en 2012.



1.2. Objetivo del proyecto

En este proyecto se desarrollará una modificación del videojuego original. Además de ello, un driver VGA para un monitor en modo 60Hz con una resolución de 640x480.
La estetica del videojuego será diferente a la original, se ha optado por una estética más *retro"* o nostálgica y jugado con las palabras en inglés *"flappy"-"floppy"*. En éste caso, el personaje del juego será un disquete y las columnas *pen-drives*.
Debido a la dificultad para obtener una imagen de lo que emite la FPGA, se ha obtado por emplear un sitio web que posee un simulador VGA. La web fué creada por Eric Eastwood[1] y, aunque no posee toda la cantidad de colores que tiene la FPGA, sirve para conocer de manera aproximada lo que emite la misma.
Aquí tenemos una imagen obtenida gracias a ese simulador del juego:



Si conectasemos la FPGA a un monitor VGA, los colores que se verían serían distintos, debido a que el simulador no posee la paleta de colores completa.

[1] El sitio web del que se habla es: https://ericeastwood.com/lab/vga-simulator/

2. Desarrollo teórico

La tarjeta *BASYS 2* dispone de un conector *DE15* cableado a la *FPGA*. Utilizando dicho conector es posible conectar directamente un monitor compatible con el estándar VGA.
Mediante una serie de contadores síncronos, generaremos todas las señales necesarias para controlar el monitor en modo 60Hz, 640x480 y 256 colores.

2.1. Uso de la fpga para transmitir por VGA

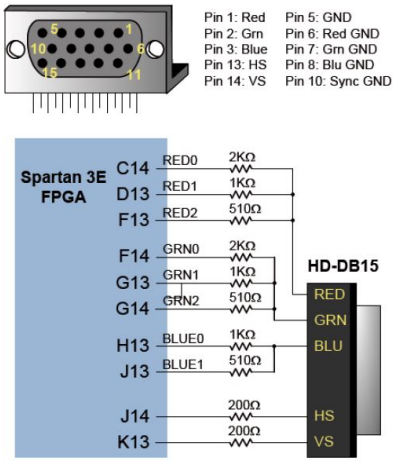
La tarjeta BASYS 2 dispone de un conector DE15 conectado a 10 puertos de la FPGA. La conexión entre el conector VGA y la FPGA se muestra en la Figura del fin de la página.
A través de este conector se pueden controlar las cinco señales necesarias para implementar el protocolo VGA:

- Dos señales digitales (activas a nivel bajo) para establecer el sincronismo: HS (*Horizontal Sync*) y VS (*Vertical Sync*).
- Tres señales analógicas para definir el color: RED, GRN y BLUE. Estas señales analógicas se controlarán a través de un bus de 8 bits: 3 bits para RED, 3 bits para GRN y 2 bits para BLUE.

Para controlar las señales analógicas de color usando las señales digitales de la FPGA se ha implementado, en placa, un convertidor Digital-Analógico basado en una serie de divisores resistivos. Mediante las resistencias de 2KΩ, 1KΩ y 510Ω, en conjunción con los 75Ω que presenta la terminación del monitor VGA, se puede convertir una señal de tres bits (RED0,RED1 y RED2) en ocho niveles analógico. Las resistencias se han dimensionado de forma que:

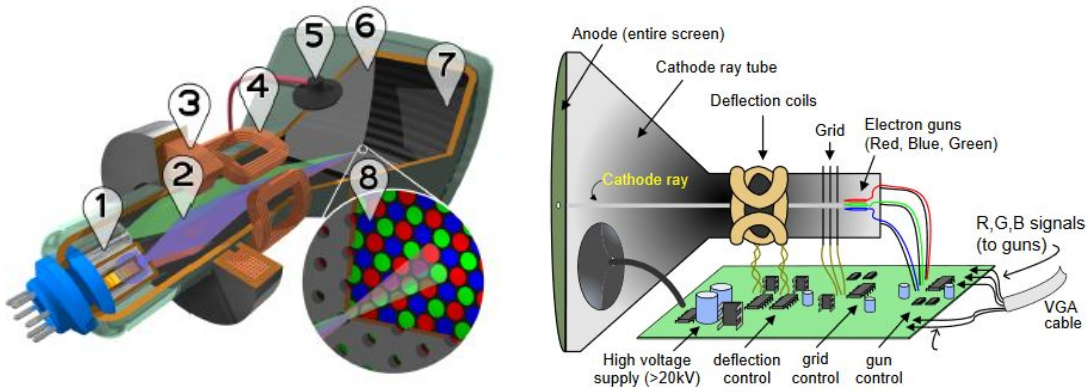
- RED = 0.7V cuando los tres bits del color están activados (*RED0='1', RED1='1' y RED2='1'*).
- RED = 0V con los tres bits a cero.
- Los ocho niveles analógicos, resultado de las diferentes combinaciones de RED(2 downto 0) estarán equiespaciados entre 0V y 0.7V.

Para la línea del verde (GRN0, GRN1 y GRN2) ocurre algo análogo. Sin embargo para el color azul se utilizan sólo dos bits (BLUE0 y BLUE1), resultando en sólo cuatro niveles de azul, ya que el ojo humano es menos sensible al tono azul. En conclusión, tenemos 8 bits de color, y por tanto 256 colores disponibles.
Para poder realizar un driver VGA es necesario comprender el funcionamiento del protocolo VGA. En el siguiente apartado se explicará brevemente el protocolo.



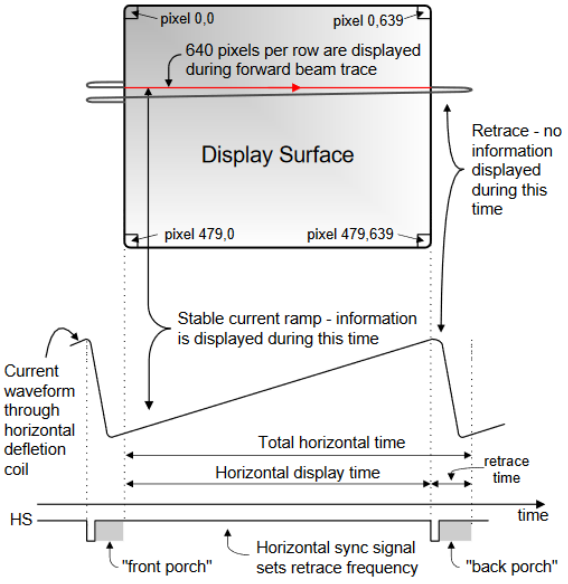
2.2. Funcionamiento del driver VGA

En este apartado se explicará cómo deben generarse las señales HS, VS, RED(2 downto 0), GRN(2 downto 0) y BLUE(1 downto 0) para representar una imagen en un monitor compatible VGA en modo 60Hz, 640x480 y 256 colores.
Para explicar el protocolo VGA es necesario conocer el funcionamiento básico de un Tubo de Rayos Catódicos (CRT: Cathode ray tube).(Aunque el tubo de rayos catódicos ha quedado obsoleto, los nuevos monitores mantienen la misma temporización para las entradas VGA).
Para entender el funcionamiento básico de un CRT debemos fijarnos en la figura X: Tres cañones de electrones *(1)* producen tres haces de electrones *(2)*, uno por cada color básico (rojo, azul y verde). Estos haces de electrones se focalizan *(3)*, utilizando una bobina, para conseguir que converjan en un punto de la pantalla.
Posteriormente, mediante un campo magnético generado por un par de bobinas *(4)* (horizontal y vertical) se direccionan al punto deseado de la pantalla. En la pantalla de visualización, los rayos son separados por una máscara *(6)* e inciden en una capa fosforescente con zonas receptivas para cada color *(7-8)*.
La intensidad luminosa dependerá de la potencia a la que se emite el haz de electrones, controlable mediante las señales analógicas RED, GRN y BLUE.



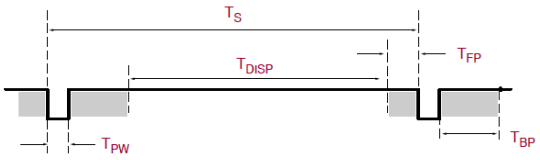
Se representarán imágenes en pantalla haciendo que el haz de electrones la recorra en una sucesión de líneas (de izquierda a derecha) comenzando por la esquina superior izquierda.
A continuación, explicaremos cómo pasa la corriente por la bobina para crear el campo magnético para la deflexión horizontal.

Se representará la imagen *(Horizontal display time)* cuando el haz de electrones va de izquierda a derecha y está apuntando correctamente a la pantalla, existiendo un tiempo *(retrace time)* para que el cañón vuelva al comienzo de la siguiente línea.
El pulso de sincronismo vertical (HS) marca el instante en que el cañón debe comenzar una nueva línea, pudiendo distinguir un tiempo anterior *(front porch)* en que el cañón está apuntando fuera de la pantalla y un tiempo posterior *(back porch)* necesario para que el haz se posicione en el inicio de la siguiente línea. Es importante que durante todo el *retrace time* los cañones de electrones estén apagados.



La frecuencia a la que se proporcionan los pulsos de sincronismos vertical y horizontal determina la velocidad y el número de veces que el haz de electrones recorre la pantalla. En nuestro caso, vamos a diseñar un controlador VGA para la resolución 640x480 píxeles y 60 Hz de refresco de pantalla. Consideraremos una frecuencia de píxel de 25MHz, para lo que dividiremos por dos la frecuencia de reloj disponible de 50MHz.

Por último, mostraremos un gráfico dónde podremos ver la temporización necesaria para una resolución de 640x480 píxeles y una tabla dónde daremos esos datos para nuestro caso (El reloj será referido al de 25MHz):



Símbolo	Parámetro	Sincronismo vertical			Sincronismo horizontal	
		Tiempo	Reloj	Líneas	Tiempo	Reloj
T_s	Periodo de sincronismo	$16.7ms$	416800	521	$32\mu s$	800
T_{DISP}	Display Time	$15.36ms$	384000	480	$25.6\mu s$	640
T_{PW}	Pulse Width	$64\mu s$	1600	2	$3.84\mu s$	96
T_{FP}	Front porch	$320\mu s$	8000	10	$640ns$	16
T_{BP}	Back porch	$928\mu s$	23200	29	$1.92\mu s$	48

En nuestro driver, los tiempos se podrán ver en los Generics empleados en algunas entidades.

En el próximo apartado, hablaremos de cada bloque que contiene nuestro proyecto, ahí, en el bloque del Driver VGA, podremos ver cómo son usados estos tiempos.

3. Realización del proyecto

3.1. Introducción a la funcionalidad de cada bloque

- **Top game:** Este bloque contendrá la máquina de estados y en él se encontrará todo el conexionado entre bloques. La máquina controlará el movimiento del Floppy y de las columnas.
- **Driver VGA:** El Driver está formado por contadores y comparadores horizontales y verticales que se encargarán de medir los tiempos en los que el rayo está pintando la pantalla.
- **Inside floppy:** Se encargará de saber si se debe pintar o no el floppy mediante una serie de comparaciones.
- **Inside pipe:** Se encargará de saber si se debe pintar o no las columnas mediante una serie de comparaciones.
- **Random:** Se encargará de generar un número aleatorio para que, de ese modo, generar columnas aleatorias. Para ello, se empleará un registro de desplazamiento retroalimentado con una puerta XOR.
- **Pipe hitbox:** Bloque que gestionará las columnas, tanto su movimiento cómo dibujarlas.
- **Artist:** Será el encargado de decirle al Driver qué debe pintar en cada momento, es decir, columnas, floppy o bien el fondo.

3.2. Entidad de cada bloque

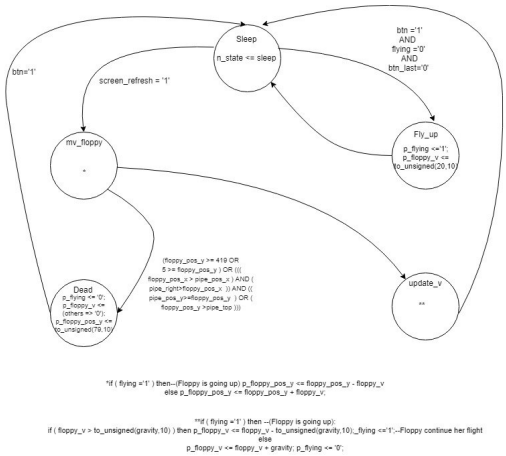
3.2.1. Top game

Top game	
Descripcion	Este bloque será el top del juego, el cuál contiene la máquina de estados.

Entidad	`entity top_game is`
	`Port ( clk : in STD_LOGIC;`
	`rst : in STD_LOGIC;`
	`btn : in STD_LOGIC;`
	`hs : out STD_LOGIC;`
	`vs : out STD_LOGIC;`
	`config0 :in STD_LOGIC;`
	`config1 : in STD_LOGIC;`
	`R_out : out STD_LOGIC_VECTOR(2 downto 0);`
	`G_out : out STD_LOGIC_VECTOR(2 downto 0);`
	`B_out : out STD_LOGIC_VECTOR(1 downto 0));`
	`end top_game;`

Descripción de puertos	
clk	Señal de reloj
rst	Reset asíncrono activo a nivel bajo
btn	Botón de la FPGA que controla el movimiento
HS	Señal de sincronismo horizontal
VS	Señal de sincronismo vertical
config0	Switch asociado a
config1	Switch asociado a
R out	Salida de rojo del controlador VGA
G out	Salida de verde del controlador VGA
B out	Salida de azul del controlador VGA

A continuación, podemos ver el diagrama de bolas que gestiona el movimiento del floppy. Cómo podemos ver, se ha reducido la máquina de estados a 5 estados: El estado de Reposo (Sleep), actualizar posición del floppy (mv floppy), actualizar velocidad del floppy (update v), el estado de salto (fly up) y por último, el estado de muerte (dead).
Se han introducido las condiciones para saltar entre un estado y otro en el esquema.



3.2.2. Driver VGA

Driver VGA	
Descripcion	Este bloque será el encargado de pintar la pantalla.

Entidad	`entity VGA is`
	`Port ( clk : in STD_LOGIC;`
	`rst : in STD_LOGIC;`
	`R : in STD_LOGIC_VECTOR ( 2 downto 0 );`
	`G : in STD_LOGIC_VECTOR ( 2 downto 0 );`
	`B : in STD_LOGIC_VECTOR ( 1 downto 0 );`
	`x_out : out UNSIGNED ( 9 downto 0 );`
	`y_out : out UNSIGNED ( 9 downto 0 );`
	`screen_refresh : out STD_LOGIC;`
	`HS : out STD_LOGIC;`
	`VS : out STD_LOGIC;`
	`RED_out : out STD_LOGIC_VECTOR ( 2 downto 0 );`
	`GRN_out : out STD_LOGIC_VECTOR ( 2 downto 0 );`
	`BLUE_out : out STD_LOGIC_VECTOR ( 1 downto 0 ) );`
	`end VGA;`

Descripción de puertos	
clk	Señal de reloj
rst	Reset asíncrono activo a nivel bajo
R	Entradas de tres bits asociada al color rojo
G	Entradas de tres bits asociada al color verde
B	Entradas de dos bits asociada al color azul
x out	Salida del contador horizontal
y out	Salida del contador vertical
screen refresh	señal de refresco de pantalla [2]
HS	Señal de sincronismo horizontal
VS	Señal de sincronismo vertical
R out	Salida de rojo del controlador VGA
G out	Salida de verde del controlador VGA
B out	Salida de azul del controlador VGA

3.2.3. Inside floppy

Inside Floppy	
Descripcion	Este bloque le dirá al driver cuándo debe el floppy

Entidad	`entity inside_fl0ppy is`
	`Port ( x : in unsigned ( 9 downto 0 );`
	`y : in unsigned ( 9 downto 0 );`
	`pos_x : in unsigned ( 9 downto 0 );`
	`pos_y : in unsigned ( 9 downto 0 );`
	`right : in unsigned( 9 downto 0 );`
	`top : in unsigned( 9 downto 0 );`
	`inside : out STD_LOGIC;`
	`index : out unsigned ( 9 downto 0));`
	`end inside_fl0ppy;`

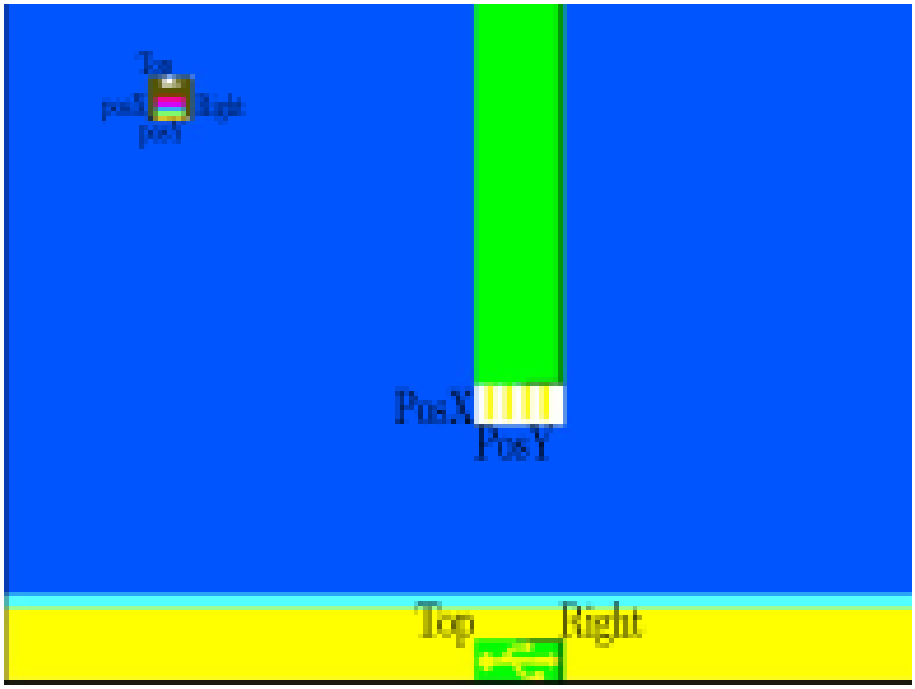
Descripción de puertos	
x	Posición X por la que va pintando la pantalla
y	Posición Y por la que va pintando la pantalla
pos x	Posición X de la hitbox del floppy
pos y	Posición Y de la hitbox del floppy
right	Esquina inferior derecha de la hitbox del floppy
top	Esquina superior izquierda de la hitbox del floppy
inside	Señal lógica que nos dirá en qué momento pintar el floppy
index	Dirección de la memoria RAM donde accederemos

3.2.4. Inside pipe

Inside Pipe	
Descripcion	Este bloque le dirá al driver cuándo debe pintar las columnas

Entidad	`entity inside_pipe is`
	`Port ( x : in unsigned ( 9 downto 0 );`
	`y : in unsigned ( 9 downto 0 );`
	`pos_x : in unsigned ( 9 downto 0 );`
	`pos_y : in unsigned ( 9 downto 0 );`
	`right : in unsigned( 9 downto 0 );`
	`top : in unsigned( 9 downto 0 );`
	`inside : out STD_LOGIC;`
	`index : out unsigned ( 11 downto 0));`
	`end inside_pipe;`

Descripción de puertos	
x	Salida del contador horizontal que nos dirá la posición X por la que va pintando la pantalla
y	Salida del contador vertical que nos dirá la posición Y por la que va pintando la pantalla
pos x	Posición X del pipe
pos y	Posición Y del pipe
right	Right position pipe hitbox
top	Top position pipe hitbox
inside	señal lógica que nos dirá en qué momento pintar la columna
index	Dirección de memoria RAM



Debido a las limitaciones del simulador en línea empleado, la imagen que muestra la pantalla del juego no es totalmente apropiada. Sin embargo, sí se puede distinguir a qué punto de los objetos se ha asociado cada variable.
En el caso del floppy, las variables *posX* y *posY* corresponden a la esquina inferior izquierda. La variable *Right*, corresponderá a la esquina inferior derecha y, por último, la variable *Top*, corresponderá a la esquina superior izquierda.
Por último, en el caso de las columnas, vemos cómo las variables *posX* y *posY* están asociadas a la esquina izquierda de la parte superior de la columna. Las variables *Top* y *Right* estarán asociadas a la esquina izquierda y derecha respectivamente de la columna inferior.

3.2.5. Generador de números aleatorios

Random	
Descripcion	Este bloque generará números aleatorios

Entidad	`entity rng is`
	`Port ( clk : in STD_LOGIC;`
	`rst : in STD_LOGIC;`
	`seed : in STD_LOGIC_VECTOR( 9 downto 0);`
	`random : out STD_LOGIC_VECTOR ( 9 downto 0));`
	`end rng;`

Descripción de puertos	
clk	Señal de reloj
rst	Reset
seed	Valor inicial de reset
random	Número aleatorio generado

3.2.6. Pipe Hitbox

Pipe hitbox	
Descripcion	Este bloque será la hitbox de las columnas

Entidad	`entity pipe_hitbox is`
	`Generic( default_pos_x : INTEGER := 500;`
	`default_pos_y : INTEGER := 100;`
	`default_gap : INTEGER);`
	`Port ( clk : in STD_LOGIC;`
	`rst : in STD_LOGIC;`
	`sr : in STD_LOGIC;`
	`rnd_number : in STD_LOGIC_VECTOR (9 DOWNTO 0);`
	`pipe_v : in unsigned ( 9 downto 0 );`
	`pipe_pos_x : out unsigned ( 9 downto 0 );`
	`pipe_pos_y : out unsigned ( 9 downto 0 );`
	`pipe_right : out unsigned( 9 downto 0 );`
	`pipe_top : out unsigned( 9 downto 0 ));`
	`end pipe_hitbox;`

Descripción de genéricos	
default pos x	Valor de la posición X por defecto
default pos y	Valor de la posición Y por defecto
default gap	Valor del ancho por defecto

Descripción de puertos	
clk	Señal de reloj
rst	reset asíncrono
sr	Señal de refresco de pantalla
pipe v	Velocidad de las columnas
pipe pos x	Posición X del pipe
pipe pos y	Posición Y del pipe
pipe right	Right position pipe hitbox
pipe top	Top position pipe hitbox

3.2.7. Artist

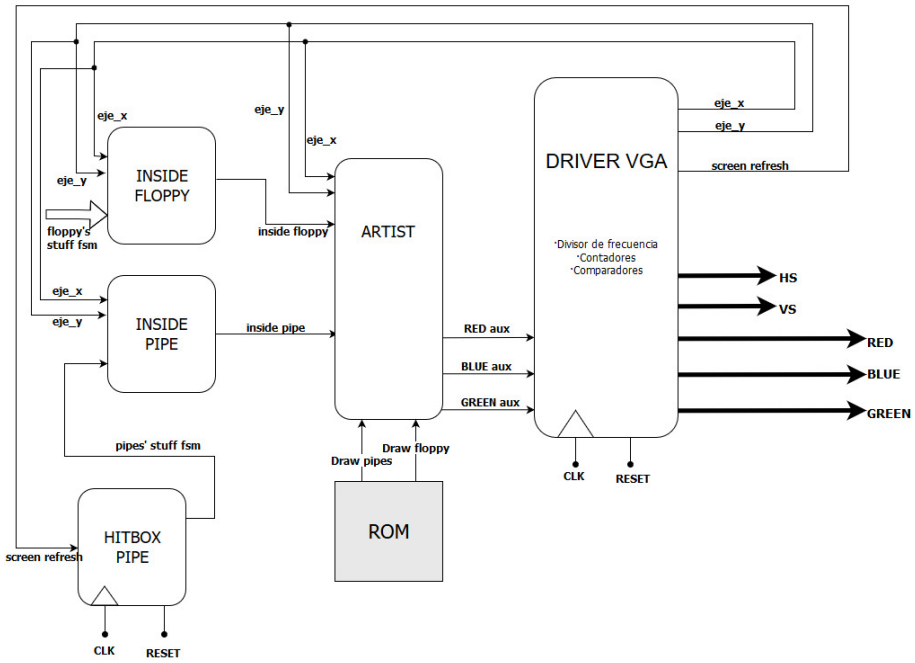
Artist	
Descripcion	Este bloque será el encargado de gestionar qué se pinta por pantalla.

Entidad	`entity artist is`
	`Port ( clk : in STD_LOGIC;`
	`en : in STD_LOGIC;`
	`inside_floppy : in STD_LOGIC;`
	`inside_pipe : in STD_LOGIC;`
	`x : in unsigned ( 9 downto 0 );`
	`y : in unsigned ( 9 downto 0 );`
	`pipe_index : in unsigned(11 downto 0);`
	`floppy_index : in unsigned(9 downto 0);`
	`R : out STD_LOGIC_VECTOR( 2 downto 0);`
	`G : out STD_LOGIC_VECTOR( 2 downto 0);`
	`B : out STD_LOGIC_VECTOR( 1 downto 0));`
	`end artist;`

Descripción de puertos	
clk	Señal de reloj
en	Señal de habilitación del bloque
inside floppy	señal lógica que nos dirá si debemos pintar floppy
inside pipe	señal lógica que nos dirá si debemos pintar columna
x	Valor del contador del eje X
y	Valor del contador del eje Y
pipe index	Salida de la ROM con la imagen
floppy index	Salida de la ROM con la imagen
R	Señal que irá al Driver del color rojo
G	Señal que irá al Driver del color verde
B	Señal que irá al Driver del color azul

3.3. Diagrama de conexionado entre bloques

En la figura de a continuación, tenemos el diagrama de conexionado de los bloques declarados.
Está representado el bloque top, el cuál contiene la máquinas de estados que gestiona el control del juego. No se ha representado dicha máquina de estados, sin embargo, si hay entradas y salidas de bloques asociadas a la máquina.
Cómo podemos ver, el driver VGA, está retroalimentado por los bloques programados, ya que es el encargado de pintar la pantalla y a la vez contiene los contadores que nos ayudarán a saber en cada momento en que posición de la pantalla se encuentra el rayo que la pinta.
Por último, se han creado 2 memorias ROM, una para las imágenes que se han introducido para pintar las columnas y otra para las imágenes del floppy.



4. Consumo de recursos de la FPGA

A continuación, haremos un análisis de la utilización de los recursos de la FPGA y comentaremos los warnings que se han obtenido en el código.

Device Utilization Summary				[-]
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Flip Flops	131	1,920	6%	
Number of 4 input LUTs	729	1,920	37%	
Number of occupied Slices	426	960	44%	
Number of Slices containing only related logic	426	426	100%	
Number of Slices containing unrelated logic	0	426	0%	
Total Number of 4 input LUTs	780	1,920	40%	
Number used as logic	729			
Number used as a route-thru	51			
Number of bonded IOBs	15	83	18%	
Number of RAMB16s	3	4	75%	
Number of BUFGMUXs	1	24	4%	
Average Fanout of Non-Clock Nets	3.02			

Cómo se puede ver, el consumo ha sido por debajo del 50 por ciento de media, sin embargo, cabe descatar el comsumo de memorias RAM.
Aunque se hayan empleado sólo 2 en la realización del proyecto, se ha implementado una tercera porque ocupabamos más memoria debido a que las imágenes de las columnas son de 32x64 pixeles.

En cuándo a los warnings que presenta el proyecto, sólo tendremos los warnings asociados a la instanciación de las memorias RAM.

Synthesis Messages - Errors, Warnings, and Infos	New
Xst:2211 - "C:/Documents and Settings/Marco Montes/Mis documentos/Descargas/Floppy_Bird_FINAL_LITE/Floppy_Bird/artist.vhd" line 39: Instantiating black box module <floppy_ROM>.	New
Xst:2211 - "C:/Documents and Settings/Marco Montes/Mis documentos/Descargas/Floppy_Bird_FINAL_LITE/Floppy_Bird/artist.vhd" line 45: Instantiating black box module <pipe_ROM>.	New
Xst:1767 - HDL ADVISOR - Resource sharing has identified that some arithmetic operations in this design can share the same physical resources for reduced device utilization. For improved clock frequency you may try to disable resource sharing.	New
Xst:2169 - HDL ADVISOR - Some clock signals were not automatically buffered by XST with BUFG/BUFR resources. Please use the buffer_type constraint in order to insert these buffers to the clock signals to help prevent skew problems.	New

Por último, cabe destacar que todo el código se encuentra comentado y ahí se puede encontrar información detallada del funcionamiento de cada bloque.