PROYECTO: SISTEMAS ELECTRÓNICOS (FPGA)

-Francisco Javier Román Cortés 54179754B

-Juan de Dios Herrera Hurtado 47562083S

FPGA “Basys 2, Spartan 3E” de Diligent

“Sliding Puzzle”

ÍNDICE

1.-INTRODUCCIÓN

El proyecto que se plantea consiste en, utilizando la FPGA que aparece en la portada (Spartan 3E Basys2), conseguir la implementación del juego “Sliding Puzzle”, del que también se adjunta una foto. Este juego consiste en una serie de piezas desordenadas y un hueco en el tablero, de manera que moviendo las piezas alrededor de dicho hueco, se consiga poner el puzzle del tablero en orden. Esto se quiere adaptar a la programación en FPGA, de manera que usando una serie de módulos (bloques VHDL) en conjunto, se gestione tanto el tablero en sí (que procede de la memoria RAM) como la forma de jugar y el hecho de pintar correctamente cada pieza.

Con esto además se usa un driver VGA para llevar esto a pantalla y ver si el funcionamiento con los botones de la placa es el que se ha comentado del puzzle descrito adaptado a esta placa.

Estos bloques se explican en el apartado **2. Bloques VHDL.**

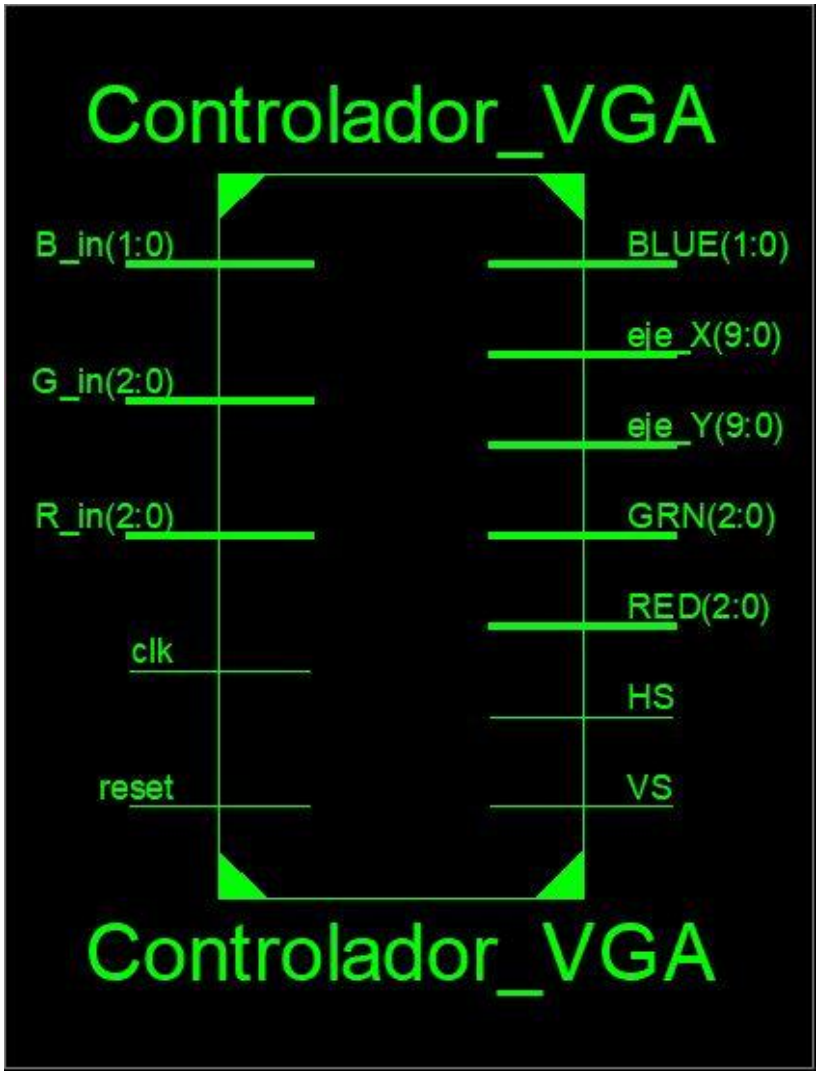
2.-BLOQUES VHDL

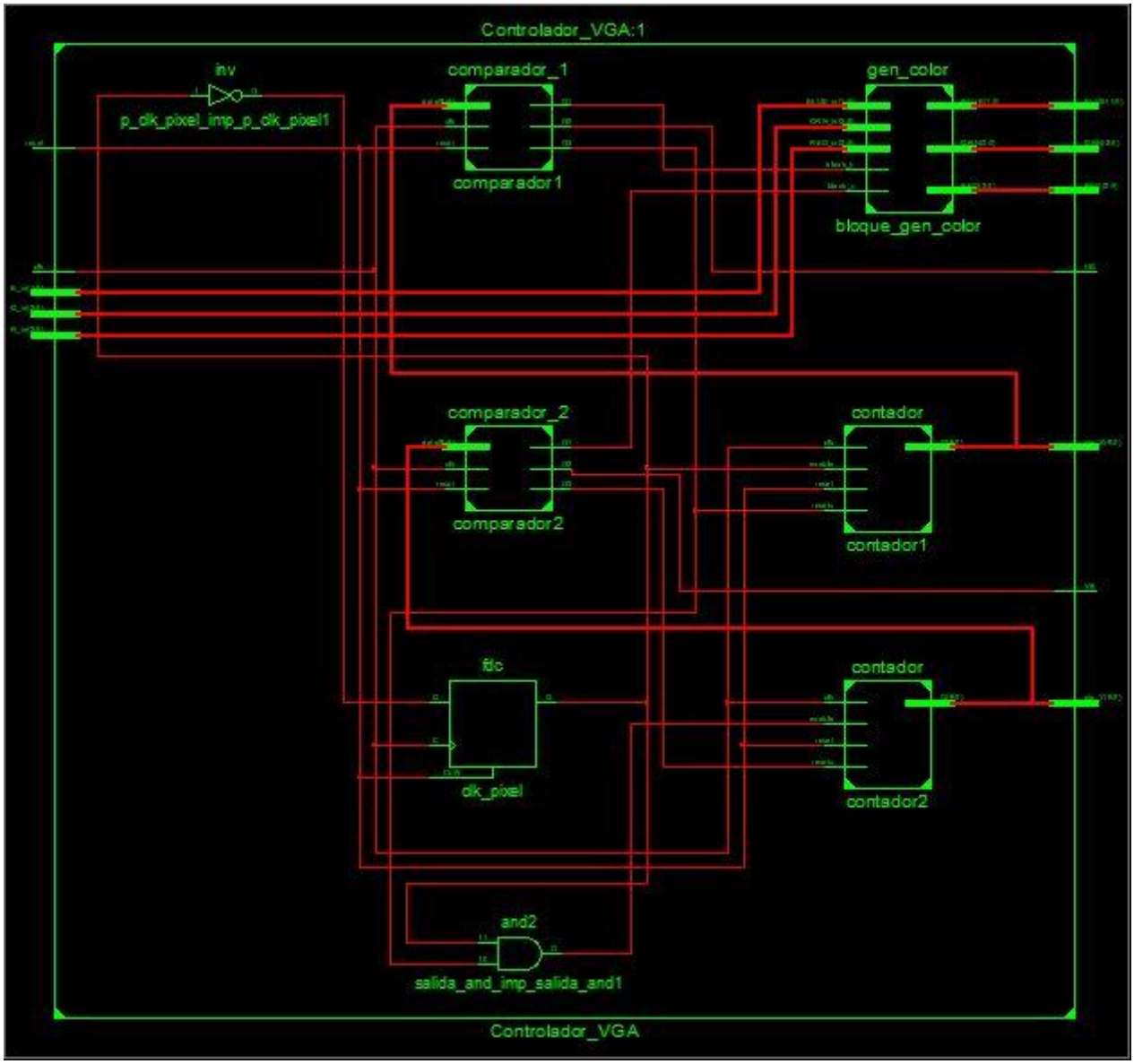
2.1.-ENTRADAS Y SALIDAS

Se adjunta a continuación unas tablas de entradas y salidas para cada bloque VHDL implementado de manera individual.

-Driver VGA:

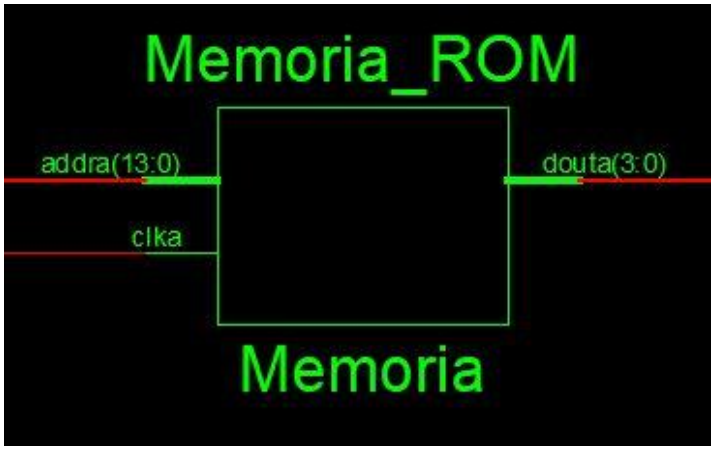
ENTRADAS	SALIDAS
clk (STD_LOGIC)	eje_X (STD_LOGIC_VECTOR (9 downto 0))
reset (STD_LOGIC)	eje_Y (STD_LOGIC_VECTOR (9 downto 0))
R_in (STD_LOGIC_VECTOR (2 downto 0))	HS (STD_LOGIC)
G_in (STD_LOGIC_VECTOR (2 downto 0))	VS (STD_LOGIC)
B_in (STD_LOGIC_VECTOR (1 downto 0))	RED (STD_LOGIC_VECTOR (2 downto 0))
	GRN (STD_LOGIC_VECTOR (2 downto 0))
	BLUE (STD_LOGIC_VECTOR (1 downto 0))




Controlador_VGA:1
inv
p_clk_pixel_imp_p_clk_pixel1
comparador_1
comparador1
gen_color
bloque_gen_color
comparador_2
comparador2
contador
contador1
fdc
clk_pixel
contador
contador2
and2
salida_and_imp_salida_and1
Controlador_VGA

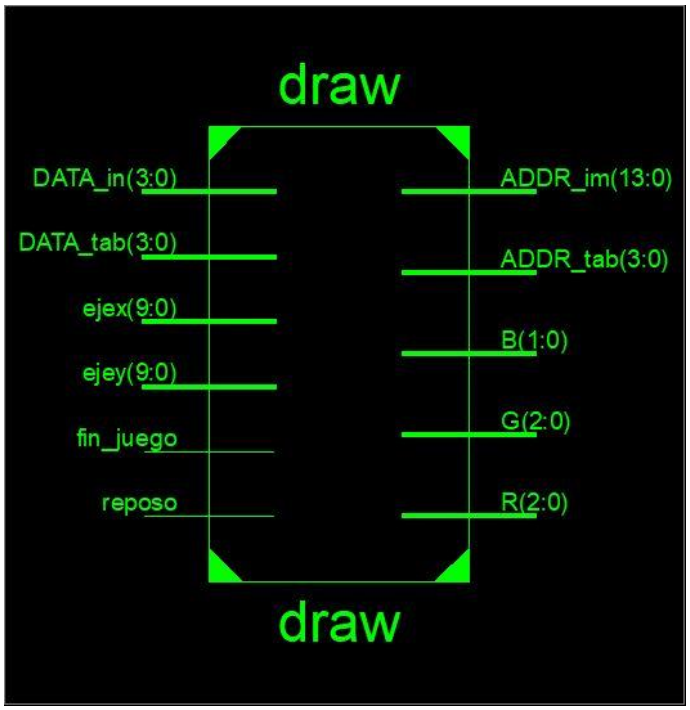
-Memoria ROM:

ENTRADAS	SALIDAS
clka (STD_LOGIC)	douta (STD_LOGIC_VECTOR (3 downto 0))
addra (STD_LOGIC_VECTOR (13 downto 0))	



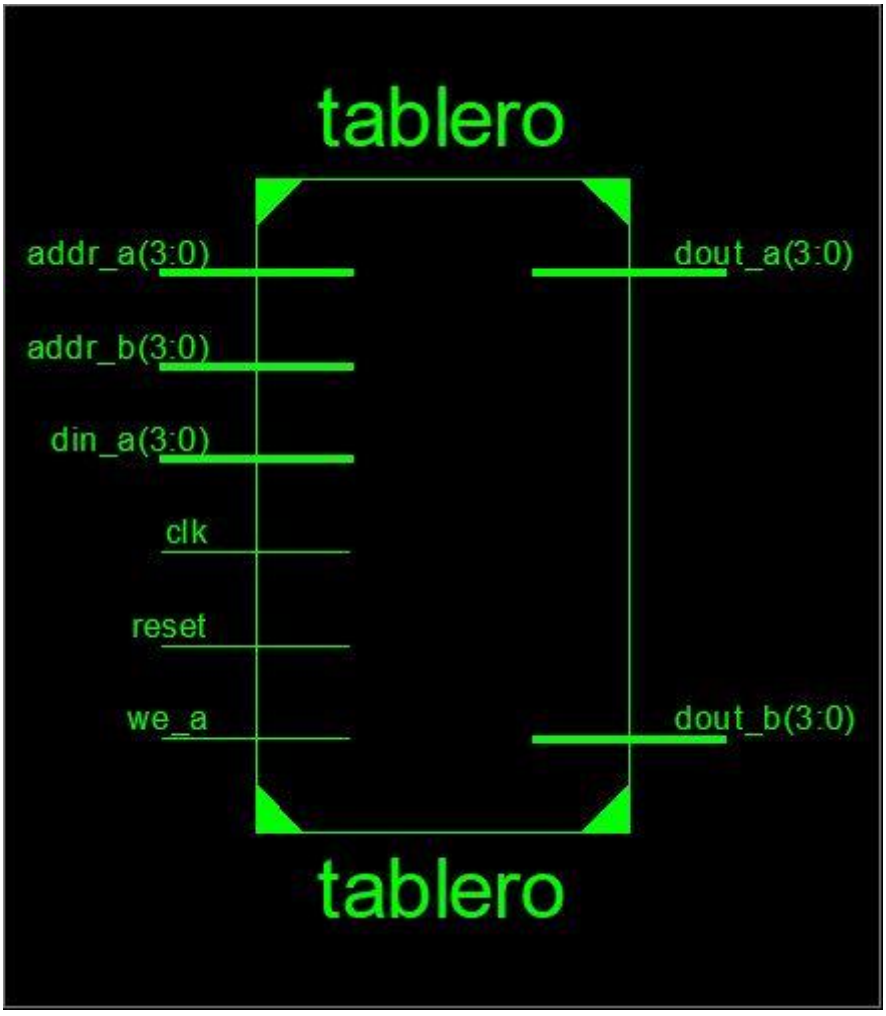
-Draw:

ENTRADAS	SALIDAS
ejex (STD_LOGIC_VECTOR (9 downto 0))	ADDR_tab (STD_LOGIC_VECTOR (3 downto 0))
ejey (STD_LOGIC_VECTOR (9 downto 0))	ADDR_im (STD_LOGIC_VECTOR (13 downto 0))
DATA_tab (STD_LOGIC_VECTOR (3 downto 0))	R (STD_LOGIC_VECTOR (2 downto 0))
DATA_in (STD_LOGIC_VECTOR (3 downto 0))	G (STD_LOGIC_VECTOR (2 downto 0))
fin_juego (STD_LOGIC)	B (STD_LOGIC_VECTOR (1 downto 0))
reposo (STD_LOGIC)	



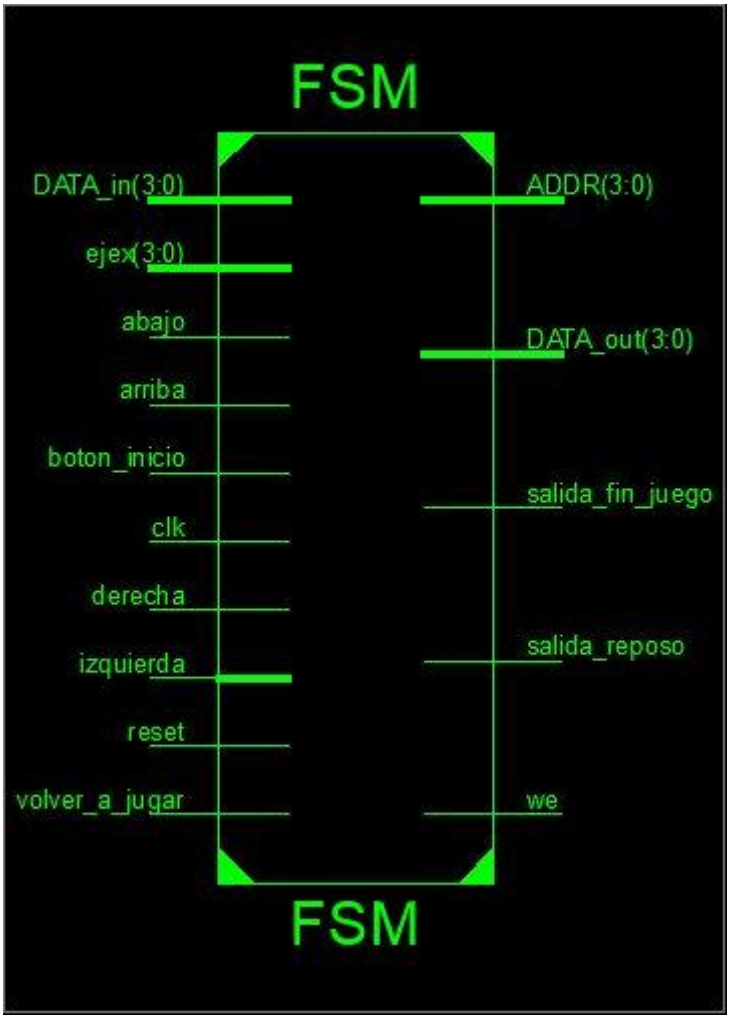
-Tablero (Memoria RAM):

ENTRADAS	SALIDAS
clk (STD_LOGIC)	dout_a (STD_LOGIC_VECTOR (3 downto 0))
reset (STD_LOGIC)	dout_b (STD_LOGIC_VECTOR (3 downto 0))
din_a (STD_LOGIC_VECTOR (3 downto 0))	
addr_a (STD_LOGIC_VECTOR (3 downto 0))	
addr_b (STD_LOGIC_VECTOR (3 downto 0))	
we_a (STD_LOGIC)	



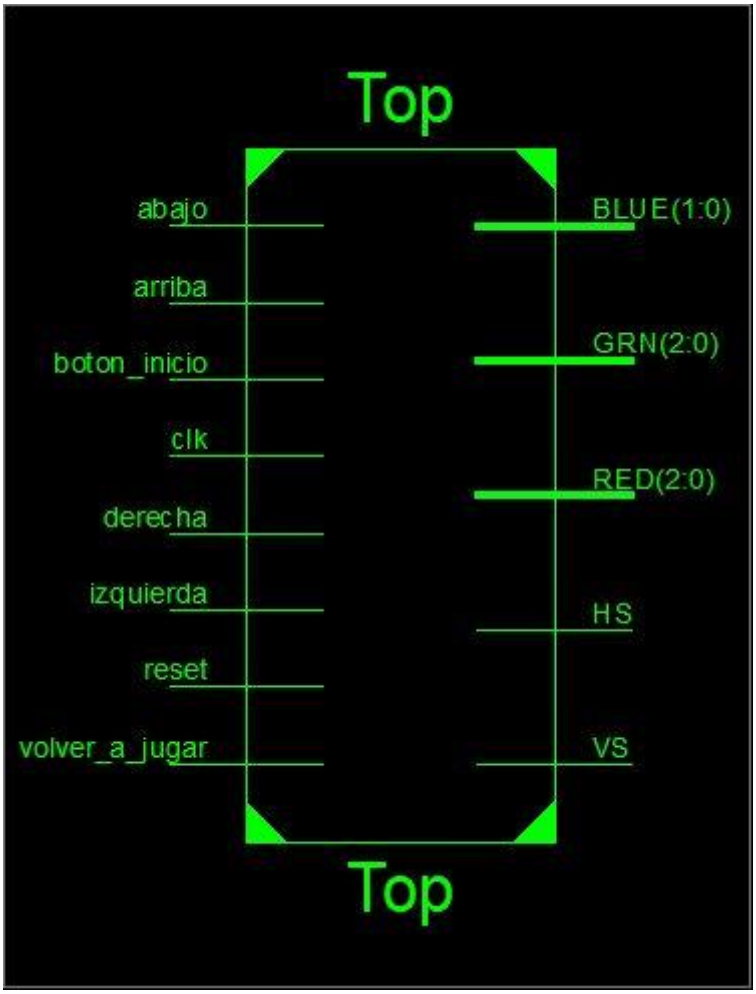
-FSM (Motor del Juego):

ENTRADAS	SALIDAS
clk (STD_LOGIC)	ADDR (STD_LOGIC_VECTOR (3 downto 0))
reset (STD_LOGIC)	DATA_out (STD_LOGIC_VECTOR (3 downto 0))
abajo (STD_LOGIC)	we (STD_LOGIC)
arriba (STD_LOGIC)	salida_fin_juego (STD_LOGIC)
derecha (STD_LOGIC)	salida_reposo (STD_LOGIC)
izquierda (STD_LOGIC)	
DATA_in (STD_LOGIC_VECTOR (3 downto 0))	
boton_inicio (STD_LOGIC)	
ejex (STD_LOGIC_VECTOR (3 downto 0))	
volver_a_jugar (STD_LOGIC)	



-Top (Bloque superior):

ENTRADAS	SALIDAS
clk (STD_LOGIC)	BLUE (STD_LOGIC_VECTOR (1 downto 0))
reset (STD_LOGIC)	GRN (STD_LOGIC_VECTOR (2 downto 0))
abajo (STD_LOGIC)	RED (STD_LOGIC_VECTOR (2 downto 0))
arriba (STD_LOGIC)	HS (STD_LOGIC)
derecha (STD_LOGIC)	VS (STD_LOGIC)
izquierda (STD_LOGIC)	
boton_inicio (STD_LOGIC)	
volver_a_jugar (STD_LOGIC)	



2.2.-Funcionalidad de los bloques

-Driver VGA:

Recibe la información del color procedente del bloque “draw”, con esto pinta la pantalla del color que corresponde y sin sobrepasar los límites definidos, a través del uso de las señales de sincronismo horizontal (HS) y vertical (VS), resultando un cuadrado de 128x128 píxeles. Además, al tener como salidas “eje_X” y “eje_Y”, proporciona al resto del sistema, el lugar exacto por donde se pinta la pantalla en ese momento.

-Memoria ROM:

Memoria ROM generada con “Core Generator” que contiene la información del tablero ordenado que se le ha pasado mediante el archivo “SlidingImage.coe”. En este bloque según la dirección que le llegue de “draw” por el puerto de entrada “addra”, gestionará el color que corresponde al punto de la pantalla que se está pintando, sacando esto por el puerto de salida “douta”.

-Draw:

Este bloque recibe por una parte la información que proporciona el driver en cuanto a la posición en la pantalla mediante las entradas “ejex” y “ejey”. También recibe de la máquina de estados (“FSM”) dos señales “fin_juego” y “reposo” que informan del estado en que se encuentra el juego, y que hará que “draw” pinte toda la pantalla de verde o rojo respectivamente, para indicar dichos estados.

Por otro lado, en caso de no estar activas ni “fin_juego” ni “reposo”, recibe la señal procedente de la salida de la Memoria ROM “douta” por el puerto de entrada “DATA_in”. Para ello por el puerto de salida “ADDR_im”, se le envía a la memoria ROM la dirección que se quiere leer. Dicha señal (DATA_in) se la pasa al driver a través de los puertos de salida “RGB” con la siguiente codificación:

```
R (2) <= DATA_in (3);
R (1 downto 0) <= DATA_in (2) & DATA_in (2);
G <= DATA_in (1) & DATA_in (1) & DATA_in (1);
B <= DATA_in (0) & DATA_in (0);
```

Ahora, con la salida “ADDR_tab”, el bloque “draw” solicita una dirección del “tablero”, enviando una señal al mismo que le entra por “addr_a”. Tras esto, el “tablero” devuelve por el puerto de salida “dout_b”, el valor del azulejo que llega a “draw” por

el puerto de entrada “DATA_tab”. Con esto, el “tablero” informa de qué azulejo hay en esas coordenadas de “ejex” y “ejey” que llegaban desde el driver.

-Tablero (Memoria RAM):

La conexión entre el “tablero” y “draw”, está descrita en el apartado correspondiente al bloque “draw”.

Según si el sistema se encuentra en modo escritura (we_a = ‘1’) o lectura (we_a = ‘0’), el “tablero” funciona de una manera u otra, siendo que esta señal procede del puerto de salida “we” de la “FSM”. Si estamos en modo lectura, la máquina de estados (“FSM”), envía por su puerto de salida “ADDR”, a la entrada del “tablero” “addr_a”, la dirección a leer. Con esto, el “tablero” envía por su puerto de salida “dout_a” el valor del azulejo que se encuentra en esa dirección del “tablero” y que le llega a la “FSM” por el puerto de entrada “DATA_in”.

Si por el contrario se encuentra en modo escritura, la dirección se envía por los mismos puertos que para la lectura, y en este caso, la máquina de estados envía el valor del azulejo que hay que escribir en esa dirección a través del puerto de salida “DATA_out”, y que recibe el “tablero” por la entrada “din_a”.

-FSM (Motor del Juego):

La máquina de estados es la que controla el funcionamiento de todo el juego, gestionando la lectura/escritura en tiempo real del “tablero”. Tiene 4 entradas (“arriba”, “abajo”, “izquierda” y “derecha”) dedicadas al movimiento de las piezas hacia la posición donde está el hueco, en caso de que el movimiento esté permitido (no salga de los límites del tablero de juego).

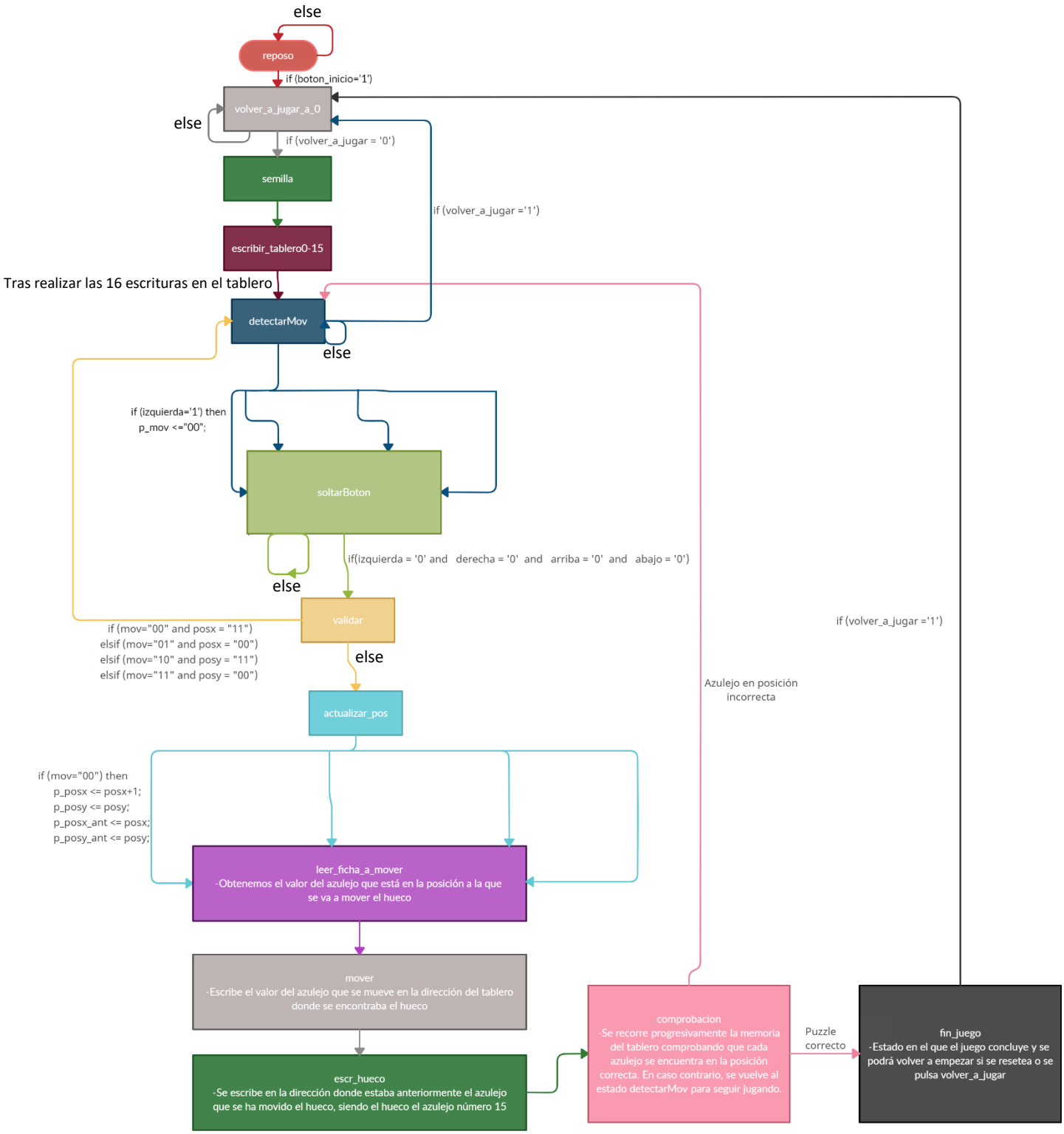
Posee además una entrada “boton_inicio”, para salir del estado de “reposo”.

Además, tiene dos salidas denominadas “salida_fin_juego” y “salida_reposo” para que, como se comentó en el bloque “draw”, se conozca si estamos en alguno de estos estados de “fin_juego” o “reposo” respectivamente, y el bloque “draw” represente las pantallas de inicio y fin si corresponde.

Se muestra a continuación un diagrama de bolas, que representa el funcionamiento de la máquina de estados de manera esquemática.

Finalmente, como ampliación para comenzar con el tablero desordenado, se ha tomado como entrada los 4 bits menos significativos del “ejex” procedente del “driver VGA”, para utilizarlo como semilla de un *Linear Feedback Shift Register* de 4 bits, que proporciona 16 valores aleatorios sin repetición entre 0 y 15. De esta manera se abarcan todos los posibles valores de los azulejos del tablero. Con esto,

introduciendo una serie de estados que lo hacen posible, se consigue que en cada partida se comience con el tablero desordenado de manera diferente.



**** Es importante destacar el hecho de que si en cualquier momento (sin importar en el estado que se encuentre) se activa el “reset”, la máquina de estados vuelve a “reposo”, poniendo todas las variables en sus valores iniciales. ****

-Top (Bloque superior):

Es el bloque de jerarquía superior, cuya función es interconectar entre sí todos los bloques anteriormente descritos. Tiene de nuevo las entradas “arriba”, “abajo”, “izquierda” y “derecha” que están asociadas en el UCF a ciertos botones y cuya información irá a la “FSM”.

También tiene como entradas “clk” y “reset”, que dominan globalmente a los “clk” y “reset” de todos los bloques interconectados en este “Top”.

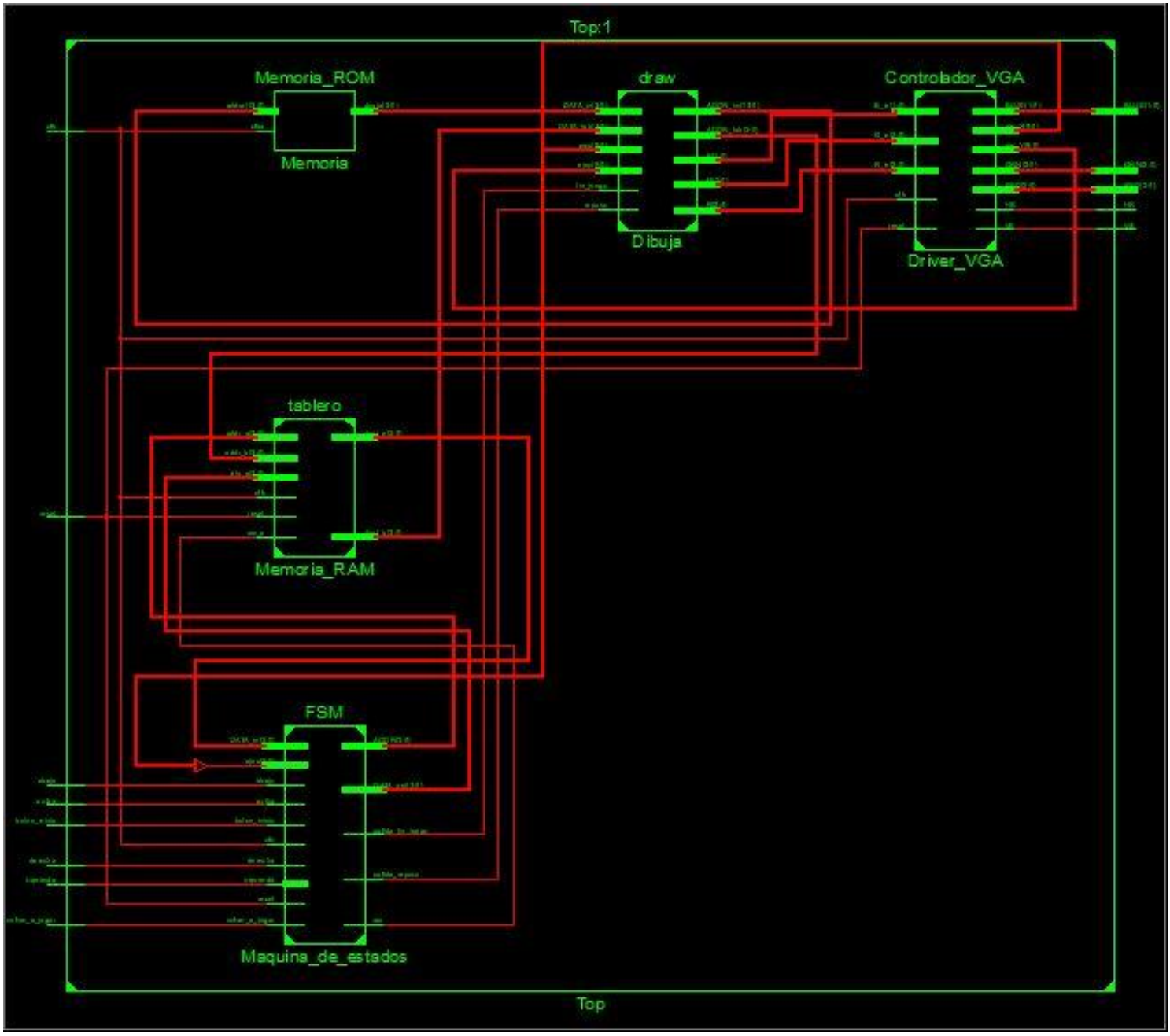
Como última entrada, tiene “boton_inicio” asociada a un switch en la placa, cuya información irá a la entrada “boton_inicio” de la “FSM”.

Además, tiene las 5 salidas necesarias (“RED”, “GRN”, “BLUE”, “HS”, “VS”), conectadas a las salidas del driver, que permiten pintar correctamente la pantalla.

Por último, se tiene una entrada “volver_a_jugar”, que permite volver a desordenar el tablero (de manera diferente a la anterior) una vez se ha resuelto el puzzle correctamente, o bien si durante la resolución del puzzle no se consigue resolver y se activa el switch asignado a dicha señal.

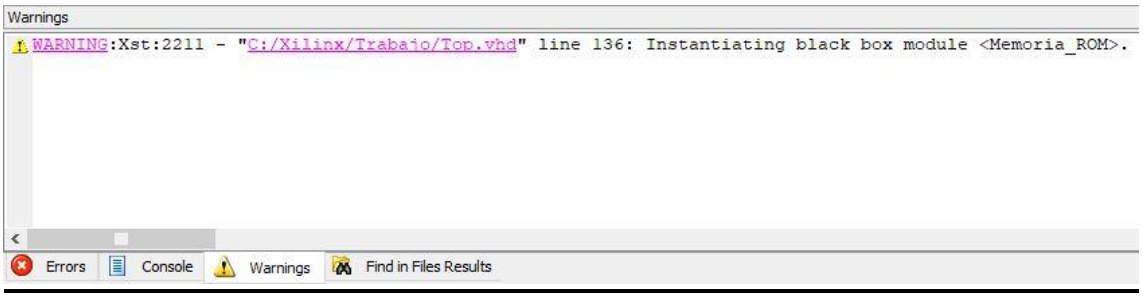
2.3.-DIAGRAMA DE CONEXIONADO COMPLETO

Se adjunta aquí el bloque superior que interconecta el conjunto de bloques implementados. Se trata del bloque “Top” (descrito sin ampliar en la página 9) expandido para ver las conexiones de manera global.



3.-INFORME DE WARNINGS Y RECURSOS CONSUMIDOS

Tras sintetizar cada bloque por separado, ninguno de ellos presentaba warnings. Sin embargo, al sintetizar el “Top”, aparece únicamente el siguiente warning:



Esto aparece al instanciar en el “Top”, la memoria ROM generado con el Core Generator, ya que como se ve en la página de Xilinx, el sintetizador genera ese warning al utilizar esa memoria instanciándola.

Solution

A black box is any instantiated component that is not represented by HDL code, but rather by another netlist format.

Synthesis tools will generally report some kind of warning when a black box (an instantiated component with no associated VHDL code) is detected.

Examples of black boxes include:

- CORE Generator modules
- Instantiated EDIF files
- Instantiated primitives

Fuente: https://www.xilinx.com/support/answers/9838.htm

Recursos consumidos de la FPGA:

Top Project Status			
Project File:	Trabajo.xise	**Parser Errors:**	
Module Name:	Top	**Implementation State:**	Programming File Generated
Target Device:	xc3s100e-5cp132	**• Errors:**	No Errors
Product Version:	ISE 14.7	**• Warnings:**	1 Warning (1 new)
Design Goal:	Balanced	**• Routing Results:**	All Signals Completely Routed
Design Strategy:	Xilinx Default (unlocked)	**• Timing Constraints:**	All Constraints Met
Environment:	System Settings	**• Final Timing Score:**	0 (Timing Report)

Device Utilization Summary				[-]
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Flip Flops	162	1,920	8%	
Number of 4 input LUTs	277	1,920	14%	
Number of occupied Slices	190	960	19%	
Number of Slices containing only related logic	190	190	100%	
Number of Slices containing unrelated logic	0	190	0%	
Total Number of 4 input LUTs	278	1,920	14%	
Number used as logic	277			
Number used as a route-thru	1			
Number of bonded IOBs	18	83	21%	
Number of RAMB16s	4	4	100%	
Number of BUFGMUXs	1	24	4%	
Average Fanout of Non-Clock Nets	3.80			

De esta captura, donde se aprecian los recursos consumidos en la pestaña de “Device Utilization Summary”, observándose que los valores consumidos son bastante bajos en porcentaje, salvo el “Number of RAMB 16s” y el “Number of Slices containing only related logic”.

4.-CONEXIONADO CON LOS PINES REALES DE LA PLACA FPGA

Se adjunta una captura del fichero UCF configurado para seleccionar los pines requeridos de la placa y otra de un esquema de los pines de la FPGA referenciando los pines utilizados:

```
1  NET "clk" LOC = "B8";
2
3  NET "reset" LOC = "P11";
4
5  NET "boton_inicio" LOC = "L3";
6
7  NET "RED<0>" LOC = "C14";
8  NET "RED<1>" LOC = "D13";
9  NET "RED<2>" LOC = "F13";
10
11 NET "GRN<0>" LOC = "F14";
12 NET "GRN<1>" LOC = "G13";
13 NET "GRN<2>" LOC = "G14";
14
15 NET "BLUE<0>" LOC = "H13";
16 NET "BLUE<1>" LOC = "J13";
17
18 NET "HS" LOC = "J14";
19
20 NET "VS" LOC = "K13";
21
22 NET "arriba" LOC = "G12";
23 NET "abajo" LOC = "C11";
24 NET "izquierda" LOC = "M4";
25 NET "derecha" LOC = "A7";
26
27 NET "volver_a_jugar" LOC = "K3";
```

-Señal de reloj

-Señal de reset—Tipo Switch

-Señal para iniciar juego—Tipo Switch

-Bits (3) de salida del color rojo

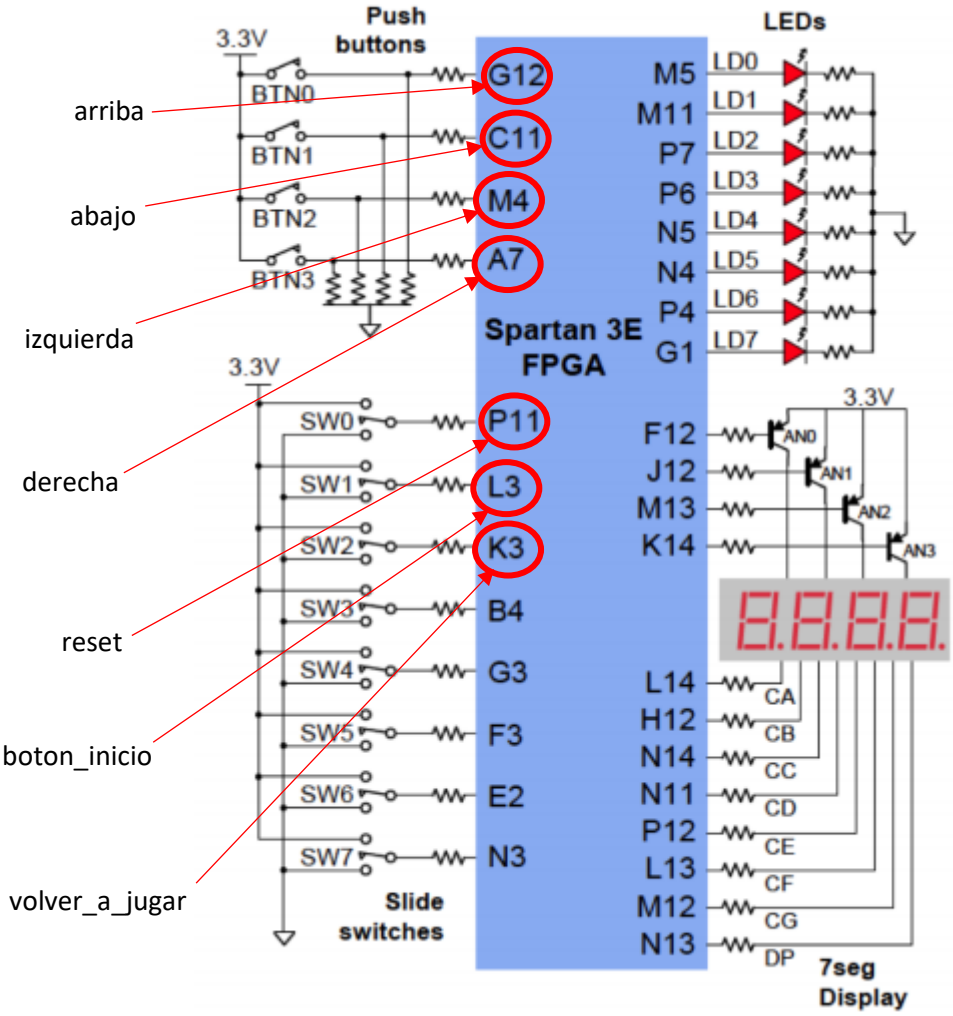
-Bits (3) de salida del color verde

-Bits (2) de salida del color azul

-Señales de sincronismo horizontal y vertical respectivamente

-Señales para controlar el movimiento del juego—Tipo Botón

-Señal para volver a iniciar otra partida una vez resuelto el puzzle correctamente, o durante la resolución si se quiere comenzar de nuevo con otro puzzle—Tipo Switch



5.-RESULTADOS

Como vimos en la práctica del Driver VGA, se pueden realizar simulaciones para comprobar el funcionamiento mediante el script de Python “show_screen.py”, mediante el cual hemos realizado varias simulaciones:

-Pantalla en el estado de reposo:

-Ejemplo de Tablero desordenado, generado tras pulsar “botón_inicio” con el uso del Linear Feedback Shift Register de 4 bits:

-Pantalla que se obtiene como resultado de tener el puzzle correcto, tras haber realizado los movimientos necesarios:

6.-CONCLUSIONES

Es importante gestionar bien la optimización de los bloques usados, ya que, en una aplicación comercial o empresarial, si nos quedamos sin memoria o sin recursos en la placa que se está usando (Basys2), implicaría recurrir a una placa FPGA mejor como, por ejemplo, la Basys3, y por tanto más cara.

Acerca del proyecto, da una buena aproximación a lo que será nuestra vida laboral en un futuro. Con esto queremos decir que nos parece una buena forma de evaluación para prepararnos de manera más práctica al trabajo real, que en cierta manera se echa de menos en cursos anteriores u otras asignaturas, donde el peso recae mayormente en, normalmente, un solo examen.

Con respecto al trabajo de Microcontroladores, me parece un factor positivo tanto el hecho de que aquel trabajo, nos preparaba para la modalidad de trabajo individual, en este caso, este proyecto nos aporta un "aprendizaje" en cuanto al desarrollo de un trabajo o tarea en grupo de varias personas, que es la forma de trabajo más común en el ámbito de la Ingeniería.

Con estos comentarios concluimos esta memoria y procedemos a "firmarla":

Francisco Javier Román Cortés, Estudiante de 3º de GIERM.

Juan de Dios Herrera Hurtado, Estudiante de 3º de GIERM.

Asignatura de Sistemas Electrónicos.

FPGA.