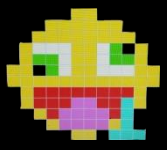
MEMORIA DE PROYECTO:

IMPLEMENTACIÓN DEL JUEGO "SNAKE" EN LA FPGA SPARTAN3E



2010

REALIZADO POR:

- RICHARD HAES ELLIS
- PABLO CAMACHO CARRELLÁN
- JOSE MARÍA ROMERO FALCÓN

GAME OVER

ÍNDICE

IDENTIFICACIÓN DE PROYECTO:

Datos de los alumnos.-

Apellidos y nombre: *Camacho Carrellán, Pablo*.

E-mail: pamcarre3@gmail.com

Apellidos y nombre: *Romero Falcón, Jose María*.

E-mail: joseromeroguetta@gmail.com

Apellidos y nombre: *Haes Ellis, Richard M*.

E-mail: richard9661h@gmail.com

Horario y fecha de entrega.-

Horario: 23:59 horas.

Fecha: Sábado 21 de Enero de 2017.

Profesor de la asignatura.-

Apellidos y nombre: Muñoz Chavero, Fernando

E-mail: fmunoz@gie.us.es

Introducción

A continuación se detallará la realización de nuestro proyecto de implementación del videojuego Snake en la FPGA Spartan3E.

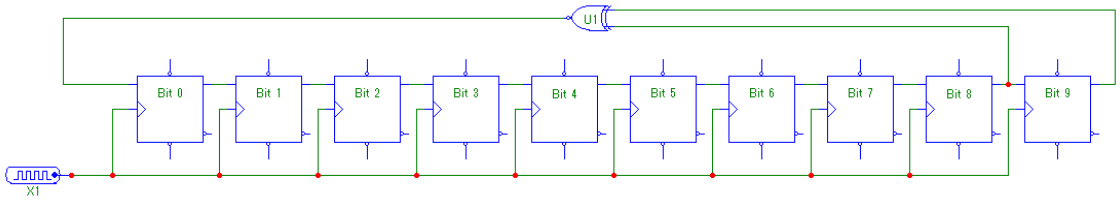
El proyecto tiene una interfaz basada en las máquinas retro arcade recreativas. Se ha procurado llevar a cabo una experiencia gráfica agradable con fluidez de movimiento avanzada.

Para su codificación se ha procedido por bloques, comentando la funcionalidad de cada apartado. Con el pensamiento de realizar un código intuitivo y fácilmente ampliable, teniendo en cuenta las limitaciones y el alcance del proyecto.

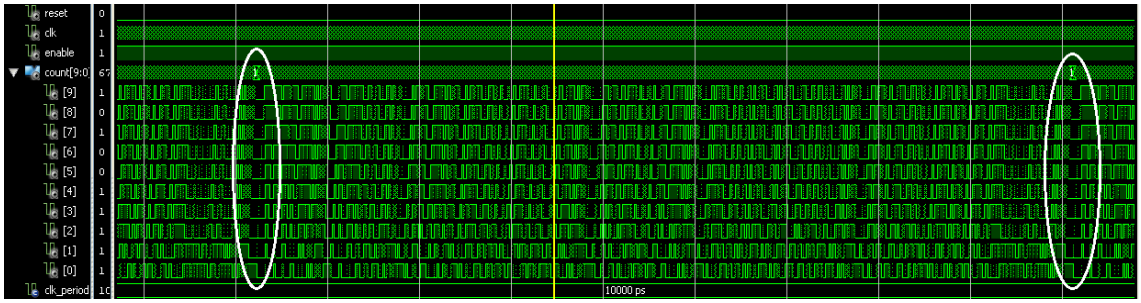
Estructura del Proyecto

El proyecto está compuesto por 9 partes que comentaremos a continuación:

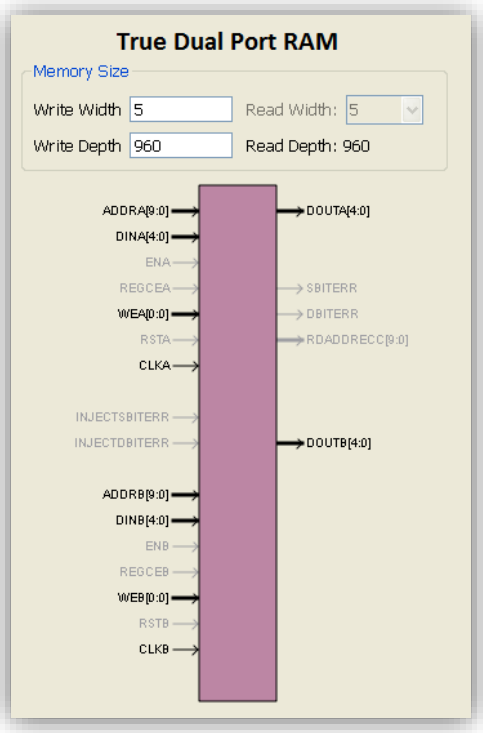
- **LFSR (Linear Feedback Shift Register)**

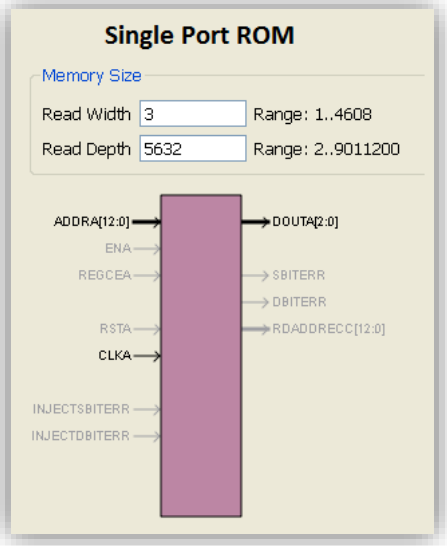


Consiste en un registro de desplazamiento de 10 bits, donde a la entrada tenemos el resultante de la operación lógica XOR del bit 8 y 9 negado. De este modo podemos crear una secuencia periódica de número “aleatorios”. En la siguiente simulación se puede observar la periodicidad de la secuencia, esta periodicidad ocurre cada 900 números aproximadamente lo cual es válida para nuestra aplicación ya que en el tablero tenemos 960 casillas de las cuales la manzana solo podrá ocupar 841 celdas si quitamos los bordes.



- **Memorias**





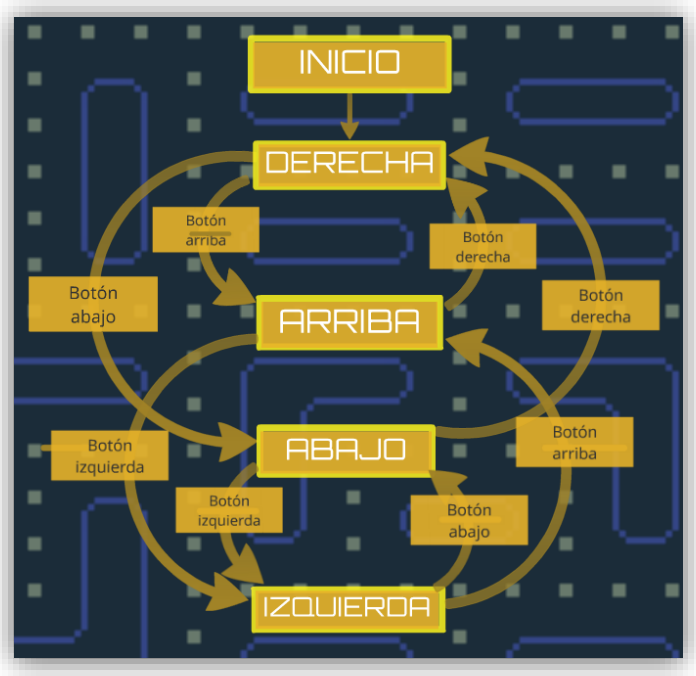
En la RAM tenemos guardado el tablero del juego que resulta ser de 30x32 casillas lo cual hacía falta 960 celadas de memoria, el ancho de palabra será de 5 bits ya que inicialmente tenemos 22 objetos en el juego que se pueden codificar con 5 bits.

Hemos usado un dual port para poder leer y escribir desde dos fuentes a la vez. Mientras la lógica del juego lee y escribe al tablero, el representador lee y representa el tablero en pantalla.

En la ROM tenemos guardados las imágenes necesarias para pintar los diferentes objetos, para ello hemos usado el programa facilitado por el profesor coeGen.jar, las imágenes son de 16x16 con 3 bits de resolución de colores.

El tamaño de palabra de la ROM tendrá que ser de 3 bits y el número de celdas de memoria será 5632 celdas.

- **Máquina de estados de la botonera**

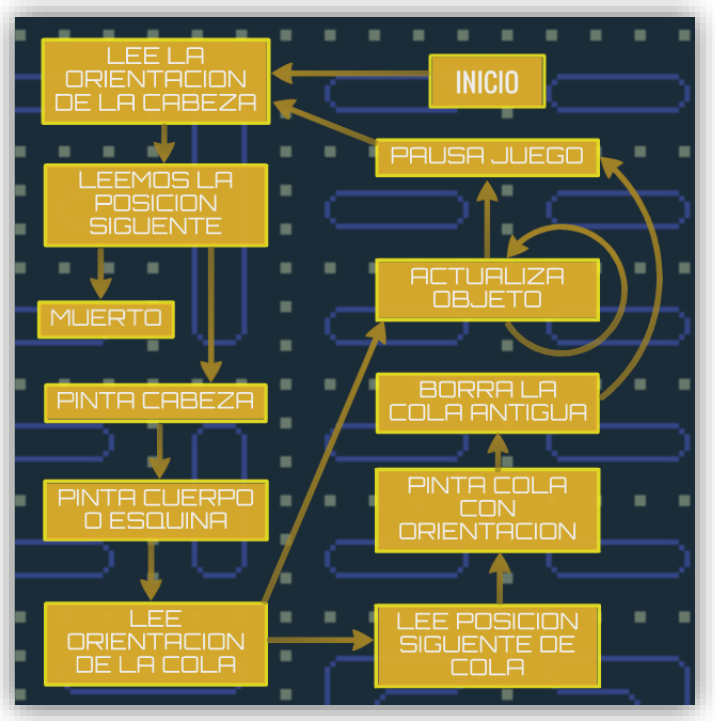


Este módulo se encarga de conectar la interacción humana con la lógica del juego. El usuario pulsará uno de los cuatro botones disponibles para la dirección de la serpiente y la máquina de estados se encargará de evitar que se puedan acceder a posiciones ilógicas en el bloque que controla la dinámica principal.

Para llevar a cabo esta función, se codificaron las diferentes direcciones mediante un vector de salida de dos bits. Se definió una variable “stat” que fuese jugando con el estado actual y el estado nuevo, de manera que en función de la tecla pulsada el nuevo estado será igual que el actual u otro estado nuevo.

En función del estado, mediante la señal “state” se le pasa al programa principal la información necesaria para que éste pueda ejecutar la siguiente acción. En el programa principal esa información se le llama “dir_snake”.

- **Máquina de estados de la lógica del juego**



En este bloque es donde ocurre toda la dinámica del juego, será el único que podrá modificar los contenidos de la RAM (El tablero de juego).

En todo momento se conoce la posición actual, posición anterior, dirección actual, y dirección anterior de la cabeza. Además de eso sabemos la posición actual y anterior de la cola y su dirección

Inicialmente el juego se encuentra parado en el estado de "INICIO" y no pasara al siguiente estado hasta que pulsemos el botón derecho de la FPGA.

En el estado de "LEE DIRECCION CABEZA", actualizamos la variable "dir_act_cbza" asignándole el valor de "dir_snke" y a la variable "dir_act_cbza" le asinamos el valor de "dir_act_cbza". Asi tenemos la dirección actual que la usaremos para calcular la posición siguiente y la dirección anterior que la usaremos para pintar el cuerpo donde estuvo la cabeza anteriormente.

En el estado de "LEER POSICION SIGUENTE", leemos lo que hay en la posición de la cabeza siguente sumándole 1 en la dirección que indique "dir_act_cbza".

Dejamos que pase un ciclo de reloj con el contador "wait_time" para descargar el dato de RAM y evaluamos el dato. Dependiendo del dato le asignamos a la variable "obj" el objeto que hay en esa casilla (vacío, pared, manzana…etc.) y saltamos al siguiente estado.

En el estado de "PINTA CABEZA", escribimos en memoria la cabeza orientada según la variable "dic_act_cbza".

En el estado de "PINTA CUERPO O ESQUINA" trataremos de pintar un cuerpo recto si "dir_act_cbza" y "dir_ant_cbza" son las mismas ó una esquina determinada según la combinación de esas direcciones si fueron diferentes.

En el estado de "LEER ORIENTACION DE COLA", leemos en la posición de la cola esperando un ciclo de reloj para cargar el dato, y le asignamos a la variable "dir_cola" la dirección correspondiente.

En el estado de "LEER POSICION SIGUENTE DE COLA", sabiendo la "dir_cola", leemos la posición siguiente que tiene que tomar la cola y determinamos si habrá que pintar la cola con la misma orientación u otra según si hay una esquina o cuerpo recto. Para ello usamos la variable "side" que tomara los valores de "left","right" o "straight" que será respecto a la cola actual.

En el estado de "PINTA COLA", pintamos la cola en la casilla siguiente de la cola según "dir_cola" y la orientación adecuada según la variable "side".

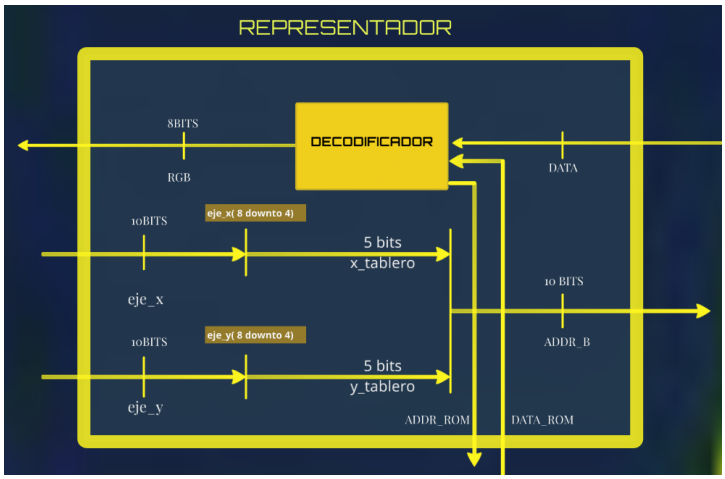
Y finalmente pasamos al estado de espera "DELAY GAME", en este estado se cuenta 15 millones de ciclos de reloj hasta que se vuelva a pasar al estado de "LEE DIRECCION CABEZA".

En el estado de "LEE ORIENTACION DE LA COLA", si resulta que "obj" fue manzana saltamos directamente al estado de "ACTUALIZA OBJETO" y nos saltamos el proceso actualizar la cola, de este modo la cola se queda como está pero la cabeza habrá crecido y así es cómo funciona el crecimiento del snake.

En el estado de "ACTUALIZA OBJETO", comprobamos que la coordenada aleatoria del LFSR es válida, es decir, si está dentro del tablero y si es menor de 960 ya que se estaría leyendo en una posición de memoria no accesible. Con esa coordenada leemos la celda, y si está vacía pintamos la manzana, en caso contrario se le da un pulso de reloj a la señal de enable del LFSR para generar la siguiente coordenada aleatoria.

Una vez actualizada el objeto se salta al estado de "DELAY GAME".

- **Representa**



El bloque representa se encarga de interpretar el contenido que se lee en la RAM, y en función de esto, recoger las imágenes necesarias de la ROM para indicarle al driver VGA los colores que se tienen que pintar en cada momento.

El representador recibe las coordenadas del “eje_x” (10 bits) y del “eje_y” (10 bits), procedentes del driver VGA.

En el caso de que los píxeles que se estén pintando estén fuera del tablero de juego (es decir, que el “eje_x” esté entre 0 y 64 píxeles o sea mayor que 575 o “eje_y” mayor que 479) entonces la salida hacia el driver VGA (“RGB” (8bits)) toma el valor de “00000000” (pinta negro dentro del rango de píxeles mencionado).

En caso de encontrarse fuera de dicho rango, restamos 64 a “eje_x” (para centrar el tablero de juego) y lo almacenamos en un vector auxiliar “x_tablero_aux” (10 bits), el cual dividimos por 16 cogiendo del bit 8 al 4 y almacenándolo a su vez en otro vector auxiliar “x_tablero” (10 bits) llenando el resto del vector con ceros. Para el “eje_y” realizamos el mismo procedimiento, dividimos por 16 “eje_y”, almacenamos en vector auxiliar “y_tablero” (10 bits) asignando al resto del vector ceros.

Con ello habremos obtenido las coordenadas del tablero “x_tablero” e “y_tablero”.

Tanto “x_tablero” como “y_tablero” los asignamos al vector “Addr_RAM” (10 bits), que será la dirección de la RAM (El “y_tablero” será los 5 bits más significativos y el “x_tablero” los 5 bits menos significativos de “Addr_RAM”). De ésta obtenemos el dato de la celda correspondiente “Data_RAM” (5 bits) que decodificamos para obtener la dirección de la imagen que hay que pintar y lo almacenamos en otro vector auxiliar “dir_imagen” (13 bits). Estos valores va de 256 en 256 (Imágenes de

16x16 pixeles), apuntando al inicio de cada imagen almacenada anteriormente en la ROM.

Para acceder al pixel exacto de cada imagen de ROM elaboramos la siguiente fórmula:

“Addr_ROM_aux <= dir_imagen + ((unsigned (eje_y)-(y_tablero sll 4)) sll 4) + (unsigned (eje_x)-64-(x_tablero sll 4)) -1;”

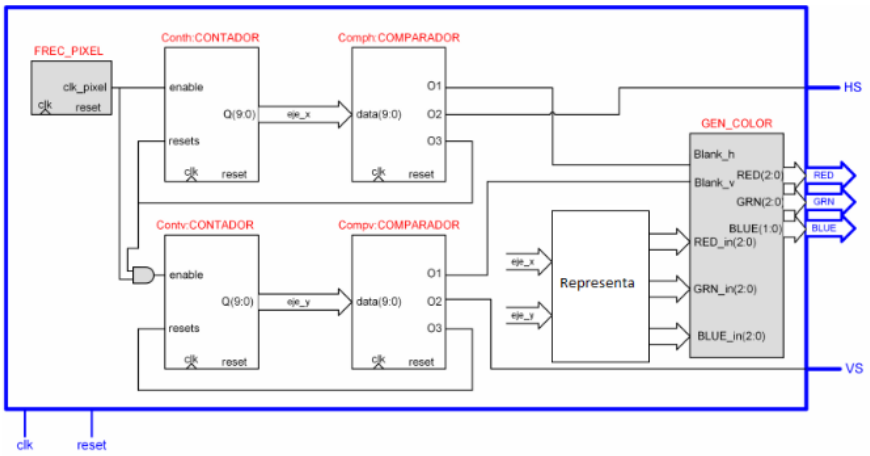
Como se puede observar, la anteriormente citada fórmula se compone de tres partes diferenciadas:

1. “Dir_imagen”: Será la dirección de comienzo de cada imagen en la ROM.
2. “((unsigned (eje_y)-(y_tablero sll 4)) sll 4)”: Cogiendo el eje_y y restándole la coordenada y del inicio de la celda correspondiente obtenemos la fila de la imagen, y como en ROM las filas van de 16 en 16, lo multiplicamos por 16.
3. “(unsigned (eje_x)-64-(x_tablero sll 4))”, De la misma forma que antes obtenemos el número de la columna correspondiente y como en ROM las columnas van de 1 en 1 no hace falta multiplicarlo por un escalar.
4. *”-1”: trata de minimizar el efecto de la limitación técnica de la velocidad de reloj y los acceso a ROM y RAM.

Una vez obtenida la dirección del pixel exacto de cada imagen, la ROM nos devolverá un vector de 3 bits “Data_ROM” que decodificamos para asignar al vector “RGB” el color correspondiente, esta luego conecta a la entrada RGB del VGA driver.

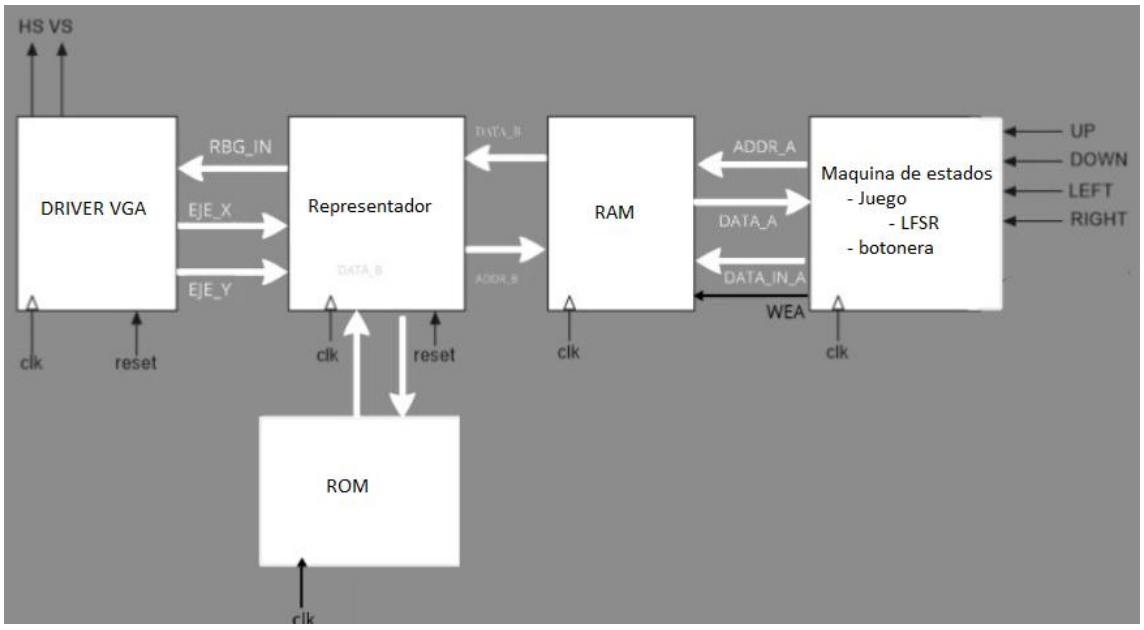
Por último, aunque no está implementado en el juego, creamos una sentencia en función de una variable (“LSD”), la cual, si está activa, cambia el decodificador anterior por otro del cual salen colores totalmente distintos.

- **Driver VGA**



El driver VGA es la misma que hemos usado en las prácticas de la asignatura, con la excepción del bloque dibuja que sería reemplazado por el bloque Representa.

- **TOP**



En el TOP tenemos declarado, instanciado e interconectado cada uno de los bloques tal y como se muestra en la imagen anterior. El LFSR está declarada e instanciada dentro de la máquina de estados del juego.

Simulaciones

Además de usar el simulador ISim, hemos hecho uso del simulador VGA de Eric Eastwood (http://ericeastwood.com/blog/8/vga-simulator-getting-started) que permite representar en la propia página web las imágenes que se representaría con la señales de la simulación del VGA en el ISim (R,G,B,VS,HS) de este modo podemos comprobar sin usar la placa que todo esté funcionando con normalidad.

Para poder usar aquella herramienta hemos guardado los valores de aquellas señales en un fichero .txt en cada instante de tiempo con el siguiente formato

[Tiempo_de _simulación unidades_de_tiempo: HS VS Red Green Blue]

Este fichero luego se sube a la página web y se especifica las especificaciones de la pantalla virtual para que se represente de manera correcta.

Limitaciones Técnicas

Dado que el reloj de la placa es de 50MHz y la frecuencia a la que se pinta cada pixel es de 25Mhz no deja precisamente 2 ciclos de reloj para que el bloque Representa lea de la RAM, lea de la ROM y con eso pinte el pixel correspondiente.

Como sabemos con leer la RAM y la ROM necesitamos dos ciclos de reloj luego al pintar, el tablero se pinta con un pixel de retraso, por ello hemos restado 1 a la fórmula de la dirección de ROM para minimizar el efecto de retraso.

Limitaciones de memoria, técnicas que se podrían usar para mejorar el código y la eficacia

Warnings

```
- "//vboxsrv/c_drive/Users/Richa/Xilinx stuff/SNAKE_GAME/TOP_MODULE.vhd" line 141: Instantiating black box module <ROM>.
- "//vboxsrv/c_drive/Users/Richa/Xilinx stuff/SNAKE_GAME/TOP_MODULE.vhd" line 142: Instantiating black box module <RAM>.
 Signal <y_tablero<9:6>> is assigned but never used. This unconnected signal will be trimmed during the optimization process.
 Signal <x_tablero_aux<9>> is assigned but never used. This unconnected signal will be trimmed during the optimization process.
 Signal <x_tablero_aux<3:0>> is assigned but never used. This unconnected signal will be trimmed during the optimization process.
 Signal <x_tablero<9:6>> is assigned but never used. This unconnected signal will be trimmed during the optimization process.
- Output <lsd_en> is never assigned. Tied to value 0.
 Signal <colas_borradas<0>> is assigned but never used. This unconnected signal will be trimmed during the optimization process.
- Node <obj_3> of sequential type is unconnected in block <MAQUINA_JUEGO>.
- Node <obj_5> of sequential type is unconnected in block <MAQUINA_JUEGO>.
- Node <obj_1> of sequential type is unconnected in block <MAQUINA_JUEGO>.
- Node <obj_0> of sequential type is unconnected in block <MAQUINA_JUEGO>.
- Node <obj_2> of sequential type is unconnected in block <MAQUINA_JUEGO>.
- Node <side_1> of sequential type is unconnected in block <MAQUINA_JUEGO>.
```

En primer lugar tenemos los Warnings correspondientes a las instancias de los bloques de memoria ROM y RAM.

Luego tenemos unos Warnings asociados a las variables del tablero del bloque representa y nos advierte de que no están conectados o que no se usan, lo mismo ocurre con las variables de objeto side y colas borradas, esto no supone ningún inconveniente ya que el sintetizador es capaz de recortar aquellas señales para que no consuma recursos innecesarios.

Conclusión

Tras la realización del proyecto hemos podido comprobar que el uso de FPGAs para la implementación de un videojuego no es eficiente, pero mediante éste hemos conseguido incrementar nuestra curva de aprendizaje rápidamente.

También hemos podido apreciar la diferente dinámica de trabajo, al realizar un proyecto en grupo con respecto al trabajo en las prácticas. Es un punto a tener en cuenta, ya que se debe adaptar la codificación a

las limitaciones que surgen al unir dos códigos creados con una estructura de pensamiento diferente.

Este proyecto también ha sido de gran ayuda de cara al aprendizaje en cuanto al desarrollo de un trabajo minimizando errores. Para esto ha sido primordial la realización de pruebas mediantes simulaciones, ya que permite trabajar de manera independiente y ahorrar futuros problemas al unir los diferentes módulos. Incluso se ha hecho uso de herramientas externas a la asignatura debido a la imposibilidad de implementarlo directamente en la placa, lo que nos ha permitido apreciar el potencial de este tipo de programación más allá de lo aprendido.

En definitiva, este ha sido un interesante proyecto en el que se nos ha evaluado tanto el conocimiento adquirido como nuestra capacidad de expresión oral, lo que nos será de gran ayuda de cara al futuro. Trabajar de manera directa con el código para el desarrollo de un proyecto, ha sido una experiencia didáctica con la que sentimos que realmente hemos adquirido conocimiento directamente aplicable.