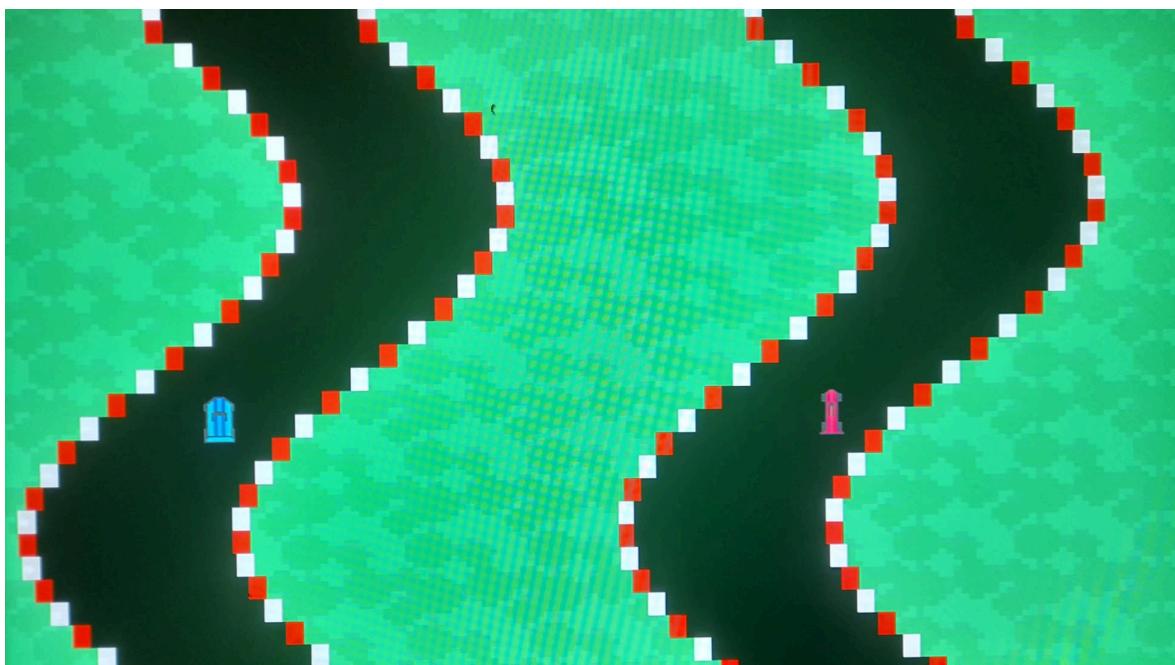


PROYECTO

“MARIO KART”

FPGA

Sistemas Electrónicos GIERM



José Chaqués Torres
Fernando Román Hidalgo
Marta Barroso Infante

ÍNDICE:

- 1. Introducción.**
- 2. Diagrama de bloques.**
- 3. VGA DRIVER.**
- 4. Gestionamiento del juego.**
- 5. Dibujo del juego.**
- 6. Comunicación por puerto serie.**
- 7. Warnings.**
- 8. Recursos utilizados.**
- 9. Conclusiones.**

INTRODUCCIÓN

OBJETIVO DEL PROYECTO

La finalidad de la realización de este proyecto es poner en práctica todos los conocimientos obtenidos en la parte de la asignatura de Sistemas Electrónicos relacionada con el contenido de FPGA.

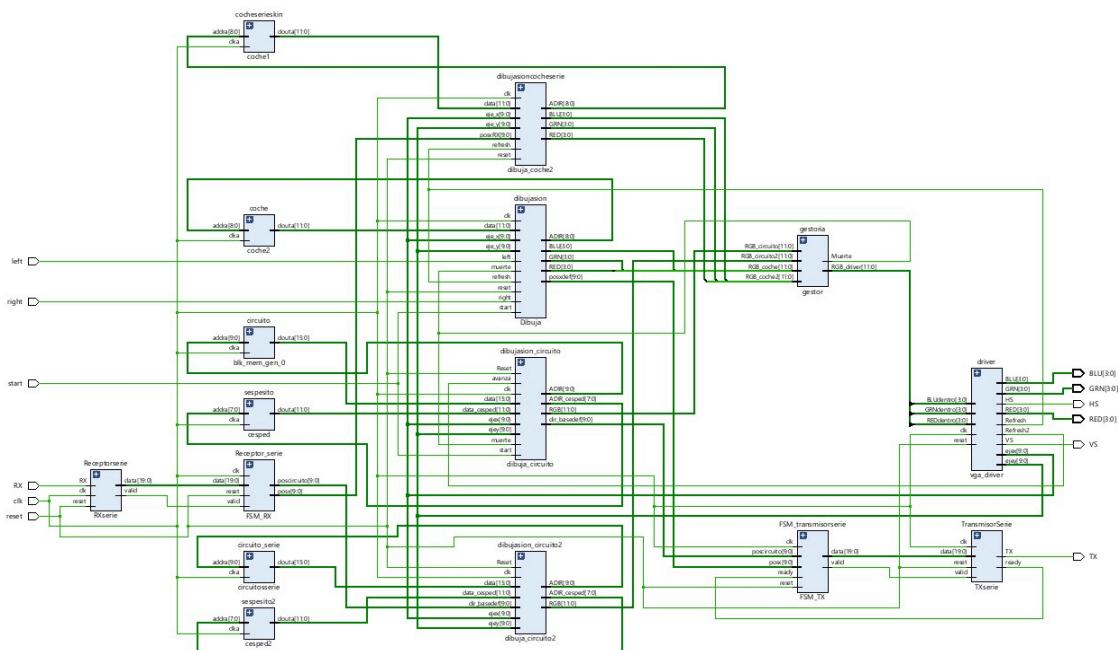
Para verlo reflejado, se ha tratado de realizar un juego similar al conocido videojuego “Mario Kart”. Se trata de presentar en pantalla un circuito con carretera y césped por el que va circulando un coche que debe intentar no salirse de la zona permitida para circular, es decir, la carretera. Además, también se quiere que el juego pueda ser multijugador, compitiendo con un adversario.

Este proyecto tiene infinidad de posibilidades y de distintas maneras creativas de ampliarlo y mejorarlo. En esta memoria además de explicar cómo se han codificado los distintos bloques que comprenden el programa, se verá reflejado cuál ha sido el alcance realizado por este grupo.

DIAGRAMA DE BLOQUES

“PADRE”

El schematic proporcionado por el software utilizado es el que se muestra en la imagen inferior. Todas las conexiones entre las diferentes partes del trabajo se realizan en el archivo *padre.vhd*. Ver el esquema ha servido de gran utilidad a la hora de solucionar algunos de los problemas que han surgido a lo largo del proceso de trabajo.



VGA DRIVER

Este bloque es la base con la que se ha partido para realizar el trabajo. Se ha usado el código realizado en la segunda práctica de la asignatura haciendo modificaciones.

El cambio más notable es que se ha extraído el componente de dibujar, ya que esta acción de presentar los dibujos por pantalla se ha realizado con bloques externos que se explicarán en posteriores apartados. Simplemente mantiene el protocolo VGA necesario para pintar los píxeles del color que indiquen los bloques que realizan esa función.

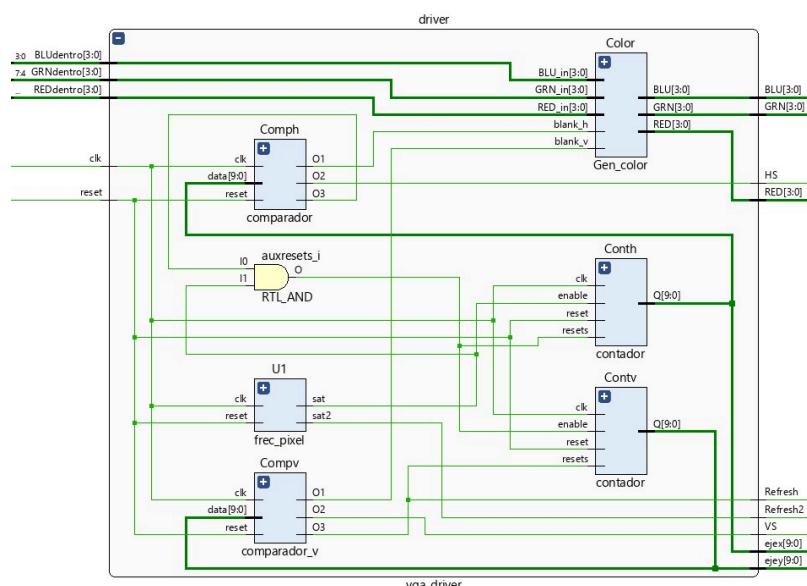
COMPONENTES

Este bloque contiene los **contadores en horizontal y vertical**, que se encargan de indicar el píxel de la pantalla en el que se va trabajando para así recorrer toda la pantalla. Es importante recordar que va de derecha a izquierda y va bajando de línea cada vez que termina de pasar por una completa.

Otro componente importante es el **divisor de frecuencias**, y tiene la función de indicar la velocidad por la que se van recorriendo los píxeles de la pantalla.

Además se dispone de dos **comparadores** que van indicando cuándo finaliza una línea, o la pantalla completa, a través de pulsos con un proceso síncrono. Al finalizar cada línea y cada pantalla habrá un tiempo en el que el VGA tendrá que recolocarse para empezar la siguiente línea o pantalla. Los comparadores se encargan de que durante este tiempo el VGA no pinte nada mediante las señales *HS* y *VS*. Esto sirve para evitar trazos indeseados en la pantalla.

Finalmente, se encuentra el **generador de colores**, que simplemente se introduce para poder colorear los píxeles.



GESTIÓN DEL JUEGO

GESTORÍA

Este bloque llamado *gestor.vhd* se encarga de todo lo relacionado con el funcionamiento del juego. Su función es saber cuándo **finaliza la partida**. Esto se realiza teniendo en cuenta las señales RGB enviadas por el coche y el circuito. Si en un mismo píxel se reciben estas dos señales en un valor distinto a cero, significará que el coche se ha salido del circuito y, por lo tanto, termina el juego. Esto se ha conseguido con el siguiente trozo de código:

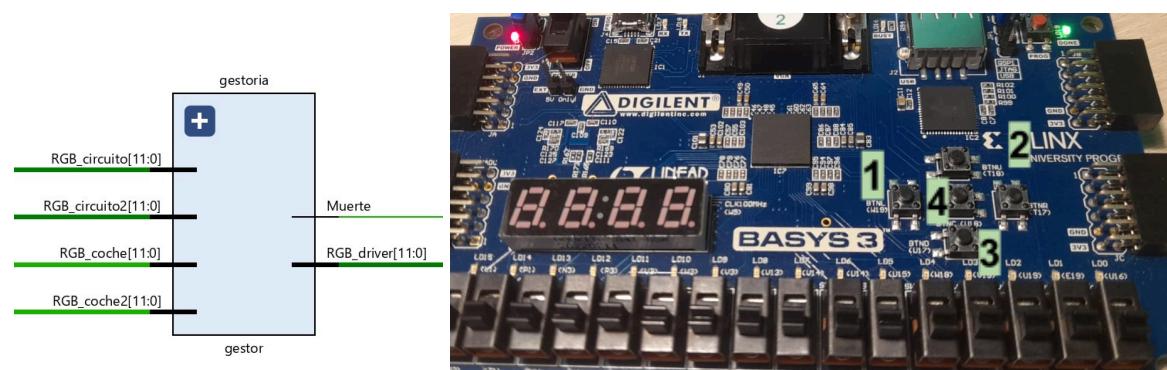
```
if(RGB_coche>"000000000000" and RGB_circuito>"000000000000") then  
Muerte<='1';  
else  
Muerte<='0';  
end if;
```

INICIO DE LA PARTIDA

Se ha habilitado un botón cuya función es darle comienzo al juego, llamado **start**. Esta señal funciona como un *enable* para que se empiece a mover el circuito y se pueda mover el coche. Antes de pulsar este botón, el juego será una imagen estática.

REINICIO DEL JUEGO

Si el coche se ha salido de la pista, la partida finalizará del modo mencionado anteriormente. Sin embargo, eso no significa que no se pueda seguir jugando. Para volver a iniciar una partida nueva basta con pulsar un botón de **reset**. Es síncrono y está habilitado para todos los componentes del programa, por lo que al ser pulsado, se vuelve al inicio del mismo.



1. Left

2. Right

3. Start

4. Reset

DIBUJO DEL JUEGO

Como se ha comentado anteriormente, los dibujos del coche y de fondo que aparecerán en pantalla se indican en los bloques llamados “*dibujasion*”. Se han realizado cuatro bloques separados ya que de esta manera hay menor probabilidad de confusión con el dibujo de cada elemento.

El profesorado ha proporcionado al alumnado una carpeta con imágenes y **archivos .coe** ya creados para poder usarlos en los trabajos. La manera de introducir estas imágenes en el programa se hace siempre de la misma manera. Habrá que crear una **memoria** por cada imagen que se quiera introducir, teniendo en cuenta que no se podrá usar una de ellas para dos cosas distintas. Tras seguir los pasos especificados en el enunciado de la tercera práctica y especificando el tamaño de cada archivo en particular, se introducen así los dibujos.

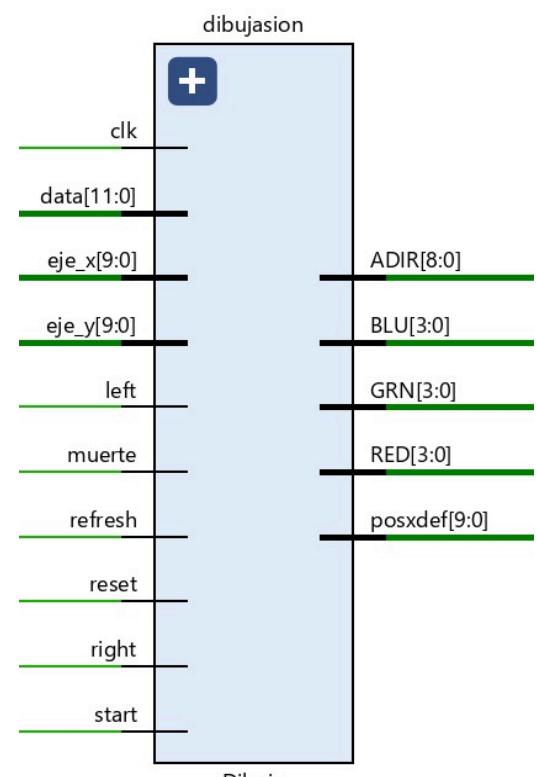
COCHE PRINCIPAL



Para el jugador 1 se ha elegido el **coche de carreras azul**, proporcionado en Enseñanza Virtual. El archivo .coe consiste en una memoria que tendrá almacenada el color de cada píxel del rectángulo de 32 x 16 que conforma el coche.

Dependiendo de la posición en la que se encuentre en la pantalla, medida con “*posx*” y del píxel en el que se está dibujando gracias a “*eje_x*” y “*eje_y*” del VGA Driver, sabrá a qué dirección de memoria ir y así, píxel por píxel, creará el sprite del coche dentro del juego.

La posición del coche variará en el eje x con los botones físicos de la pantalla **left & right**. Para que el coche se mueva de la manera deseada, se ha añadido una señal de *refresh* que actuará como un enable para el movimiento y evitará que en una pulsación se mueva de forma indeseada. Este refresh lo generará el divisor de frecuencias.



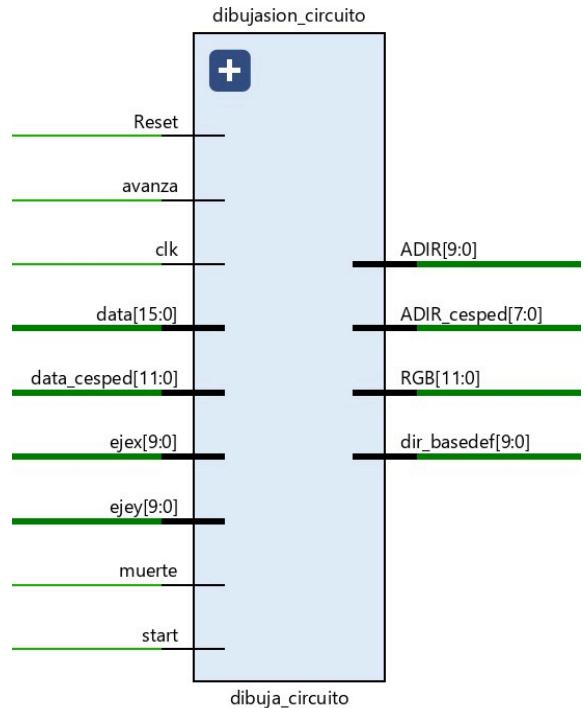
Por otro lado, hasta que no se pulse el botón de *start*, no se podrá mover. Además, cuando la señal “*muerte*” esté activa, se bloqueará el movimiento.

La **posición x del coche** será un valor que se enviará por el *puerto serie* hacia la otra placa que usará el contrincante.

CIRCUITO PRINCIPAL

Este bloque se encarga de dibujar el circuito por donde va el coche del jugador 1, además de añadir el sprite de césped al fondo y unos pianos para el circuito.

El funcionamiento es similar al del coche. Se crea una memoria con el archivo *circuito.coe*, en la que habrán almacenados vectores que contienen los límites izquierdos y derechos del circuito. Fuera de estos límites se pintará la pantalla con el césped. Se comienza leyendo en una dirección base, y se le suma el cuarto bit del eje y, para que a lo largo de la pantalla el circuito varíe. Gracias a la señal “avanza”, creada con el divisor de frecuencias, la dirección base de la memoria disminuirá, y con esto parecerá que el circuito se mueve.



Para incluir el sprite del **césped**, se crea otra memoria con el archivo *cesped.coe*, y cuando se esté fuera de los límites del circuito, dependiendo de la posición en la pantalla, se seleccionará una dirección de memoria que contendrá el color de ese píxel de césped.

Para añadir los **pianos**, teniendo los límites de izquierda y derecha, se les añade una franja de diez píxeles, que serán de color rojo o blanco dependiendo de la posición en el eje y.

Hasta que no se pulsa start el circuito no avanza, y cuando la señal de muerte está activa, dejará de hacerlo.

Se enviará la **dirección base de memoria del circuito** al jugador 2 por el *puerto serie*.

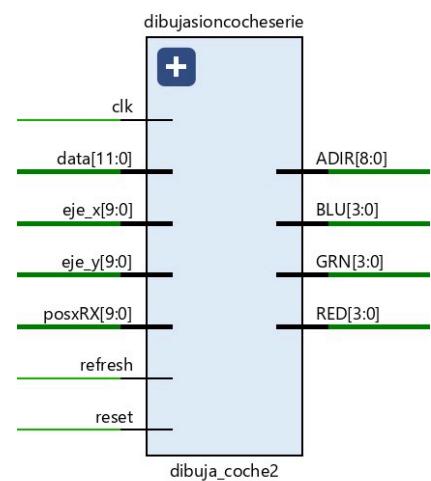
```

if((unsigned(ejex)>(lim_izda) and unsigned(ejex)<lim_izda+10) or
(unsigned(ejex)>lim_dcha+10) and (unsigned(ejex)<(lim_dcha+20))) then
    if(ejey(4)='1') then
        RGB<="110100000000";
    else
        RGB<="111111111111";
    end if;
else if(unsigned(ejex)<(lim_izda+10) or unsigned(ejex)>(lim_dcha+10)) then
    ADIR_cesped(3 DOWNTO 0) <= STD_LOGIC_VECTOR((unsigned(ejex(5 DOWNTO 2)))); 
    ADIR_cesped(7 DOWNTO 4) <= STD_LOGIC_VECTOR((unsigned(ejey(5 DOWNTO 2)) )); 
    RGB<=data_cesped;
else
    RGB <= ( others => '0'); -- estoy dentro de los 1 mites de la carretera, es negro
end if;
end if;
else
    RGB <= ( others => '0'); -- estoy dentro de los 1 mites de la carretera, es negro
end if;
    
```

COCHE ADVERSARIO

El coche del adversario se dibuja de la misma forma que el principal con la ventaja de que la posición x del coche llega por el puerto serie. A esta posición se le suman **320** píxeles para que el coche se ubique en el lado derecho de la pantalla. Por lo tanto, se prescinde de las señales *left*, *right*, *start* y *muerte* debido a que cuando el coche adversario se mueve, llegará su nueva posición por puerto serie.

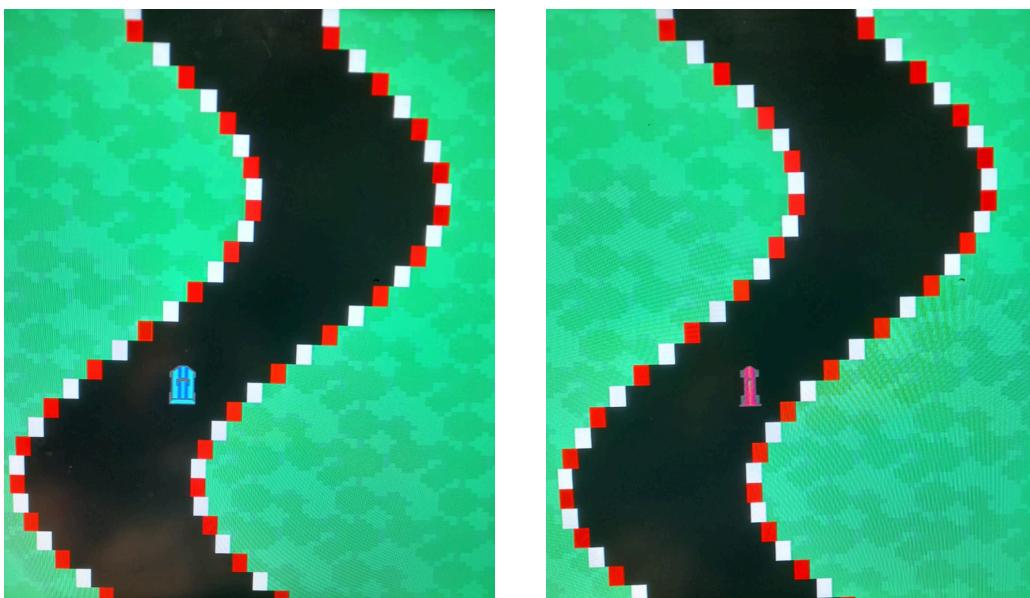
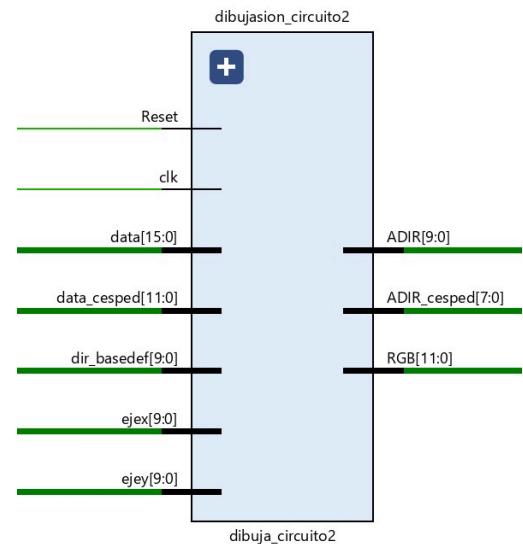
En este caso se ha usado el sprite del **coche de carreras rojo**, por lo que se ha creado una nueva memoria siguiendo el procedimiento con el que se incluyó el coche principal.



CIRCUITO ADVERSARIO

El circuito del contrincante se dibuja de forma similar al circuito principal. La dirección base del circuito llega desde el puerto serie. Con esta dirección se obtienen los límites del circuito, a los que se le suman también 320 píxeles para que este circuito aparezca en la zona derecha de la pantalla. El resto del dibujo (césped y pianos), sigue la misma estructura descrita previamente en el circuito principal.

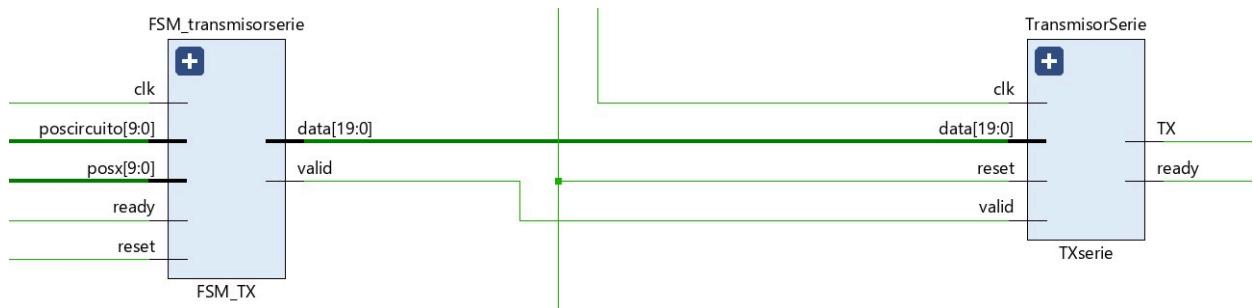
Cabe destacar que aunque se quiera el mismo circuito que el del jugador 1, habrá que **crear otra memoria** distinta ya que la memoria solamente es accesible por un puerto.



COMUNICACIÓN POR PUERTO SERIE

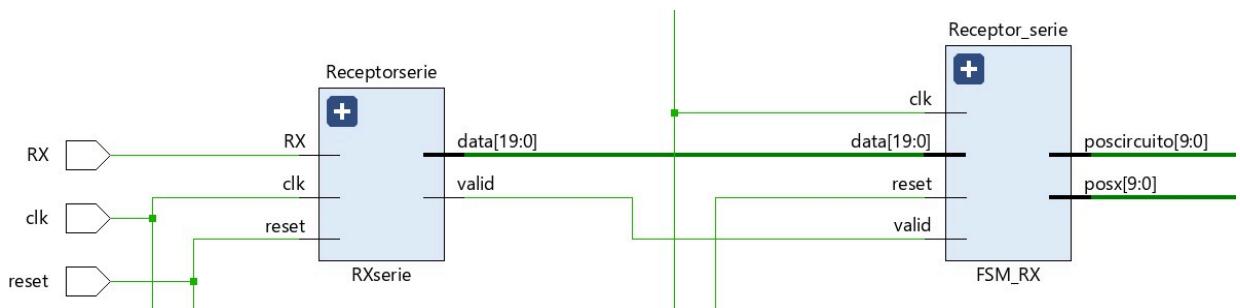
BLOQUE EMISOR

- **FSM_TX**: se encarga de juntar los datos a transmitir, posición x del coche y la dirección base del circuito, en un solo vector, y de mandarlo a TXserie cuando este esté listo para recibir, es decir, cuando la señal “*ready*” esté activa.
- **TXserie**: se encarga de transmitir bit por bit el vector deseado. Este vector estará formado y listo para transmitirse cuando la señal “*valid*” esté activa.



BLOQUE RECEPTOR

- **RXserie**: recibe los bits transmitidos mediante el puerto serie uno a uno. Junta estos bits en un solo vector para su posterior manipulación.
- **FSM_RX**: divide este vector en dos siendo estos posición x y la dirección base del circuito, y se encarga de sólo recibir la información cuando esté completa mediante la señal “*valid*”, que indica cuándo se han terminado de recibir todos los bits en RXserie.



WARNINGS

Durante todo el proceso de creación de código han ido apareciendo cantidad de errores más o menos importantes al ir comprobando cada parte del juego que se iba finalizando, todos resueltos con éxito.

Un error recurrente durante el trabajo ha sido relacionado con el **clock period** de las memorias, pero desapareció al descubrir cómo solucionar este fallo en las opciones disponibles en el menú de creación de las mismas.

Otros fallos habituales fueron relacionados con problemas de registros, debidos a la gran cantidad de conexiones existentes y a la facilidad de la aparición de despistes por ese mismo motivo. Sin embargo, ver el **schematic** de todos los bloques unidos entre sí ha sido la gran herramienta para encontrar los errores de forma sencilla.

Finalmente, es importante recordar que era requisito necesario para la entrega del proyecto que el número de latches presentes en el listado de warnings fuera nulo, cosa que se ha conseguido gracias a la habilidad adquirida para resolverlos a lo largo del proceso. Por lo tanto, las únicas advertencias que aparecen son relacionadas con la implementación debido a que las memorias creadas usando los archivos .coe usan un **reset asíncrono**, mientras que el resto del programa trabaja con resets síncronos.

> ● [DRC REQP-1840] RAMB18 async control check: The RAMB18E1 circuito/U0/inst_blk_mem_gen/gnram.gnatvebmg.native_blk_mem_gen/valid.cstr/ramloop[0].ram/r/prim_init.ram/DEVICE_7SERIES.NO_BMM_INFO.SP.SIMPLE_PRIM18.ram has an input control pin circuito/U0/inst_blk_mem_gen/gnram.gnatvebmg.native_blk_mem_gen/valid.cstr/ramloop[0].ram/r/prim_init.ram/DEVICE_7SERIES.NO_BMM_INFO.SP.SIMPLE_PRIM18.ram/ADDRARDADDR[12] (net: circuito/U0/inst_blk_mem_gen/gnram.gnatvebmg.native_blk_mem_gen/valid.cstr/ramloop[0].ram/r/prim_init.ram/addrA[8]) which is driven by a register (*driver/Contv/cuenta_reg[4]*) that has an active asynchronous set or reset. This may cause corruption of the memory contents and/or read values when the set/reset is asserted and is not analyzed by the default static timing analysis. It is suggested to eliminate the use of a set/reset to registers driving this RAMB pin or else use a synchronous reset in which the assertion of the reset is timed by default. (19 more like this)

RECURSOS UTILIZADOS

Se muestra en la tabla inferior los recursos que utiliza la placa con el alcance que ha tenido este trabajo:

Name	Slice LUTs (20800)	Slice Registers (41600)	F7 Muxes (16300)	Slice (8150)	LUT as Logic (20800)	Block RAM Tile (50)	Bonded IOB (106)	BUFGCTRL (32)
↳ N padre	370	200	1	145	370	3	21	1
> sespesito2 (cesped2)	1	0	0	1	1	0.5	0	0
> sespesito (cesped)	0	0	0	0	0	0.5	0	0
> driver (vga_driver)	210	52	0	86	210	0	0	0
↳ dibujasioncocheserie (dibuja_coche2)	17	10	0	17	17	0	0	0
↳ dibujasion_circuito2 (dibuja_circuito2)	9	0	0	13	9	0	0	0
↳ dibujasion_circuito (dibuja_circuito)	25	12	0	21	25	0	0	0
↳ dibujasion (Dibuja)	37	12	0	19	37	0	0	0
> cocheserieskin (coche1)	0	0	0	0	0	0.5	0	0
> coche (coche2)	1	3	0	2	1	0.5	0	0
> circuito_serie (circuitosserie)	1	1	0	2	1	0.5	0	0
> circuito (blk_mem_gen_0)	1	1	0	1	1	0.5	0	0
↳ TransmisorSerie (TXserie)	28	22	0	14	28	0	0	0
↳ Receptorserie (RXserie)	45	42	0	20	45	0	0	0
↳ Receptor_serie (FSM_RX)	1	21	0	10	1	0	0	0
↳ FSM_transmisorserie (FSM_TX)	7	24	1	8	7	0	0	0

CONCLUSIONES

Esta idea de proyecto sobre el juego del Mario Kart ha sido una gran motivación para aprender el temario de esta parte de la asignatura de Sistemas Electrónicos. Se ha notado una importante mejoría en la capacidad de resolución de problemas con este lenguaje y entorno de programación debido a todo lo que se ha aprendido en el proceso de creación del juego a base de prueba y error.

En este grupo se ha priorizado la perfección de los requisitos básicos del funcionamiento del juego, como la capacidad de un buen funcionamiento de la transmisión por puerto serie para que el contrincante pudiera jugar exactamente de la misma manera que el jugador principal para tener partidas dignas, antes de añadir otras mejoras al proyecto.

Por lo tanto, el grupo ha quedado más que satisfecho con este trabajo que ha supuesto todo un reto.