Universidad
Rey Juan Carlos

VIDEOJUEGO IMPLEMENTADO EN FPGA

Diseño de Circuitos y Sistemas Electrónicos

Grado en Ingeniería Electrónica Industrial y Automática

Universidad Rey Juan Carlos

Curso Académico 2023-24

Curso 4º - Cuatrimestre 1º

Adrián Martín Moreno

Memoria de proyecto para desarrollar un videojuego cuya representación gráfica en pantalla consiste en un circuito de carreras con dos vehículos uno controlado por el usuario y otro denominado fantasma, cuyo movimiento dirige el sistema.

Este proyecto se encuentra sujeto a un plazo de entrega establecido, por ello, una vez finalizada la parte básica del proyecto y habiéndose planteado realizar mejoras, se desarrollarán únicamente aquellas que puedan finalizarse y presentarse en plazo.

Índice

1.Introducción

Cursando la asignatura de Diseño de Circuitos y Sistemas Electrónicos, se adquieren conocimientos como manejo de memorias, multiplexores, máquinas de estados y contadores, que nos permiten realizar un sistema funcional que interactúe con una pantalla HDMI.

Como práctica final de la asignatura es necesario desarrollar un proyecto cuyo objetivo principal es la integración y aplicación de los conocimientos teórico-prácticos adquiridos, reforzando las habilidades de diseño digital y de hardware, así como la capacidad de integrar componentes complejos para crear un sistema interactivo y funcional.

En este contexto se desarrolla este proyecto de implementación de un videojuego, utilizando la FPGA ZYBO Z7 de Digilent (Zybo Z7-10, XC7Z010-ICLG400C).

Obtendremos como resultado final un juego interactivo que presenta un fantasma de 16x16 píxeles, se mueve en una cuadrícula de 32x30 celdas rebotando en las paredes del campo de juego, y un jugador controlado mediante pulsadores. Fantasma y sprite de jugador almacenados en memoria.

Para llevar a término el proyecto necesitamos coordinar diferentes sistemas o elementos electrónicos que permitan crear imagen y movimiento. Para la implementación de este videojuego en concreto, utilizaremos una FPGA, driver HDMI que incluye estándar VGA, para la salida de video y el entorno de desarrollo Vivado de Xilinx. Estos sistemas son creados utilizando lenguaje de programación VHDL (VHSIC Hardware Description Language).

Las FPGA's (Field Programmable Gate Arrays) son dispositivos hardware, conformados por circuitos digitales que pueden ser programados por el usuario en función de su objetivo. En el ámbito del diseño digital, destacan como herramientas fundamentales por su flexibilidad y capacidad de procesamiento paralelo, permitiendo el desarrollo de sistemas embebidos altamente personalizados.

Los estándares de video VGA (Video Graphics Adapter) y HDMI (High Definition Multimedia Interface) se utilizan para conectar un equipo a un dispositivo de salida. La principal diferencia entre ambos es el formato en el cual se transmiten los datos. VGA es un estándar analógico y HDMI es un estándar digital.

Vivado es una herramienta de diseño electrónico, fue desarrollada por Xilinx (ahora parte de AMD) para trabajar con sus FPGA (Field-Programmable Gate Array) y SoCs (System-on-Chip). Es una suite integral que permite diseñar, implementar, simular, y programar circuitos digitales en hardware reconfigurable.

VHDL (VHSIC Hardware Description Language) es un lenguaje de descripción hardware que permite describir el diseño en diferentes niveles de abstracción, ya sea mediante su estructura física con puertas lógicas o bien mediante su comportamiento.

Para comenzar el proyecto se establece en primer lugar un orden de ejecución de tareas recogido en el índice. Tras finalizar cada una de ellas se realiza un testbench verificando así el funcionamiento correcto antes de continuar. Esto facilitará tanto la localización de errores, como a posteriori, realizar alguna modificación de ser necesaria.

En el inicio contamos con dos módulos necesarios que son proporcionados previamente, HDMI y PINTAc. Si bien HDMI se encuentra totalmente funcional en el momento de la entrega, PINTAc ha sido proporcionado en su versión básica, que si bien permite comprobar el correcto funcionamiento del módulo de sincronismo hdmi, requiere modificaciones. Estas modificaciones irán realizándose con el avance del proyecto. Debemos tener en cuenta la relevancia que le corresponde como módulo, al ser el encargado de administrar los datos de salida al HDMI (lo que se “pinta” en la pantalla).

Una vez finalizada la arquitectura básica del proyecto se implementarán tres mejoras, variación en la velocidad cuando sprite cruce el borde de la pista, cambio de orientación del Sprite del jugador en función del sentido de dirección y color a la Pista de carreras y Sprite.

Teniendo en cuenta que el objetivo principal de este proyecto es mostrar los conocimientos y habilidades adquiridas en el curso de la asignatura, además de las anteriormente expuestas, académicamente el desarrollo del proyecto refuerza las habilidades de diseño digital y de hardware, destacando la capacidad adquirida de integrar componentes complejos para crear un sistema interactivo y funcional.

En opinión, este proyecto no solo permite consolidar habilidades en diseño digital y modularidad, sino que también, tras su manejo, nos permite considerar herramientas como Vivado y las FPGAs pilares esenciales en el desarrollo de sistemas complejos con aplicaciones en el ámbito académico, industrial y de entretenimiento.

En el presente quedan recogidos y /o documentados módulos implementados, simulaciones, resultados obtenidos, y análisis de FPGA. Demostrando su efectividad logrando los objetivos técnico-académicos.

2. Implementación arquitectura básica

La arquitectura del proyecto se basa en varios módulos principales que a su vez contienen otros menores. Obtenemos como resultado final el Top_VGA en el cual quedan todos los módulos

englobados. En consecuencia el Top_VGA irá modificándose y evolucionando durante el proceso de desarrollo.

2.1 Módulo de sincronismo hdmi

La comunicación con la pantalla se realiza mediante un cable HDMI (es el estándar de la FPGA), tal como se explicó con anterioridad, el módulo HDMI nos fue proporcionado, por lo que para demostrar el conocimiento adquirido, se implementan el diseño del sistema y la sincronización de video conforme al estándar VGA. Para realizar dicha implementación se requiere el desarrollo de un módulo de sincronización horizontal y vertical, esencial para comprender cómo se construyen imágenes rasterizadas en pantallas.

Considerando que el módulo de sincronismo hdmi lo forman PLL, SYNCVGA, PINTAc y HDMI, para dar comienzo al proyecto necesitamos desarrollar PLL y SYNCVGA. Una vez desarrollados estos módulos es necesario hacer uso del PINTAc en su versión básica para comprobar el correcto funcionamiento del módulo de sincronismo hdmi (denominado en el codigo Top_VGA,) quedando esquemáticamente representado en su inicio en la ilustración 1.

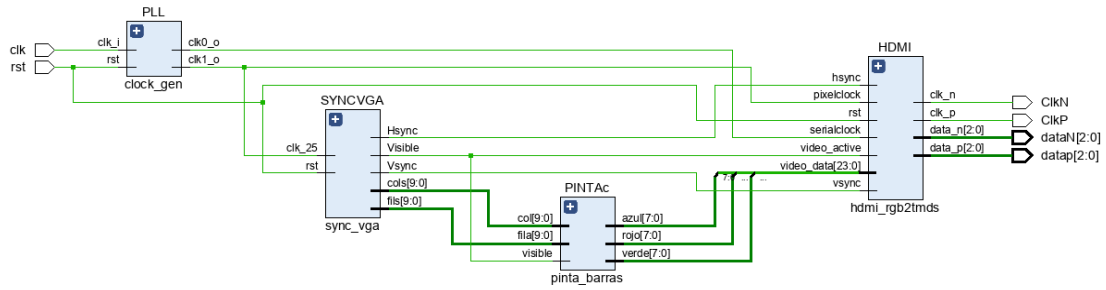


Ilustración 1. Módulo Top_VGA inicial.

El Módulo PLL se desarrolla siguiendo la entidad del modelo aprendido en la asignatura. Proporciona dos señales de reloj que marcaran la velocidad base de funcionamiento de los módulos. Ambas identificadas en el esquema previo (ilustración 1) Clk0_0 = 125Mhz y Clk1_0 = 25Mhz

Módulo SYNCVGA, implementa el estándar VGA utilizando lógica de conexión y 2 contadores genéricos que debemos desarrollar. Considerando que a posteriori será necesario el uso de más contadores, como previsión, se desarrolla uno genérico que nos permita instanciar varios a partir del mismo. Desarrollar este tipo de contador resuelve la necesidad momentánea de dos contadores, facilita la tarea y ahorra tiempo en próximos módulos a desarrollar.

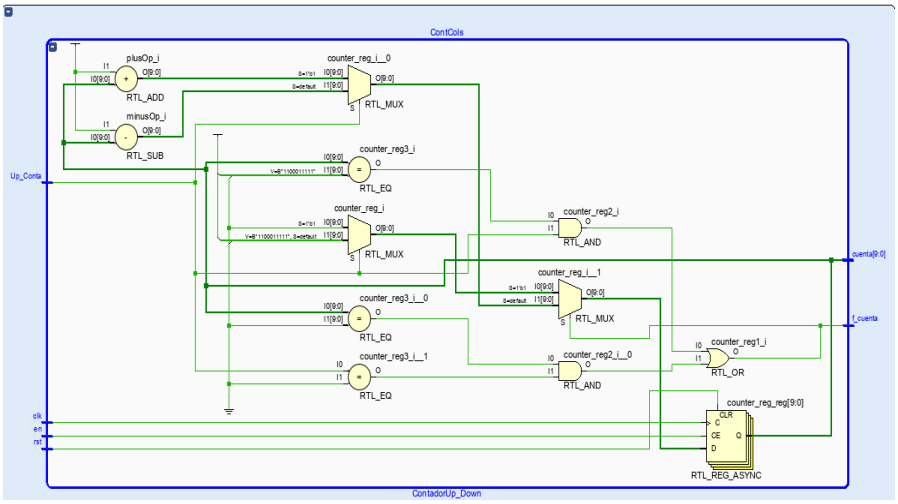


Ilustración 2. Estructura del contador.

- Entradas
    - Up_Conta (lógica 0-1): Establece conteo ascendente o descendente
    - Clk (lógica 0-1): Señal de reloj
    - En (lógica 0-1): Señal de habilitación
    - Rst (lógica 0-1):Reinicio del contador
- Salidas
    - Cuenta (vector de 10 bits): Valor de cuenta actual
    - F_cuenta (lógica 0-1): Indica valor máximo alcanzado.

Se realiza el *testbench* para comprobar el correcto funcionamiento y ajustar los posibles errores hasta obtener un resultado favorable (resultado 6.1).

Una vez realizados, ambos contadores se unen a la lógica necesaria determinada por el funcionamiento del estándar VGA para obtener la estructura del SYNCVGA:

- Entradas
    - Clk_25(lógica 0-1), corresponde a la señal de reloj.
    - Rst (lógica 0-1): Reinicio.
- Salidas
    - Hsync (lógica 0-1), destinada a la sincronización horizontal.
    - Vsync(lógica 0-1), destinada a la sincronización vertical.
    - Cols (vector de 10bits), transporta el número de columna.
    - Fils (vector de 10bits), transporta el número de fila.
    - Visible (lógica 0-1), permite determinar la zona visible.

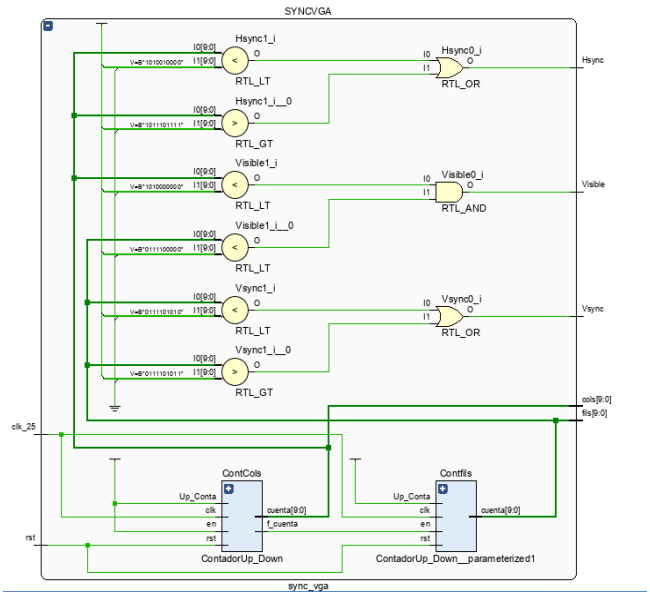


Ilustración 3 Estructura SYNCVGA.

El uso conjunto de Cols y Fils permite recorrer todos los pixeles. Es el momento de realizar un testbench del Top_VGA inicial (resultado 6.2). Con un resultado favorable se da por finalizado el Top_VGA inicial (equivalente al módulo de sincronismo hdmi). Como ya se mencionó, será nuestro bloque "Top System" el cual será modificado y ampliado para albergar todos los componentes necesarios, modificando en consecuencia su testbench a lo largo del desarrollo. No es imprescindible realizar así el desarrollo, ya que otra opción sería tratar el módulo de sincronismo hdmi como un módulo más e incluirlo a su vez en uno mayor.

2.2 Campo de juego

El campo de juego consiste en una rejilla de 32 columnas x30 filas, donde cada celda queda formada por 16x16 pixeles. La técnica para dibujar la cuadrícula consiste en separar filas y columnas del módulo de sincronismo. Deben separarse en dos grupos seleccionando filas y columnas interiores (4 bits menos significativos) y filas y columnas de bits más significativos.

Puesto que SYNCVGA nos proporciona los vectores cols y fils de 10 bits cada uno podemos descomponer ambos vectores en dos partes, bits más significativos y menos significativos.

De esta manera con los bits más significativos se realiza la división en cuadrículas la pantalla mientras que los bits menos significativos permiten recorrer los pixeles dentro de una cuadrícula especificada.

Finalizado el proceso anterior se hace necesario implementar en PINTAc dos entradas que le permitirán gestionar la información a mostrar por pantalla. Siendo esta su primera modificación. Puesto que el campo de juego de 32 x30 no ocupa toda la pantalla, utilizaremos los valores proporcionados por el vector cols para pintar en negro los píxeles sobrantes, alrededor del campo de juego hasta completar la pantalla (a partir de la columna 512).

2.3 El pacman (personaje controlado por el usuario)

Para completar el pacman además de su silueta y color debemos desarrollar su movimiento y velocidad, por lo que su construcción se realiza en fases individuales. En este caso iniciamos el proceso desarrollando en primer lugar la estructura encargada del movimiento. El objetivo es desplazar el pacman por la superficie de la rejilla en función de la interacción del usuario.

Comenzamos su construcción utilizando una cuadrícula elegida al azar que nos servirá como punto de partida para iniciar el movimiento y cambiando su color la convertimos en la ubicación provisional del pacman..

Necesitamos crear un nuevo módulo que gestione los dos vectores necesarios para determinar la ubicación del pacman que debe estar localizado permanentemente, y, que transmita dicha ubicación al módulo PINTAc (encargado de gestionar lo mostrado en pantalla). Estos vectores reciben la denominación FilasPacaman y ColumnasPacman identificando cada uno de ellos, según correspondencia (fila y columna), el lugar que ocupa el pacman dentro de la rejilla o lo que es lo mismo, la casilla a colorear. Teniendo en cuenta las medidas del campo de juego los vectores deben ser de 5 bit.

El nuevo módulo se denomina CELDAPACMAN.

Por otra parte, el usuario jugador enviará la orden de desplazamiento de color de una casilla a otra, utilizando 4 pulsadores que informan de la dirección en que debe producirse dicho cambio. Para recoger esta información es necesario realizar el conteo de casillas, por lo que se implementa un contador por vector. Estos contadores manejan las modificaciones de los valores que se obtienen en función de las pulsaciones realizadas por el usuario jugador. Para evitar la necesidad de realizar tantas pulsaciones en el botón como cuadrículas se desee desplazar el Pacman, debemos instanciar un nuevo contador (el tercero) encargado de establecer el muestreo de los pulsadores cada 100 milisegundos.

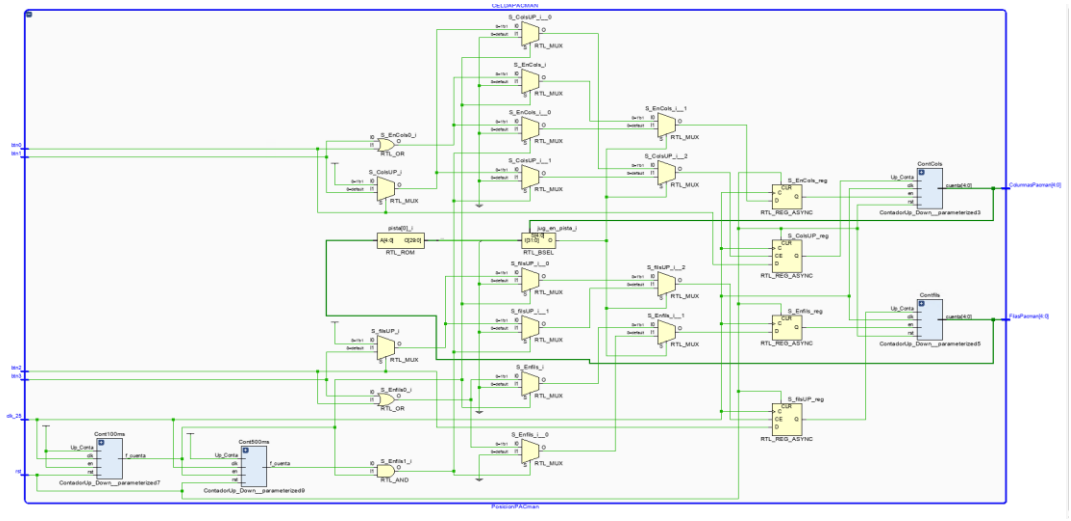


Ilustración 4 Estructura CELDAPACMAN con un cuarto contador (Cont500ms) explicado más adelante.

- Entradas
    - Btn0 (Entrada lógica 0-1): Movimiento hacia la derecha
    - Btn1 (Entrada lógica 0-1): Movimiento hacia la izquierda
    - Btn2 (Entrada lógica 0-1): Movimiento hacia abajo
    - Btn3 (Entrada lógica 0-1): Movimiento hacia arriba
    - Clk_25 (Entrada lógica 0-1): Reloj 25Mhz
    - Rst (Entrada lógica 0-1): Reinicio
- Salidas
    - ColumnasPacman (vectores 5bits): Establece en que columna se encuentra el jugador, valores variables entre 0-31.
    - FilasPacman (vectores 5bits): Establece en que fila se encuentra el jugador, valores variables entre 0-29.

Realizamos un testbench para conseguir el funcionamiento deseado. Este será modificado a posteriori para comprobar el uso con la mejora (resultado 6.3).

2.3.1 Pintar el Pacman desde memoria

Necesitamos una matriz o memoria en la que se encuentran almacenadas 16 imágenes de 16x16 píxeles, la obtenemos de un repositorio, consta de dos entradas (addr vector de 8 bits y clk señal de reloj) y una salida (dout vector de 16bits). La denominamos MemoriaPSJ.

Cada imagen se identifica con los bits más significativos de la dirección de memoria. Concretamente, el pacman se corresponde con los bits más significativos “0011”.

Si consideramos la memoria como un mapa de cuadrículas donde cada cuadrícula contiene una imagen (de forma análoga a la segmentación de la pantalla por cuadrículas) buscaremos con los bits más significativos la imagen deseada (su celda) y utilizaremos los bits menos significativos para recorrerla extrayendo toda la información de la imagen. Esta información debe llegar al módulo PINTAc al tiempo que PINTAc indica a la memoria la imagen que desea pintar (acciones sincronizadas). Es por tanto necesario crear en PINTAc una salida PSJ y una entrada din.

Utilizamos un multiplexor en PINTAc que extraerá de los datos recibidos de la memoria el valor del pixel a pintar.

Para obtener la salida correcta de la memoria es importante tener claro el contenido de su dirección de memoria (addr). Esta dirección la componen los bits que determinan la casilla elegida (personaje deseado, salida PSJ módulo PINTAc) y los bits menos significativos del vector fils provenientes de SYNCVGA.

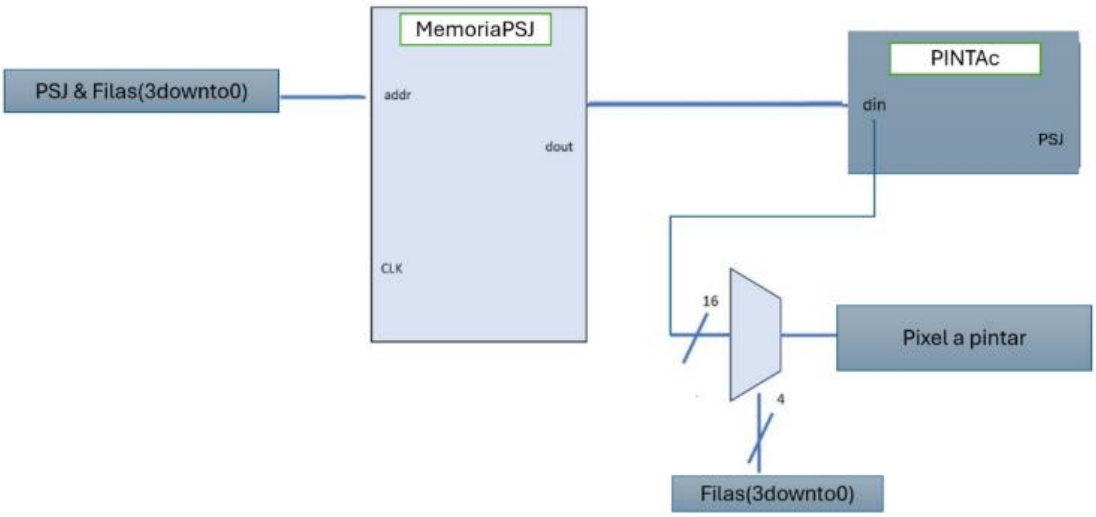


Ilustración 5 Estructura implementada para pintar el personaje desde memoria.

2.4 Movimiento y dibujo del fantasma

Se inicia su desarrollo siguiendo el patrón en la creación del PACMAN, en primer lugar implementando el movimiento que deseamos para este personaje y con la misma metodología, seleccionando una celdilla a la que se le da color.

El movimiento del fantasma es controlado o dirigido por una máquina de estados, se realiza en diagonal con rebotes en las paredes del campo de juego y con una actualización de posición cada 100 ms, es decir, el fantasma cambiará de cuadrícula con ese periodo de refresco.

Creamos una máquina de estados que permita mover un cuadrado en diagonal por la pantalla y 2 vectores (ColumnasGhost y FilasGhost) encargados de almacenar la posición del Ghost y de transmitir la información al PINTAc que será modificado añadiendo dos nuevas entradas. Por tanto, se necesita un contador por vector y un tercero para temporizar el cambio de casilla (los 100ms).

Al elaborar la máquina de estados resulta imprescindible considerar los límites de pantalla o los límites de juego. En horizontal, desde la columna 0 hasta la 31 y en vertical desde la fila 0 a la 29. Siendo estos los indicadores a tener en cuenta para realizar los cambios de estado.

En cada estado se ajustará el modo de funcionamiento de los contadores responsables de los vectores (ColumnasGhost y FilasGhost). Implementando la máquina de estados obtenemos el módulo CeldaGHOST.

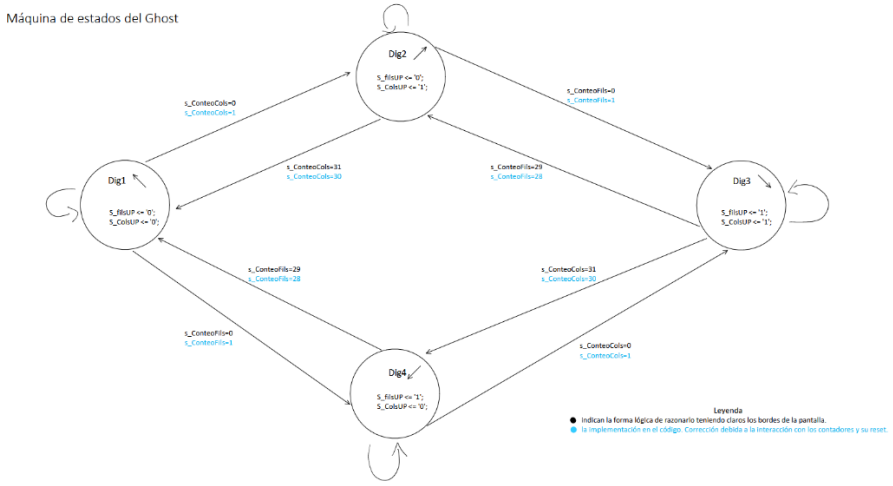


Ilustración 6 Máquina de estados del fantasma.

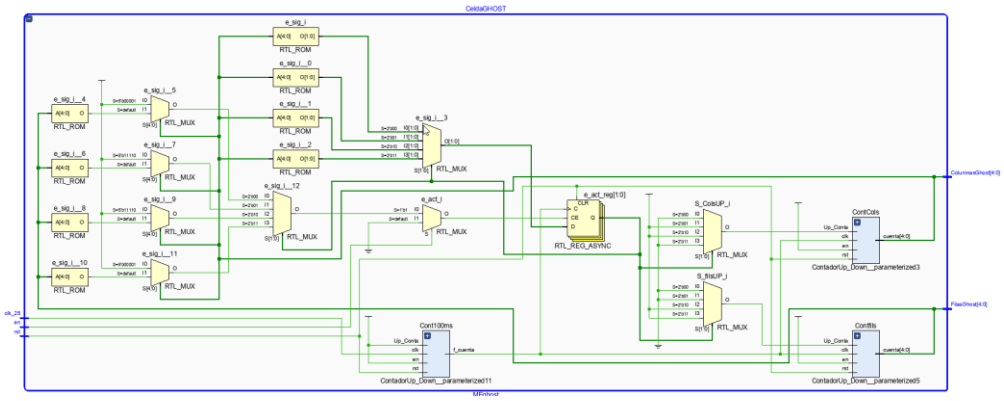


Ilustración 7 Estructura interna CeldaGhost

Se realiza el testbench para conseguir el funcionamiento adecuado (resultado 6.4).

Para mostrar el dibujo del ghost se sustituyen los bits significativos del dibujo del pacman por los bits más significativos del dibujo del ghost y se utiliza la infraestructura ya creada (ilustración 5).

2.5 Dibujar la pista de carreras

Para comenzar el proceso descargamos desde el repositorio la matriz necesaria que se instancia en el Top_VGA con la denominación de MemoriaMap. Es una memoria de 32x30 píxeles en la que se encuentra almacenado el mapa con dos entradas (addr vector de 5 bits y clk señal de reloj) y una salida (dout vector de 32bits).

El procedimiento estructuralmente es el mismo que el realizado con los personajes salvo por dos pequeños cambios conceptuales.

1º Esta memoria almacena únicamente el mapa, por lo que no es necesario realizar concatenación.

2º No se pinta en cuadrícula, se realiza un escalado x16 siguiendo el patrón de multiplicar por 2 elevado cada vez que se realiza un desplazamiento hacia un bit más significativo.

En este caso son necesarios cuatro desplazamientos resultando como ubicación final el bit 4, quedando la dirección de la memoria con los bits desde el 4 al 8.

Para recibir los datos extraídos del MemoriaMap, PINTAc debe ampliarse con una entrada más(dinMapa).

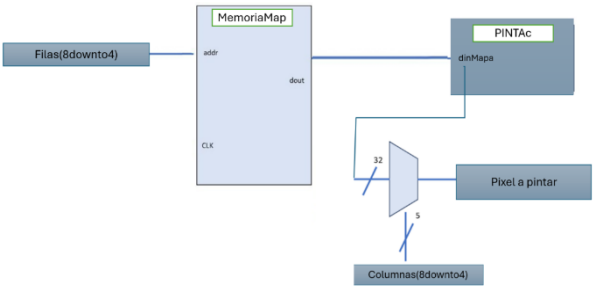


Ilustración 8 Estructura implementada para pintar el personaje desde memoria.

3.Implementación arquitectura de mejoras

3.1 Cambio de velocidad

Para tener una experiencia más realista, cuando el jugador saque el pacman de la pista este reducirá su velocidad. Para conseguir esta reducción de velocidad, en lugar de avanzar una cuadrícula cada 100 ms lo hará cada 500 ms.

Conocer cuál es la posición del Pacman, dentro o fuera de la pista, es un dato necesario que puede obtenerse con la lectura de la memoria, de encontrarse fuera de la pista, se aplica la lógica necesaria para realizar el cambio de velocidad. Esta reducción de velocidad puede realizarse también mediante la implementación, incluyendo una constante que almacene el mapa de la pista. Con esta última opción se elimina la preocupación que pueda generar la limitación de la memoria de la FPGA.

El proceso de implementación comienza descargando el paquete con la constante que contiene el mapa de la pista. Esta constante es una matriz que permite realizar la consulta a partir de las filas y columnas de la cuadrícula en las que se encuentre el pacman.

Una vez conocida la ubicación del pacman, en caso de encontrarse fuera de la pista, aumentando el tiempo de muestreo de los pulsadores a 500ms obtendremos la reducción de velocidad deseada. Para conseguir este objetivo se modifica el módulo CELDAPACMAN creando un nuevo contador partiendo de los 100ms que permita temporizar los 500 milisegundos.

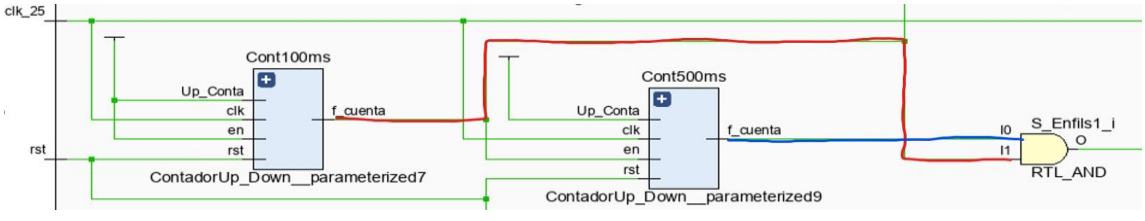


Ilustración 9 Contadores en cascada.

Tras realizar un primer testbench (resultado 6.3) con un resultado de funcionamiento incorrecto, fue necesario realizar varias pruebas hasta localizar el error, este se debía a la falta de una puerta AND de conexión entre los finales de cuenta de ambos contadores, necesaria al ser una estructura en cascada (al mantenerse a f_cuenta=1 de Cont500ms el sistema funcionaba de forma indeseada).

3.2 Orientación del sprite

Previo a comenzar el desarrollo de esta mejora se considera oportuno elaborar un resumen-guía de conocimientos considerados mínimos indispensables para realizar la técnica (anexo III).

Para que el jugador experimente una sensación más agradable, se reorienta el Sprite del personaje pacman en función del sentido de dirección que el jugador solicita. En nuestro proyecto el movimiento del pacman puede realizarse en 8 sentidos de dirección, arriba, abajo, derecha, izquierda, y las 4 diagonales

Para el movimiento en las diagonales se sustituye el Sprite del pacman por un vehiculo.

Para realizar estas mejoras se incorpora un nuevo módulo (RTCPSJ) encargado de manipular la dirección enviada a la memoria y de enviar un vector btns al PINTAc para que este sepa cómo debe interpretar los datos que recibe de la memoria y obtener la visualización correcta.

Las operaciones necesarias para la rotación se implementan utilizando multiplexores y lógica combinacional. En función de la dirección requerida se realiza la selección de las coordenadas invertidas y/o transpuestas al acceder a la memoria del sprite. Esta lógica garantiza que el sprite se oriente correctamente en la dirección deseada.

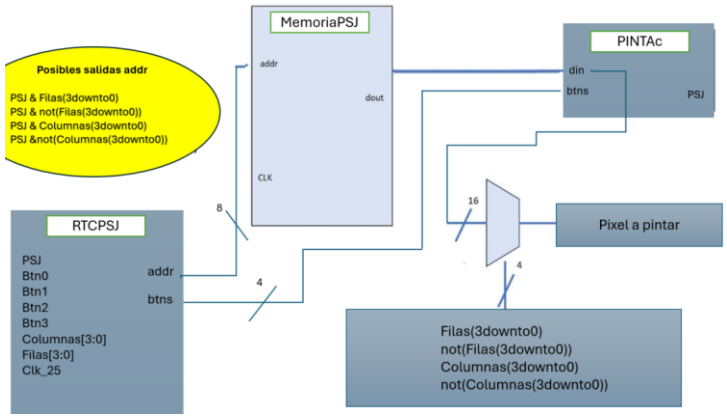


Ilustración 10 Estructura para la rotación de sprites.

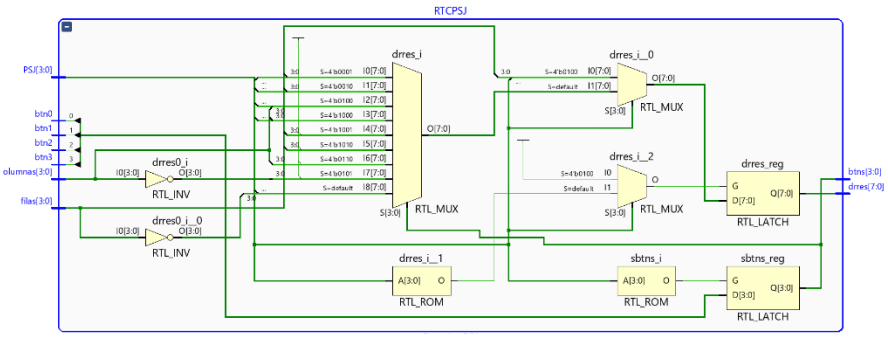


Ilustración 11 Estructura módulo RTCPSJ.

- Entradas
    - PSJ(vector de 4 bits), indica el personaje que se desea dibujar
    - Btn0 (entrada lógica 0-1), valor del botón externo 0
    - Btn1 (entrada lógica 0-1), valor del botón externo 1
    - Btn2 (entrada lógica 0-1), valor del botón externo 2
    - Btn3 (entrada lógica 0-1), valor del botón externo 3
    - Columnas (vector de 4 bits), contiene los bits menos significativos del vector de las cols procedentes de SYNCVGA
    - Filas (vector de 4 bits), contiene los bits menos significativos del vector fils procedentes de SYNCVGA
    - Clk_25(entrada lógica 0-1), reloj
- Salidas
    - btns (vector de 4 bits), almacena los valores obtenidos en los botones, se envía al PINTAc
    - drres (vector de 8 bits), contiene la dirección a enviar a la memoria ya formada (incluye ya el PSJ).

En este momento de desarrollo del proyecto, en opinión, el uso de Vivado resulta clave para el diseño, simulación, síntesis y análisis de los recursos implementados en la FPGA. La herramienta permite validar cada módulo del sistema y optimizar el diseño en función de las limitaciones de hardware.

3.3 Pista de carreras y sprites de color

Se desarrollan individualmente.

Pista. Para desarrollar esta mejora debemos incorporar las dos memorias de pista que descargamos del repositorio (MemoriaMapgreen, MemoriaMapblue).

Son necesarias dos nuevas entradas en PINTAc (dinmapaGreen y dinmapaBlue) que obtenemos como resultado de implementar las memorias en el interior del Top_VGA con técnica de replique de la estructura de MemoriaMap (ilustración 8). El proceso finaliza estableciendo la lógica para el entrelazado de datos.

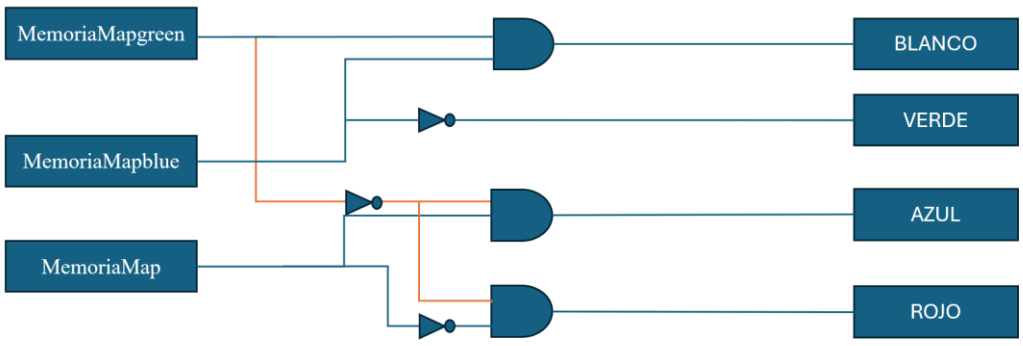


Ilustración 12 Lógica establecida en PINTAc

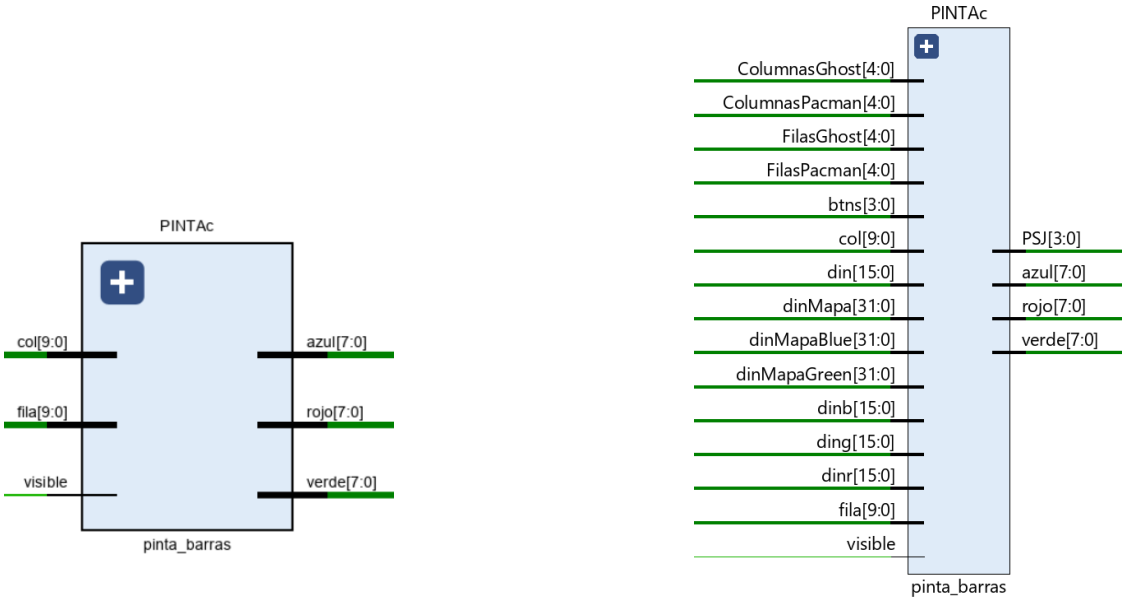
Sprites. Para colorearlos son necesarias tres memorias diferentes (MemoriaPSJred, MemoriaPSJgreen, MemoriaPSJblue), una para cada canal (RGB). Una vez instanciadas dentro del Top_VGA siguiendo la misma estructura de la MemoriaPSJ (ilustración 5), se implementa la lógica en PINTAc y se añaden las entradas (dinr,ding,dinb). Finalizado este proceso los sprites quedan coloreados y la MemoriaPSJ solo se utilizará para suministrar el sprite del coche en movimiento diagonal.

Podría ser eliminada (reduciendo recursos) y mostrar el coche a color utilizando las 3 memorias implementadas para el RGB, pero no se realiza ya que no son necesarios conocimientos nuevos y conlleva tiempo a emplear en otras tareas.

De implementar el coche con las memorias RGB, la MemoriaPSJ Podría ser eliminada reduciendo(recursos). En este proyecto no se encuentra utilidad ya que no se muestran nuevos conocimientos.

El incluir mejoras opcionales, como el uso de color, orientación dinámica de los sprites y lógica avanzada para movimientos, permite valorar como destacable el potencial de las FPGAs para tareas concurrentes y personalizables.

4. Módulo PINTAc



INICIAL FINAL

Ilustración 13 Evolución módulo PINTAc.

Con sus respectivas estructuras en anexo IV.

Ya en la introducción destacamos la relevancia que se consideraba tendría PINTAc en el desarrollo de este proyecto. Finalizado el mismo, queda demostrada gráficamente su importancia indiscutible. A la vista de la imagen comparativa, resulta imprescindible un buen manejo de su implementación para llevar a término este proyecto.

Reorganizando sus entradas y salidas explicadas durante el desarrollo.

- Entradas (15)
    - ColumnasGhost y FilasGhost (vectores 5 bits), recibe la ubicación del fantasma.
    - ColumnasPacman y FilasPacman (vectores 5 bits), recibe la ubicación del pacman
    - Btns (vector 4 bits), recibe el estado de los botones
    - din (vector 16 bits), recibe la información de los sprites de la MemoriaPSJ.
    - dinMapa (vector 32 bits), recibe la información de la MemoriaMap.
    - dinMapaBlue (vector 32 bits), recibe la información de la MemoriaMapBlue.
    - dinMapaGreen (vector 32 bits), recibe la información de la MemoriaMapGreen.
    - dinb (vector 16 bits), recibe la información de la MemoriaPSJblue.
    - ding (vector 16 bits), recibe la información de la MemoriaPSJgreen.
    - Dinr (vector 16 bits), recibe la información de la MemoriaPSJred.
    - fila (vector 10 bits), recibe el vector fila procedente de SYNCVGA.
    - col (vector 10 bits), recibe el vector columna procedente de SYNCVGA.
    - visible, procedente de SYNCVGA permite determinar la zona visible ubicando el momento para pintar los pixeles.
- Salidas
    - PSJ (vector 4 bits), se envían los bits más significativos del sprite deseado.
    - Azul (vector 8 bits), contiene la información de los pixeles para el canal azul.
    - Rojo (vector 8 bits), contiene la información de los pixeles para el canal rojo.
    - Verde (vector 8 bits), contiene la información de los pixeles para el canal verde.

5. Esquema final Top_VGA

Imagen y PDF anexo I.

6. Resultados obtenidos

6.1 Cronometro

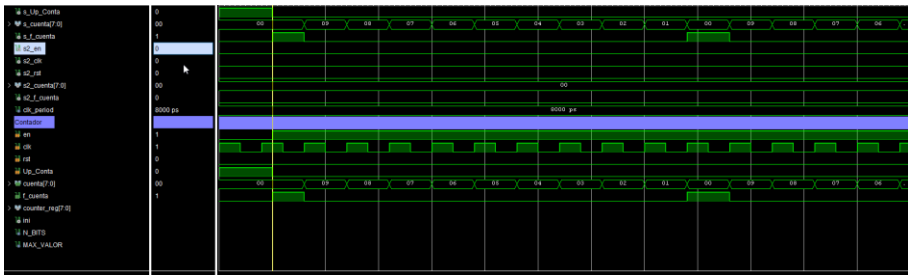


Ilustración 14 Resultados simulación archivo Test_vga_tb.

Manteniendo Up_Conta=0 fijándonos en el vector cuenta[7:0] observamos en el vector cuenta[7:0] el correcto reseteo del conteo descendente.

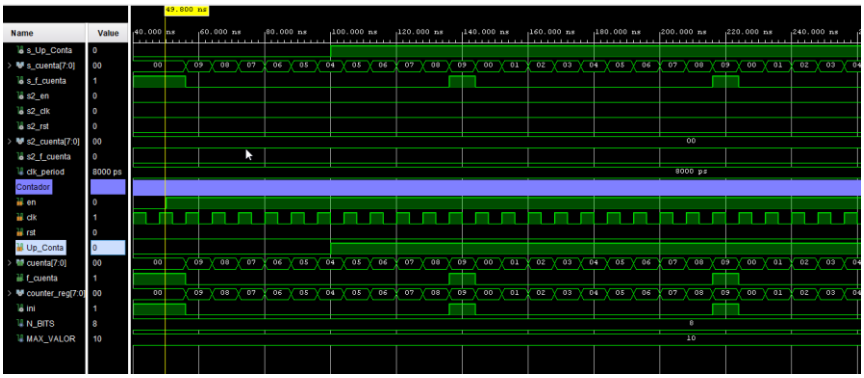


Ilustración 115 Resultados simulación archivo Test_vga_tb.

Observando el vector cuenta [7:0] tras modificar el valor de Up_Conta a 1, se produce un conteo ascendente en mitad del proceso. Resultado correcto.

Las ilustraciones 14 y 15 demuestran que si el en=0, el contador permanece parado, manteniendo el ultimo valor de f_cuenta y de cuenta [7:0].

6.2 Top_VGA inicial

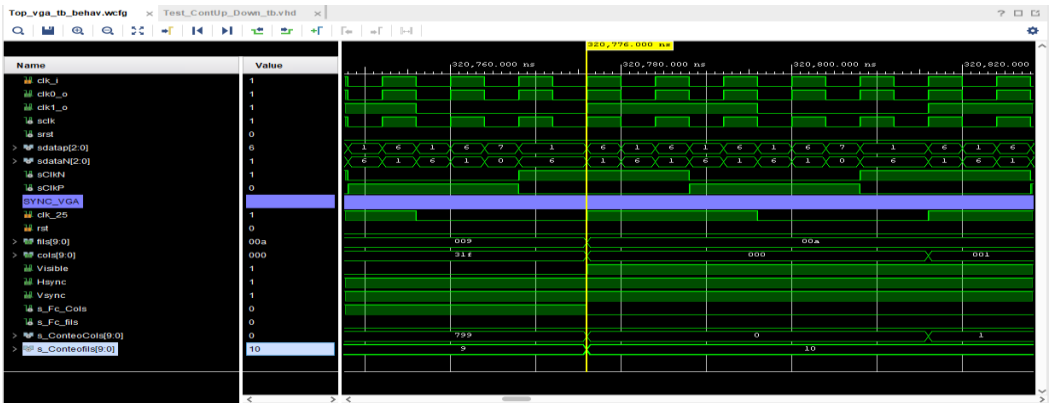


Ilustración 12 Resultados simulación archivo Top_vga_tb en su versión inicial.

Salida de datos correcta sdataN y sdataP en recorrido de pantalla. Recorrido completo de columnas (s_conteoCols llega a 799) previo a fila siguiente (aumenta una unidad s_Conteofils).

6.3 CELDAPACMAN

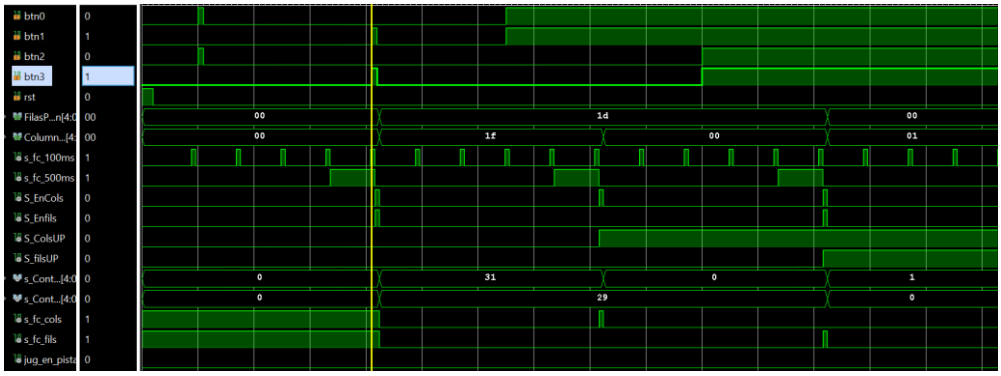


Ilustración 17 Resultados simulación archivo PosiconPacman_tb.

Al coincidir algún btn pulsado (en este caso 1 y 3) junto al muestreo de botón (s_fc_100ms, s_fc_500ms), estos son detectados, el valor de las columnas y filas se ajusta (línea amarilla).

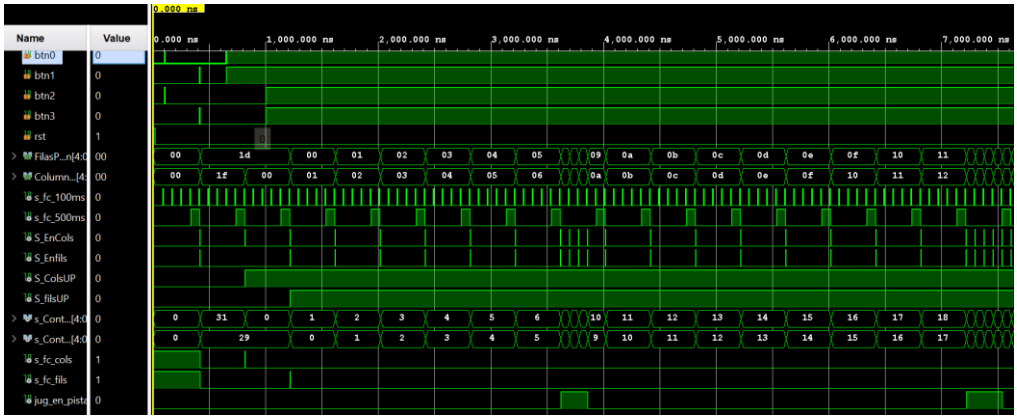


Ilustración 138 Resultados simulación archivo PosiconPacman_tb.

Se observa que en función de jug_en_pista, se realiza el muestreo de los botones a una velocidad u otra. Si jug_en_pista=1 el valor de columnas y filas varia más rápido (equivale a dentro de la pista)

6.4 CeldaGHOST

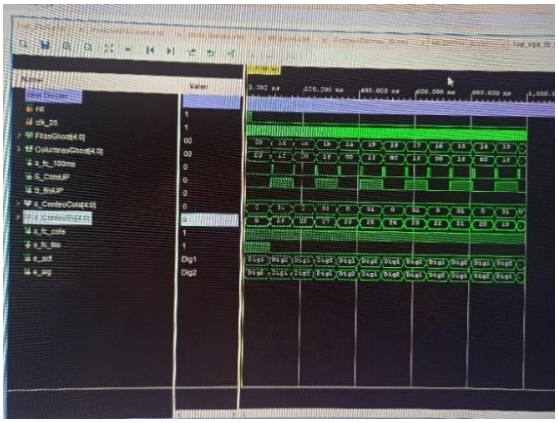


Ilustración 149 Comportamiento erróneo, simulación archivo MEGHOST_tb

Fue necesario realizar ajustes en las condiciones de cambio de estado para dar solución a problemas derivados del comportamiento conjunto, cambio de estado-reset de los contadores.

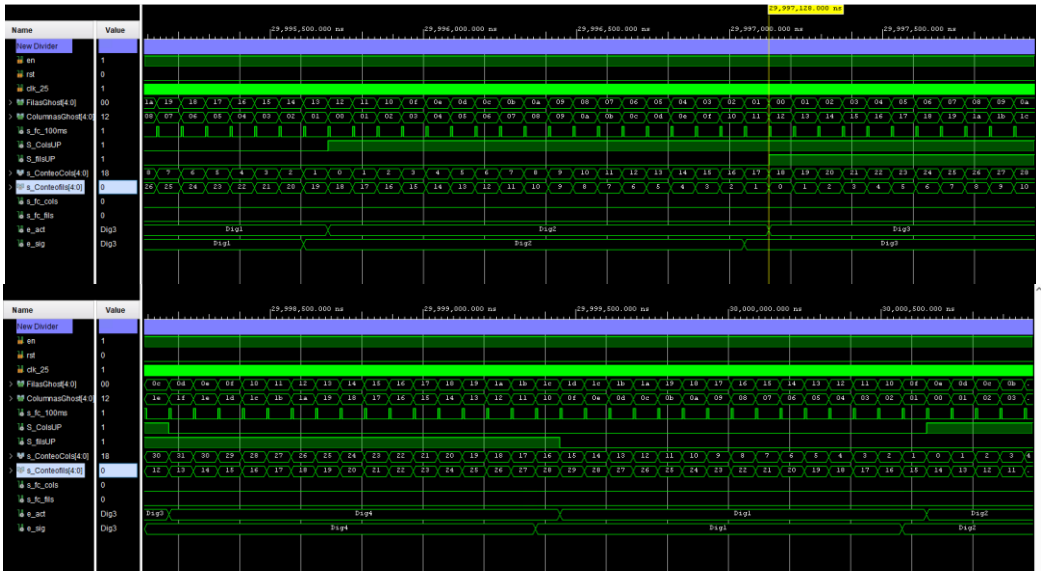


Ilustración 20 Resultados simulación archivo MEGHOST_tb

Modificando las condiciones de cambio de estado obtenemos un funcionamiento correcto.

6.5 Tabla de consumos anexo II

6.6 Repositorio

Código del proyecto en rama main https://github.com/adrielmicom/ProyectoSed2.git.

Contenido audiovisual, diferentes etapas del proyecto, en OneDrive ProyectoSed2.

Video final de proyecto en OneDrive VideoExplicacionProyectoFinal.mp4.

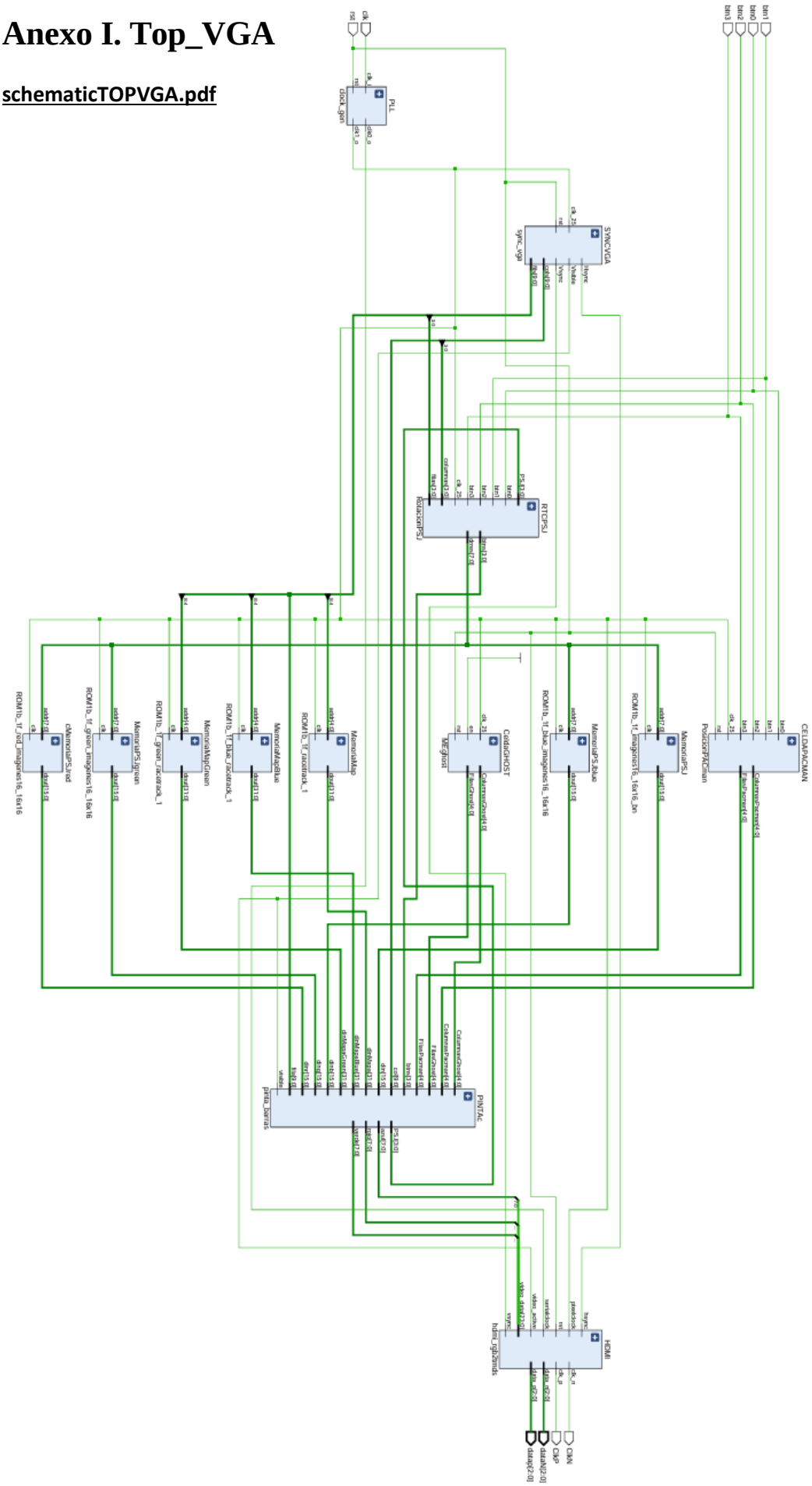
6.7 Bibliografía

Memorias descargadas del repositorio

https://github.com/felipe-m/img_rom/tree/master/examples/vhd.

Anexo I. Top_VGA

schematicTOPVGA.pdf

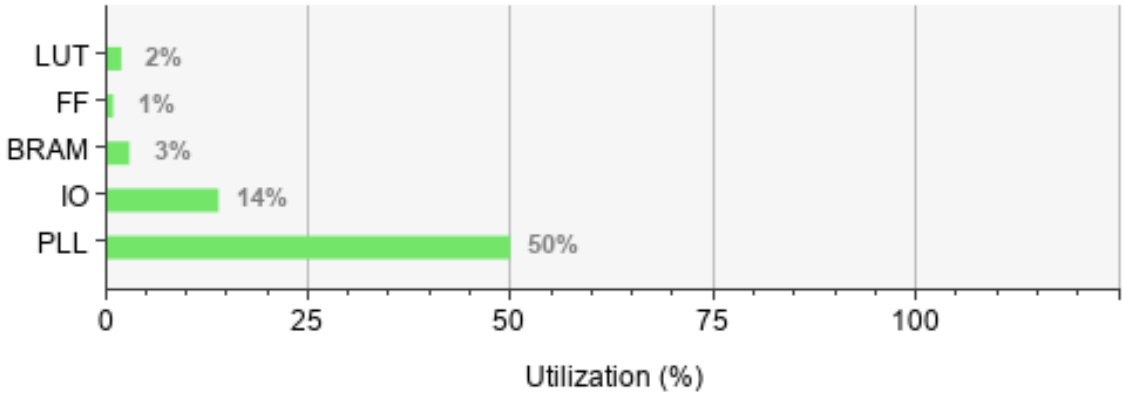


Anexo II. Tablas de Recursos

Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BRAM	URAM	DSP	Start	Elapsed	Run Strategy	Report Strate
synth_1	constrs_1	synth_design Complete!															Vivado Synthesis Defaults (Vivado Synthesis 2020)	Vivado Synth
impl_1	constrs_1	write_bitstream Complete!	30.935	0.000	0.190	0.000	0.000	0.337	0	272	170	2.0	0	0	12/5/24, 9:22 AM	00:01:36	Vivado Implementation Defaults (Vivado Implementation 2020)	Vivado Imple

Name	Slice LUTs (17600)	Slice Registers (35200)	F7 Muxes (8800)	F8 Muxes (4400)	Slice (4400)	LUT as Logic (17600)	Block RAM Tile (60)	Bonded IOB (100)	OLOGIC (100)	BUFGCTRL (32)	PLLE2_ADV (2)
Top_VGA	272	184	11	2	95	272	2	14	8	2	1
CeldaGHOST (MEghost)	36	34	0	0	14	36	0	0	0	0	0
CELDAPACMAN (PosicionPACman)	46	39	4	1	20	46	0	0	0	0	0
cMemoriaPSJred (ROM1b_1f_red_imagenes16_16x16)	6	0	0	0	3	6	0.5	0	0	0	0
HDMI (hdmi_rgb2tmds)	12	24	0	0	10	12	0	0	8	0	0
MemoriaMap (ROM1b_1f_racetrack_1)	23	25	1	0	10	23	0	0	0	0	0
MemoriaMapGreen (ROM1b_1f_green_racetrack_1)	9	28	4	0	8	9	0	0	0	0	0
MemoriaPSJ (ROM1b_1f_imagenes16_16x16_bn)	18	0	0	0	6	18	0.5	0	0	0	0
MemoriaPSJblue (ROM1b_1f_blue_imagenes16_16x16)	25	0	2	1	10	25	0.5	0	0	0	0
MemoriaPSJgreen (ROM1b_1f_green_imagenes16_16x16)	0	0	0	0	0	0	0.5	0	0	0	0
PINTAc (pinta_barras)	5	3	0	0	5	5	0	0	0	0	0
PLL (clock_gen)	0	0	0	0	0	0	0	0	0	2	1
RTCPSJ (RotacionPSJ)	26	11	0	0	13	26	0	0	0	0	0
SYNCVGA (sync_vga)	66	20	0	0	29	66	0	0	0	0	0

Resource	Utilization	Available	Utilization %
LUT	272	17600	1.55
FF	184	35200	0.52
BRAM	2	60	3.33
IO	14	100	14.00
PLL	1	2	50.00



Anexo III. Teórica para la rotación de sprites

La manipulación de las coordenadas de filas y columnas durante el acceso a la memoria en consonancia con la lectura posterior de datos de salida con el uso de un multiplexor conforman la base de rotación de sprites en función del sentido de dirección. Este proceso aprovecha las propiedades de la indexación en matrices y la lógica de inversión y permutación de bits para modificar la orientación del sprite representado en la pantalla.

• Invertir filas: se realiza aplicando una operación lógica not sobre las coordenadas de fila que modifican el sentido de lectura, los datos son leídos en sentido inverso.

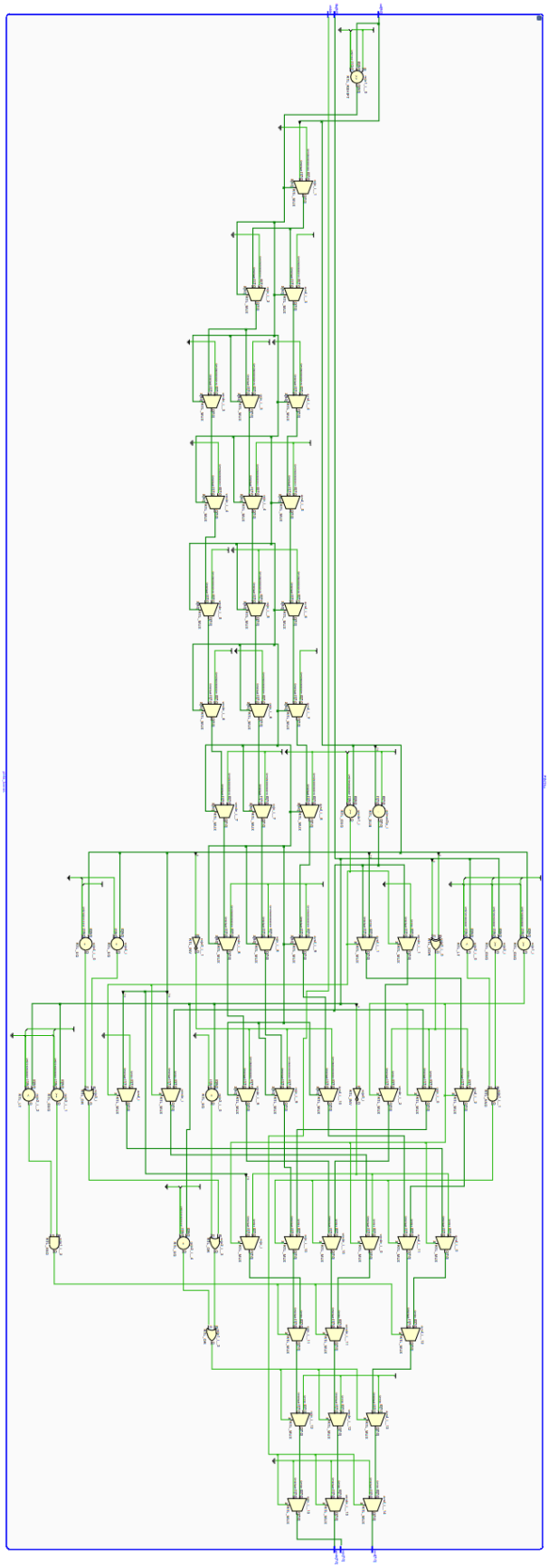
• Invertir columnas: al aplicar not a las columnas, el orden de lectura se invierte horizontalmente. Esto provoca que la imagen parezca mirar hacia el lado opuesto en la pantalla.

• Intercambio de filas y columnas la fila de se convierte en la columna correspondiente y viceversa generando una rotación de 90 grados en la orientación del sprites de realizarse una inversión previa o posterior de las filas o columnas, se pueden lograr rotaciones en múltiplos de 90 grados (90°, 180°, 270°).

• Combinando las operaciones de inversión y transposición, es posible orientar el sprite en cualquier dirección deseada.

Anexo IV. Estructura interna PINTAc

INICIAL, **PINTAcInicialschematic.pdf**



FINAL, **PINTAcschematic.pdf**