Sede Heredia

Proyecto Final

PONG

Pr. Final:Diseño y Valid Circuito Int C.

CPSC-08

Profesor:

Jonathan Martin Guevara Soto

Estudiante:

Kevin Josué García Hernández

Heredia, Costa Rica

Agosto, 2025

Tabla de contenido

Introducción

El presente trabajo se centra en el diseño de circuitos digitales mediante RTL y FPGA, utilizando la FPGA Basys 3, el lenguaje Verilog y Vivado como entorno de desarrollo. El propósito principal del proyecto es adquirir conocimientos y experiencia práctica en diseño digital, así como familiarizarse con la implementación de sistemas funcionales en FPGA. Para ello, se recrearon siete proyectos descritos en el sitio web Nandland, siendo seis de estos proyectos la base para construir un proyecto final: la recreación del juego clásico PONG de los años 70. Durante el desarrollo, se logró implementar la proyección del juego a través de VGA, la visualización de las paletas y la pelota, así como un marcador de puntaje en un display de 7 segmentos. Este proyecto permitió enfrentar retos propios del diseño digital, consolidando habilidades en programación en Verilog y en la integración de sistemas de hardware y visualización.

Objetivos

Objetivo General:

El objetivo principal de este proyecto es aprovechar un proyecto existente, en este caso PONG, como medio de aprendizaje práctico de Verilog. Se busca adquirir conocimiento a partir de los problemas y dificultades encontradas durante la elaboración e implementación del proyecto, al mismo tiempo que se consigue familiaridad con el lenguaje y se desarrolla soltura en su uso.

Objetivos Específicos:

1. Elaborar el juego PONG partiendo de proyectos previos, utilizando estos como bloques fundamentales para el desarrollo del proyecto final.

2. Implementar y verificar circuitos que permitan la lectura y control de señales externas, asegurando su estabilidad mediante técnicas como debounce.

3. Diseñar y controlar un display de 7 segmentos para representar datos, integrando la lógica secuencial y combinacional previamente aprendida.

4. Desarrollar la comunicación serial mediante UART, tanto para recepción como transmisión de datos, entendiendo el manejo de bits de inicio, datos, paridad y parada.

5. Generar señales de video compatibles con VGA, comprendiendo la temporización, sincronización y representación de colores en pantalla.

6. Integrar los distintos módulos para recrear el juego PONG, aplicando los conocimientos adquiridos en Verilog y resolviendo los problemas surgidos durante la implementación.

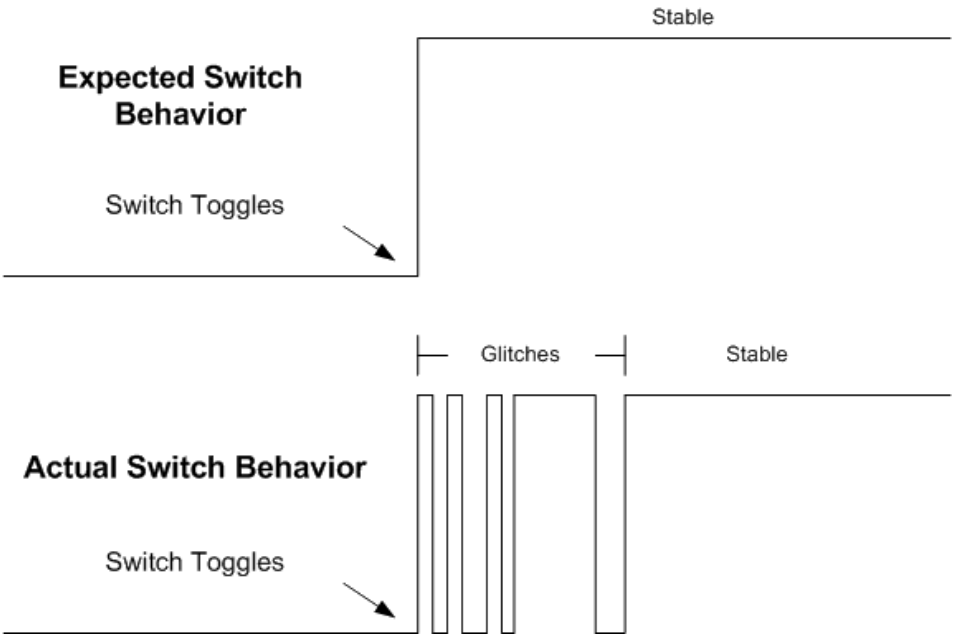
7. Adquirir soltura y confianza en el uso de Verilog para diseñar, implementar y depurar sistemas digitales completos.

Marco Teórico

Debouncing

Bouncing:

Es un fenómeno que se produce a la hora de interactuar con un switch o botón mecánico. Durante un breve periodo de tiempo el estado del switch es inestable, y rebota entre los estados alto y bajo.



Debouncing en Verilog

Debouncing es una técnica mediante la cual se filtran estos rebotes para únicamente capturar los momentos donde el switch se encuentra estable. En Verilog se puede crear un modulo que se encargue de recibir la señal de entrada y mediante un contador y un registro, se determine en que momento el estado del switch es estable.

Seven Segment Display

Un display de 7 segmentos es un indicardor visual utilizado normalmente para mostrar números, en este caso se quiere hacer uso de este para mostrar el conteo de los puntos obtenidos en el juego de PONG

Para poder controlarlo mediante Verilog se plantea el uso de un modulo que reciba una entrada binaria del valor que se quiere representar y devuelva los estados de cada segmento para representar el valor esperado.

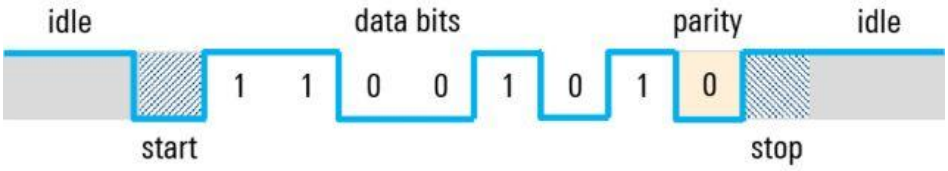
Para la elaboración del módulo respectivo se tomará de base la siguiente tabla de codificación hexadecimal.

Digit	gfedcba	abcdefg	a	b	c	d	e	f	g
0	0×3F	0×7E	**on**	**on**	**on**	**on**	**on**	**on**	off
1	0×06	0×30	off	**on**	**on**	off	off	off	off
2	0×5B	0×6D	**on**	**on**	off	**on**	**on**	off	**on**
3	0×4F	0×79	**on**	**on**	**on**	**on**	off	off	**on**
4	0×66	0×33	off	**on**	**on**	off	off	**on**	**on**
5	0×6D	0×5B	**on**	off	**on**	**on**	off	**on**	**on**
6	0×7D	0×5F	**on**	off	**on**	**on**	**on**	**on**	**on**
7	0×07	0×70	**on**	**on**	**on**	off	off	off	off
8	0×7F	0×7F	**on**	**on**	**on**	**on**	**on**	**on**	**on**
9	0×6F	0×7B	**on**	**on**	**on**	**on**	off	**on**	**on**
A	0×77	0×77	**on**	**on**	**on**	off	**on**	**on**	**on**
b	0×7C	0×1F	off	off	**on**	**on**	**on**	**on**	**on**
C	0×39	0×4E	**on**	off	off	**on**	**on**	**on**	off
d	0×5E	0×3D	off	**on**	**on**	**on**	**on**	off	**on**
E	0×79	0×4F	**on**	off	off	**on**	**on**	**on**	**on**
F	0×71	0×47	**on**	off	off	off	**on**	**on**	**on**

UART

UART (receptor/transmisor asíncrono universal) es un protocolo que establece normas para el intercambio de datos en serie entre dos dispositivos. UART utiliza únicamente dos hilos entre el transmisor y el receptor para transmitir y recibir datos en ambas direcciones. Ambos extremos tienen una conexión a masa. La comunicación en UART puede ser simplex (los datos se envían en una sola dirección), semidúplex (cada extremo se comunica, pero solo uno al mismo tiempo) o dúplex completo (ambos extremos pueden transmitir simultáneamente). En UART, los datos se transmiten en forma de tramas.

Las tramas de UART contienen bits de inicio y de parada, bits de datos, y un bit de paridad.



Como en la mayoría de los sistemas digitales, un nivel de tensión "alto" se utiliza para indicar un "1" lógico, mientras que un nivel de tensión "bajo" se emplea para representar un "0" lógico. Dado que el protocolo UART no especifica tensiones o rangos de tensión precisos para estos niveles, se suele denominar al nivel alto como "marca" y al bajo como "espacio". Cabe destacar que en el estado de reposo (cuando no

se están transmitiendo datos), la línea se mantiene en un estado alto. Esto facilita la detección de una línea o un transmisor defectuoso.

Bits de inicio y de parada

Debido a que UART es asíncrono, el transmisor necesita señalizar que los bits de datos están siendo enviados. Esta señalización se realiza mediante el bit de inicio, una transición del estado de reposo alto a un estado bajo, seguido inmediatamente por los bits de carga útil (datos). Una vez finalizados los bits de datos, el bit de parada indica el fin de la carga útil. El bit de parada puede ser una transición de retorno al estado alto o de reposo, o bien la permanencia en el estado alto durante un tiempo adicional. Se puede configurar un segundo bit de parada (opcional) para dar al receptor tiempo para prepararse para la siguiente trama, aunque esta práctica no es muy común.

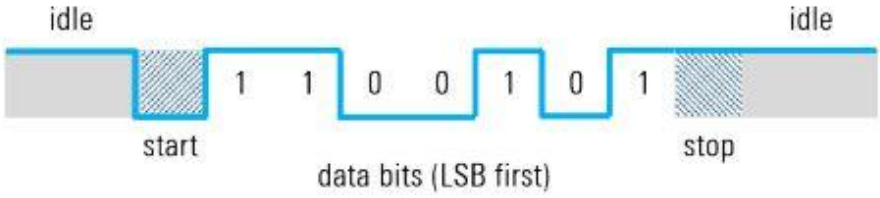
Bits de datos

Los bits de datos representan la carga útil o bits "útiles" y llegan inmediatamente después del bit de inicio. Puede haber entre 5 y 9 bits de carga útil, siendo más comunes 7 u 8 bits. Estos bits de datos suelen transmitirse con el bit menos significativo primero.

Por ejemplo, si se desea enviar la letra mayúscula "S" en ASCII de 7 bits, la secuencia de bits es 1010011. Primero se invierte el orden de los bits para colocarlos en el orden del bit menos significativo, es decir, 1100101, antes de enviarlos. Una vez enviados los últimos bits de datos, el bit de parada se utiliza para finalizar la trama y la línea regresa al estado de reposo.

"S" en ASCII con 7 bits (0x52) = 1010011

Orden del bit menos significativo = 1100101



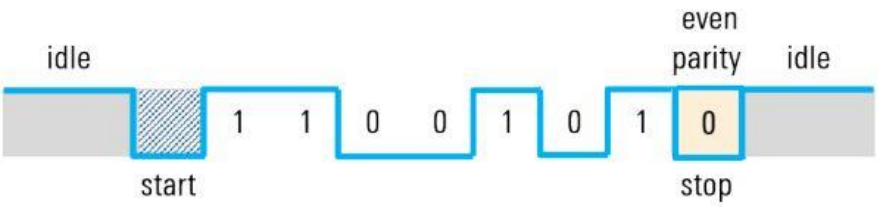
Bit de paridad

Una trama UART también puede incluir un bit de paridad opcional para la detección de errores. Este bit se inserta entre los bits de fin de datos y el bit de parada. El valor del bit de paridad depende del tipo de paridad utilizado (par o impar):

- En la **paridad par**, este bit se ajusta de tal forma que el número total de unos en la trama sea par.
- En la **paridad impar**, este bit se ajusta para que el número total de unos en la trama sea impar.

Por ejemplo, la letra "S" mayúscula (1010011) contiene tres ceros y cuatro unos. Si se utiliza la paridad par, el bit de paridad es cero, ya que el número de unos es par. Si se utiliza la paridad impar, el bit de paridad será uno para que la trama tenga un número impar de unos. El bit de paridad solo puede detectar un único bit invertido; si hay más de un bit invertido, no se detectará con fiabilidad utilizando un solo bit de paridad.

Ejemplo de bit de paridad



VGA

La tecnología VGA (Video Graphics Array) surgió originalmente como un estándar de hardware de visualización, introducido en 1987 por IBM® en su línea de computadoras PS/2. Con el tiempo, el término VGA se ha expandido para designar tanto los estándares de video analógico establecidos por VESA®, como el conector DE-15 (denominado comúnmente conector VGA) y, en muchos contextos, la resolución base de 640×480 píxeles.

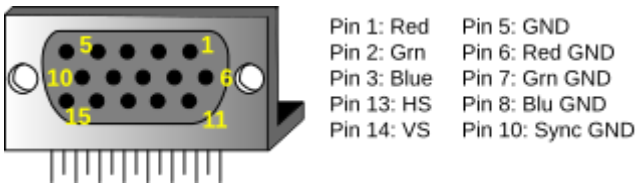
Dicho estándar de visualización analógica es desarrollado y administrado por la organización VESA (Video Electronics Standards Association). Para este proyecto, se utiliza como referencia la temporización correspondiente a una resolución VGA clásica de 640×480 píxeles.

Conector VGA

El conector DE-15 o VGA corresponde a un conector D-subminiatura de 15 pines distribuidos en tres filas, denominado así por su carcasa metálica en forma de letra "D". Aunque este conector gestiona múltiples señales, en este trabajo se consideran principalmente cinco: Red, Green, Blue (señales analógicas que determinan el color de cada píxel), y HS (Horizontal Sync) y VS (Vertical Sync), que sirven como referencias posicionales para la correcta ubicación de la imagen en pantalla.

Controlar de manera adecuada estas señales permite representar cualquier imagen o información en pantalla, siempre que se cumplan las especificaciones de

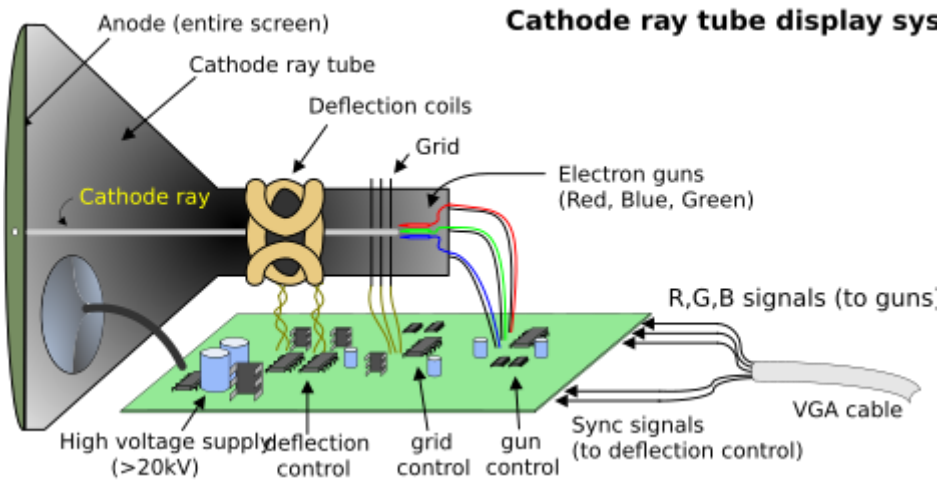
temporización establecidas para VGA. Comprender cómo generar dichas señales requiere analizar el principio de funcionamiento de los dispositivos de visualización.



Funcionamiento de los Monitores

Los monitores VGA basados en tubos CRT (Cathode Ray Tube), generan imágenes mediante haces de electrones modulados en amplitud que inciden sobre una superficie interna recubierta de fósforo, produciendo así la luminiscencia necesaria para formar la imagen visible. En contraste, los monitores LCD (Liquid Crystal Display) operan utilizando una matriz de diminutos elementos que, al someterse a variaciones de voltaje eléctrico, modifican sus propiedades ópticas, lo cual permite regular el paso de la luz y controlar de manera precisa la luminosidad de cada píxel.

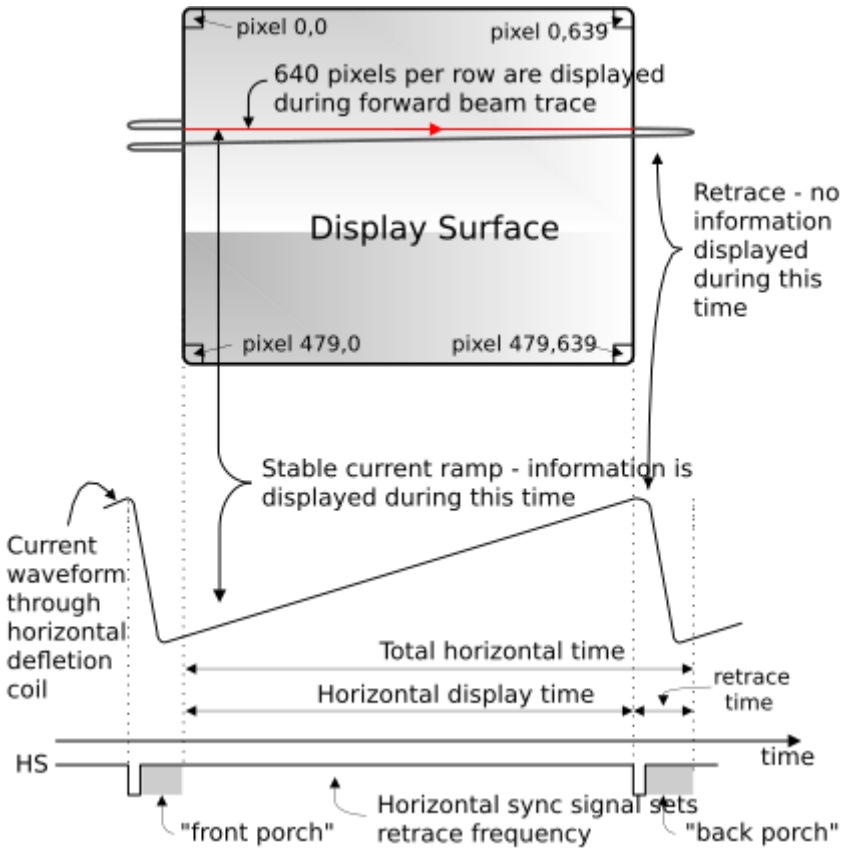
Aunque la siguiente descripción está centrada en pantallas CRT, los monitores LCD modernos han sido diseñados para operar utilizando los mismos esquemas de temporización, lo que permite compatibilidad en términos de señalización entre ambos tipos de tecnologías.



Los monitores CRT en color emplean tres haces de electrones, correspondientes a los colores rojo, verde y azul. Estos haces inciden sobre el recubrimiento de fósforo ubicado en la parte interna de la pantalla, provocando la emisión de luz y formando las imágenes. Los haces de electrones se generan en los denominados cañones de electrones, constituidos por cátodos calentados ubicados cerca de una rejilla cargada positivamente, la cual ejerce fuerza electrostática sobre los electrones, dirigiéndolos hacia la pantalla. La fuerza electrostática de la rejilla atrae electrones desde los cátodos, creando un haz cuya intensidad está determinada por la corriente suministrada al cañón. Tras pasar por la rejilla, los haces de electrones son acelerados por la elevada diferencia de potencial generada por la carga de la pantalla de fósforo (comúnmente de 20kV o más), permitiendo que impacten con la energía suficiente para iluminar el fósforo. Al atravesar la rejilla, los haces se concentran en un haz fino y finalmente impactan la superficie recubierta de fósforo. Este impacto genera luminiscencia, manteniéndose visible durante un breve lapso tras retirar el haz. La intensidad luminosa de cada punto en pantalla depende directamente de la corriente suministrada al cañón de electrones.

Entre la rejilla y la superficie del fósforo, el haz atraviesa el cuello del CRT, donde se encuentran dos bobinas encargadas de generar campos magnéticos que permiten la deflexión precisa del haz. Dado que el haz está compuesto de electrones, estos pueden ser desviados mediante campos magnéticos, posibilitando el barrido de la imagen sobre la superficie del monitor. Las bobinas generan campos magnéticos oscilantes que desvían el haz de electrones para cubrir toda la pantalla en un patrón de barrido horizontal y vertical (raster), permitiendo construir las imágenes línea por línea. Durante el recorrido del haz sobre la superficie, la variación en la corriente de los cañones de electrones modula la intensidad de cada punto, generando distintas tonalidades de color.

La visualización ocurre únicamente durante el movimiento activo del haz, mientras que en los periodos de retorno horizontal o vertical (blanking) no se representa información en pantalla, lo cual es necesario para reposicionar el haz de forma estable. En consecuencia, una fracción considerable del tiempo de operación se destina a estos intervalos de blanking, reduciendo el tiempo útil de visualización por cuadro. La resolución final de la imagen depende tanto del grosor de los haces de electrones como de la velocidad de barrido y de la capacidad de modulación de los cañones electrónicos.



Especificación de Temporización VGA

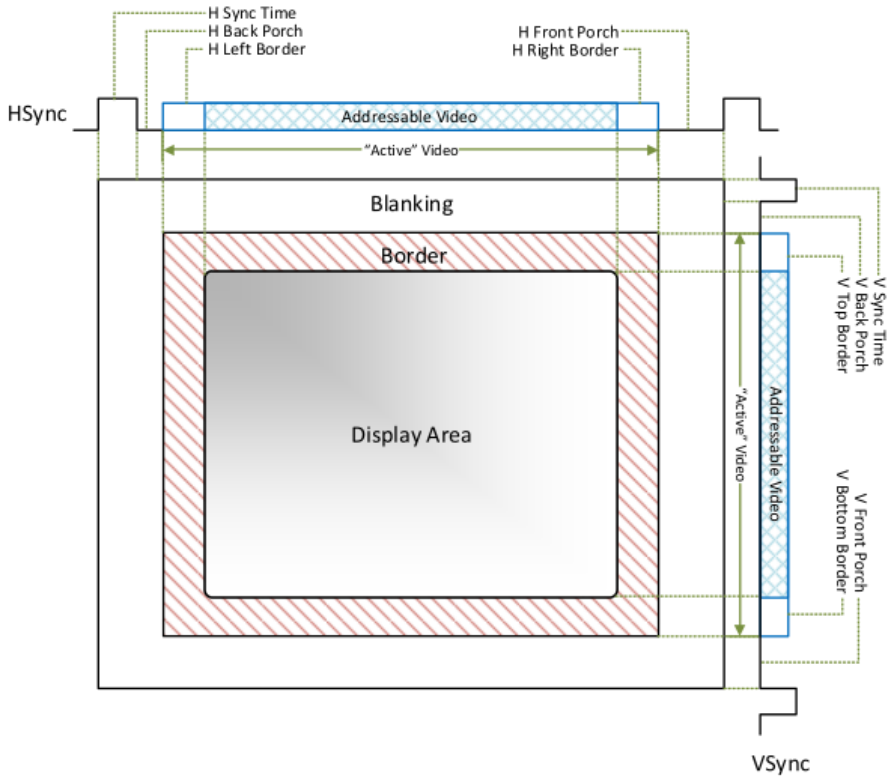
Los monitores VGA modernos admiten diversas resoluciones, cuya selección depende del diseño de un circuito controlador encargado de generar las señales de temporización. Este controlador produce impulsos de sincronización, típicamente de 3.3V o 5V, que determinan la frecuencia de barrido en las bobinas de deflexión, y

sincroniza la entrega de datos de video para garantizar que la imagen se proyecte en el momento preciso sobre la pantalla. Las pantallas de tipo raster organizan la imagen en filas y columnas, donde las filas representan las pasadas horizontales del haz y las columnas delimitan las zonas asignadas a cada píxel. Habitualmente, las resoluciones varían entre 240 y 1200 líneas verticales, y entre 320 y 1600 píxeles horizontales, siendo el tamaño del píxel resultado directo de estas dimensiones. Los datos de video generalmente provienen de una memoria de refresco, con uno o más bytes asignados a cada píxel (por ejemplo, la Nexys4 usa 12 bits por píxel, mientras que Nexys 2, Nexys 3 y Basys2 usan 8 bits). El controlador debe indexar esta memoria conforme el haz avanza por la pantalla, recuperando y aplicando los datos de video con precisión temporal sobre cada píxel.

Un controlador VGA tiene la función de producir las señales de sincronización horizontal (HS) y vertical (VS), y de coordinar la entrega de datos de video, siguiendo la cadencia impuesta por el reloj de píxel. Este reloj marca el tiempo disponible para representar cada punto de imagen, mientras que la señal VS determina la frecuencia con la que la pantalla se actualiza por completo, parámetro que depende tanto de las características del fósforo como de la intensidad del haz. En la práctica, las frecuencias de refresco oscilan entre 50Hz y 120Hz. La cantidad de líneas visualizadas a determinada frecuencia establece la frecuencia de retorno horizontal.

Especificación de Temporización para 640x480@60Hz

La siguiente imagen y tabla las especificaciones de temporización para el estándar VGA en resolución 640×480, operando a una frecuencia de refresco de 60Hz.



Description	**Notation**	**Time**	**Width/Freq**
Pixel Clock	tclk	39.7 ns (± 0.5%)	25.175MHz
Hor Sync Time	ths	3.813 µs	96 Pixels
Hor Back Porch	thbp	1.907 µs	48 Pixels
Hor Front Porch	thfp	0.636 µs	16 Pixels
Hor Addr Video Time	thaddr	25.422 µs	640 Pixels
Hor L/R Border	thbd	0 µs	0 Pixels
V Sync Time	tvs	0.064 ms	2 Lines
V Back Porch	tvbp	1.048 ms	33 Lines
V Front Porch	tvfp	0.318 ms	10 Lines
V Addr Video Time	tvaddr	15.253 ms	480 Lines
V T/B Border	tvbd	0 ms	0 Lines

Implementación en FPGA

Primero, se requiere un divisor de frecuencia para generar el reloj de píxel, el cual actúa como referencia temporal para las señales HS y VS. Aunque la especificación indica una frecuencia de 25.175 MHz, un reloj de 25 MHz es aceptable (dentro del margen de ±0.5%) y es fácil de generar en las tarjetas FPGA mediante divisores de reloj.

Segundo, se necesitan dos contadores: uno horizontal para contar píxeles por línea, y otro vertical para contar líneas por cuadro. El contador horizontal debe reiniciarse al llegar al final de la línea (799 en este caso), y al hacerlo debe generar una señal de Terminal Count que habilite al contador vertical para incrementar en uno, iniciando así una nueva línea. De igual manera, el contador vertical se reinicia al alcanzar el final del cuadro. Por tanto, se deben adaptar modificaciones a los contadores previamente implementados.

partir de los valores de los contadores, se pueden comparar con las constantes definidas para generar las señales HS y VS. Cabe destacar que las señales Red, Green y Blue deben mantenerse en GND fuera del área activa de visualización. La siguiente imagen muestra cómo generar las señales HS y VS según los valores de los contadores.

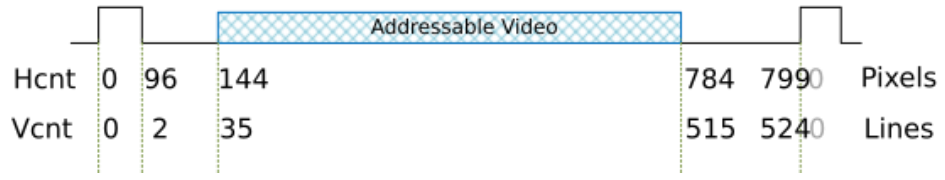
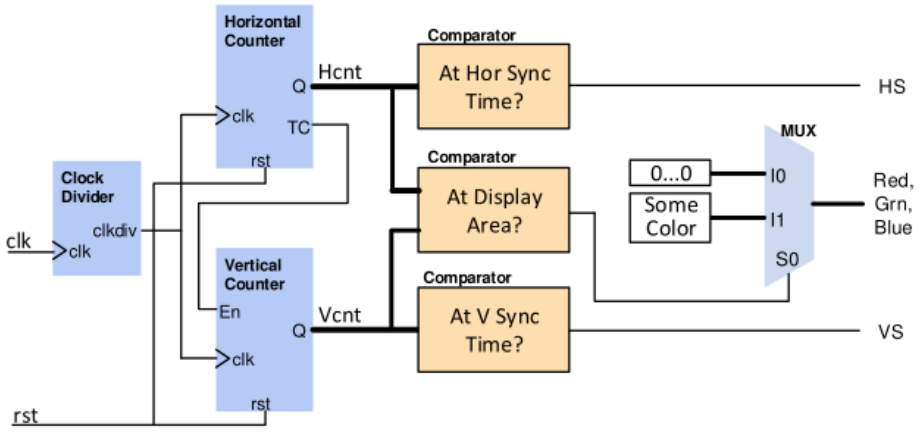


Diagrama de bloques



Desarrollo del proyecto

Proyecto 1: Flip-Flop

Descripción:

Este proyecto introduce el uso del Flip-Flop tipo D, uno de los bloques fundamentales en el diseño digital con FPGAs. Mientras que la lógica combinacional permite realizar operaciones en el momento, los Flip-Flops añaden la capacidad de almacenar información sincronizada con el reloj, lo que hace posible construir sistemas secuenciales como contadores, máquinas de estados o registros.

El objetivo práctico es lograr que un LED cambie de estado cada vez que se libera un interruptor. Para esto se emplean registros que permiten recordar el estado anterior del switch y detectar el flanco descendente de la señal (cuando pasa de 1 a 0). Una vez detectado, el valor almacenado en el registro del LED se invierte, logrando el comportamiento de "toggle".

A nivel de implementación, el diseño combina lógica secuencial (Flip-Flops para almacenar los estados) con lógica combinacional (detección del flanco), mostrando cómo ambos tipos de lógica trabajan en conjunto dentro de la FPGA.

Aprendizaje:

Durante la realización de este proyecto se observó que la mayoría de los flip-flops en la industria son de tipo D. Se exploró una técnica para detectar un falling edge precedido por un rising edge, utilizando un registro que detecta cuando la señal cambia de alto a bajo. Esto resulta útil, por ejemplo, en un bloque always sincronizado con un reloj, si se desea que al presionar un botón un LED se encienda no al momento de

presionarlo, sino al soltarlo. Este enfoque permitió comprender mejor el comportamiento de los registros para los futuros proyectos

Proyecto 2: Debounce a Switch

Descripción:

Este proyecto aborda el problema del rebote mecánico de los switches, un fenómeno en el que, al presionar o soltar un botón, la señal no cambia de manera limpia de 0 a 1, sino que oscila rápidamente entre ambos estados durante unos milisegundos. En un sistema digital, esto puede provocar que una sola pulsación sea interpretada como múltiples entradas.

El objetivo es diseñar un circuito de "debounce" dentro de la FPGA, de manera que el sistema solo reconozca un cambio de estado cuando la señal del switch se mantenga estable durante un breve tiempo. Para lograrlo, se combinan Flip-Flops y lógica secuencial: primero se sincroniza la entrada del switch con el reloj del sistema, y luego se emplea un mecanismo de verificación (como un contador) para asegurarse de que la señal permanezca estable antes de aceptar el cambio.

En la práctica, este proyecto enseña cómo limpiar y estabilizar señales provenientes del mundo real antes de usarlas en un diseño digital. Gracias a esto, una pulsación del botón produce exactamente una acción controlada, como encender o apagar un LED, sin errores por rebotes.

Modificaciones:

Se realizaron ajustes al código original, específicamente en el contador, para que concordara con la frecuencia de reloj de la Basys 3.

Aprendizaje:

Se aprendió a realizar debouncing en RTL, utilizando un contador que determina cuándo el estado del switch es estable. La duración del conteo depende del tiempo durante el cual la señal debe mantenerse sin cambios; en este caso, se determinó un tiempo de 10 ms para garantizar estabilidad. Este conocimiento fue fundamental para asegurar entradas confiables en los módulos posteriores del proyecto.

Proyecto 3: 7-Segment Display

Descripción:

Este proyecto presenta el uso de la pantalla de 7 segmentos en la FPGA, un periférico que permite mostrar números de forma visual. El sistema se basa en un contador binario que se incrementa cada vez que se libera un switch. Para evitar lecturas erróneas causadas por el rebote mecánico, se reutiliza el circuito de debounce diseñado en el proyecto anterior, garantizando que cada pulsación sea interpretada una sola vez.

El valor del contador, que avanza de 0 a 9 y luego vuelve a cero, se convierte en la representación adecuada para el display mediante un módulo llamado Binary_To_7Segment, el cual traduce el número binario a las señales que encienden los segmentos correctos. De esta manera, cada liberación del switch se refleja directamente como un cambio visible en la pantalla.

Este ejercicio integra conceptos aprendidos previamente —como flip-flops, detección de flancos y debounce— con un nuevo componente de salida. Además, muestra cómo combinar lógica secuencial (contador) con lógica combinacional (decodificación binaria a segmentos) para controlar un periférico externo.

Problemas:

A diferencia de la Go Board utilizada en los proyectos de Nandland, la Basys 3 tiene un display de 4 dígitos, mientras que la Go Board tenía dos displays de un digito. Esto implicó diseñar una lógica diferente para controlar correctamente los cuatro dígitos. Para este proyecto, se trabajó el display como si fuera un solo dígito, mostrando el mismo número en los cuatro, dejando el control independiente de cada dígito para el proyecto siguiente.

Aprendizaje:

Se utilizó codificación hexadecimal para asignar fácilmente el estado a cada segmento según cada carácter y número, y se generaron las salidas correspondientes a los segmentos del display. Este proyecto permitió consolidar conceptos de multiplexado y control de displays en Verilog.

Proyecto 4: Receptor UART

Descripción:

Este proyecto introduce la comunicación serial UART, un protocolo ampliamente utilizado para intercambiar datos entre una computadora y un dispositivo digital. El enfoque está en construir un receptor UART dentro de la FPGA, capaz de leer la información enviada por el PC a través de un puerto serial.

El sistema se basa en una máquina de estados finita (FSM) que controla el proceso de recepción: detecta el bit de inicio, lee los bits de datos de forma ordenada según el baud rate configurado, valida el bit de parada y finalmente entrega el byte completo a la lógica interna. Gracias a esto, la FPGA puede interpretar correctamente cada carácter enviado desde la computadora.

Este proyecto trabaja cómo implementar una FSM para manejar protocolos de comunicación y cómo integrar la entrada de datos externos con otros módulos, como un display de 7 segmentos.

Problemas:

Dado que el display de la Basys 3 es diferente al de la Go Board, se requirió ajustar la lógica para mostrar correctamente la información en los tres dígitos del display. Con ayuda de ChatGPT, se modificó el módulo para recibir 16 bits (4 por dígito) y multiplexar las salidas de manera que cada digito estuviera prendido por un periodo de tiempo y que el cambio fuera tan rápido que los 4 dígitos parecieran encendidos simultáneamente.

Aprendizaje:

Se adquirieron habilidades en el uso del display de 7 segmentos multiplexado, en la creación de máquinas de estado en Verilog, y en la implementación del protocolo UART para recepción de datos.

Proyecto 5: Transmisor UART

Descripción:

Este proyecto amplía el trabajo iniciado en la Parte 1 de UART, pasando ahora de recibir información a enviarla desde la FPGA hacia una computadora. Para lograrlo, se diseña un transmisor UART, encargado de tomar un byte de datos interno, convertirlo en una secuencia serial y enviarlo al puerto de comunicación.

El transmisor organiza la información en el formato estándar del protocolo UART: primero un bit de inicio (start bit), luego los bits de datos enviados uno a uno en orden, y finalmente un bit de parada (stop bit) que marca el final del paquete. Todo esto

se realiza respetando el baud rate configurado, de manera que el receptor (la computadora) pueda interpretar los datos correctamente.

Con este diseño, la FPGA no solo puede “escuchar” al PC (como en la Parte 1), sino también responder o transmitir sus propios datos, habilitando una comunicación bidireccional.

Aprendizaje:

Se completó la comprensión del protocolo UART, incluyendo la gestión de bits válidos y la sincronización de transmisión. Se puso especial atención a la importancia de mantener la línea en alto cuando no se está transmitiendo, y de registrar correctamente los datos recibidos y enviados.

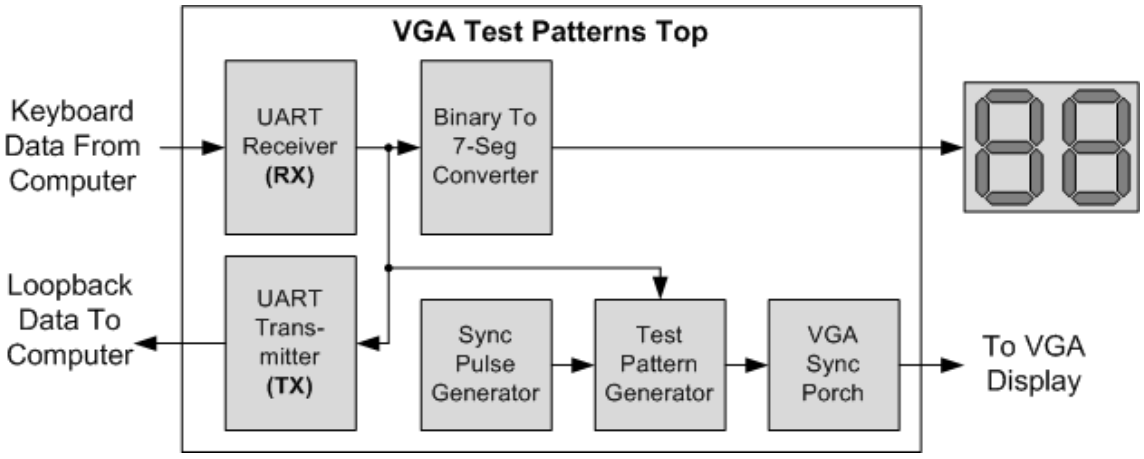
Proyecto 6: VGA

Descripción:

Este proyecto tiene como objetivo enseñar cómo usar una FPGA para generar señales de video compatibles con un monitor VGA. La idea central es controlar las señales de sincronización horizontal (HSYNC) y vertical (VSYNC), junto con las señales de color (RGB), de manera que el monitor pueda mostrar información visual correctamente.

Para ello, se generan patrones de prueba básicos, como líneas horizontales, líneas verticales y cuadros de colores, que permiten verificar que la FPGA está enviando las señales en el momento y formato correcto. Estos patrones no solo sirven para confirmar la conexión con el monitor, sino también para entender cómo se estructura la información de video en términos de sincronización y color.

El siguiente diagrama de bloques muestra como se compone este proyecto.



Sync Pulse Generator: Genera los pulsos de sincronización HSync y VSync usando un reloj de 25 MHz, definiendo las áreas activas e inactivas del marco. Permite ajustar fácilmente el tamaño del marco mediante parámetros de entrada.

Test Pattern Generator: Determina qué píxeles dibujar según la posición actual (fila/columna) y genera los diferentes colores de los patrones de prueba, usando los pulsos de sincronización del módulo anterior.

VGA Sync Porch: Ajusta los pulsos de sincronización para incluir los porches frontales y traseros necesarios, entregando las señales VGA finales a los pines del puerto.

Sync to Count: Calcula los índices de fila y columna del píxel activo, proporcionando esta información tanto al Test Pattern Generator como al VGA Sync Porch. Reutiliza el mismo módulo en dos lugares para optimizar el código.

Problemas

Problema 1: El primer problema que se identificó antes de elaborar el proyecto es que por los estándares de comunicación VGA, era necesario contar con un reloj de 25MHz, y el de la basys 3 es de 100MHz.

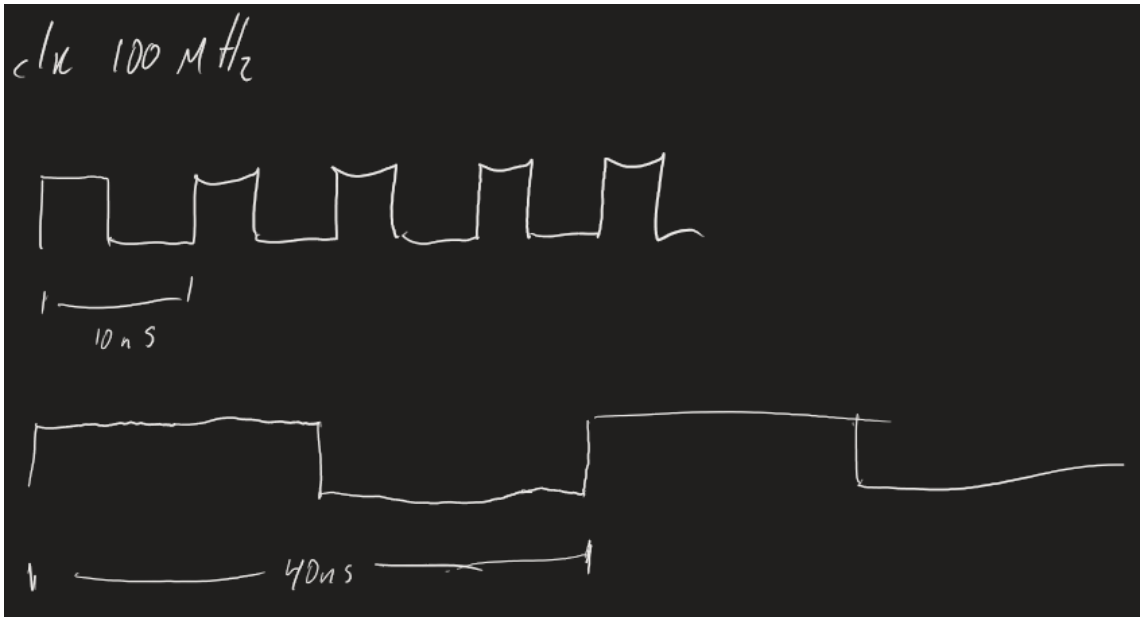
La solución: Se implementó de inmediato, creando un módulo encargado de dividir el reloj para convertirlo en uno de 25 MHz

Problema 2: Al implementar el proyecto en la FPGA, se observa que el monitor no da imagen, sino que muestra el mensaje “Fuera de Rango”, esto significaba que la señal que se le estaba enviando al monitor no se ajustaba al estándar del VGA.

Hipótesis: La primera hipótesis sugería un problema en las señales de sincronización vertical y horizontal, VSync y HSync.

Primera revisión: durante la primera revisión se corrigen parámetros incorrectos de correspondientes a los porches horizontales y verticales, los cuales no se correspondían con los establecidos en el estándar de VGA para 640x480. Sin embargo, al probarlo el problema continuo.

Segunda revisión: Se elaboró un testbench para revisar las señales de sincronización, las señales de color, y los relojes; se observo que el reloj de 25MHz no estaba cumpliendo con las características propias de un reloj de 25MHz el ciclo del reloj era mayor, es decir el divisor implementado no estaba funcionando correctamente.



El reloj de 25MHz debería de tener un ciclo por cada 4 ciclos del reloj de 100MHz, sin embargo, en la simulación se evidencio que el reloj resultante del divisor tenia un ciclo de correspondiente a 6 ciclos del reloj de 100MHz.

El problema radicaba en que al elaborar el divisor se estableció un cambio de semiciclo cada 2 ciclos del reloj original, pero se escribió que el contador contara hasta 2, sin tomar en cuenta que el 0 contaba por lo que en realidad pasaban 3 ciclos.

Solución: Arreglar el reloj de 25MHz para que, si entregara esa frecuencia, tras resolver este problema el monitor dejo de dar el mensaje y mostro la imagen que se estableció.

Problema 3: En algunos patrones los colores no tenían brillo, tras analizar la situación detenidamente y revisando todos los patrones y sus diferencias se planteó la siguiente hipótesis.

Hipótesis: Si se envía datos de color fuera de la zona activa del VGA el monitor no muestra el color de la zona activa correctamente.

Solución: se modificaron los patrones para que solo enviaran datos durante la zona activa y el problema se solucionó.

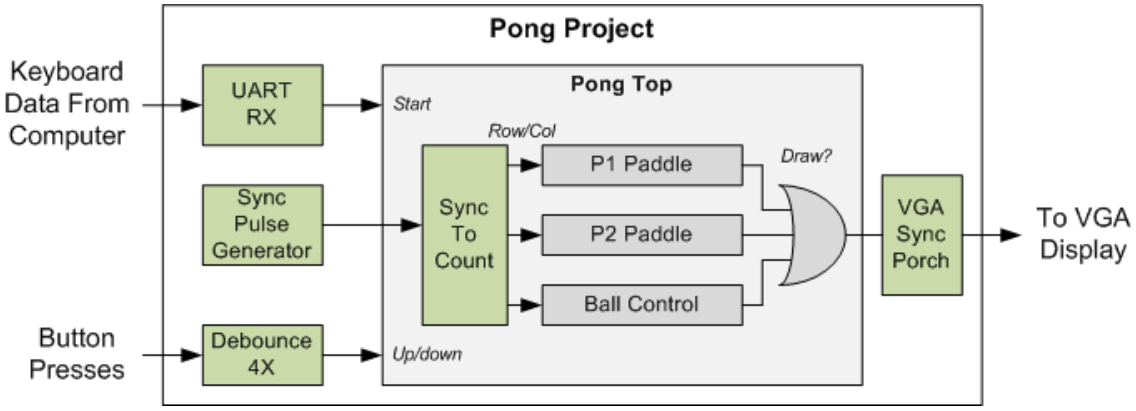
Proyecto Final: PONG

Descripción:

Este proyecto representa la culminación de todos los proyectos anteriores, consiste en recrear el clásico juego arcade PONG utilizando una FPGA y un monitor VGA. Cada jugador controla una paleta mediante los botones integrados en la FPGA: el

Jugador 1 usa los switches 1 y 2, y el Jugador 2 los switches 3 y 4. La lógica del juego se implementa en Verilog, organizando el código en módulos independientes que manejan cada paleta, la pelota y la salida VGA.

En el siguiente diagrama de bloques se muestra como se va a organizar este proyecto, y como se implementa cada bloque, siendo los verdes los provenientes de proyectos anteriores. Y los grises los nuevos que a implementar.



El tablero de PONG se configura en una resolución de 40×30 píxeles, ya que 40×30 es igual a 640×480 dividido por 16.

En las FPGA, realizar divisiones es complicado, a menos que se divida por una potencia de 2. Al dividir por una potencia de 2, simplemente se descartan los bits menos significativos necesarios para representar el divisor. En este ejemplo, se dividió por 16, lo que requiere 4 bits para representar; eliminando los 4 bits menos significativos de los índices de fila y columna, se logra la división por 16. Esto evita tener que implementar circuitos complejos de multiplicación o división.

El proyecto se compone de varios módulos clave:

UART RX: Permite iniciar el juego mediante un pulso de datos válidos, se reutiliza un módulo de proyectos anteriores en lugar de añadir un botón extra, esto debido a que la Go Board solo cuenta con 4 botones.

Generador de pulsos de sincronización, Sync To Count, VGA Sync Porch: Se reutilizan directamente de proyectos anteriores, encargándose de la sincronización horizontal y vertical de la pantalla.

Debounce de switches: Limpia las señales de los botones físicos, reutilizado de un proyecto previo para asegurar entradas confiables.

P1/P2 Paddle Control: Este es el primer código nuevo real. La lógica es la misma para ambos jugadores; la única diferencia se indica mediante un Parameter en Verilog.

Este módulo recibe la fila/columna actual que se está dibujando en la pantalla y mantiene la posición de la paleta si el jugador la mueve arriba o abajo. Si el índice de fila/columna activa coincide con la posición de la paleta, dibuja la paleta en pantalla. La salida o_Draw_Paddle en 1 dibuja un píxel en esa posición.

Ball Control: Este módulo mantiene la ubicación de la pelota en fila/columna. Si el píxel activo coincide con la posición de la pelota, la salida o_Draw_Ball se activa en 1; de lo contrario, es 0.

Big Or Gate: No es un módulo independiente. Su función es combinar las salidas de P1 Paddle Control, P2 Paddle Control y Ball Control. Si cualquiera indica dibujar en ese píxel, la salida se activa.

Problemas

Se presentaron pequeños problemas de compatibilidad de versiones entre la mía y la original para la Go Board, debido a las diferencias entre el hardware, además, se

cambiaron algunos de los nombres y forma de definir las entradas y salidas. Tras revisar el código y asegurar la compatibilidad con el resto de los módulos.

Problema 1: Al realizar las pruebas ya en la FPGA, se observa que, aunque las paletas funcionan la bola no se mueve.

Hipótesis 1: se plantea la posibilidad que haya un problema con la señal game active, la cual es la que se utiliza para determinar cuando la bola debe moverse o permanecer en el centro.

Revisión 1: se cambia la forma en la que se obtiene esta señal ya que se utilizaba lógica combinacional, que estaba siendo registrada por vivado como una advertencia critica, se cambio por un registro que se actualiza de acuerdo con el estado en el que se encuentre la máquina de estados.

Tras esta revisión la bola ahora si se movía, pero lo hacia en todo momento y rebotaba en todas las paredes, es decir no se reiniciaba o volvía al centro cuando no se encontraba con ninguna paleta como se suponían que debía ser, si la maquina de estados se encontraba en el estado de RUNNING

Hipótesis 2: la bola se está moviendo cuando la máquina de estados se encuentra en IDLE, y la forma en la que se estableció la actualización del registro no estaba siendo efectiva para que la bola se mantuviera en el centro. Esta hipótesis se reforzo al ver apretando el botón de START la bola volvía al centro y ahora no rebotaba si no tocaba una paleta, pero tras cada punto, la maquina de estados vuelve a IDLE, por lo que cuando se suponía que estaba en este estado la bola se movía y rebotaba en todos los bordes.

Revisión 2: Esta vez se cambio cuando el registro se actualizaba, se cambió para que mientras estaba en IDLE, se mantuviera actualizando de forma recursiva, y solo

cambiara en el momento que cambiaba a estado RUNNING. Tras realizar estas correcciones el comportamiento del juego paso a ser el esperado, la bola solo comenzaba a moverse tras presionar el botón de START.

Aprendizajes del proyecto:

Durante el desarrollo de los proyectos de Nandland, incluido PONG, se adquirió una mejor comprensión del funcionamiento y uso de los registros, es decir, los flip-flops, y de técnicas fundamentales como el debouncing para garantizar entradas estables. También se profundizó en el funcionamiento del protocolo de comunicación UART y en la forma de trabajar con la señalización VGA para la visualización en pantalla. Se aprendió a elaborar y utilizar máquinas de estado en Verilog, así como la importancia de aplicar técnicas de manipulación de datos en RTL, como realizar divisiones o multiplicaciones mediante desplazamiento de bits. De manera destacada, se comprendió que un proyecto debe construirse mediante bloques pequeños, reutilizables y probables de forma independiente, asegurando un avance eficiente y una estructura sólida, clara y metodológica.

Mejoras futuras

Momento:

Sería la implementación de la variabilidad de la velocidad de movimiento de la bola según el movimiento de la paleta al impactar en ella, se podría agregar un modificador de velocidad que compruebe la posición anterior de la paleta y el tiempo desde el último movimiento para de esta manera aumentar o disminuir el parámetro de velocidad de la bola.

Texto en pantalla

Una gran forma de mejorar este proyecto seria mediante la implementación de texto en pantalla, se podría utilizar para mostrar una especie de interfaz o solo para presentar el juego, al mismo tiempo que se podría utilizar para mostrar el puntaje de cada jugador, directamente en pantalla en lugar de un display de siete segmentos.

Uso de teclado

Se investigó la posibilidad de utilizar un teclado para el control de las paletas de PONG, pensando en recibir las teclas presionadas mediante comunicación UART. Sin embargo, existe un problema principal: los programas utilizados en Windows para enviar datos por protocolo serial normalmente emplean una interfaz de texto que solo puede enviar un carácter a la vez, por lo que no es posible transmitir múltiples teclas simultáneamente. Para superar esto, existen dos opciones:

La opción más práctica es desarrollar un programa propio que capture el estado completo del teclado y lo transmita por UART a la FPGA.

La opción mucho más compleja es implementar un USB Host directamente en la FPGA, para que ésta pueda comunicarse con el teclado USB de manera nativa.

Conclusión

El desarrollo del proyecto permitió implementar exitosamente los siete proyectos descritos en Nandland, utilizando la FPGA Basys 3, Vivado y Verilog. Los primeros seis proyectos sirvieron como base para construir el juego PONG, y aunque algunos presentaron problemas debido a diferencias de hardware entre la FPGA original (Go Board) y la utilizada, mediante el análisis y planteamiento de hipótesis todos estos problemas fueron solucionados y los proyectos completados satisfactoriamente.

En el proyecto PONG se lograron implementar los módulos de control de la pelota y las paletas, integrándolos sobre el módulo de VGA para la visualización del juego. Además, se incorporaron módulos auxiliares, como el de debouncing para asegurar entradas estables de los botones, y se añadió la funcionalidad de mostrar el puntaje en un display de 7 segmentos, mejorando el proyecto original de Nandland que manejaba el score solo a nivel interno. Finalmente, se dejó preparada la comunicación UART para futuras mejoras.

Este proyecto permitió consolidar conocimientos y habilidades en diseño digital, programación en Verilog e integración de módulos de hardware, cumpliendo con el objetivo de adquirir experiencia práctica mediante la recreación de proyectos existentes.

Referencias

Proyectos de Nandland

[1] Nandland, “Project 10: PONG,” The Go Board, [En línea]. Disponible: https://nandland.com/project-10-pong/

[2] Nandland, “Project 9: VGA Introduction - Driving Test Patterns to VGA Monitor,” The Go Board, [En línea]. Disponible: https://nandland.com/project-9-vga-introduction-driving-test-patterns-to-vga-monitor/

[3] Nandland, “Project 8: UART Part 2 - Transmit Data to Computer,” The Go Board, [En línea]. Disponible: https://nandland.com/project-8-uart-part-2-transmit-data-to-computer/

[4] Nandland, “Project 7: UART Part 1 - Receive Data from Computer,” The Go Board, [En línea]. Disponible: https://nandland.com/project-7-uart-part-1-receive-data-from-computer/

[5] Nandland, “Project 5: Seven Segment Display,” The Go Board, [En línea]. Disponible: https://nandland.com/project-5-seven-segment-display/

[6] Nandland, “Project 4: Debounce a Switch,” The Go Board, [En línea]. Disponible: https://nandland.com/project-4-debounce-a-switch/

[7] Nandland, “Project 3: The Flip Flop (aka Register),” The Go Board, [En línea]. Disponible: https://nandland.com/project-3-the-flip-flop-aka-register/

Recursos Adicionales

[8] Digilent, “VGA Display Controller,” [En línea]. Disponible: https://digilent.com/reference/learn/programmable-logic/tutorials/vga-display-congroller/start

[9] Rohde & Schwarz, “¿Qué es UART?,” [En línea]. Disponible: https://www.rohde-schwarz.com/es/productos/test-y-medida/essentials-test-equipment/digital-oscilloscopes/que-es-uart_254524.html