UNIVERSIDAD NACIONAL DE ASUNCIÓN

FACULTAD DE INGENIERÍA

Implementación del juego PONG usando un microprocesador MIPS

Proyecto Final – Sistemas digitales 2

Ingeniería Mecatrónica

8vo Semestre

Integrantes: Sebastian Marin y Jose Carlos Rios

Profesores: PhD. Vicente Gonzalez Ayala y Msc. Edgar Maquedad

San Lorenzo - Paraguay

2022

Índice

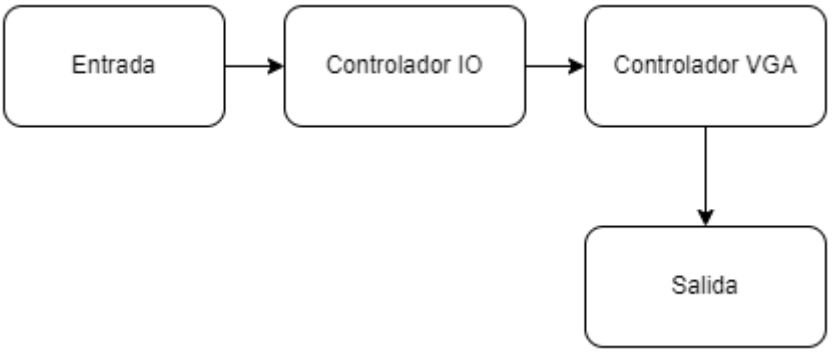
Introducción

En el siguiente trabajo se presenta la implementación con código Assembler y diseño de hardware en VHDL del juego Pong.

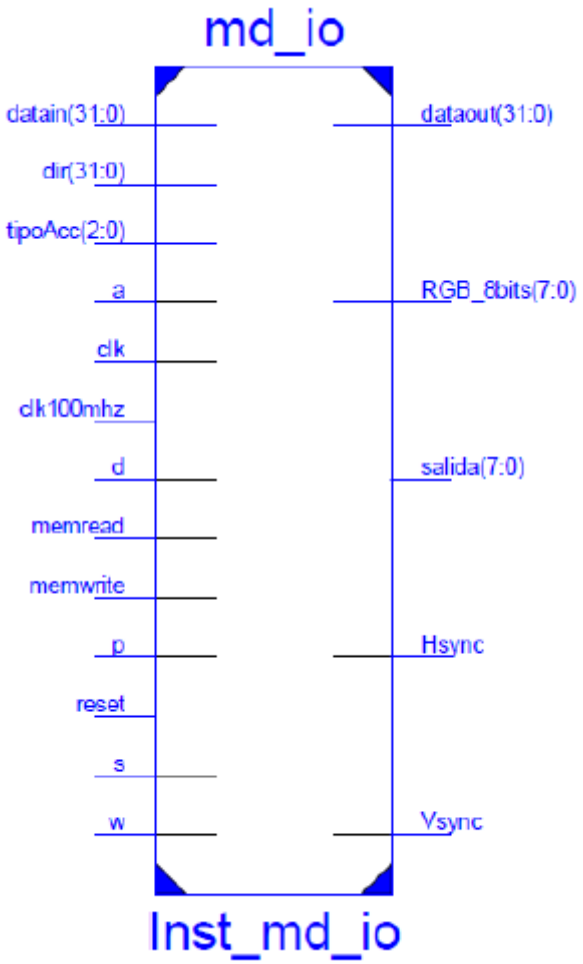
El juego Pong es un videojuego de la primera generación de videoconsolas publicado por Atari, creado por Nolan Brushlnell y lanzado el 29 de noviembre de 1972. Pong está basado en el deporte tenis de mesa (o ping pong) que en dos dimensiones simula su jugabilidad. El jugador controla en el juego una paleta moviendo verticalmente en la parte izquierda de la pantalla, y puede competir tanto contra un oponente controlado por computadora, como contra otro jugador humano que controla la segunda paleta en la parte opuesta. Los jugadores pueden usar las paletas para pegarle a la pelota hacia un lado y otro. El objetivo consiste en que uno de los jugadores consiga 4 puntos antes que su rival. Estos puntos se obtienen cuando el jugador opuesto falla en su intento de devolver la pelota.

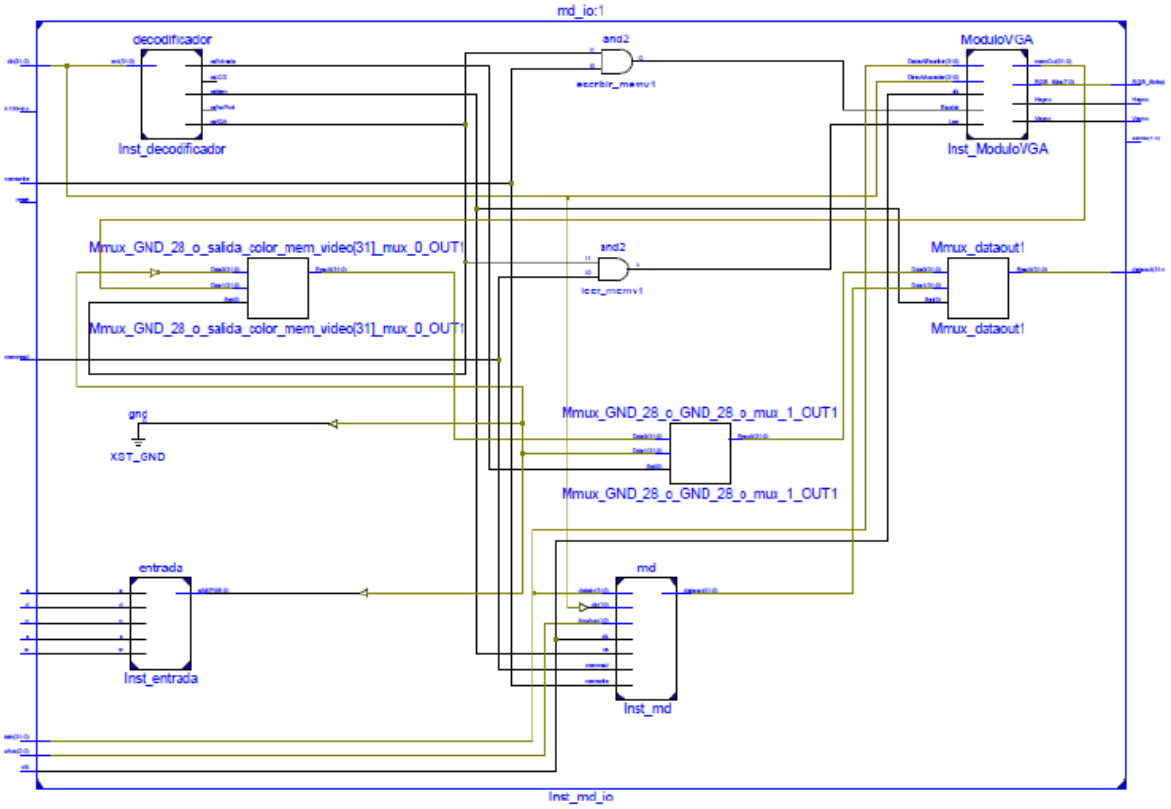
El código se programa en el lenguaje Ensamblador (Assembler) por medio del entorno MARS. El diseño del hardware con los módulos de entrada/salida y control de VGA se diseñan en VHDL en el entorno XILINX. Ambos procesos están detallados en los siguiente capítulos del trabajo.

Diagrama de bloques

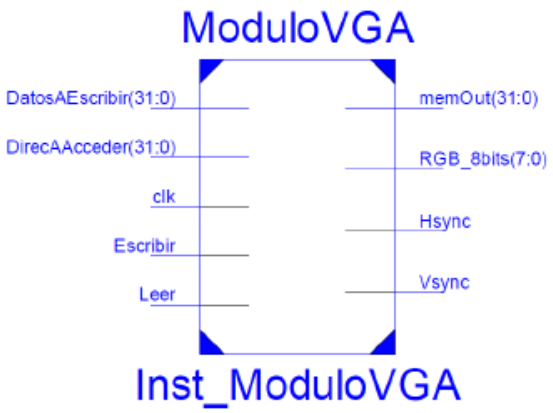


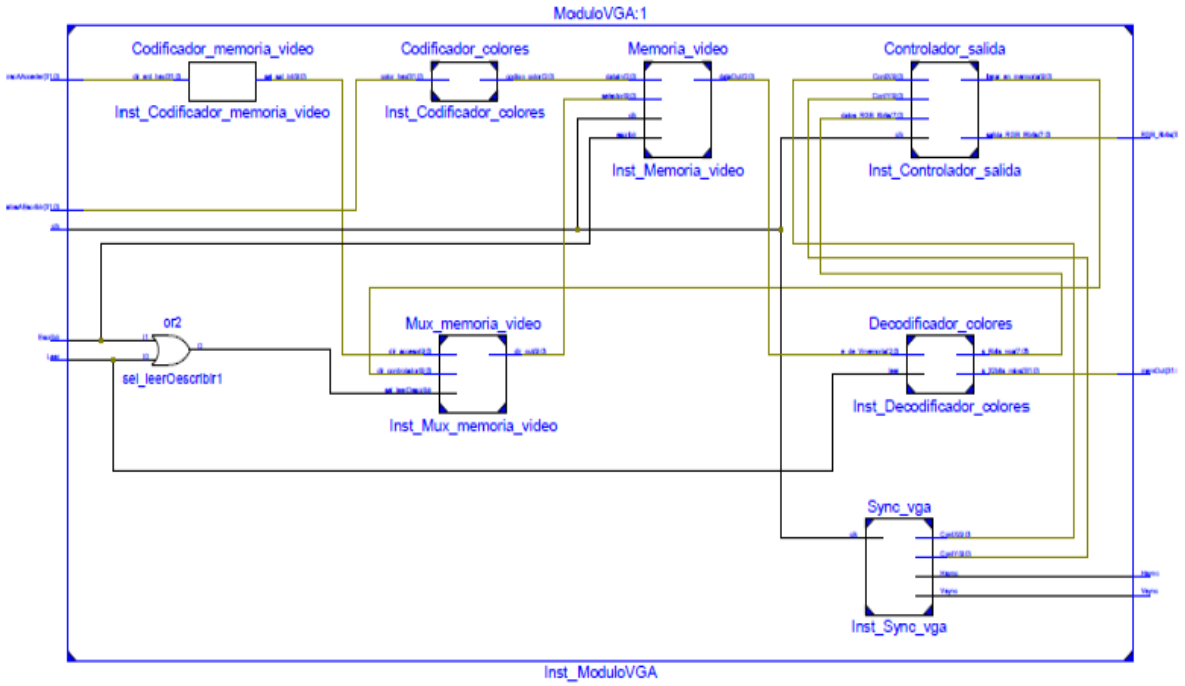
Módulos en VHDL




md_io:1
decodificador
Inst_decodificador
and2
escribir_memv1
ModuloVGA
Inst_ModuloVGA
Mmux_GND_28_o_salida_color_mem_video[31]_mux_0_OUT1
Mmux_GND_28_o_salida_color_mem_video[31]_mux_0_OUT1
and2
leer_memv1
Mmux_dataout1
Mmux_dataout1
gnd
XST_GND
Mmux_GND_28_o_GND_28_o_mux_1_OUT1
Mmux_GND_28_o_GND_28_o_mux_1_OUT1
entrada
Inst_entrada
md
Inst_md
Inst_md_io

Módulo VGA


ModuloVGA
DatosAEscribir(31:0)
DirecAAcceder(31:0)
clk
Escribir
Leer
memOut(31:0)
RGB_8bits(7:0)
Hsync
Vsync
Inst_ModuloVGA


ModuloVGA:1
Codificador_memoria_video
Inst_Codificador_memoria_video
Codificador_colores
Inst_Codificador_colores
Memoria_video
Inst_Memoria_video
Controlador_salida
Inst_Controlador_salida
or2
sel_leerOescribir1
Mux_memoria_video
Inst_Mux_memoria_video
Decodificador_colores
Inst_Decodificador_colores
Sync_vga
Inst_Sync_vga
Inst_ModuloVGA

Componentes Hardware implementados

Decodificador

El decodificador establece la comunicación entre el MIPS, el módulo VGA y las entradas digitales. Este nos ayuda a verificar el momento de activación y desactivación de los dispositivos mencionados según la dirección de los datos a los cuales se está accediendo.

Tenemos tres componentes conectados al módulo Decodificador: csVGA, csMem y csEntrada. El primero se trata de un habilitador del módulo VGA que se activa cuando se modifica algún dato de la memoria de video del VGA con una función **sw** del MIPS, en las direcciones que empiezan con “0x10008”. Además se habilita la lectura cuando sea necesario.

El módulo csMem es un habilitador de la memoria principal que tiene la función principal de activar y desactivar la memoria de datos cuya dirección de habilitación comienza con “0x1001”

El módulo csEntrada es el habilitador de datos del MIPS el cual permite la comunicación de los botones con el programa, su dirección es “0xFFFF0004”.

Entrada

En el módulo entrada se configuran las entradas de los botones para el movimiento de la paleta en el juego, la salida se concatena con ceros cuando el habilitador csEntrada tiene un 1.

Módulo VGA

Codificador de colores

Recibe como entrada un valor de 32 bits correspondiente a alguno de los 8 colores establecidos, este se codifica a un valor de 3 bits y se pasa a la entrada de datos de memoria.

Decodificador de colores

Devuelve al controlador de salida un color en formato de 8 bits correspondiente al dato de memoria que está siendo solicitado por el mismo. En el caso que el mips solicite algún dato de la memoria de video. El decodificador devuelve al MIPS el dato en formato de 32 bits.

Codificador de memoria de video

Se encarga de convertir la dirección de acceso a la memoria (es decir a la dirección a la cual se está queriendo escribir o solicitar un dato de memoria) en un número entero capaz de ser utilizado como selector por la memoria.

Multiplexor de memoria de video

Controla el acceso a memoria, si es que este se da por una solicitud del MIPS o por una solicitud del controlador de salida. Este último se encarga de imprimir en pantalla lo que hay en la memoria de video.

Memoria de video

La memoria de video es un arreglo de 1024 posiciones con 3 bits para cada posición. Es esta la encargada de guardar la información necesaria para imprimir en pantalla el color del píxel perteneciente al juego.

Sync VGA

Este módulo se encarga de utilizar la señal de reloj para iterar un contador horizontal y vertical, además de mandar las señales de Hsync y Vsync necesarias para hacer funcionar al monitor de forma correcta. Utilizamos una resolución de 640x480 a una tasa de refresco de 60Hz. Por tanto los contadores deben estar en el rango de 0 a 799 para el contador horizontal y de 0 a 524 para el vertical, ya que existen unas franjas vertical y horizontal de píxeles denominadas “porches” que no se utilizan para la salida de video pero deben considerarse en la configuración del VGA.

Controlador de salida

Encargado de dar la posición absoluta en pantalla por medio de los contadores del sync, y también de mandar la señal a la pantalla con los valores adecuados que fueron solicitados de la memoria de video, escalando al mismo tiempo el juego a la resolución nativa de la pantalla.

Software implementado

Primeramente se declaran las constantes a utilizar, se define el stack_end que es la dirección donde va a comenzar el stack pointer y las variables a utilizar para los jugadores, sus respectivas posiciones en X e Y, sus respectivos puntajes (que puede ir de 0 a 4), dos offsets, uno en X y otro en Y y por último una variable de colisión que se pone en '1' cuando la 'pelota' colisiona con una pared o una paleta.

En el main se inicializan las direcciones y los valores default (imagen 1)

```
main:
        #Stack
        #la             $sp, stack_end
        li direccionX, 1
        li direccionY, 0
        sw $0, colision

        li $t7, 0
        sw $t7, puntajeJugador1
        sw $t7, puntajeJugador2
        sw $t7, yOffset
        addi $t7,$t7,1
        sw $t7, xOffset

        la $sp, stack_end
        #addi $sp, $0, stack_end
```

Imagen 1

Seguidamente se renderizan las paletas de ambos jugadores usando la función llamada drawVertLine, la cual se le proporcionan los valores de posición en X e Y, indiceDelColor y cantidad de píxeles a pintar. Por ejemplo para el jugador 1 se inicia en la posición (1, 16) con color blanco y dibuja 7 pixeles hasta la posición (1, 22). Las coordenadas comienzan en la esquina superior izquierda (0, 0) y termina con la coordenada (31, 31) en la esquina inferior derecha.

Se dibujan las paredes que representan el borde de la cancha de juego con funciones similares a drawVertLine, el puntaje de ambos jugadores con la función drawScore, la pelota en el medio de la cancha con la función drawDot la cual es llamada cada vez que la pelota cambia de posición.

En todo momento para realizar cualquier movimiento se borra la posición de la pelota y se procede a preguntar si existe una colisión para decidir si hay que cambiar la dirección y posición de la pelota (imagen 2). Si se verifica que no existe colisión en ninguno de los ejes se mueve la pelota a su siguiente posición. Luego de cualquier movimiento de la pelota se realiza la consulta de si existe un input de botón para mover alguna paleta.

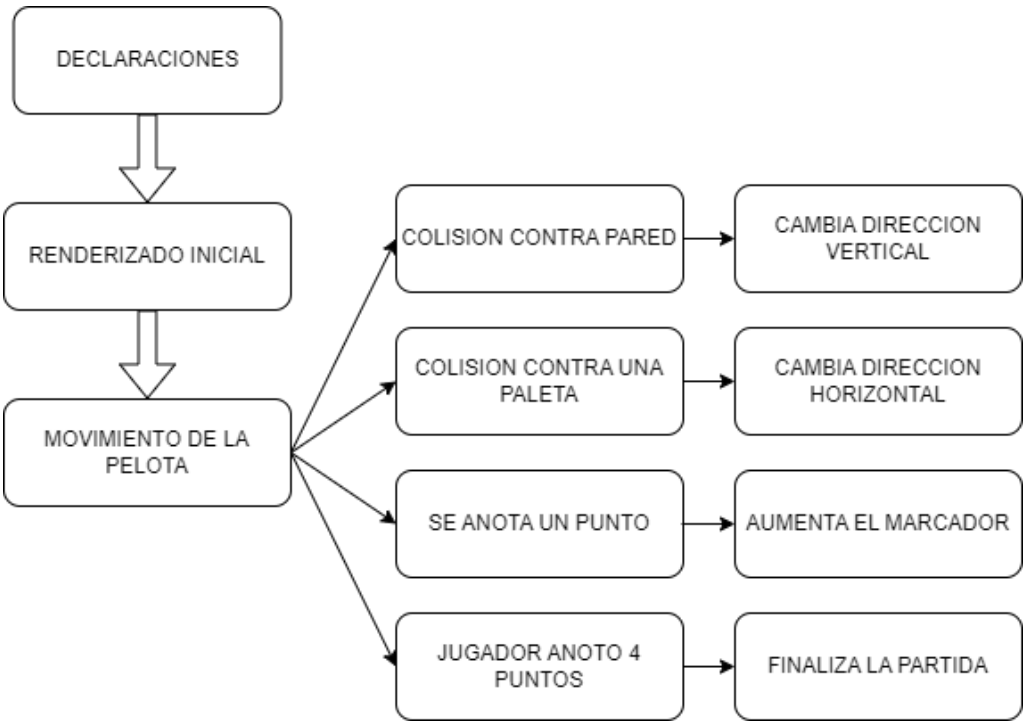


Imagen 2

Si ocurre algún input de botones se vuelve a llamar a la función para dibujar las paletas arriba o abajo, dependiendo del input, la cual previamente fue borrada su posición anterior. Si ocurre el caso de que la pelota colisionó con una pared lateral se le concede el punto al jugador del lado opuesto cambiando su marcador y reiniciando la pelota al centro del campo de juego. Por último si uno de los jugadores llega a los 4 puntos, se imprime en pantalla que es el ganador y seguidamente se reinicia el juego.

Estado final del trabajo

Actualmente el juego corre de forma óptima en la simulación dentro del entorno MARS. En la implementación con el FPGA funcionan correctamente los botones así como el controlador de VGA para la salida en pantalla, la codificación de colores RGB funciona de forma correcta. El problema a solucionar es el ocasionado por colisiones incorrectas que se producen. La posible causa de esto podría ser el manejo de la memoria, la cual sobreescribe espacios incorrectos como así también podría deberse a que no se está asignando memoria suficiente.

Podemos concluir que se logró la implementación del juego a un nivel satisfactorio si bien no en la práctica ya que siguen ocurriendo errores pero si en la didáctica ya que el proceso de aprendizaje fue provechoso para nosotros. A lo largo del proyecto se describieron múltiples componentes de hardware que pueden ser aprovechados para futuros proyectos.

Creemos que los conocimientos adquiridos durante el proyecto son valiosos y resultan indispensables para la elaboración de cualquier proyecto que involucre la implementación de hardware que utilice FPGAs.

Referencias Utilizadas

- Diapositivas de las clases de la cátedra Sistemas Digitales 2 de la Facultad de Ingeniería de la Universidad Nacional.
- Assemblers, Linkers, and the SPIM Simulator, James R. Larus.
- MIPS Assembly Game - Final Project for CSCB58
- https://en.wikipedia.org/wiki/8-bit_color
- https://numato.com/docs/mimas-v2-spartan-6-fpga-development-board-with-ddr-sdram/
- https://es.wikipedia.org/wiki/Pong