Instituto Tecnológico y de Estudios Superiores de Monterrey

Diseño con lógica programable

TE2002B - Grupo 100

Evidencia: Reto

Equipo 9

Hortencia Alejandra Ramírez Vázquez	A01750150
Karen Lizette Rodríguez Hernández	A01197734
Carlos Gaeta López	A01611248
José Gustavo Buenaventura Carreón	A01570891

Profesores:

Enrique González Guerrero
Raúl Peña Ortega

Fecha de entrega:

16 de marzo de 2022

Introducción

La evolución de la tecnología ha crecido exponencialmente en las últimas décadas, cada día diseñando nuevos dispositivos para que el ser humano desarrolle nuevas herramientas que le faciliten la vida. Todos estos requieren diferentes componentes para elaborarlos exitosamente. A nivel mundial todas las compañías al crear un nuevo dispositivo, componente o cualquier otro objeto pasan por un proceso que conlleva diferentes etapas de creación desde la lluvia de ideas del proyecto, la construcción del plano, la manufacturación de los materiales a utilizar entre otras fases. Una de estas etapas es la del prototipaje del proyecto donde se aportan ideas de lo que se quiere diseñar y las crean para probarlas y ver si son 100% funcionales al igual de apreciar qué cambios se le pueden hacer para el producto final. Todo esto se hace con la finalidad de crear un producto que sea funcional y que realmente las personas puedan aprovecharlo. Por esta razón en este bloque se va a diseñar y programar un videojuego utilizando un monitor VGA y una placa FPGA.

Justificación del problema

A lo largo de la historia el ser humano ha creado diferentes maneras de entretenerse, una de estas maneras de diversión son los videojuegos. El primer videojuego que se creó fue el “Tres en raya” utilizando un computador llamado EDSAC elaborado por Alexander Douglas y a partir de ahí el crecimiento de los videojuegos fue totalmente exponencial. Atari, una de las compañías de videojuegos más grandes de todos los tiempos, eventualmente nació y con ella surgió el primer juego comercial conocido, su nombre era “PONG”. A partir de ahí el reto de crear videojuegos fue subiendo y demás compañías visionarias siguieron los pasos de la Atari, creando nuevas y mejores tecnologías hasta llegar a la época actual donde los videojuegos superan la ficción.

En esta unidad de formación se han visto diferentes conceptos y herramientas para la elaboración de procesadores y dispositivos integrables a sistemas digitales como el uso de las placas FPGA, el lenguaje de ensamblador, la aplicación de VHDL en Quartus entre otras cosas; temas que nos demuestran el uso y la importancia de estas tecnologías en nuestras vidas.

Por esa misma razón, en este bloque se dio a la tarea de crear un prototipo funcional para recrear algún videojuego utilizando las herramientas vistas en clase como el uso del monitor VGA, el lenguaje ensamblador y VHDL. De esta misma manera podemos comprender el

funcionamiento completo de un procesador pues el problema se basa principalmente en crear uno con las herramientas que tenemos e integrar diferentes componentes para que funcione a la perfección.

Metodología

Para abordar esta problemática, lo primero que realizamos fué hacer una lluvia de ideas entre los integrantes del equipo para definir en cuál videojuego basar nuestro prototipo. Después de identificar ventajas y desventajas de cada una de ellas, se llegó a una conclusión. Sin embargo, al momento de empezar a probar y desarrollar el prototipo, nos dimos cuenta de que lo que queríamos lograr estaba fuera de nuestro alcance debido al tiempo con el que contábamos. Es por ello, que decidimos cambiar nuestro enfoque y diseñar un prototipo con base en los juegos populares Piano Tiles y Guitar Hero.

Después, definimos cuáles entradas y salidas se van a utilizar en el prototipo: las entradas serían los 10 interruptores y los 2 botones, y las salidas serían 2 displays de 7 segmentos y un monitor conectado por el puerto VGA.

Por consiguiente, nos dedicamos a implementar todo lo necesario para nuestro prototipo. Para lograrlo, utilizamos una tarjeta que contiene un FPGA llamada DE10 lite, en donde con ayuda de la aplicación Quartus Prime, pudimos escribir código en el lenguaje de programación VHDL y sintetizar los programas en la tarjeta. Lo primero que desarrollamos fue el diseño base en la interfaz del puerto VGA y lo mapeamos al procesador tipo “Gumnut” que realizamos.

Al estar estos dos componentes funcionando correctamente en nuestro sistema, decidimos diseñar e implementar una máquina de estados que describa el funcionamiento de nuestro videojuego. Además, incluímos que el puntaje se incrementará con la ayuda de un programa en Ensamblador y el procesador “Gumnut”. Finalmente, implementamos el funcionamiento de los botones (pausa y reset), agregamos las limitaciones del juego y comprobamos el correcto funcionamiento de nuestro prototipo.

Manual de Instrucciones para el juego o aplicación

Diseño

Para poder jugar Power Chore, es necesario entender las partes que lo componen. En la parte de abajo de la pantalla, se encuentran cuadros de color blanco que representan cada uno de los interruptores incorporados en la tarjeta, son 10 en total. Arriba de ellos, se encuentra una línea que marca el rango en donde se puede activar el interruptor y tocar la nota. Por último, en el resto de la pantalla van a aparecer cuadros, de diferentes colores, que representan notas, van a ir descendiendo desde la parte de arriba hasta los cuadros blancos. En la tarjeta, se puede observar en el display el puntaje que se va acumulando.

Objetivo

El objetivo de Power Chore es tocar la mayor cantidad de notas, esto se hace mediante la activación de los interruptores correctos dependiendo del lugar en donde está bajando la nota, es decir, si la nota se encuentra en la columna 3, el interruptor 3 debe de estar encendido entre la línea y los cuadros blancos para que el puntaje incremente. Entre más tiempo esté encendido el interruptor en el rango, mayor puntaje se obtiene. Con los botones dentro de la tarjeta se puede pausar (las notas dejan de aparecer) presionando el primer botón, o se puede resetear el puntaje presionando el segundo botón.

Código de colores

Las notas tienen diferente color dependiendo de la columna en la que se encuentren, para que al jugador sea más sencillo identificar el interruptor que se debe de encender. Además, los cuadros blancos cambian de color para reconocer el estado en el que se encuentran: si su color cambia a cian [Figura 1], significa que se activo el interruptor correspondiente a esa columna; si el color cambia a rojo [Figura 2], quiere decir que no se activo el interruptor correcto, por lo tanto no se asignan puntos por esa nota; y si el color es verde [Figura 3], significa que el interruptor correcto se activo y se asignan puntos por esa nota.

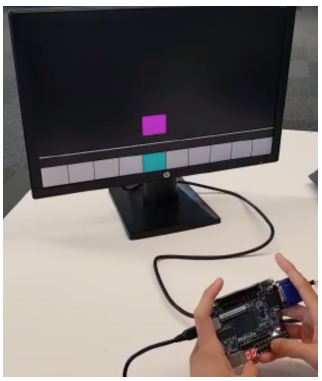

Figura 1

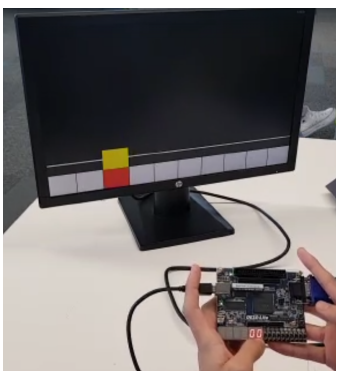

Figura 2

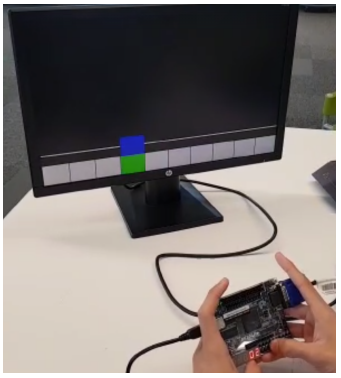

Figura 3

Limitantes

Es importante notar, que para que el puntaje aumente, es necesario activar solamente el interruptor correcto, ya que si se activan más (incluso incluyendo el interruptor correcto) no se darán puntos por esa nota.

Marco Teórico

FPGA

Las FPGAs, que por sus siglas en inglés significan Matrices de Compuertas Programables en Campo, son dispositivos lógicos programables de campo que constan de varias unidades lógicas, los módulos de conexión y las múltiples conexiones que hay entre ellas. Estos dispositivos tienen un nivel de complejidad más alto que los dispositivos lógicos programables simples, pero no deben confundirse con los dispositivos lógicos programables complejos. Las FPGAs nos permiten probar circuitos complejos descritos en VHDL o algún otro lenguaje de descripción y se sintetizan aquí. Podemos llegar a sintetizar desde compuertas lógicas simples hasta procesadores.

Máquina de estados

Las máquinas de estados finitas, o FSM por sus siglas en inglés, son modelos lógicos en donde las salidas de algún sistema dependen de estados, y estos estados se cambian dependiendo del estado en donde te encuentres, las entradas que tienes activas o inactivas y las señales de reloj. Existen primordialmente dos máquinas de estados, las máquinas de Moore, en donde las salidas dependen únicamente del estado actual, y las máquinas de Mealy, que dependen del estado actual y las entradas activas.

DE10-lite

La placa DE10-lite es una placa desarrollada por terasIC en donde se integra una FPGA Altera MAX 10 10M50DAF484C7G con un conjunto de periféricos como switches, botones, indicadores de 7-segmentos, salida VGA, acelerómetro, entre otros, y que cuenta con un conector USB para poder sintetizar circuitos diseñados en VHDL. Sirve principalmente para probar circuitos y tener una manera sensible de ver que funcionan.

Decoder

Un decoder es un circuito simple en donde tomamos una entrada de n-bits que representan números en binario desde 0 hasta 2^n -1, y regresamos diferentes salidas para cada número. Para el caso de nuestro proyecto, utilizamos un decoder de 4 bits para un display de 7 segmentos, así que para cada número de 0 a 15 nos daba las señales que el display necesitaba para representar el carácter en hexadecimal.

VGA

VGA son siglas que significan Matriz de Gráficos de Video. Es un protocolo para mostrar imágenes en algún monitor utilizando señales de reloj y las señales de color RGB. Para mostrar la imagen mandamos pulsos de sincronización tanto vertical y horizontal junto con las señales de color RGB para mostrar pixel por pixel todo el display.

ROM

Read Only Memory, o memoria de sólo lectura, es un dispositivo programable de fábrica que almacena datos. En una ROM, le damos de entrada un valor en binario que es la dirección de la memoria que queremos acceder y nos regresa el dato que está almacenado. Las memorias pueden almacenar desde 1 hasta varios bits. La cantidad de memoria que puede almacenar depende de los bits de dirección, si tenemos m bits de entrada de dirección, podemos tener hasta 2^m alocaciones de memoria

Gumnut

Gumnut es un procesador simple que puede sintetizarse en nuestra FPGA mediante Quartus lite. Este procesador se maneja mediante el lenguaje de ensamblador, cuenta con 8 registros que se acceden de manera rápida y una memoria donde igual se pueden almacenar más datos pero su tiempo de acceso es mayor. Puede realizar tareas de asignación, aritméticas y lógicas, al igual que de input y output.

Diagrama de la máquina de estados

Para describir el funcionamiento lógico bajo el que se ejecuta el juego, se diseñó una máquina de estados que cuenta con tres estados, a través de los cuales podemos visualizar la transición de los cuadros de arriba de la pantalla hacia abajo.

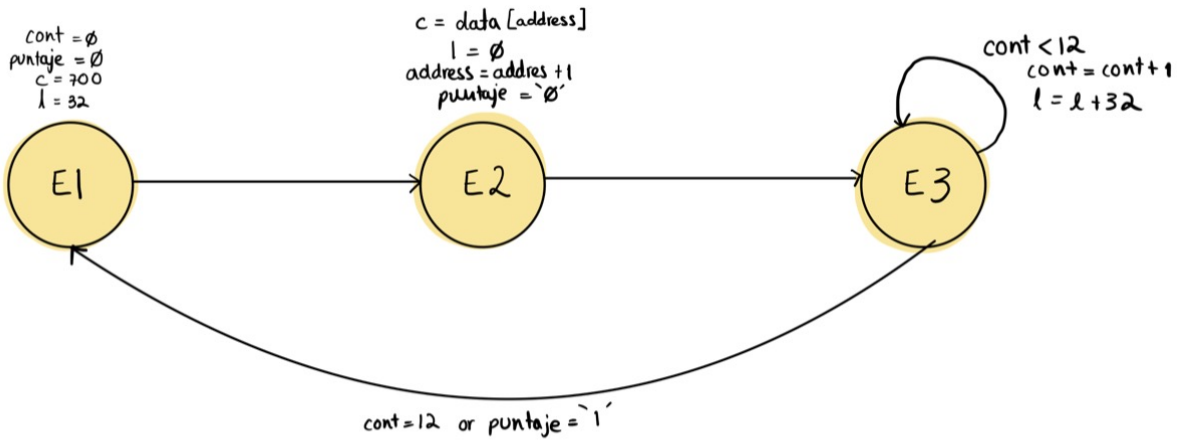

Figura 4. Diseño de una máquina de estados

La máquina de estados comienza en el estado 1, donde *cont* hace referencia al contador para que el cuadro haga su transición de arriba hacia abajo, *puntaje* es la señal para determinar si el switch se accionó mientras el cuadro pasó por el rango delimitado, *c* y *l* son variables que corresponde a las posiciones iniciales donde inicia el cuadro para el caso del estado 1, aparece fuera del rango de visualización. Dentro del estado 2, mediante la implementación de una memoria rom que almacena una lista de valores random entre el rango de todas las columnas a partir de las cuales puede posicionarse un cuadro, en este caso la variable c toma como valor dicho dato almacenado, y la posición de *l* equivale a que el cuadro inicie al principio de la pantalla, *address* corresponde a la señal para seleccionar el valor almacenado en la memoria rom. Cuando pasa al estado 3, existen varias condiciones para determinar si permanece dentro del estado 3, esto ocurre si el contador es menor a 12, el contador lo aumenta al igual que mueva la posición del cuadro visualizando una transición. Pero en caso contrario en el que el contador sea igual a 12 o el puntaje sea una señal de 1, en la máquina de estados regresa al estado 1.

RTL del prototipo

Utilizando la herramienta RTL viewer de Quartus, se generó el diagrama en el que podemos visualizar todos los componentes necesarios para realizar el funcionamiento del juego.

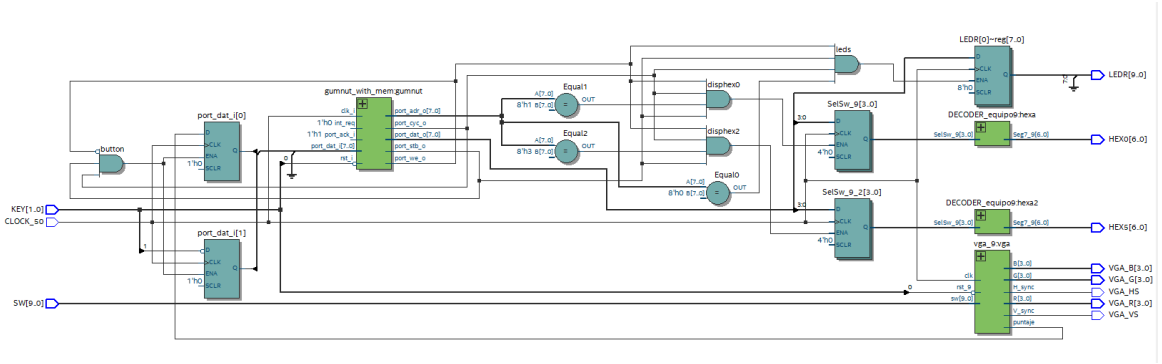

Figura 5. RTL Viewer del diseño del videojuego

En la figura se puede visualizar el procesador gumnut realizado, los decoders implementados para utilizar los displays de 6 segmentos, el funcionamiento del juego que se encuentra dentro del componente vga, así como otros componentes necesarios que en conjunto realizan la ejecución y visualización del videojuego realizado.

Explicación de partes fundamentales del código de VHDL

Máquina de estados

De la máquina de estados diseñada previamente en la sección de *diagrama de máquina de estados*, se desarrolló el código en VHDL, que muestra las condiciones para pasar de un estado a otro, además de las condiciones establecidas previamente para determinar si permanece en el estado 3 o si pasa del estado 3 al estado 1.

```
--fsm
PROCESS( clk_fsm, rst_9 )
   BEGIN
      IF( rst_9 = '1' )THEN
         pr_state_9 <= state_1_9;
      ELSIF rising_edge( clk_fsm ) THEN
         pr_state_9 <= nx_state_9;
   END IF;
END PROCESS;

-- State transitions / conditions
PROCESS( pr_state_9 )
   BEGIN
      CASE pr_state_9 IS
         --Estado 1
         when state_1_9 =>
            nx_state_9 <= state_2_9;

         --Estado 2
         WHEN state_2_9 =>
            nx_state_9 <= state_3_9;

         --Estado 3
         WHEN state_3_9 =>
            if cont = 12 or puntaje = '0' then
               nx_state_9 <= state_1_9;
            elsif cont < 12 then
               nx_state_9 <= state_3_9;
            end if;

      END CASE;
END PROCESS;
```

Figura 6. Código de máquina de estados (condiciones)

Para el caso de la sección de las acciones que se ejecutan, se muestran a continuación el cambio de las señales que hay durante el funcionamiento de la máquina de estados, tomando como base el diseño de la máquina de estados que se desarrolló previamente. Aquí se también se puede visualizar el funcionamiento de los interruptores en el que a partir de su localización en cierto rango y si es el caso en el que el switch está activado y los demás switches están en apagado, manda una señal de '0' al puntaje, esto para indicar que si se activó la señal ya que la señal de puntaje tiene un funcionamiento de una señal negada. Esto se repite para los demás casos if, de los demás switches.

```
IF rising_edge( clk_fsm ) THEN
   CASE pr_state_9 IS
      --Estado 1
      WHEN state_1_9 =>
         cont <= 0;
         puntaje <= '1';
         c := 700;
         l := 32;

      --Estado 2
      WHEN state_2_9 =>
         c:= data_out; -- lista de valores
         l:= 0;
         address <= address + 1;
         puntaje <= '1';


      --Estado 3
      WHEN state_3_9 =>
         if cont < 12 then
            cont <= cont + 1;
            l := l + 32;
         else
            cont <= cont;
         end if;

         if ( l > 316 and l < 415) then
            if  (sw(9) = '1' and sw(8) = '0' and sw(7) = '0' and sw(6) = '0' and sw(5) = '0' and sw(4) = '0' a

                  and sw(3) = '0' and sw(2) = '0' and sw(1) = '0' and sw(0) = '0' and c = 0 )then

               puntaje <= '0';
```

Figura 7. Código de máquina de estados (acciones)

Diseño de los gráficos

Para el diseño de los gráficos y poder representar la cuadrícula de la parte inferior se realizaron una serie de condiciones para determinar la posición entre líneas y columnas y dependiendo del encendido de los interruptores se determina el color del cuadro, ya sea en cian, verde o rojo. Si se encendia el interruptor el cuadro se pintaba de color cian, el el caso de acertar en la nota, se pinta de color verde y por otro lado si se falla en la nota se pinta de color rojo.

```
IF dena = '1' THEN
      if (line_count > 415 and line_count < 480 and colum_count > 0 and colum_count < 64) then

         if (sw(9) = '1' and l < 316 ) then
            R <= ( OTHERS => '0' );
            G <= ( OTHERS => '1' );
            B <= ( OTHERS => '1' );

         elsif (sw(9) = '1' and l > 316 and l < 415 and c = 0 and and1(9) = '1' ) then
            R <= ( OTHERS => '0' );
            G <= ( OTHERS => '1' );
            B <= ( OTHERS => '0' );

         elsif ((sw(9) = '0' and l > 316 and l < 415 and c = 0) or (sw(9) = '1' and and1(9) = '0' and c = 0) ) then
            R <= ( OTHERS => '1' );
            G <= ( OTHERS => '0' );
            B <= ( OTHERS => '0' );

         else
            R <= ( OTHERS => '1' );
            G <= ( OTHERS => '1' );
            B <= ( OTHERS => '1' );
         end if;
```

Figura 8. Gráficos de cuadrícula inferior del juego

Para el caso de los cuadros que van cayendo de arriba hacia abajo, se utiliza la siguiente sección de código que se muestra a continuación a partir de las condiciones para dibujar un cuadro como se hizo anteriormente, y a partir del funcionamiento de la máquina de estados va moviendo el cuadrito, y partir de condiciones de la posición de c que es recibida a través de la memoria rom que se diseñó para generar una lista de valores aleatorios, determina el color bajo el cual se pintaran los cuadrados que van cayendo.

```
elsif (line_count > l and line_count < (l + 64) and (colum_count > c) and (colum_count < (c + 64))) then
   if c = 0 then
      --verde
      R <= ( OTHERS => '0' );
      G <= ( OTHERS => '1' );
      B <= ( OTHERS => '0' );

   elsif c = 64 then
      --rojo
      R <= ( OTHERS => '1' );
      G <= ( OTHERS => '0' );
      B <= ( OTHERS => '0' );

   elsif c = 128 then
      --amarillo
      R <= ( OTHERS => '1' );
      G <= ( OTHERS => '1' );
      B <= ( OTHERS => '0' );

   elsif c = 192 then
      --azul
      R <= ( OTHERS => '0' );
      G <= ( OTHERS => '0' );
      B <= ( OTHERS => '1' );
```

Figura 9. Gráficos de los cuadros que caen de arriba hacia abajo

Diseño de una memoria ROM

Se realizó el diseño de una memoria ROM, como se explica anteriormente se generan cuadros que caen de cierta forma aleatoria, esto se hizo con ayuda de python en el que se genero una lista de valores que tuviera los valores deseados de las columnas y que estos se generarán de manera random. Estos se almacenan en la memoria ROM y se accede a ellos dentro de la máquina de estados por medio del valor de address.

```
-- rom memory
signal   reg_address :  integer  range 0 to 15;
      type     memory is array (0 to 99) of natural;

      constant myrom :  memory :=   (
         0 => 576,
         1 => 256,
         2 => 128,
         3 => 128,
         4 => 320,
         5 => 0,
         6 => 384,
         7 => 512,
         8 => 448,
         9 => 384,
         10 => 64,
         11 => 448,
         12 => 384,
```

```
------------------
  --rom
     process (clk_fsm)
     begin
        if (clk_fsm'event and clk_fsm='1') then
           reg_address <= address;
        end if;
     end process;

  --Get unregistered output
     data_out <= myrom(reg_address);


  ----------------------
```

Figura 10. Diseño de una ROM para almacenar valores

Diseño de una Gumnut

Para la implementación de un procesador Gumnut, se implementó el diseño de un contador que realice la función de mostrador de puntaje cada vez que el interruptor por el que está pasando el gráfico del recuadro la señal de puntaje mencionada en la máquina de estados es enviada al ensamblador y de ahí es recibida como una señal y de esta forma va sumando el valor del contador. Al igual que se integró una señal de reset y cuando esta señal esté en activo el contador regresa a cero. Y estos valores se envían al VHDL para ser mapeados y mostrados en los displays de 7 segmentos.

```
            text
            org 0x000 ; start here on reset
            jmp loop
; Data memory layout
            data
leds:       bss 1
disp7hex1:  bss 1
button:     bss 1
disp7hex2:  bss 1
cont1:      byte 9
cont2:      byte 15

; Main program
            text
            org 0x010


loop:       inp r1, button
            jsb delay
            and r0, r1, 2
            bnz reset
            and r0, r1, 1
            bnz loop
            ldm r2, cont1
            ldm r3, cont2
            out r3, disp7hex2
            out r2, disp7hex1
            add r2, r2, 1
            stm r2, cont1
            sub r0, r2, 10
            bnz loop
            jmp counter2


counter2:   stm r0, cont1
            ldm r3, cont2
            add r3, r3, 1
            stm r3, cont2
            jmp loop
```

```
reset:      stm r0, cont1
            stm r0, cont2
            add r2, r0, 0
            add r3, r0, 0
            out r2, disp7hex1
            out r3, disp7hex2
            jmp loop


delay:      add r7, r0, 0
again3:     add r6, r0, 0
again2:     add r5, r0, 0
again1:     add r5, r5, 1
            sub r0, r5, 0xFF
            bnz again1
            add r6, r6, 1
            sub r0, r6, 0xFF
            bnz again2
            add r7, r7, 1
            sub r0, r7, 0x09
            bnz again3
            ret
```

Figura 11. Diseño de un contador en ensamblador

```
vga: vga_9 port map (CLOCK_50, rst_i, VGA_HS , VGA_VS, BLANKn, SYNCn, VGA_R , VGA_G, VGA_B, puntaje, sw);

leds     :   PROCESS (clk_i)
               BEGIN
                  IF rising_edge(clk_i) THEN
                     IF port_adr_o = "00000000" and port_cyc_o = '1' and port_stb_o = '1' and port_we_o = '1' THEN
                        LEDR(7 DOWNTO 0) <= port_dat_o(7 DOWNTO 0);
                     END IF;
                  END IF;
            END PROCESS;

disphex0 :   PROCESS (clk_i)
               BEGIN
                  IF rising_edge(clk_i) THEN
                     IF port_adr_o = "00000001" and port_cyc_o = '1' and port_stb_o = '1' and port_we_o = '1' THEN
                        SelSw_9(3 DOWNTO 0) <= port_dat_o(3 DOWNTO 0);
                     END IF;
                  END IF;
            END PROCESS;

hexa    :   DECODER_equipo9 port map(SelSw_9(3 downto 0), HEX0(6 downto 0));

button :   PROCESS (clk_i)
               BEGIN
                  IF rising_edge(clk_i) THEN
                     IF port_cyc_o = '1' and port_stb_o = '1' and port_we_o = '0' THEN
                        port_dat_i(1 DOWNTO 0) <= (not KEY(1)) & puntaje;
                     END IF;
                  END IF;
            END PROCESS;


disphex2 :   PROCESS (clk_i)
               BEGIN
                  IF rising_edge(clk_i) THEN
                     IF port_adr_o = "00000011" and port_cyc_o = '1' and port_stb_o = '1' and port_we_o = '1' THEN
                        SelSw_9_2(3 DOWNTO 0) <= port_dat_o(3 DOWNTO 0);
                     END IF;
                  END IF;
            END PROCESS;

hexa2    :   DECODER_equipo9 port map(SelSw_9_2(3 downto 0), HEX5(6 downto 0));
```

Figura 11. Mapeo de las señales que son recibidas por el VHDL en el Gumnut

Conclusión y reflexiones individuales de tu aprendizaje

Karen

Considero que aprendí mucho con este reto, ya que al principio de la clase, cuando plantearon el reto, no tenía ninguna idea de cómo podríamos llegar a lograr algo como esto. Me gustaron mucho los temas y todos los conceptos que aprendimos, ya que antes no entendía cómo funcionaban internamente las computadoras, y esta clase me ayudó a darme cuenta la importancia y mi gusto por los sistemas digitales. Hablando del reto, me enorgullece saber que entiendo completamente el prototipo, y que todos participamos activamente durante su desarrollo, así como que solucionamos todos los problemas que nos enfrentamos.

Gustavo

En este nuevo bloque llegué emocionado pues cuando me contaron de que se trataría el reto quería ver todo lo que podía llegar a hacer con una tarjeta y un monitor. Claramente esto no iba a ser una tarea fácil pero aun así la emoción no se me apagaba. Todos los temas y las clases que tuvimos fueron de gran ayuda y utilidad pues no solo retomé algunos conceptos que en algún momento llegué a ver en internet, sino también pude ver y aprender nuevas cosas que me ayudaran en el reto. El apoyarme de mis amigos y compañeros del bloque fue de gran utilidad pues mientras elaboramos el reto surgieron dudas y problemas que una cabeza no sería capaz de resolver tan rápido. Gracias a esta clase pude ver realmente el uso de los sistemas digitales y cómo implementarlos en mi vida, no solo para trabajos o para la vida laboral sino también para crear nuevos proyectos por mi cuenta como algún pasatiempo. Creo que tanto la clase y el reto me brindaron problemas complejos pero solucionables y me sirvieron para comprender totalmente cómo desarrollar un prototipo funcional en equipo en el cual cada uno aportó su grano de arena.

Hortencia

El desarrollo del videojuego en la tarjeta DE10-Lite de Intel realmente nos permitió integrar muchos de los temas vistos durante la unidad de formación. El resultado final al integrar todo fue el desarrollo de un prototipo funcional que a través de pequeños laboratorios que hicimos durante clase pudimos integrar y que dieron paso al desarrollo de un videojuego. Sin duda a lo largo del desarrollo del reto nos encontramos con diversas dificultades, sin embargo con la investigación que realizamos y al visualizar las múltiples aplicaciones que pudiera tener los diferentes conceptos podemos solucionarlos y como resultado a ello el aprendizaje obtenido

fue enorme de cada una de las situaciones que se nos presentaron. De manera particular me gusto mucho realizar el desarrollo del reto ya que aunque se trató de un proceso elaborado y de trabajo continuo a lo largo de todas estas 5 semanas pudimos comprobar que la programación en VHDL tiene grandes aplicaciones y que los pequeños conceptos nos llevan a desarrollar grandes proyectos como fue el videojuego que desarrollamos.

Referencias

VIU. (22 de noviembre del 2021). *¿Cuál fue el primer videojuego?* [Página Web]. Recuperado de: https://www.universidadviu.com/es/actualidad/nuestros-expertos/cual-fue-el-primer-videojuego