

RETO

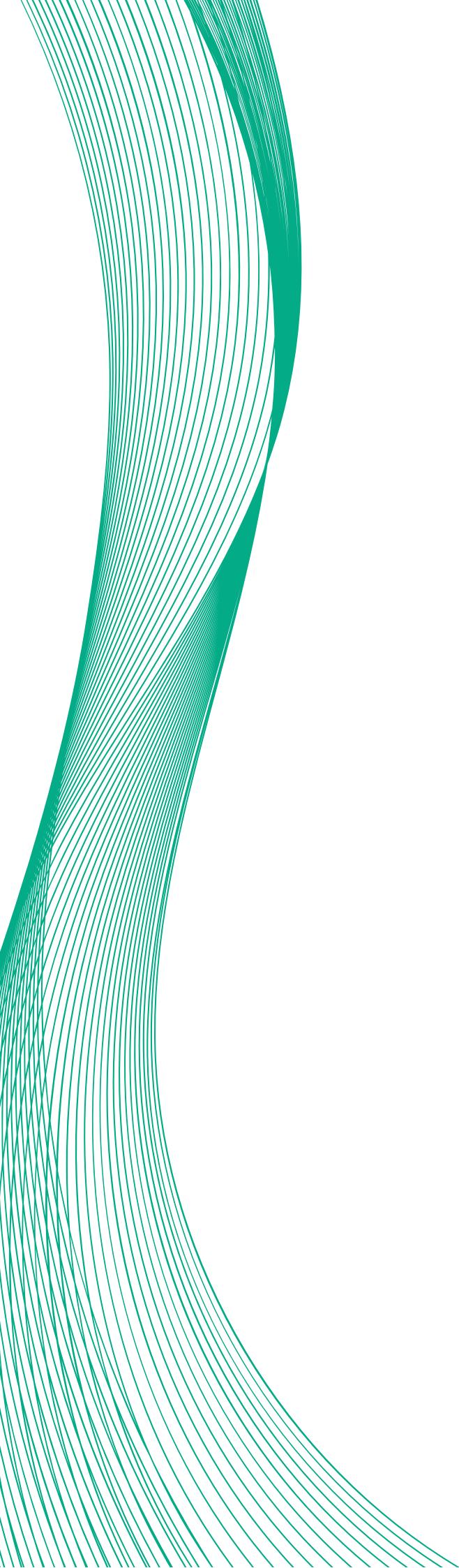
---

# VIDEOJUEGO: DINO

Marzo 16, 2023

Autores: Melisa Saucedo Sánchez A01748077  
Leonardo Madrid Morales A01747964

Diseño con lógica programable  
Grupo 101  
Prof.: Dr. Andrés David García García, Dr.  
Francisco Javier Ortiz Cerecedo



## ÍNDICE

---

**01**

Descripción del proyecto

**02**

Justificación del proyecto

**03**

Explicación de la Top Level Entity

**04**

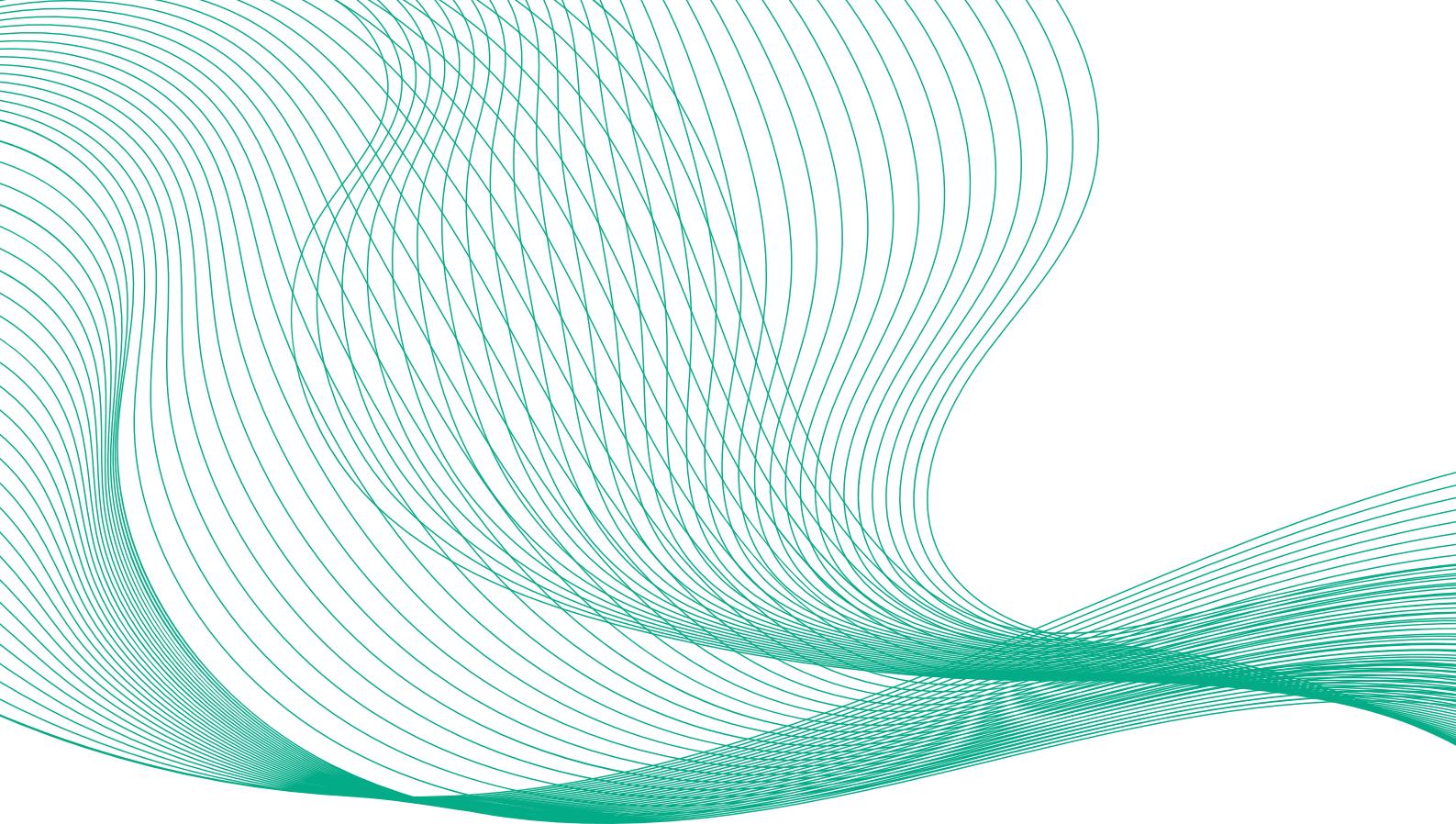
Explicación de los componentes empleados

**12**

Resultados

**13**

Conclusiones individuales



## DESCRIPCIÓN DEL PROYECTO

---

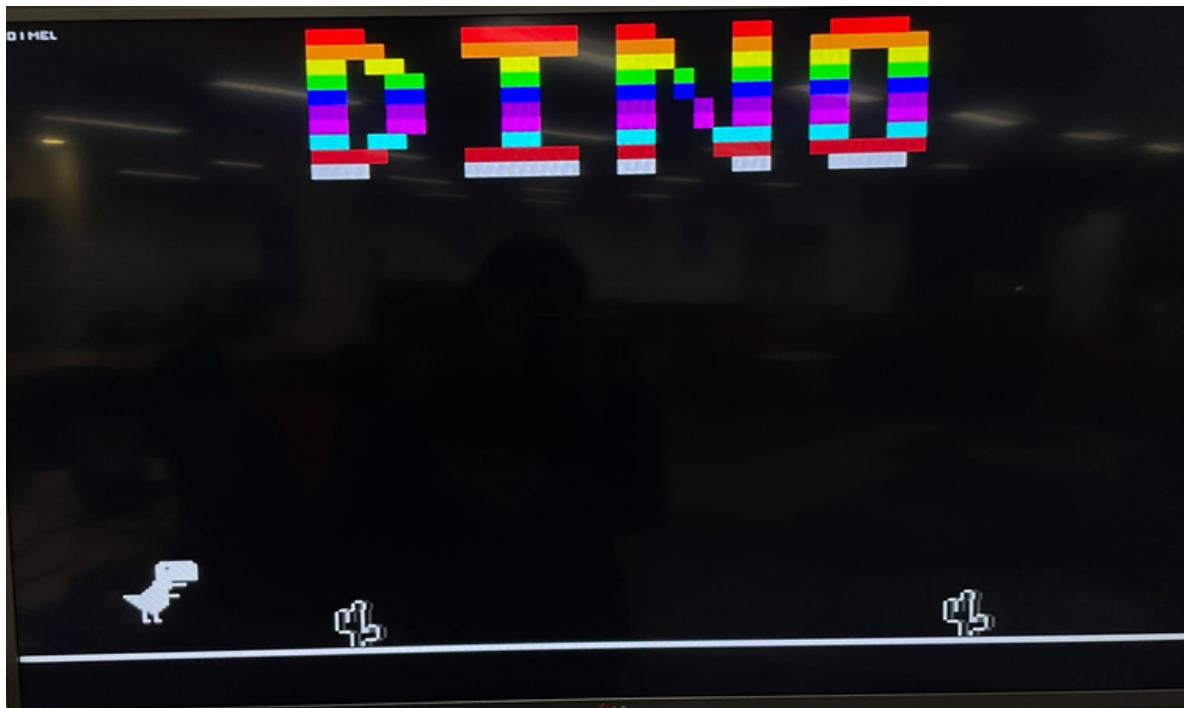
Nuestro proyecto consistió en recrear el videojuego de "Dino Run", el famoso minijuego que aparece en el buscador Google Chrome cuando uno se queda sin conexión a internet. Esto por medio de la comunicación entre un FPGA (Field Programmable Gate Array) y un monitor con interfase de VGA (Video Graphics Adapter). Para establecer la comunicación se codificó en el lenguaje descriptivo VHDL. A muy grandes rasgos, con el uso de contadores y máquinas de estado hizo que pudiéramos definir el funcionamiento y diseño visual de nuestro videojuego Dino.



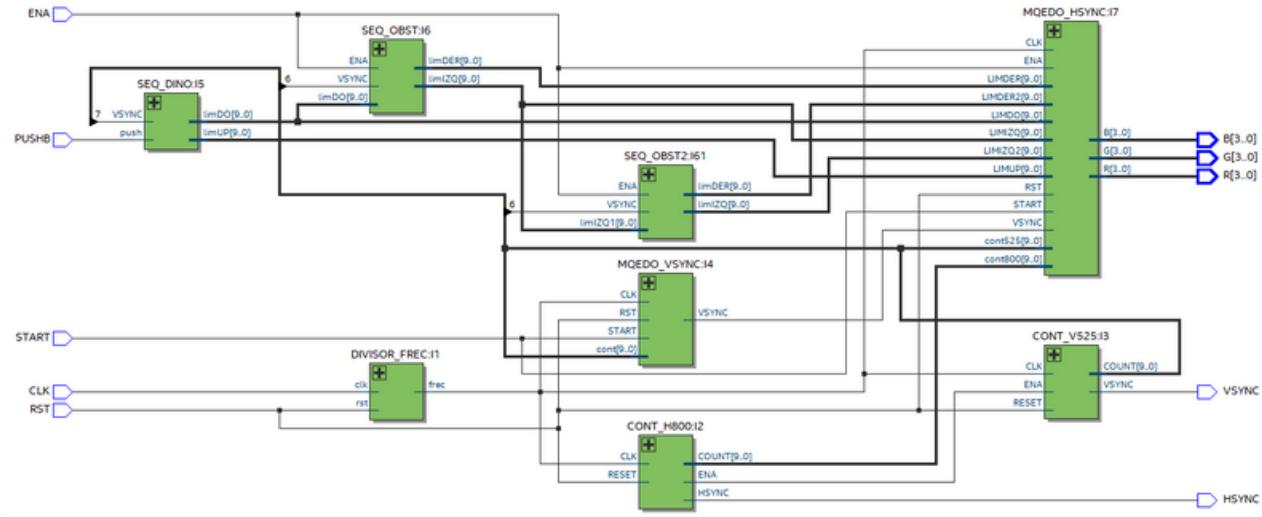
## JUSTIFICACIÓN DEL PROYECTO

---

Decidimos que nuestro proyecto fuera de esa manera porque por la parte creativa queríamos explorar un poco más de lo aprendido en clase. La propuesta de nuestro videojuego apuesta por los detalles en el diseño visual. En cuanto a la parte funcional determinamos que queríamos recrear el minijuego de Chrome, por varias razones. Probar hacer secuencia de contadores, con determinados rangos y posiciones. Probar diferentes escenarios a la hora de la interacción entre objetos, cuando salta, cuando lo toca, cuando aparecen más objetos, etc. Porque se nos hace un juego entretenido, sin necesidad de tener otro jugador, y sin hacerse tedioso con el tiempo. Haciendo del videojuego Dino una experiencia satisfactoria tanto en su realización como a la hora de jugarlo.



# EXPLICACIÓN DE LA ARQUITECTURA (TOP LEVEL ENTITY)



En nuestro top level entity entran el clock de 50 MHz, un reset, start, enable y un push button. Lo que sale son las señales sincrónicas vertical y horizontal, y los bits del RGB (Red-Green-Blue). Para que se puedan visualizar los cuadros por segundo, primero el clock pasa por el divisor de frecuencia, su salida se va a todas las entradas de clock de los demás componentes. Con la señal del reset, todos los componentes empiezan a trabajar; cuando el contador 800 termina de contar, le pasa una señal enable para que empiece a contar el contador 525. Cada contador saca las señales de sincronía. Mientras los contadores cuentan, la máquinas de estado reciben los números para indicar en qué parte del VGA se encuentran. La máquina vertical envía la señal de sincronía a la máquina de estado horizontal y si está activa, esta ejecuta las instrucciones dentro de la horizontal.

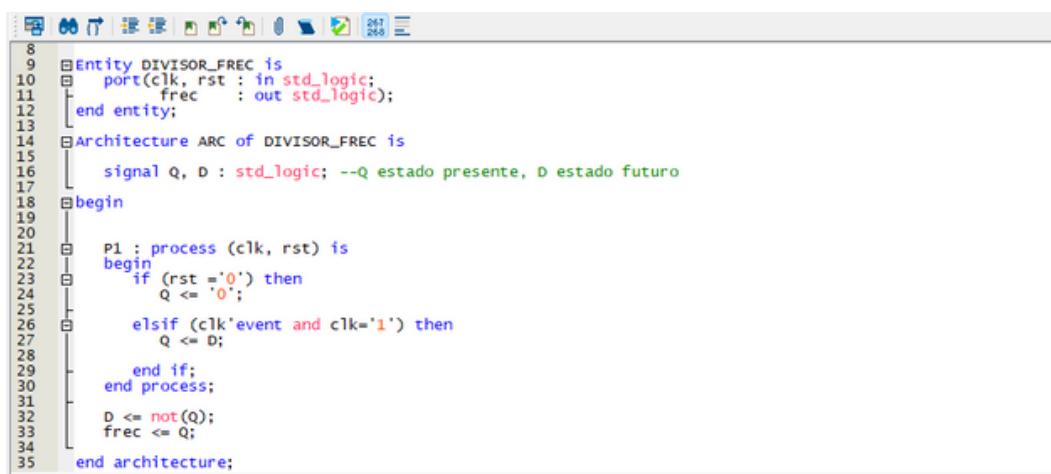
En la horizontal se reciben varios datos de ciertos componentes. El componente secuencia dinosaurio al recibir la señal del push button, realiza el salto y envía los límites inferior y superior a la máquina horizontal. La secuencia obstáculo al recibir la señal enable principal de la top level entity, realiza el movimiento horizontal de derecha a izquierda y manda los límites izquierda y derecha a la máquina horizontal. Lo mismo con el componente secuencia obstáculo 2. La máquina horizontal en el estado de zona visible al tener estos datos los compara con if's y dependiendo el resultado, pinta los bits R-G-B en los pixeles marcados.

# EXPLICACIÓN DE LOS COMPONENTES EMPLEADOS

---

## 01 DIVISOR DE FRECUENCIA

Un videojuego tiene varios cuadros de imágenes por segundo, si no son los suficientes por cada segundo, no se produce la sensación de movimiento. Se vuelve necesario tener una rápida frecuencia de tiempo para generar los varios cuadros por segundo y atienda a la persistencia del ojo humano, fenómeno del ojo que hace que podamos percibir el "movimiento". Haciendo cálculos y revisando el data sheet de la tarjeta DE10-Lite, se necesitan 25MHz de muestreo para que haya un cuadro visible de 640 x 480 pixeles a 60 cuadros/segundo. En el divisor de frecuencia, dividimos el clock de 50MHz a los 25Mhz.



```
8
9  Entity DIVISOR_FREC is
10  port(clk, rst : in std_logic;
11      freq : out std_logic);
12  end entity;
13
14  Architecture ARC of DIVISOR_FREC is
15
16    signal Q, D : std_logic; --Q estado presente, D estado futuro
17
18  begin
19
20    P1 : process (clk, rst) is
21    begin
22      if (rst = '0') then
23        Q <= '0';
24
25      elsif (clk'event and clk='1') then
26        Q <= D;
27
28      end if;
29    end process;
30
31    D <= not(Q);
32    freq <= Q;
33
34  end architecture;
```

# EXPLICACIÓN DE LOS COMPONENTES EMPLEADOS

---

## 02 CONTADOR QUE ENUMERA HASTA 800

Este componente sirve para establecer los 800 pixeles horizontales del VGA. Los cuales 640 pixeles son los que se muestran en la zona visible. Se conforma por las entradas del clock y reset, y por las salidas de un enable, la señal sincrónica horizontal y la cuenta de 10 bits. En particular, el clock recibe la frecuencia de los 25MHz y el enable sale cada vez que el contador llega hasta el 799 para que vaya al contador vertical.

```
11  ENTITY CONT_H800 IS
12    PORT(CLK,RESET : IN STD_LOGIC;
13      ENA : OUT STD_LOGIC;
14      HSYNC : OUT STD_LOGIC;
15      COUNT : OUT STD_LOGIC_VECTOR(9 DOWNTO 0));
16  END ENTITY;
17
18  ARCHITECTURE ARC_OF_CONT_H800 IS
19
20    SIGNAL C1 : STD_LOGIC_VECTOR(9 DOWNTO 0);
21    SIGNAL E, HS : STD_LOGIC;
22
23  BEGIN
24
25    PR1 : PROCESS(CLK,RESET)
26    BEGIN
27
28      IF RESET = '0' THEN
29        C1 <= (OTHERS => '0');
30      ELSIF (CLK'EVENT and CLK = '1') THEN
31        IF C1 = "1100011111" THEN --799
32          C1 <= (OTHERS => '0');
33          E <= '1';
34        ELSE
35          C1 <= C1 + 1;
36          E <= '0';
37        END IF;
38      END IF;
39    END PROCESS;
40
41    COUNT <= C1;
42    ENA <= E;
43
44    PR2: PROCESS (C1)
45    BEGIN
46      IF (C1 > "0001011111") THEN --95
47        HS <= '1';
48      ELSE
49        HS <= '0';
50      END IF;
51    END PROCESS;
52
53    HSYNC <= HS;
54
55  
```

# EXPLICACIÓN DE LOS COMPONENTES EMPLEADOS

## 03 CONTADOR QUE ENUMERA HASTA 525

Este componente es básicamente el mismo que el anterior, pero este sirve para establecer los 525 pixeles verticales. Aquí habrá 480 pixeles que estarán en la zona visible. El enable del contador 800 entra en este para que sea una señal de que empiece a contar. Por último, en lo particular, también manda su señal sincrónica.

```
10 ENTITY CONT_V525 IS
11  PORT(CLK, RESET : IN STD_LOGIC;
12    ENA           : IN STD_LOGIC;
13    VSYNC         : OUT STD_LOGIC;
14    COUNT         : OUT STD_LOGIC_VECTOR(9 DOWNTO 0));
15 END ENTITY;
16
17 ARCHITECTURE ARC OF CONT_V525 IS
18
19  SIGNAL C1 : STD_LOGIC_VECTOR(9 DOWNTO 0);
20  SIGNAL E, VS : STD_LOGIC;
21
22 BEGIN
23
24  PR1 : PROCESS(CLK,RESET)
25  BEGIN
26
27    IF(RESET = '0') THEN
28      C1 <= (OTHERS => '0');
29    ELSIF (CLK'EVENT and CLK = '1' and ENA = '1') THEN
30      IF C1 = "1000001100" THEN
31        C1 <= (OTHERS => '0');
32      ELSE
33        C1 <= C1 + 1;
34      END IF;
35    END IF;
36  END PROCESS;
37
38  COUNT <= C1;
39
40  PR2: PROCESS (C1)
41  BEGIN
42    IF (C1 > "0000000001") THEN
43      VS <= '1';
44    ELSE
45      VS <= '0';
46    END IF;
47  END PROCESS;
48
49  VSYNC <= VS;
50
51
52
53
54
55 END ARCHITECTURE;
```

## EXPLICACIÓN DE LOS COMPONENTES EMPLEADOS

### 04 MÁQUINA DE ESTADOS DE LA SINCRONÍA VERTICAL

Su función es que pueda indicar en que parte la señal del VGA vertical está. Pues primero entra en el pulso de sincronía, después al back porch, la zona visible y el front porch. Cada uno estos estados cuenta con ciertos pixeles, entonces con el contador 525, cada vez que llegue al número determinado de pixeles, se le indica en que estado está. Las salidas son el estado lógico de la señal sincrónica vertical.

```
9  Entity MQEDO_VSYNC is
10   port(CLK, RST, START : in std_logic;
11      cont            : in std_logic_vector(9 downto 0);
12      VSYNC           : out std_logic;
13      --VSYNC_TOP      : out std_logic);
14  end entity;
15
16  Architecture ARC of MQEDO_VSYNC is
17
18    type ESTADOS is (IDLE, E1, E2, E3, E4); --estados
19    signal EDO, EDO_F : ESTADOS; --EDO (presente), EDO_F (futuro)
20
21  begin
22
23    --CONTROL DE TRANSICION DE FLIP-FLOPS (FFD)
24    PIFD : process (CLK, RST, START)
25    begin
26      if RST = '0' then --RST asincrono en bajo
27        EDO <= IDLE;
28      elsif (CLK'event and CLK = '1') then
29        if (START = '1') then
30          EDO <= EDO_F;
31        end if;
32      end if;
33    end process;
34
35    --PROCESO DE TRANSICIONES
36    P2TRANSC : process (EDO, cont)
37    begin
38      case EDO is
39        when idle => if START='0' then
40          EDO_F <= idle;
41        else
42          EDO_F <= E1;
43        end if;
44
45        when E1 => if (cont = "0000000001") then --2(need 2lines) /1(cont)
46          EDO_F <= E2;
47        else
48          EDO_F <= E1; --pulso
49        end if;
50
51        when E2 => if (cont = "0000100010") then --33/34
52          EDO_F <= E3;
53        else
54          EDO_F <= E2;
```

# EXPLICACIÓN DE LOS COMPONENTES EMPLEADOS

## 05 MÁQUINA DE ESTADOS DE LA SINCRONÍA HORIZONTAL

Esta máquina sirve para igual indicar en que parte la señal del VGA horizontal está, sin embargo, sus salidas son los colores RGB (Red-Green-Blue) de 4bits. Pues en la sincronía horizontal, en la zona visible es donde el VGA "pinta". Entonces además de indicar los estados, sirve para establecer los pixeles que queremos que muestre con los colores que hemos determinado. Para lograr aquello le deben de entrar los dos contadores, el vertical (525) y el horizontal (800), con el fin de que pinte en los dos ejes: el "x" y el "y".

```
9  Entity MQEDO_HSYNC is
10  port(CLK, RST, START : in std_logic;
11    VNA : in std_logic;
12    Cont525 : in std_logic_vector(9 downto 0);
13    Cont800 : in std_logic_vector(9 downto 0);
14    LIMP : in std_logic_vector(9 downto 0);
15    LMDO : in std_logic_vector(9 downto 0);
16    LIMZQ : in std_logic_vector(9 downto 0);
17    LIMDER : in std_logic_vector(9 downto 0);
18    LIMZQ2 : in STD_LOGIC_VECTOR(9 DOWNTO 0);
19    LIMDER2 : in STD_LOGIC_VECTOR(9 DOWNTO 0);
20    VSYNC : in std_logic;
21    R : out std_logic_vector(3 downto 0);
22    G : out std_logic_vector(3 downto 0);
23    B : out std_logic_vector(3 downto 0));
24    --HSYNC : out std_logic;
25 end entity;

84  --SALIDAS
85  P3SALIDAS : process (EDO)
86 begin
87   case EDO is
88    when IDLE => R <= "0000";
89    G <= "0000";
90    B <= "0000";
91    --HSYNC <= '0';
92
93    when E1 => R <= "0000"; --pulso
94    G <= "0000";
95    B <= "0000";
96    --HSYNC <= '0';
97
98    when E2 => R <= "0000"; --back porch
99    G <= "0000";
100   B <= "0000";
101   --HSYNC <= '1';
102
103  when E3 => if (VSYNC="1") then --zona visible
104    R <= "0000";
105    G <= "0000";
106    B <= "0000";
107
108    IF (cont525 = 40) then
109      IF (cont800 = 150 OR cont800 >= 154 AND cont800 <= 157) OR (cont800 >= 159 AND cont800 <= 162) OR cont800 = 166 OR cont800
110      R <= "1111";
111      G <= "1111";
112      B <= "1111";
113    END IF;
114
115    IF (cont525 = 41) then
116      IF (cont800 = 150 OR cont800 = 154 OR cont800 = 159 OR cont800 = 162 OR cont800 = 166 OR cont800 = 170 OR cont800 = 171 OR c
117      R <= "1111";
118      G <= "1111";
119      B <= "1111";
120    END IF;
121
122    IF (cont525 = 42) then
123      IF (cont800 = 150 OR (cont800>= 154 AND cont800<= 156) OR cont800 = 159 OR cont800 = 162 OR cont800 = 166 OR cont800 = 170 O
124      R <= "1111";
125      G <= "1111";
126      B <= "1111";
127    END IF;
128  end if;
```

# EXPLICACIÓN DE LOS COMPONENTES EMPLEADOS

## 06 SECUENCIA DINOSAURIO

La secuencia dinosaurio es el componente que define el salto del dinosaurio. Sus entradas son un push y la señal vsync, y sus salidas son los límites superior e inferior en el eje y. La vsync nos determina la velocidad del salto. Por cada vez que recibe la señal del push button, resta los límites superiores o inferiores, es decir los sube en la pantalla y completa el salto, si no recibe la señal del push, se queda en su posición inicial.

```
11  port(push : in std_logic;
12    VSYNC : in std_logic;
13    limUP : out std_logic_vector(9 downto 0); --posicion en y
14    limDO : out std_logic_vector(9 downto 0)); --posicion en y
15 end entity;
16
17
18 architecture ARC of SEQ_DINO is
19
20   signal posUP : std_logic_vector(9 downto 0);
21   signal posDO : std_logic_vector(9 downto 0);
22   signal flag : std_logic;
23   signal cont : std_logic_vector(6 downto 0);
24
25 begin
26
27   P1 : process(push)
28   begin
29
30     if(VSYNC'event and VSYNC = '1') then
31
32       if(push='0' and cont < 1) then
33         flag <= '1';
34         posUP <= posUP - 1; --restar es subir
35         posDO <= posDO - 1;
36         cont <= cont + 1;
37
38       elsif(flag = '1' and cont < 80) then --SUBE 80 PXLS, saltode36
39         posUP <= posUP - 1;
40         posDO <= posDO - 1;
41         cont <= cont + 1;
42
43       elsif(flag = '1' and cont > 79) then --si ya esta mayor que 80, envia la SEÑAL DE BAJAR
44         flag <= '0';
45
46       elsif(flag = '0' and cont > 0) then --señal bajar, como esta en 80, empieza a RESTAR CONT
47         posUP <= posUP + 1;
48         posDO <= posDO + 1;
49         cont <= cont - 1;
50
51       elsif(flag='0' and cont< 1) then
52         posUP <= "010101110"; --430
53         posDO <= posUP + 44;
54         cont <= "0000000";
55
56     end if;
57
58   end process P1;
59
60
61 end architecture ARC;
```

# EXPLICACIÓN DE LOS COMPONENTES EMPLEADOS

## 07 SECUENCIA OBSTÁCULO

Este componente define el movimiento del primer obstáculo, un nopal. El movimiento que definimos es que cruce toda la pantalla horizontalmente de derecha a izquierda y lo repita hasta que reciba la señal de "game over" en la máquina de estados, un estado lógico que significa que ya colisionó con el dinosaurio. Las salidas que tiene este componente son los límites en la posición x de izquierda y derecha para mandarlos a la máquina de estados a fin que los pueda pintar.

Cuenta con un "enable" para que cuando se active el VGA no se mueva automáticamente, si no hasta que el enable este activo.

```
11  port( ENA      : in std_logic;
12    VSYNC    : in std_logic;
13    --limUP   : in std_logic_vector(9 downto 0);
14    limDO   : in std_logic_vector(9 downto 0);
15    limZZQ  : out std_logic_vector(9 downto 0); --posición en x
16    limDER  : out std_logic_vector(9 downto 0)); --posición en x
17  end entity;
18
19
20  Architecture ARC of SEQ_OBST is
21
22    signal posIZQ : std_logic_vector(9 downto 0);
23    signal posDER : std_logic_vector(9 downto 0);
24    signal gameov : std_logic;
25
26
27  begin
28
29    P1 : process(ENA)
30    begin
31
32      if(VSYNC'event and VSYNC = '1') then
33
34        if (ENA='1' and posIZQ > 144) then -- "inicializa" / cont=0
35          posIZQ <= posIZQ - 1; --restar es izq
36          posDER <= posDER - 1;
37
38        elsif(ENA='1' and posIZQ < 145) then --POSC A IZQ 657 PXLS
39          posIZQ <= "1011110111"; --759
40          posDER <= "1100001111";
41
42
43        ELSIF(ENA='0') THEN
44          posIZQ <= "1011110111"; --759
45          posDER <= "1100001111";
46          gameov <= '0';
47
48
49      end if;
50    end if;
51
52  end process;
53
54  limIZQ <= posIZQ;
55  limDER <= posDER;
56 end architecture;
```

# EXPLICACIÓN DE LOS COMPONENTES EMPLEADOS

## 08 SECUENCIA OBSTÁCULO 2

La secuencia del obstáculo 2 es básicamente la misma que la anterior, solo que a la hora de establecer su movimiento se le indica la separación que habrá entre el otro obstáculo y este.

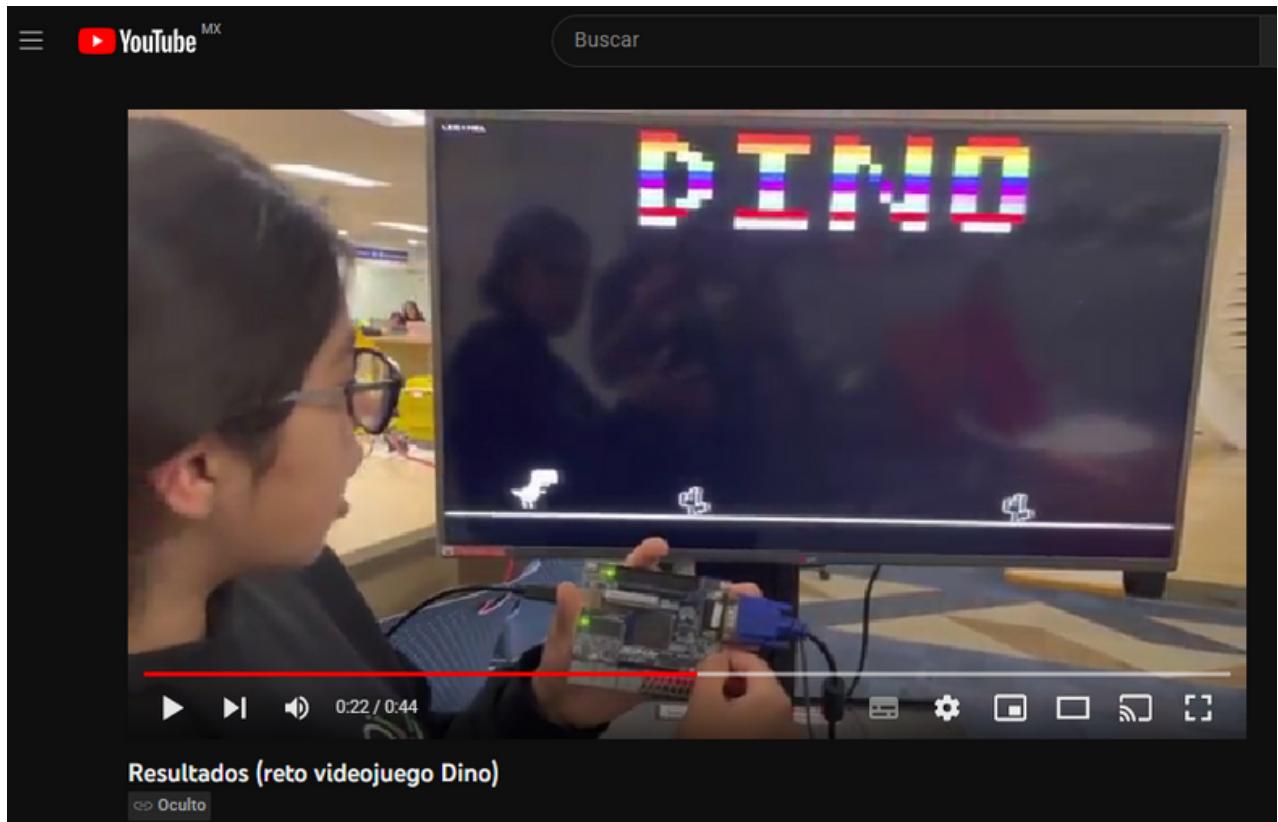
```
15      TimIZQ : out std_logic_vector(9 downto 0); --posicion en x
16      TimDER : out std_logic_vector(9 downto 0)); --posicion en x
17  end entity;
18
19  architecture ARC of SEQ_OBST2 is
20
21      signal posIZQ : std_logic_vector(9 downto 0);
22      signal posDER : std_logic_vector(9 downto 0);
23      signal CONT : std_logic_vector(8 DOWNTO 0);
24
25
26  begin
27
28      P1 : process(ENA)
29      begin
30
31          if(VSYNC'event and VSYNC = '1') then
32
33              IF (ENA='1' AND CONT < 330) THEN
34                  CONT <= CONT + 1;
35
36              ELSIF (ENA='1' AND CONT > 329 and posIZQ > 144) then -- "initializa" / cont=0
37                  posIZQ <= posIZQ - 1; --restar es izq
38                  posDER <= posDER - 1;
39
40              ELSIF(ENA='1' AND CONT > 329 and posIZQ < 145) then --POSC A IZQ 657 PXLS
41                  posIZQ <= "1011110111"; --759
42                  posDER <= "1100001111";
43
44
45              ELSIF(ENA='0') THEN
46                  posIZQ <= "1011110111"; --759
47                  posDER <= "1100001111";
48                  CONT <= "000000000";
49
50
51          end if;
52      end if;
53
54
55      end process;
56
57      limIZQ <= posIZQ;
58      limDER <= posDER;
59
60  end architecture;
```

## RESULTADOS

---

Enlace del video de youtube para el video demostrativo de los resultados:

<https://youtu.be/wo0NT3Kd68U>



# CONCLUSIONES

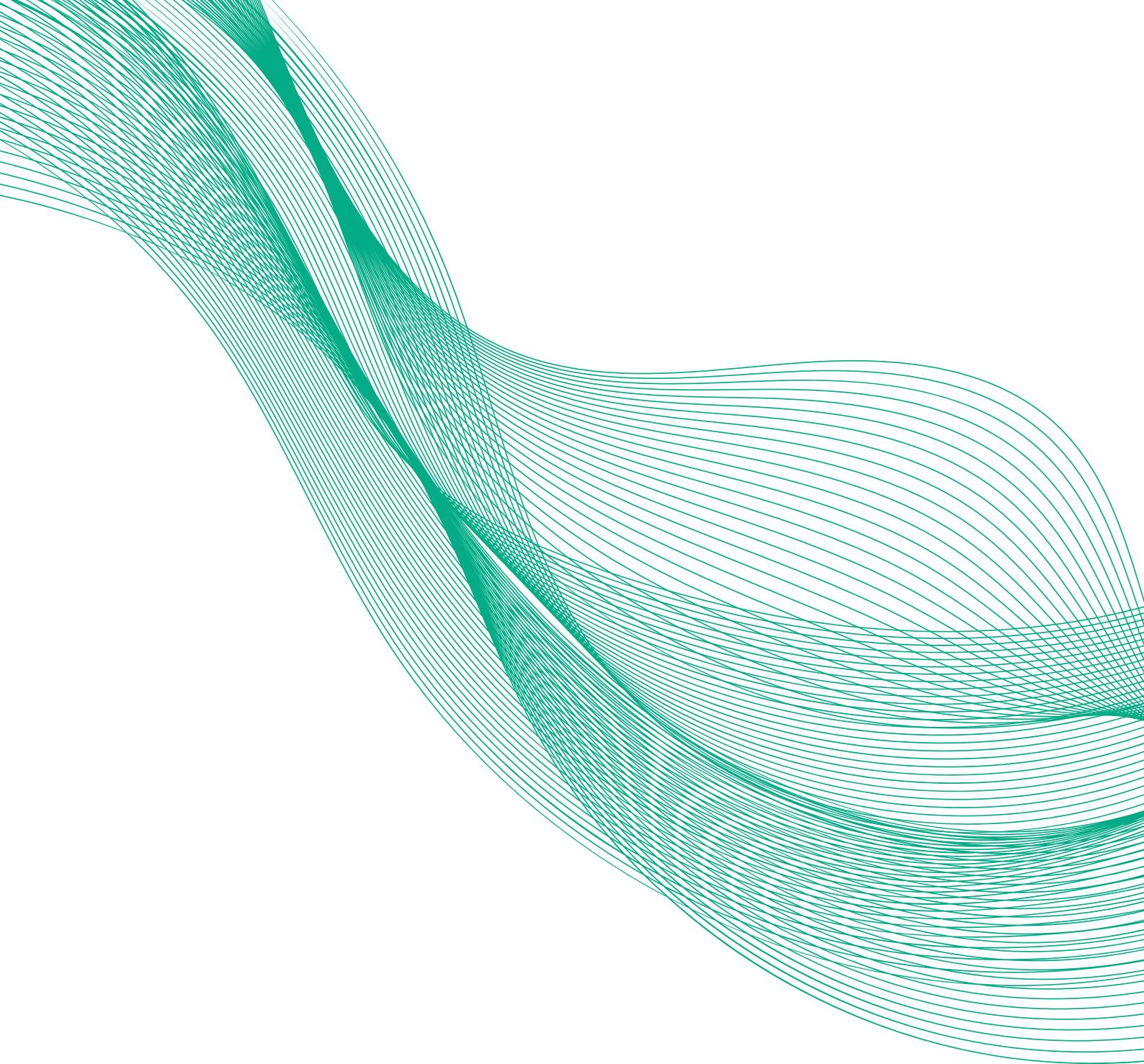
---

## LEONARDO MADRID

El objetivo fue el demostrar el dominio del conocimiento que aprendimos durante el transcurso del curso. Por lo cual se muestran nuestro entendimiento del mismo y las nuevas habilidades que destrollamos. Lo que más trabajo y tiempo me costó fue la parte del diseño del juego, por asignar los pixeles específicos. La parte del movimiento fue complicada de razonar al ser diferente y más compleja que la del pong pero una vez comprendida, se pudo implementar rápidamente. Estoy muy feliz y orgulloso con mi trabajo y el haber logrado el objetivo propuesto.

## MELISA SAUCEDO

En resumen, con el VGA pudimos implementar todo el aprendizaje obtenido desde la primera semana; definitivamente cada entrega iba aportando para este proyecto final. Este proyecto fue retador en cuanto a crear lo que diera nuestra imaginación a partir de las herramientas dadas, es decir, el ir más allá desafía la formación en clase. Quisimos balancear un juego entretenido para el reto con un diseño atractivo del mismo, pues es una parte importante de cualquier videojuego. Aunque pueda parecer lo contrario, hacer las secuencias de los objetos fue intrincado al principio, pues uno tiene que tomar en cuenta cada posible escenario dentro de estas secuencias. Pero al final, ha dado como producto un buen trabajo y un buen aprendizaje.



**Gracias**