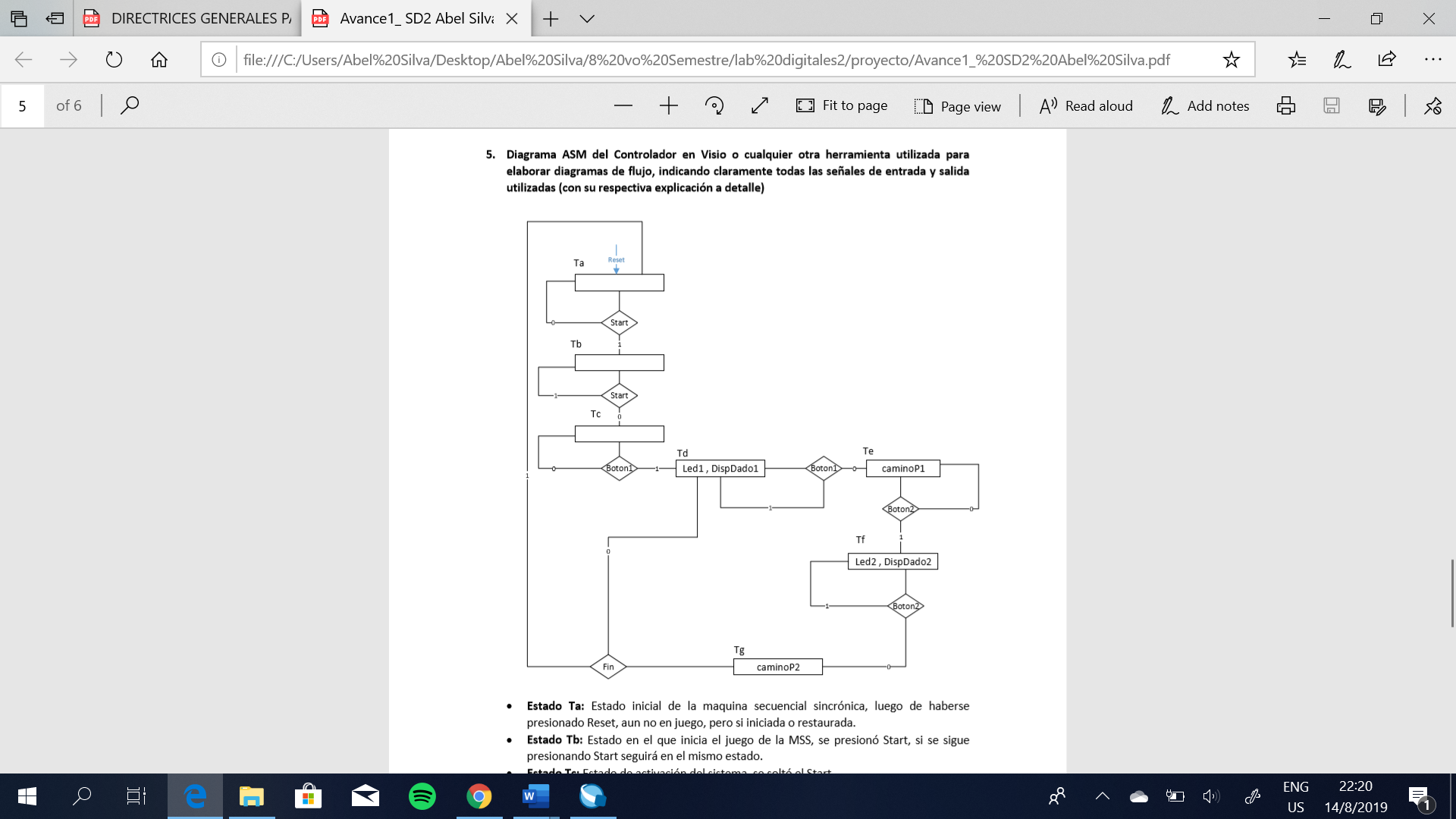
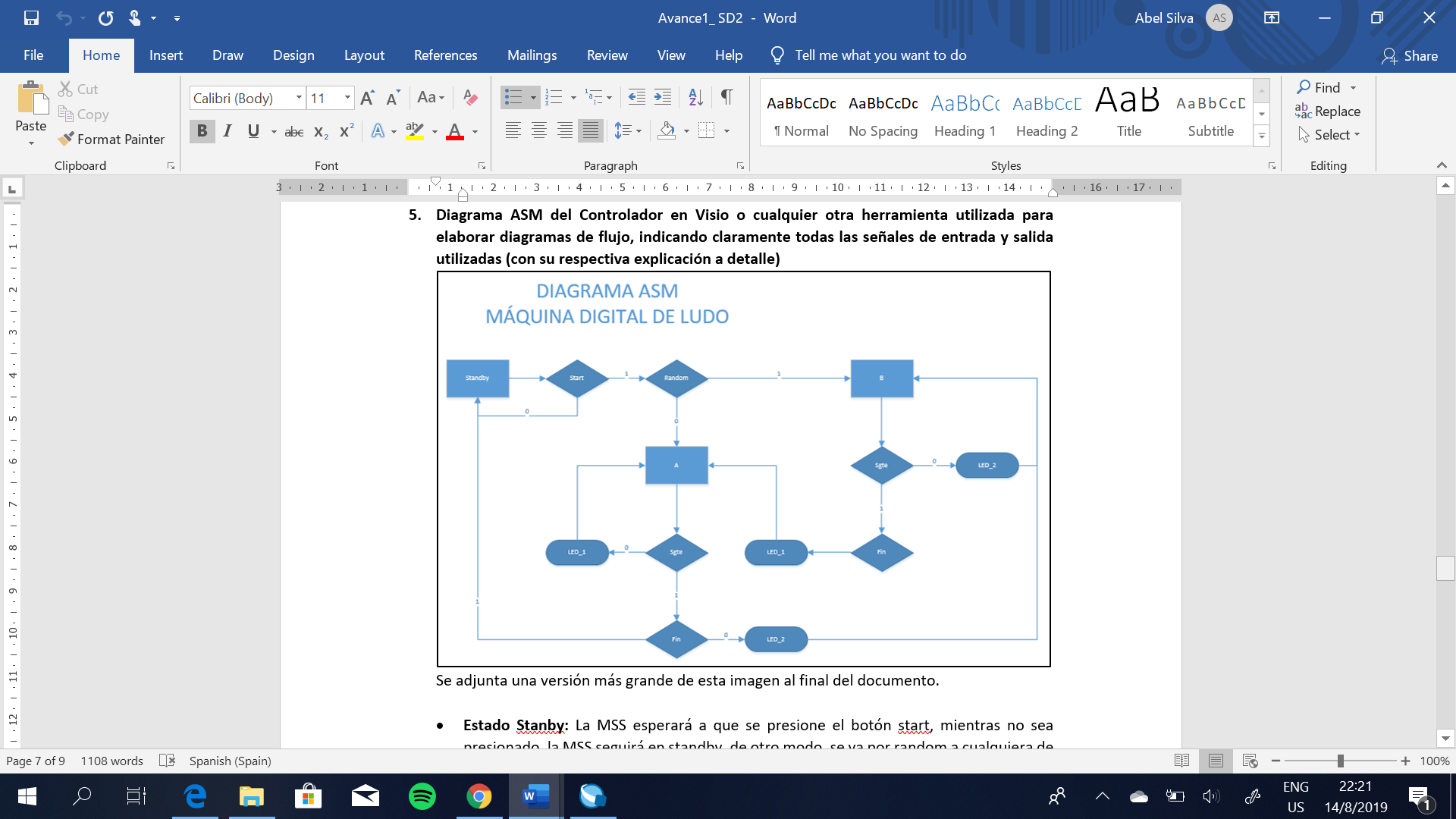
|  |  |
| --- | --- |
| LABORATORIO DE SISTEMAS DIGITALES II  Práctica de laboratorio | **Paralelo del laboratorio:**  Paralelos: 103 y 104  **Avance Final**  **Proyecto:**  **Máquina de Ludo**  **Fecha de entrega:**  15 de agosto de 2019  **Nombre del estudiante:**  Nelson Antonio Tubay Hidalgo  Abel Stuard Silva Platón  **Nombre del profesor de laboratorio:**  MSc. Sara Ríos  **Termino académico:**  2019–I |

1. **Justificación del Proyecto**

**Diagrama ASM 1**



***Imagen 1 – Diagrama ASM propuesta 1***

**Diagrama ASM 2**

***Imagen 2 – Diagrama ASM propuesta 2***

Para la realización del proyecto se tuvieron en cuenta muchos factores, la solución escogida fue la del diagrama ASM 2, ambas ideas para realizar el desarrollo del proyecto eran muy parecidas, presentabas discrepancias es etapas muy objetivas que se presentan a continuación:

* Por efectos didácticos la correcta interpretación de los turnos aleatorios generados, especificados en la partición funcional, donde se aclara mediante una señal [led jugador], el encendido de un led por jugador.
* La correcta interpretación, al momento de generar un numero aleatorio que, mediante una frecuencia muy alta, se genera la salida de un numero del 1 al 6 realizando de esta forma la simulación de un dado digital.
* La implementación correcta de las salidas del dado mediante el decodificador de 7 segmentos, etapa que en la propuesta numero 1, no era aclarada.
* El manejo adecuado de los números generados por el dado, es decir la suma y la acumulación de estos valores que serán actualizados en cada turno, hasta que el juego termine, que será al momento de que llega a 32 uno de los dos jugadores.

1. **Diagrama de Bloques del sistema digital**

Generación Turno Aleatorio

Dado

Encendido LEDs

Cambio de jugador

Guardado y Acumulación dado

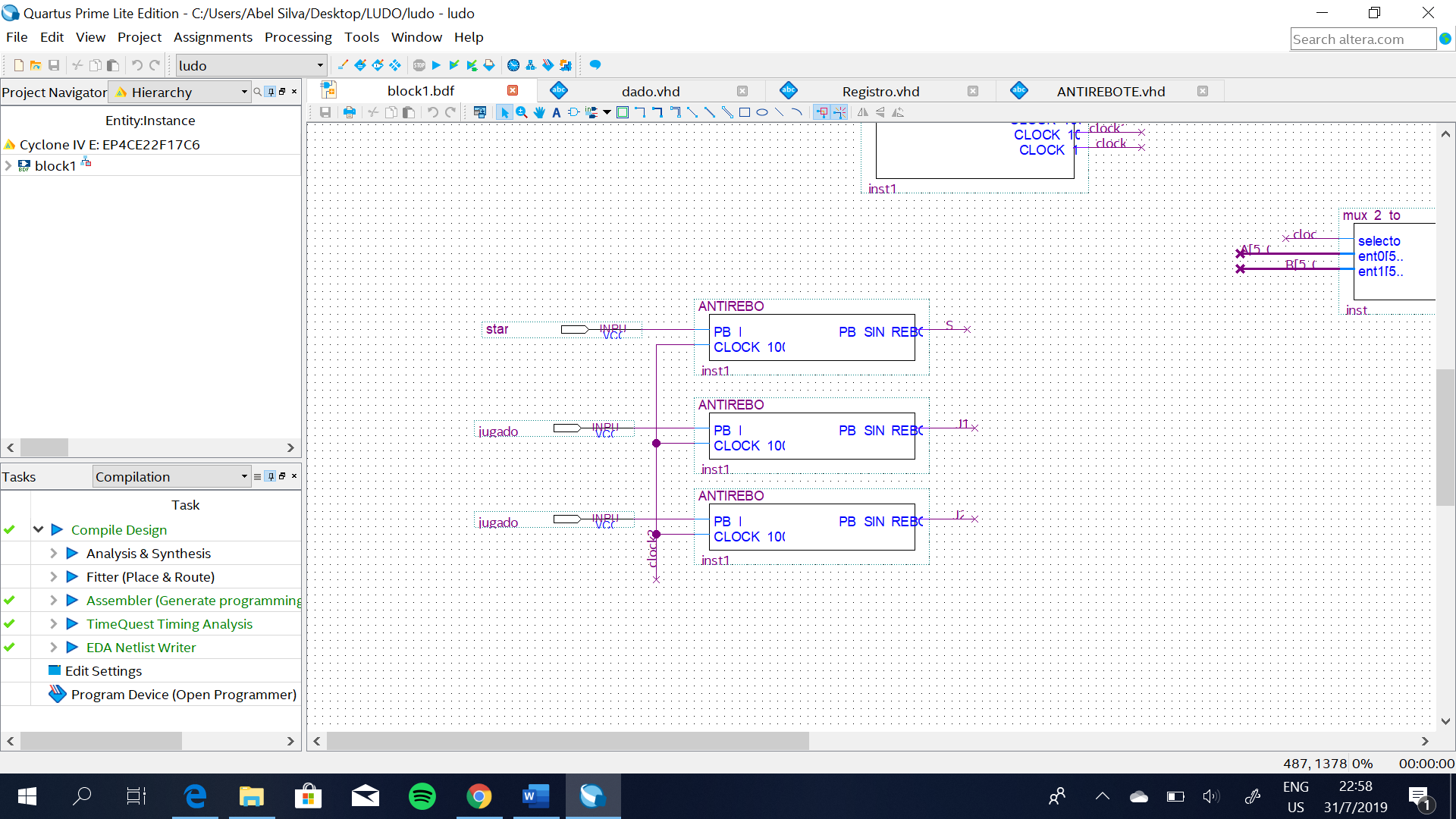
Comparación >= 32

Reinicio Juego

Fin Juego

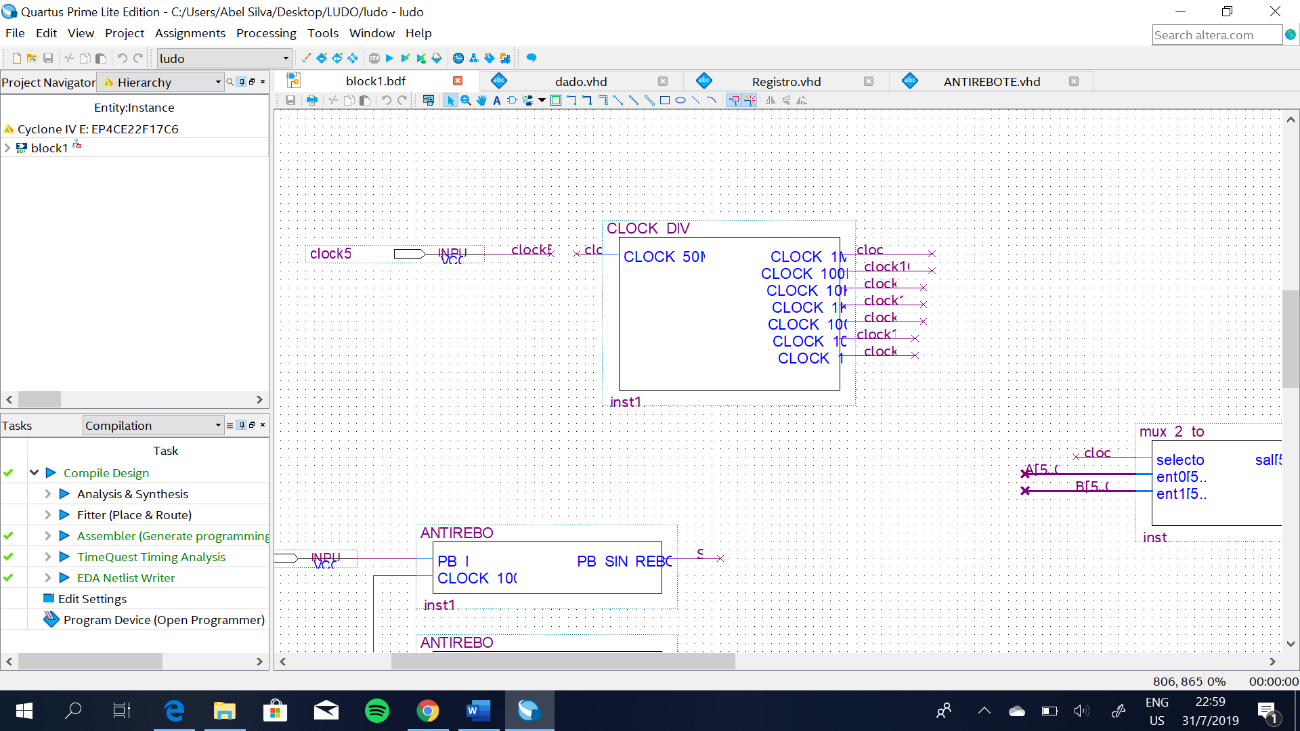
1. **Partición Funcional del Sistema Digital**

**Bloque Anti-rebote**



***Imagen 3 – Bloque Anti-Rebote***

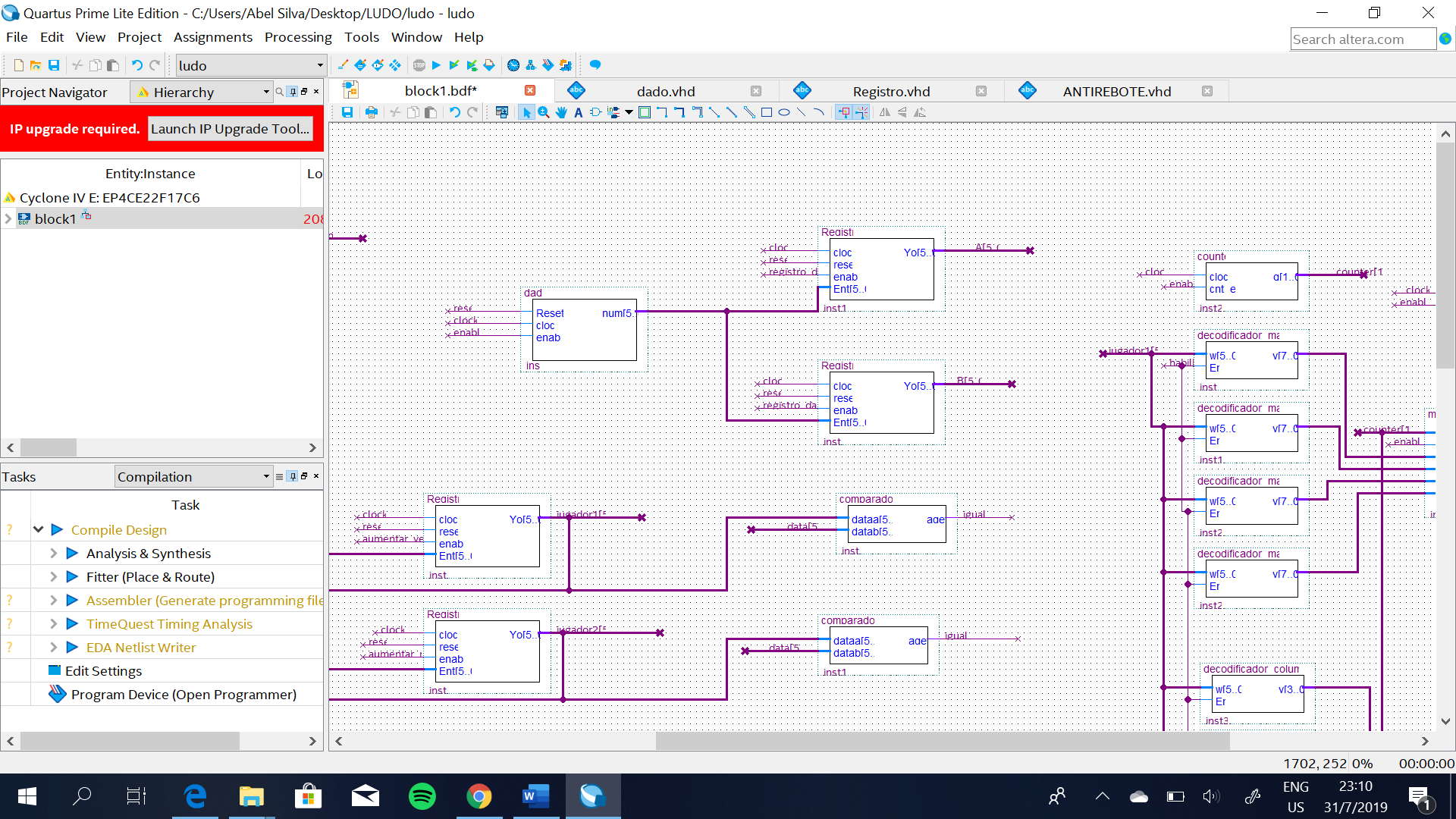
En el sistema se disponen de tres botoneras, por lo que para cada una de ellas incluye el sistema anti rebote, para poder evitar de esta manera algún comportamiento no deseado dentro de la pulsación de las mismas.

**Bloque de Reloj**

***Imagen 4 – Bloque Reloj***

Parte muy importante del sistema implementado, permite el control del sistema mediante el control de los flancos de la MSS, se produce en este bloque la salida de una señal que realiza el efecto random del dado.

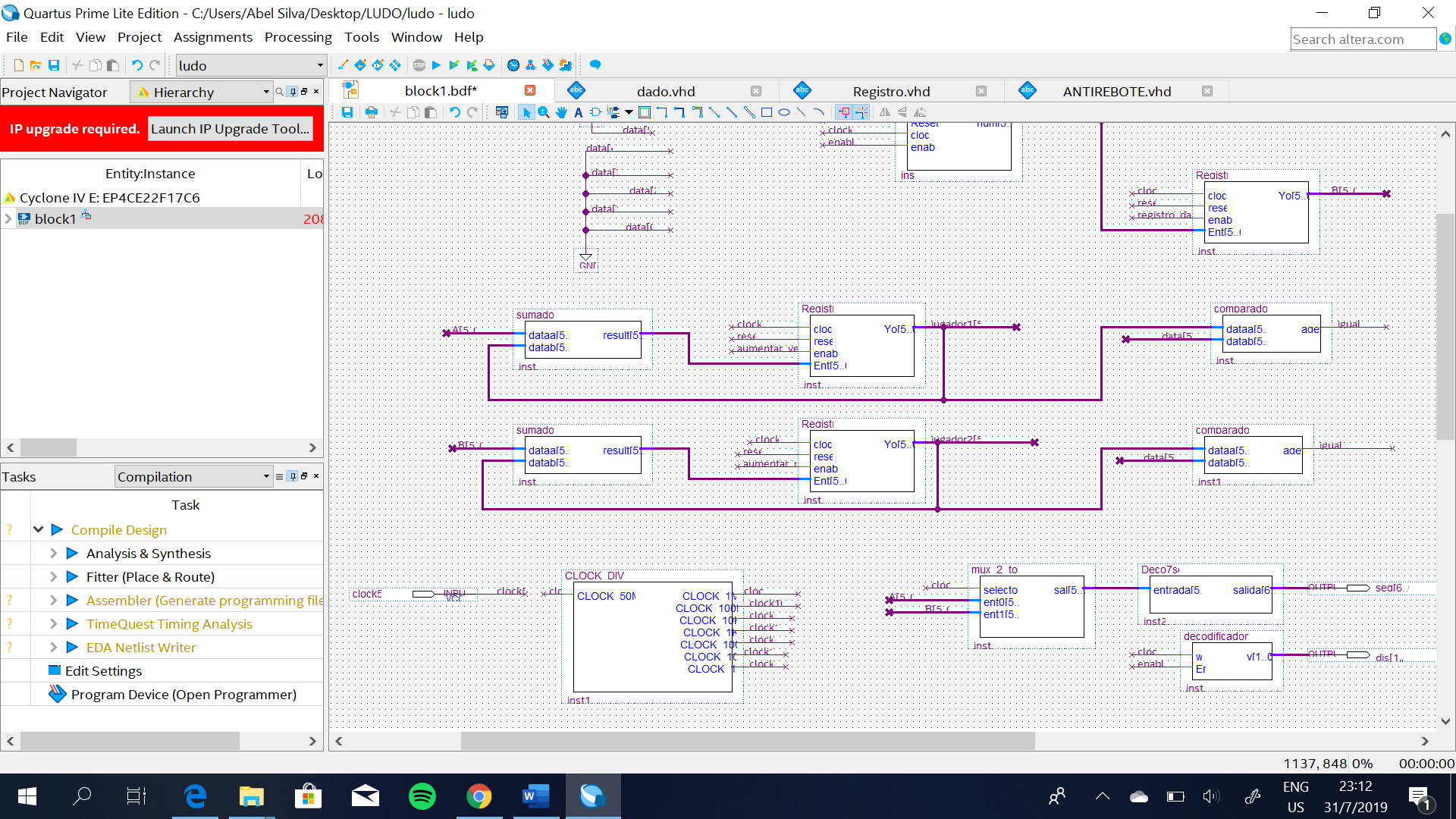
**Dado y Registro**



***Imagen 5 – Bloque Dado y Registro***

Bloque que permite la generación de un numero aleatorio, la señal CLOCK que le ingresa al dado, presenta una muy alta frecuencia, lo que permite obtener el efecto random del dado, los bloques de registro se realizan para cada jugador lo que permite la acumulación del valor de la salida del dado mediante la señal num [5..0].

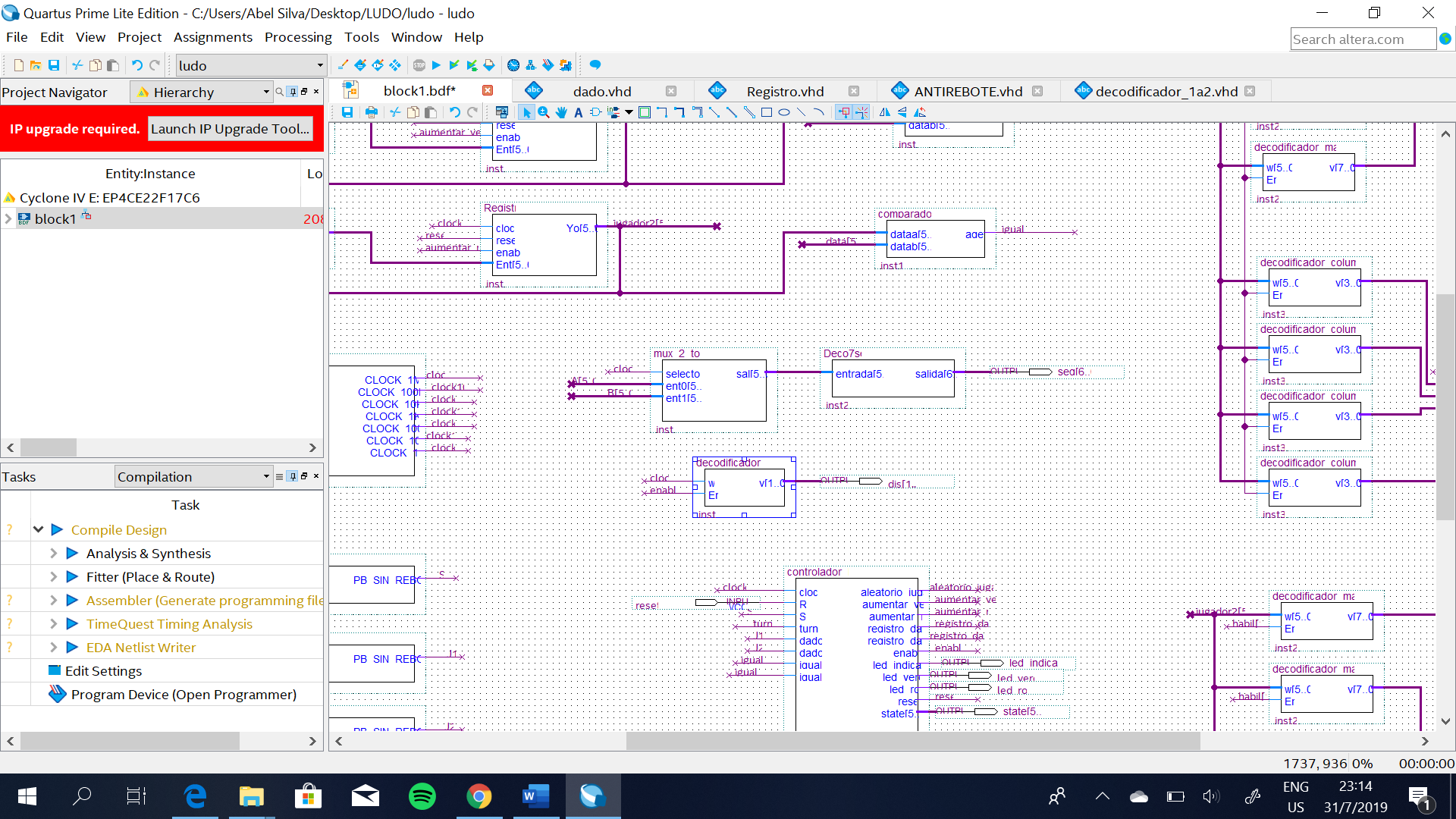
**Sumador, Registro y Comparador**



***Imagen 6 – Bloque Sumador y Comparador***

Parte del sistema que realiza la acción de suma, acumulación y comparación de los valores para cada jugador, una vez que se tenga un numero de salida se suma, para permitir el incremento de los valores, es por eso que presenta la retroalimentación, la parte final, tiene como entrada el valor de salida del registro, y el numero 32, realiza la comparación entre estos dos valores para los dos jugadores del sistema.

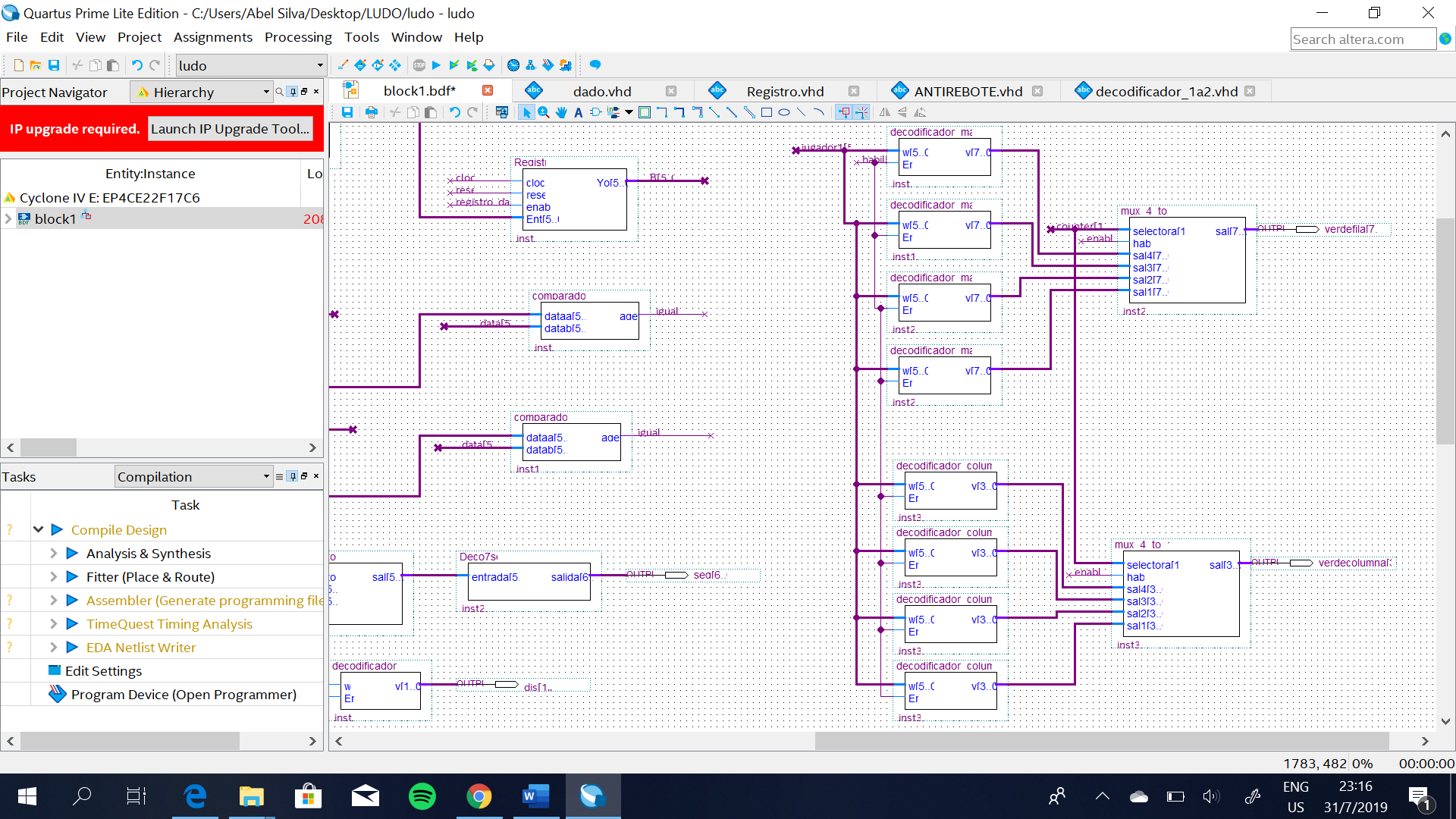
**Decodificador 7 Segmentos**



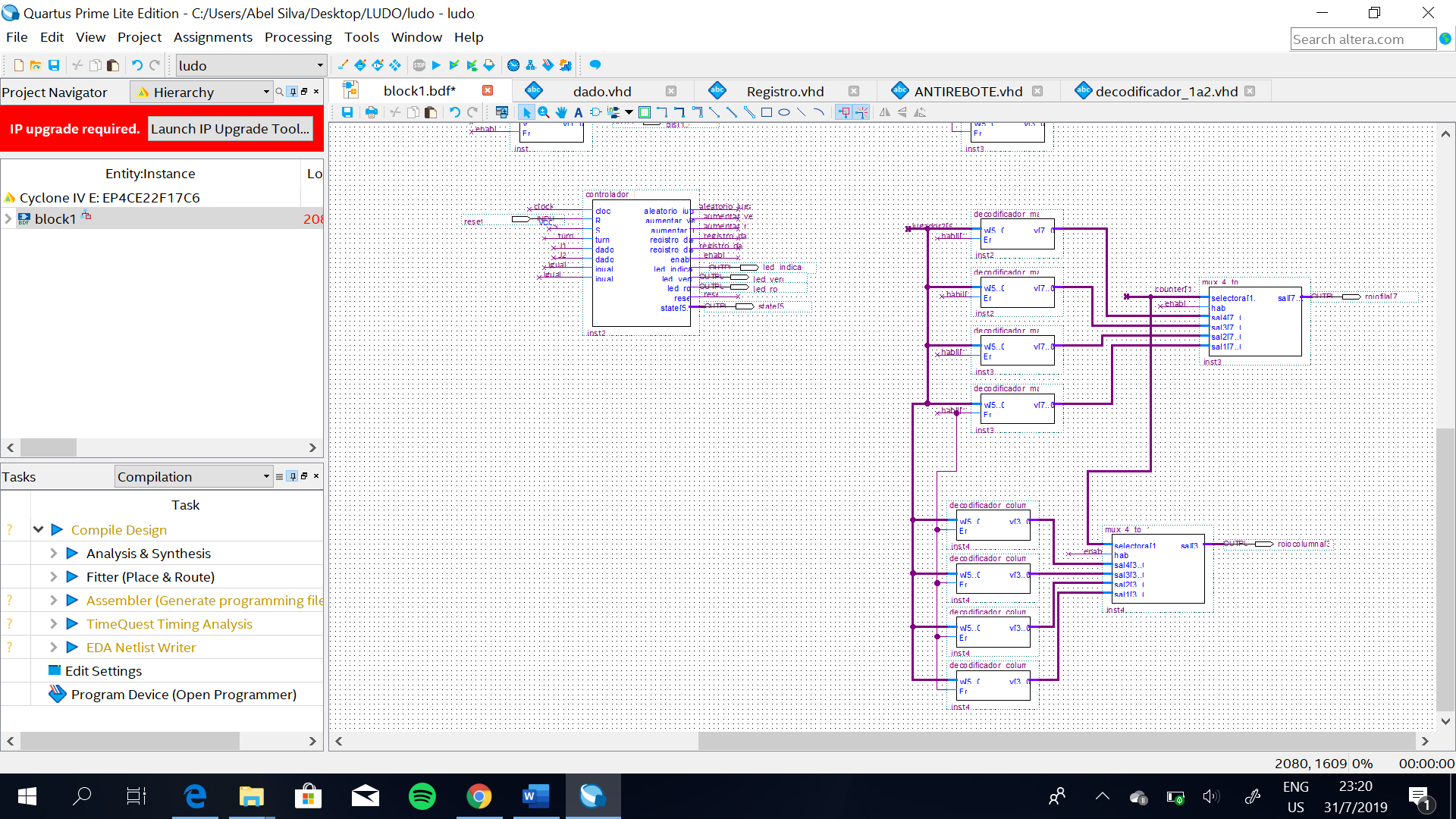
***Imagen 7 – Salida del Decodificador 7 segmentos***

Bloque del sistema donde previamente a mostrar la salida en el led 7 segmentos se realiza la elección de la señal mediante un mux de dos a 1, donde recibe como datos la salida del dado, tanto para jugador A como para jugador B.

**Decodificador Salida de Matriz Jugador A y B**

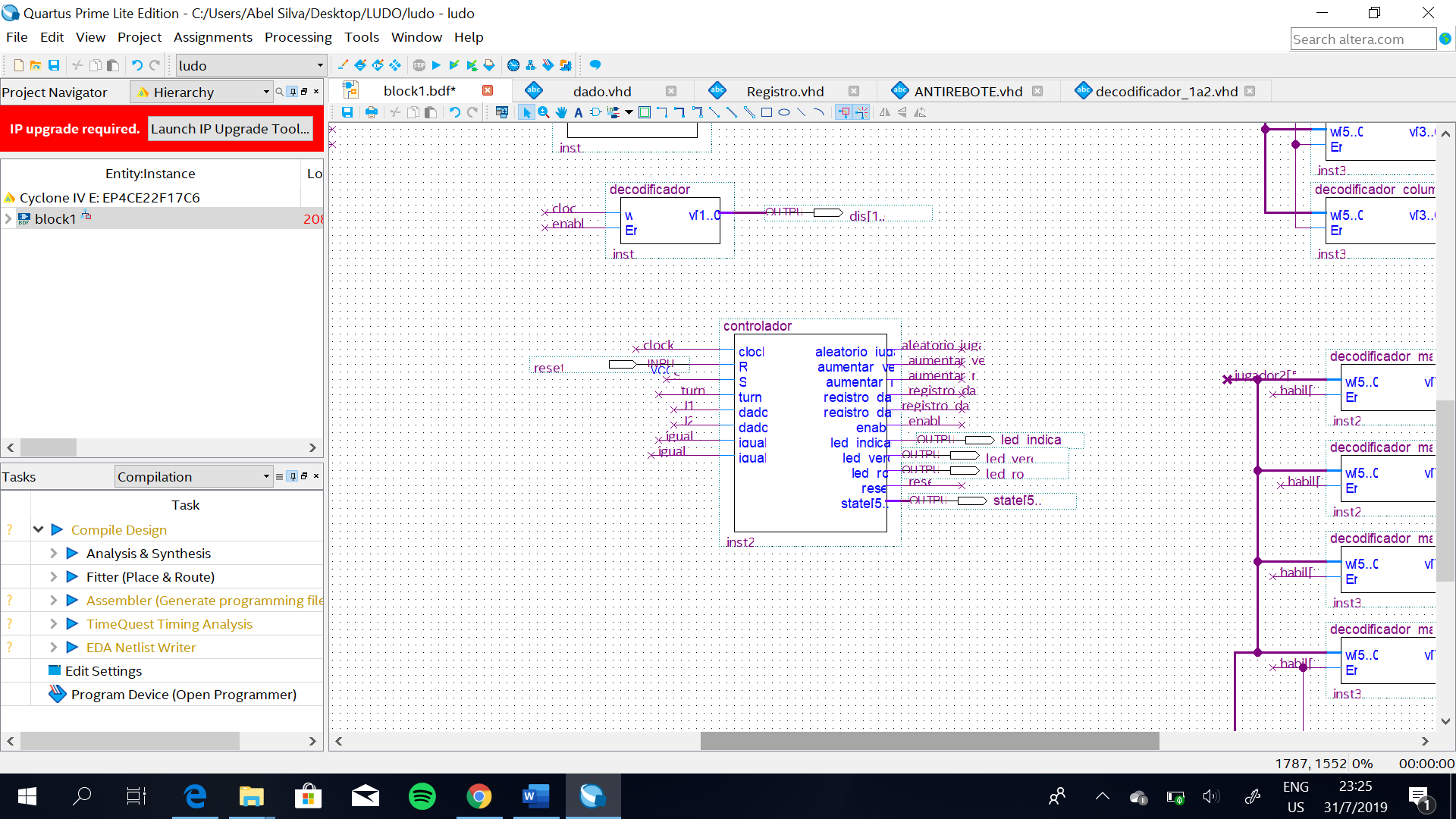


***Imagen 8 – Decodificador Jugador A***



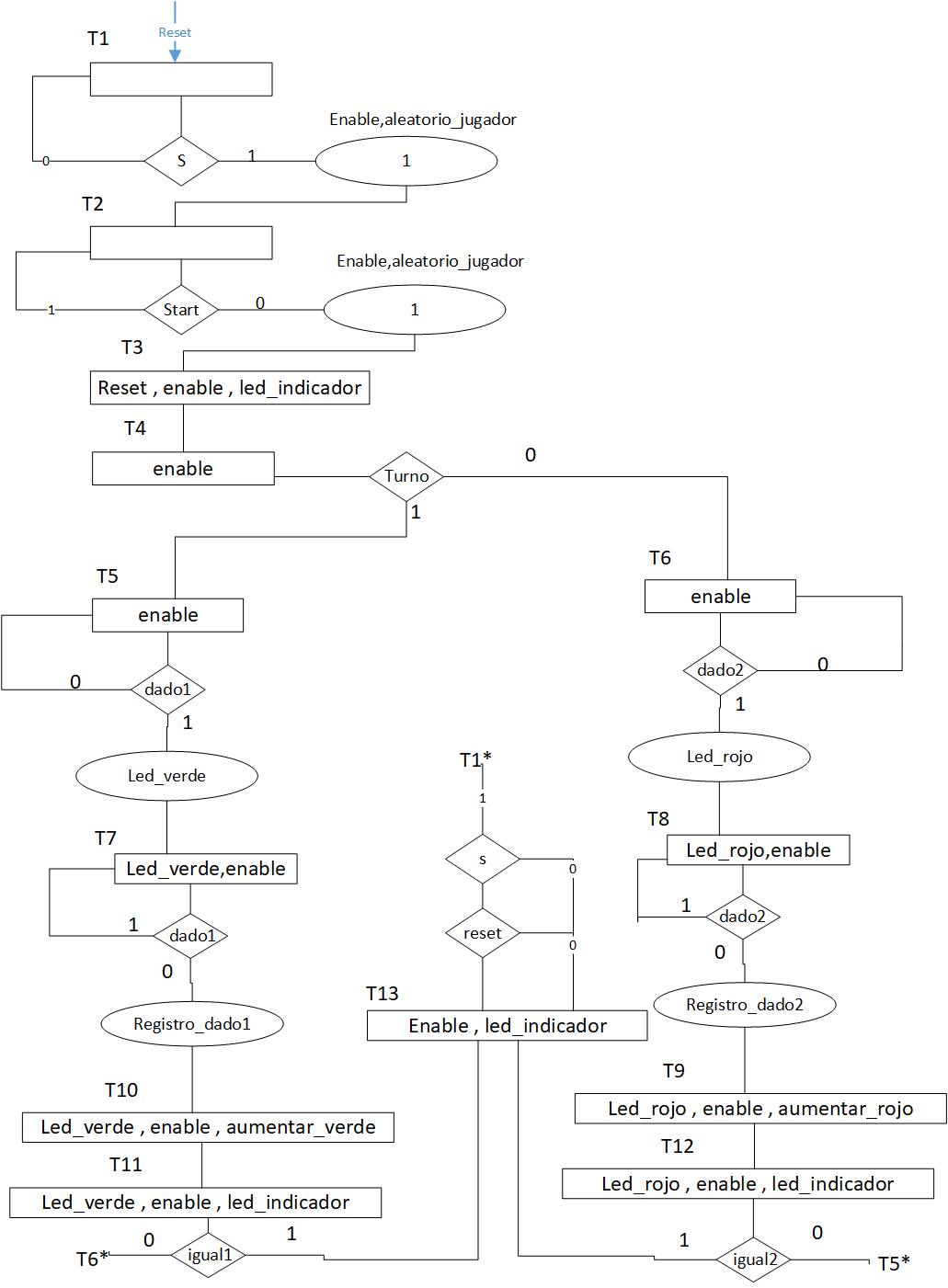
***Imagen 9 – Decodificador Jugador B***

Para las salidas de los LEDs, no se usó 32 focos como tal por cada jugador, por cuestiones de practicidad se realizó la implentación de una matriz de 64 LEDs, de los cuales se dividió en dos partes de 32 cada una, en la aplicación física, se realiza multiplexación tanto por columnas como por filas, teniendo así el comportamiento deseado, para los dos jugadores se aplicó el mismo principio.

**Bloque Controlador**

***Imagen 10 – Controlador del sistema***

Controlador del sistema, bloque donde se realiza el control de los estados y el tratamiento de todas las señales tanto de entrada como de salida, permitiendo de esta forma el correcto comportamiento de cada de uno de los parámetros para obtener finalmente el sistema digital totalmente operativo.

1. **Diagrama ASM del controlador**

***Imagen 11 – Diagrama ASM del sistema***

El diagrama ASM de la mss que coordinará el circuito del ludo cuenta con 13 estados en total. El primer estado esperará por la señal Start para proseguir hacia el estado T2, activando sus respectivos habilitadores, entre los cuales se encuentra el habilitador para genera el primer turno, el cual hay que recordar, debe ser completamente aleatorio. En el estado T2, por el contrario, se comprobará que la señal de start no se encuentre activada para poder proseguir con los estados T3 y T4, que respectivamente se encargan de realizar una pequeña pausa y elegir hacia qué segmento se dirigirá a trabajar el circuito a continuación: bloque de jugador A o B; los bloques de cada jugador son similares, donde cada uno comprobará si ya se jugó su respectivo dado, una vez que reciba este habilitador, pasará al siguiente estado donde habilitará el registro y adición del número sacado en el dado a su posición actual en el tablero. Posteriormente, cada uno revisará si alguno ya completó los 32 leds que se necesitan para llenar el camino y ganar el juego; si no se han completado, simplemente el turno pasará a ser el del siguiente jugador, por el contrario, si ya se completó alguno de los dos caminos, la maquina volverá al estado inicial T1 a la espera del botón de start.ç

1. **Código VHDL de la MSS**

library ieee;

use ieee.std\_logic\_1164.all;

entity controlador\_ludo is

**- -Declaración de cada una de las entradas y salidas del sistema - -**

Port(

clock: in std\_logic;

R, S, turno, dado1, dado2, igual1, igual2 : in std\_logic;

aleatorio\_jugador, aumentar\_verde, aumentar\_rojo,registro\_dado1,

registro\_dado2, enable, led\_indicador,led\_verde, led\_rojo,reset: out std\_logic;

state: out std\_logic\_vector(5 downto 0));

end controlador\_ludo;

**- -Declaración de todos los estados - -**

Architecture sol of controlador\_ludo is

type estado is (T1, T2, T3, T4, T5, T6, T7, T8, T9, T10,

T11, T12, T13);

signal y:estado;

begin

--transiciones

**- -Process para decodificador de estados siguientes - -**

Process(R, clock)

begin

if R = '1' then y <= T1;

elsif clock'event and clock = '1' then

case y is

when T1 => if S = '1' then y<=T2; else y<=T1; end if;

when T2 => if S = '0' then y<=T3; else y<=T2; end if;

when T3 => y<=T4;

when T4 => if turno = '1' then y<=T5;

else y<=T6; end if;

when T6 => if dado2 = '1' then y<=T8; else y<=T6; end if;

when T8 => if dado2 = '0' then y<=T9; else y<=T8; end if;

when T5 => if dado1 = '1' then y<=T7; else y<=T5; end if;

when T7 => if dado1 = '0' then y<=T10; else y<=T7; end if;

when T10 => y<=T11;

when T9 => y<=T12;

when T11 => if igual1 = '1' then y<=T13;

else y<=T6; end if;

when T12 => if igual2 = '1' then y<=T13;

else y<=T5; end if;

when T13 => y<=T1;

end case;

end if;

end process;

--Salidas

**- -Process para decodificador de salidas - -**

process(R, clock)

begin

aleatorio\_jugador<= '0'; aumentar\_verde<= '0'; aumentar\_rojo<= '0';registro\_dado1<= '0';

registro\_dado2<= '0'; enable<= '0'; led\_indicador<= '0';led\_verde<= '0'; led\_rojo<= '0';reset<= '0';

case y is

when T1 => state <= "000001";enable<= '1';aleatorio\_jugador<= '1';

when T2 => state <= "000010";enable<= '1';

when T3 => state <= "000011";reset<='1';enable<= '1';led\_indicador<= '1';

when T4 => state <= "000100"; enable<= '1';

when T5 => state <= "000101"; led\_verde<= '1';enable<= '1';

when T6 => state <= "000110"; enable<= '1';led\_rojo<= '1';

when T7 => state <= "000111"; led\_verde<= '1';enable<= '1'; registro\_dado1<= '1';

when T8 => state <= "001000";enable<= '1';led\_rojo<= '1';registro\_dado2<= '1';

when T9 => state <= "001001";enable<= '1';led\_rojo<= '1';aumentar\_rojo<= '1';

when T10 => state <= "001010";led\_verde<= '1';enable<= '1';aumentar\_verde<= '1';

when T11 => state <= "001011";led\_verde<= '1';enable<= '1';led\_indicador<= '1';

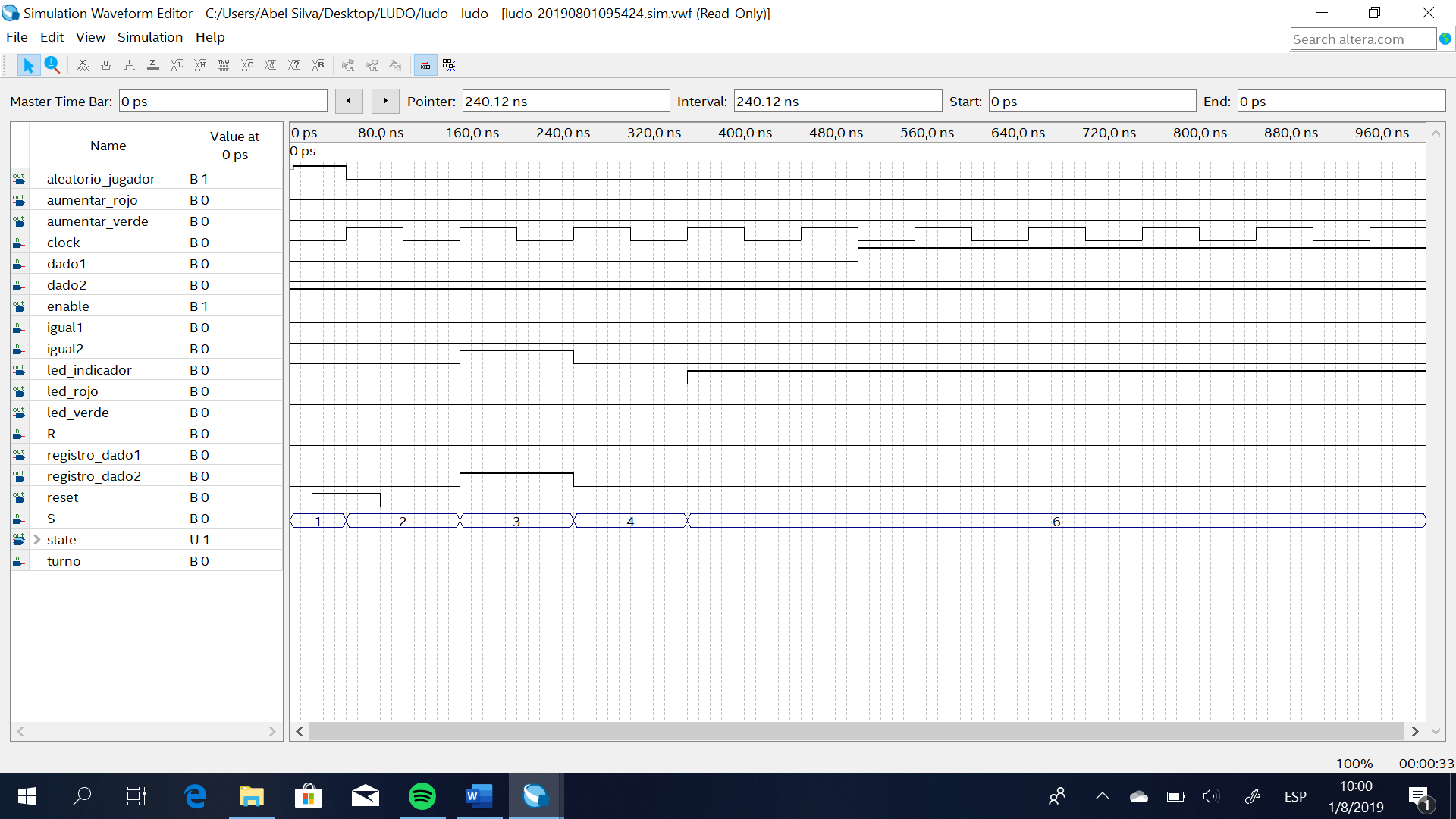
when T12 => state <= "001100";enable<= '1';led\_rojo<= '1';led\_indicador<= '1';

when T13 => state <= "001100";enable<= '1';led\_indicador<= '1';

end case;

end process;

end sol;

1. **Diagrama de tiempo de la MSS**

***Imagen 12 – Diagrama de tiempo sistema***

Se muestra la simulación del controlador con cada uno de los estados como se puede visualizar, la señal state cambia los valores como debería ser el comportamiento del sistema , la señal aleatorio\_jugador se inicia y simula cuando el sistema genera un numero aleatorio , la señal Clock muestra los flancos en los que se activa , la señal led\_rojo demuestra en el sistema que se activó un jugador por lo tanto se puede verificar que el sistema está funcionando de forma correcta , como se quiere , es necesario indicar que la salida 7 segmentos y la salida de la matriz son controladas por otros bloques , es por eso que esas señales no son mostradas en el controlador .

1. **Código VHDL completo de todos los bloques**

Bloque Anti-rebote creado para las botoneras principales del sistema, start, jugador a y jugador b siendo el mismo código, para cada uno de ellos, solo variando las salidas de cada botonera, se realiza la implementación de este bloque para evitar respuestas no deseadas, al momento de que se suelta cada una de estas botoneras.

LIBRARY IEEE;

USE IEEE.STD\_LOGIC\_1164.all;

USE IEEE.STD\_LOGIC\_ARITH.all;

USE IEEE.STD\_LOGIC\_UNSIGNED.all;

-- Debounce Pushbutton: Filters out mechanical switch bounce for around 40Ms.

ENTITY ANTIREBOTE IS

PORT(PB\_N, CLOCK\_100Hz : IN STD\_LOGIC;

PB\_SIN\_REBOTE : OUT STD\_LOGIC);

END ANTIREBOTE;

ARCHITECTURE a OF ANTIREBOTE IS

SIGNAL SHIFT\_PB : STD\_LOGIC\_VECTOR(3 DOWNTO 0);

BEGIN

-- Debounce clock should be approximately 10ms or 100Hz

PROCESS

BEGIN

WAIT UNTIL (clock\_100Hz'EVENT) AND (clock\_100Hz = '1');

-- Use a shift register to filter switch contact bounce

SHIFT\_PB(2 DOWNTO 0) <= SHIFT\_PB(3 DOWNTO 1);

SHIFT\_PB(3) <= NOT PB\_N;

IF SHIFT\_PB(3 DOWNTO 0)="0000" THEN

PB\_SIN\_REBOTE <= '0';

ELSE

PB\_SIN\_REBOTE <= '1';

END IF;

END PROCESS;

END a;

Bloque de reloj, creado para controlar cada flanco positivo del reloj, y de esta manera poner en estado activo nuestro sistema, también para controlar la duración de cada proceso que se da en la realización del juego.

LIBRARY IEEE;

USE IEEE.STD\_LOGIC\_1164.all;

USE IEEE.STD\_LOGIC\_ARITH.all;

USE IEEE.STD\_LOGIC\_UNSIGNED.all;

ENTITY CLOCK\_DIV\_50 IS

PORT

( CLOCK\_50MHz :IN STD\_LOGIC;

CLOCK\_1MHz :OUT STD\_LOGIC;

CLOCK\_100KHz :OUT STD\_LOGIC;

CLOCK\_10KHz :OUT STD\_LOGIC;

CLOCK\_1KHz :OUT STD\_LOGIC;

CLOCK\_100Hz :OUT STD\_LOGIC;

CLOCK\_10Hz :OUT STD\_LOGIC;

CLOCK\_1Hz :OUT STD\_LOGIC);

END CLOCK\_DIV\_50;

ARCHITECTURE a OF CLOCK\_DIV\_50 IS

SIGNAL count\_1Mhz: STD\_LOGIC\_VECTOR(5 DOWNTO 0);

SIGNAL count\_100Khz, count\_10Khz, count\_1Khz: STD\_LOGIC\_VECTOR(2 DOWNTO 0);

SIGNAL count\_100hz, count\_10hz, count\_1hz: STD\_LOGIC\_VECTOR(2 DOWNTO 0);

SIGNAL clock\_1Mhz\_int,clock\_100Khz\_int,clock\_10Khz\_int,clock\_1Khz\_int: STD\_LOGIC;

SIGNAL clock\_100hz\_int, clock\_10hz\_int, clock\_1hz\_int: STD\_LOGIC;

BEGIN

PROCESS

BEGIN

-- Divide by 50

WAIT UNTIL clock\_50Mhz'EVENT and clock\_50Mhz = '1'; -- 24 Mhz

IF count\_1Mhz < 49 THEN

count\_1Mhz <= count\_1Mhz + 1;

ELSE

count\_1Mhz <= "000000";

END IF;

IF count\_1Mhz < 4 THEN

clock\_1Mhz\_int <= '0';

ELSE

clock\_1Mhz\_int <= '1';

END IF;

-- Ripple clocks are used in this code to save prescalar hardware

-- Sync all clock prescalar outputs back to master clock signal

clock\_1Mhz <= clock\_1Mhz\_int;

clock\_100Khz <= clock\_100Khz\_int;

clock\_10Khz <= clock\_10Khz\_int;

clock\_1Khz <= clock\_1Khz\_int;

clock\_100hz <= clock\_100hz\_int;

clock\_10hz <= clock\_10hz\_int;

clock\_1hz <= clock\_1hz\_int;

END PROCESS;

-- Divide by 10

PROCESS

BEGIN

WAIT UNTIL clock\_1Mhz\_int'EVENT and clock\_1Mhz\_int = '1';

IF count\_100Khz /= 4 THEN

count\_100Khz <= count\_100Khz + 1;

ELSE

count\_100Khz <= "000";

clock\_100Khz\_int <= NOT clock\_100Khz\_int;

END IF;

END PROCESS;

-- Divide by 10

PROCESS

BEGIN

WAIT UNTIL clock\_100Khz\_int'EVENT and clock\_100Khz\_int = '1';

IF count\_10Khz /= 4 THEN

count\_10Khz <= count\_10Khz + 1;

ELSE

count\_10Khz <= "000";

clock\_10Khz\_int <= NOT clock\_10Khz\_int;

END IF;

END PROCESS;

-- Divide by 10

PROCESS

BEGIN

WAIT UNTIL clock\_10Khz\_int'EVENT and clock\_10Khz\_int = '1';

IF count\_1Khz /= 4 THEN

count\_1Khz <= count\_1Khz + 1;

ELSE

count\_1Khz <= "000";

clock\_1Khz\_int <= NOT clock\_1Khz\_int;

END IF;

END PROCESS;

-- Divide by 10

PROCESS

BEGIN

WAIT UNTIL clock\_1Khz\_int'EVENT and clock\_1Khz\_int = '1';

IF count\_100hz /= 4 THEN

count\_100hz <= count\_100hz + 1;

ELSE

count\_100hz <= "000";

clock\_100hz\_int <= NOT clock\_100hz\_int;

END IF;

END PROCESS;

-- Divide by 10

PROCESS

BEGIN

WAIT UNTIL clock\_100hz\_int'EVENT and clock\_100hz\_int = '1';

IF count\_10hz /= 4 THEN

count\_10hz <= count\_10hz + 1;

ELSE

count\_10hz <= "000";

clock\_10hz\_int <= NOT clock\_10hz\_int;

END IF;

END PROCESS;

-- Divide by 10

PROCESS

BEGIN

WAIT UNTIL clock\_10hz\_int'EVENT and clock\_10hz\_int = '1';

IF count\_1hz /= 4 THEN

count\_1hz <= count\_1hz + 1;

ELSE

count\_1hz <= "000";

clock\_1hz\_int <= NOT clock\_1hz\_int;

END IF;

END PROCESS;

END a;

Bloque de dado y registro, en este bloque se realiza la generación de los números del 1-6 en el dado mediante el efecto de una frecuencia muy alta, y de esta manera poder obtener un valor random, debido a la velocidad en la que se cambian los números.

library ieee;

use ieee.std\_logic\_1164.all;

entity dado is

Port(

Resetn, clock, enable: in std\_logic;

num: out std\_logic\_vector(5 downto 0)

);

end dado;

Architecture sol of dado is

type estado is (T1, T2, T3, T4, T5, T6);

signal y:estado;

begin

Process(resetn, clock)

begin

if Resetn = '1' then y <= T1;

elsif clock'event and clock = '1' then

if(enable='1') then

case y is

when T1 => y<= T2;

when T2 => y<= T3;

when T3 => y<= T4;

when T4 => y<= T5;

when T5 => y<= T6;

when T6 => y<= T1;

end case;

else y<=y;

end if;

end if;

end Process;

process(resetn, clock)

begin

case y is

when T1 => num <= "000001";

when T2 => num <= "000010";

when T3 => num <= "000011";

when T4 => num <= "000100";

when T5 => num <= "000101";

when T6 => num <= "000110";

end case;

end process;

end sol;

Registro, bloque que realiza el guardado del número generado por el dado, existen dos bloques debido a que son dos jugadores, ambos bloques funcionan bajo el mismo dado, almacenando el valor que sale en el dado.

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.std\_logic\_unsigned.all;

entity Registro is

PORT(clock,reset,enable: IN STD\_LOGIC;

Ent : IN STD\_LOGIC\_VECTOR(5 downto 0);

Yo : OUT STD\_LOGIC\_VECTOR (5 downto 0));

END Registro;

architecture operacion of Registro is

signal temp: STD\_LOGIC\_VECTOR(5 downto 0);

BEGIN

PROCESS(clock,reset)

BEGIN

if reset='1' then temp<="000000";

elsif (clock'event and clock='1') then

if(enable='1') then

temp<=Ent;

end if;

end if;

end PROCESS;

Yo<=temp;

end operacion;

Sumador, Registro y Comparador bloque integrado que realiza la suma (sumador), de los valores arrojados por el dado, el acumulador se encarga de irlos acumulando y el comparador final es el que verifica cuando el valor es mayo a 32, que es donde el juego finaliza, se realiza la implementaciónón de cada uno de estos bloques por cada jugador tenga el sistema.

**Registro**

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.std\_logic\_unsigned.all;

entity Registro is

PORT(clock,reset,enable: IN STD\_LOGIC;

Ent : IN STD\_LOGIC\_VECTOR(5 downto 0);

Yo : OUT STD\_LOGIC\_VECTOR (5 downto 0));

END Registro;

architecture operacion of Registro is

signal temp: STD\_LOGIC\_VECTOR(5 downto 0);

BEGIN

PROCESS(clock,reset)

BEGIN

if reset='1' then temp<="000000";

elsif (clock'event and clock='1') then

if(enable='1') then

temp<=Ent;

end if;

end if;

end PROCESS;

Yo<=temp;

end operacion;

Decodificador 7 segmentos, bloque que realiza primero mediante un mux la selección de que jugador se trata y ya con el valor de dado generado se muestra en la salida de 7 segmentos el valor que le toca avanzar.

**MUX**

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.std\_logic\_unsigned.all;

entity mux\_2\_to\_1 is

port(selectora : in std\_logic;

ent0,ent1 : in std\_logic\_vector(5 downto 0);

sal : out std\_logic\_vector (5 downto 0));

end mux\_2\_to\_1;

architecture arq of mux\_2\_to\_1 is

begin

with selectora select

sal <= ent0 when '0',

ent1 when '1';

end arq;

**Decodificador 7 segmentos**

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.std\_logic\_unsigned.all;

entity Deco7seg is

port(entrada : in std\_logic\_vector(5 downto 0);

salida: out std\_logic\_vector(6 downto 0));

end Deco7seg;

architecture arq of Deco7seg is

SIGNAL y: std\_logic\_vector (6 downto 0);

begin

with entrada select

y <= "1111110" when "000000",--0

"0110000" when "000001",--1

"1101101" when "000010",--2

"1111001" when "000011",--3

"0110011" when "000100",--4

"1011011" when "000101",--5

"1011111" when "000110",--6

"1110000" when "000111",--7

"1111111" when "001000",--8

"1111011" when "001001",--9

"0000000" when others;

salida <= y;

end arq;

Decodificador de salida de Matriz, primero en este bloque se realiza la decodificación de la matriz por columna y por filas, para poder controlar la cantidad de leds a encender dependiendo del jugador y del valor, para luego con el uso del multiplexor, realizar la selección de que valor nos dieron los multiplexores, este procedimiento es realizado para los jugadores.

**MUX Filas**

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.std\_logic\_unsigned.all;

entity mux\_4\_to\_1 is

port(selectora : in std\_logic\_vector(1 downto 0);

habil : in std\_logic;

sal4 : in std\_logic\_vector(7 downto 0);

sal3 : in std\_logic\_vector(7 downto 0);

sal2 : in std\_logic\_vector(7 downto 0);

sal1 : in std\_logic\_vector(7 downto 0);

sal : out std\_logic\_vector (7 downto 0));

end mux\_4\_to\_1;

architecture arq of mux\_4\_to\_1 is

begin

process(habil, selectora)

begin

if habil = '1' then

case selectora is

when "00" => sal <= sal1;

when "01" => sal <= sal2;

when "10" => sal <= sal3;

when "11" => sal <= sal4;

when others => sal <= "ZZZZZZZZ";

end case;

end if;

end process;

end arq;

**MUX Columnas**

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.std\_logic\_unsigned.all;

entity mux\_4\_to\_1\_2 is

port(selectora : in std\_logic\_vector(1 downto 0);

habil : in std\_logic;

sal4 : in std\_logic\_vector(3 downto 0);

sal3 : in std\_logic\_vector(3 downto 0);

sal2 : in std\_logic\_vector(3 downto 0);

sal1 : in std\_logic\_vector(3 downto 0);

sal : out std\_logic\_vector (3 downto 0));

end mux\_4\_to\_1\_2;

architecture arq of mux\_4\_to\_1\_2 is

begin

process(habil, selectora)

begin

if habil = '1' then

case selectora is

when "00" => sal <= sal1;

when "01" => sal <= sal2;

when "10" => sal <= sal3;

when "11" => sal <= sal4;

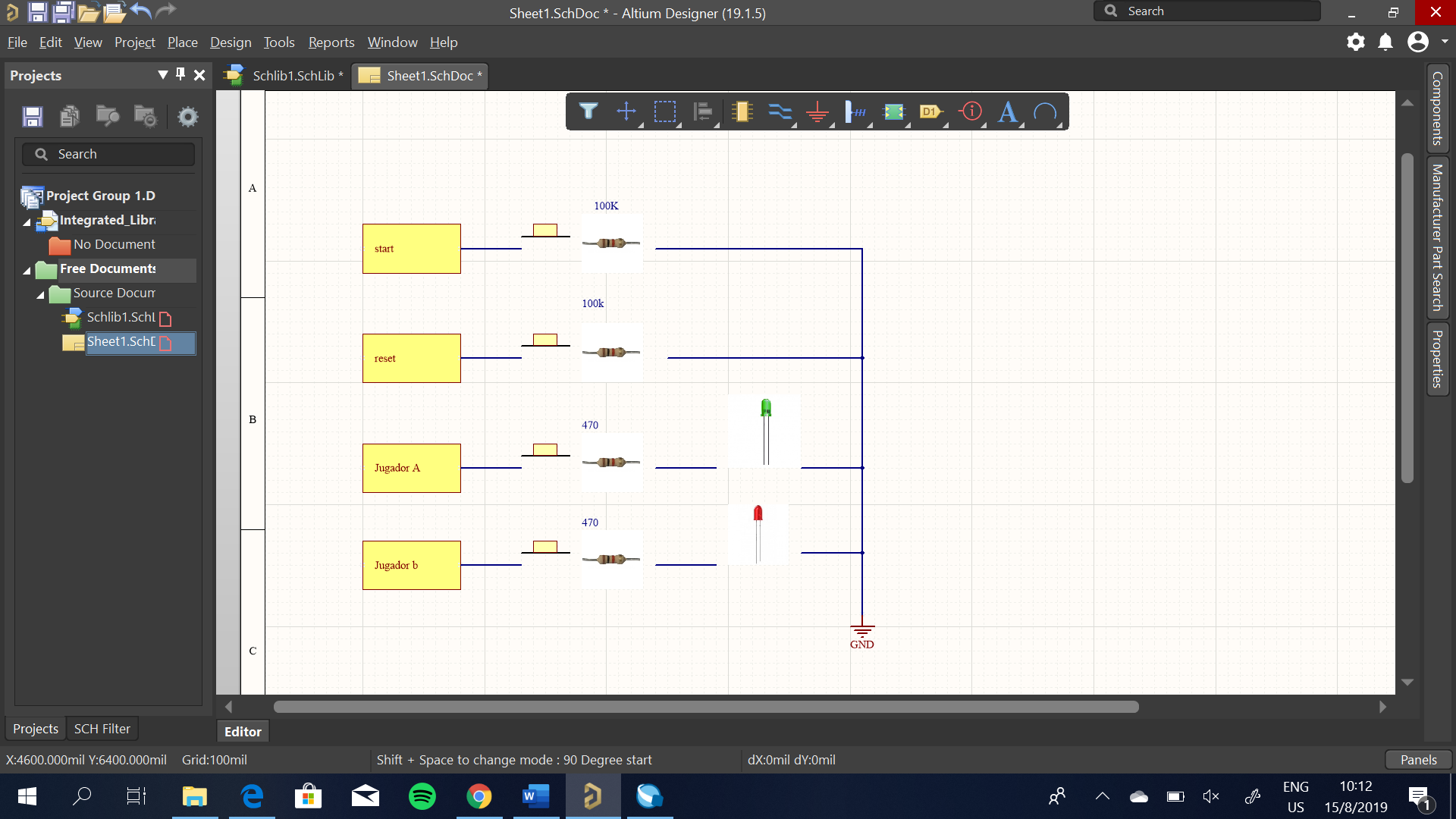
when others => sal <= "ZZZZ";

end case;

end if;

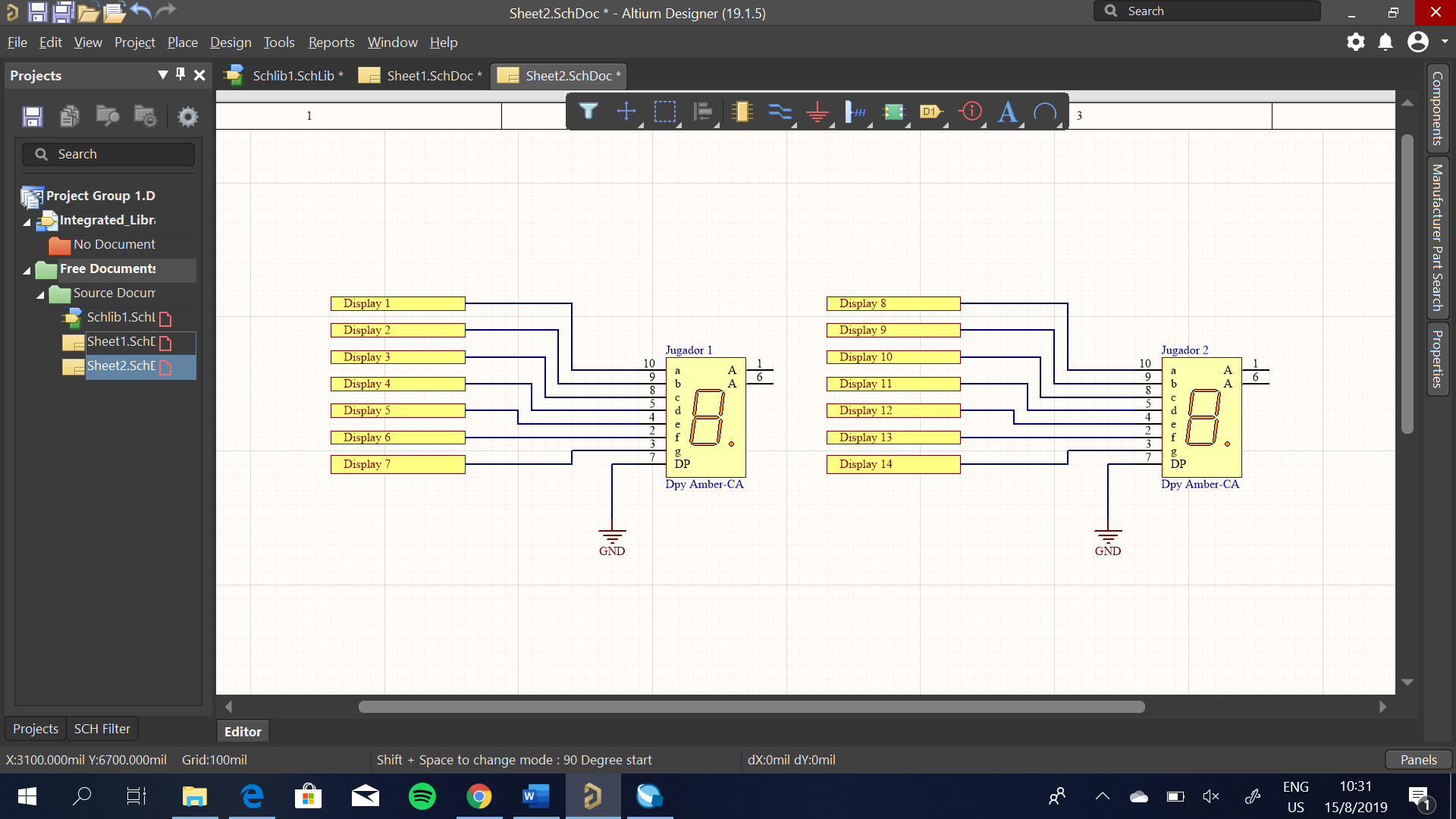
end process;

end arq;

1. **Diagrama Esquemático**

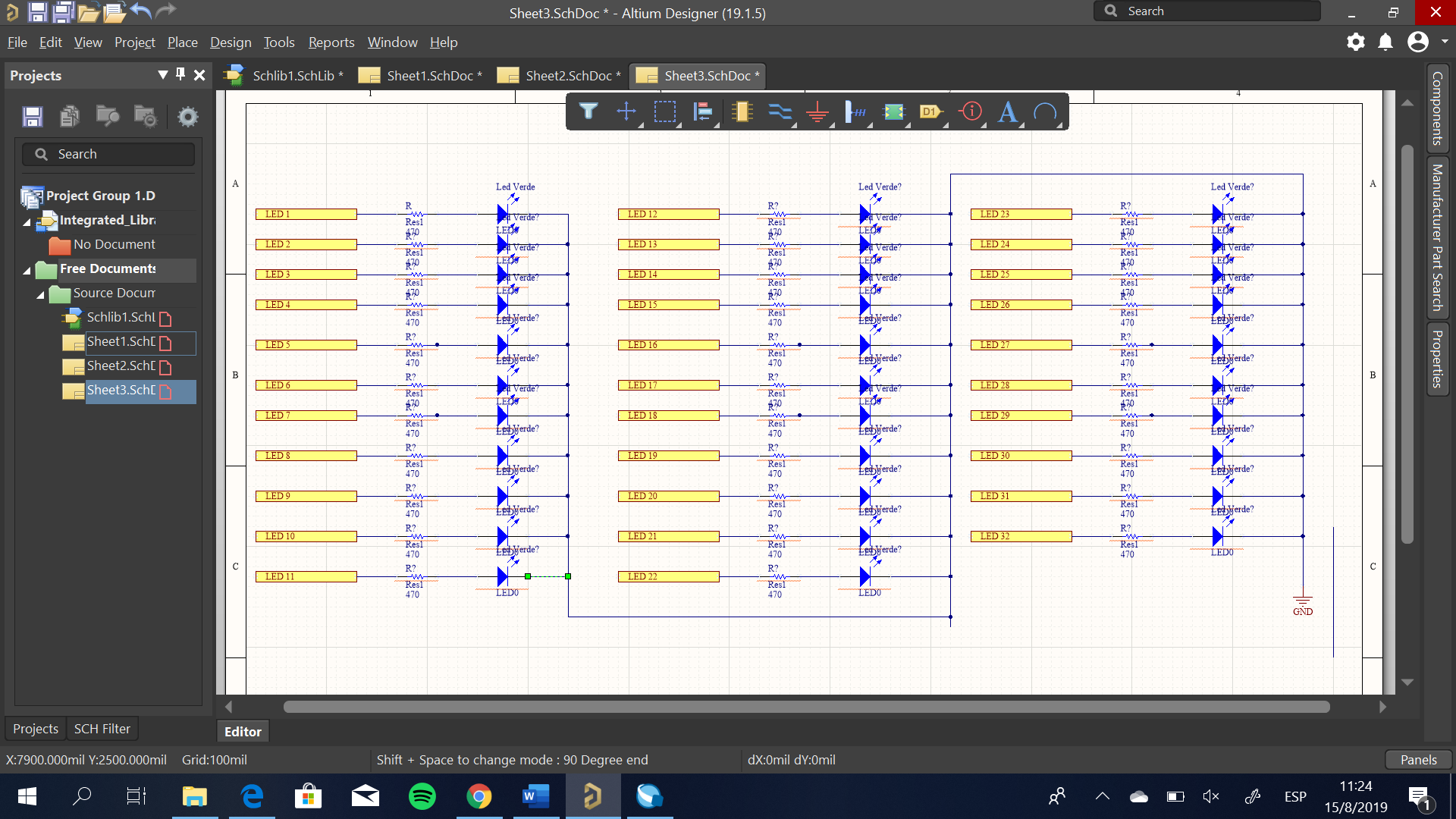
***Imagen 13 – Diagrama esquemático botonera***

Se tiene todas las botoneras de nuestro sistema, start es el que inicia el funcionamiento del juego, Reset me reinicia el sistema y las botoneras correspondientes a cada jugador A y B, que son las que me generarán el número del dado del jugador.



***Imagen 14 – Diagrama esquemático salida 7 segmentos***

En esta parte se evidencia la salida del display 7 segmentos, dado que son dos jugadores se usan dos displays para presentar de forma independiente los valores obtenidos por el dado.



***Imagen 15 – Diagrama esquemático salida camino LEDs***

Se presenta las salidas LEDs del juego, 32 focos que se encienden por cada jugador, indicando de forma la trayectoria y los valores de la misma, al momento de que un jugador pueda completar el camino completo es el ganador de la partida.

Para el diagrama mostrando la salida de los leds, se especifica el uso de una matriz de 64 leds, en la cual se dividió, 32 leds para cada jugador, dado de que en Altium no se encontró un dispositivo parecido, se realizó el diagrama del circuito emulando el funcionamiento con el uso de 32 leds para cada jugador.

1. **Conclusiones y Recomendaciones**

**Conclusiones**

* La presente programación, VHDL, es muy versátil; permite diseñar bloques con varias funcionalidades para posteriormente sincronizarlas utilizando una MSS. Es un funcionamiento analógico a programar distintas funciones y luego coordinarlas en un main.
* Para conseguir un orden correcto de funcionamiento de los distintos bloques, es necesario que haya un intercambio de señales habilitadoras entre cada bloque y la MSS para que estos sepan qué hacer y en qué momento hacerlo, por esto estas señales deben ser muy específicas y distintas entre sí.
* El bloque anti rebotes es necesario para evitar falsos ingresos de señales que pueden perjudicar el funcionamiento del circuito.

**Recomendaciones**

* Para fines de pruebas, es más fácil comprobar el funcionamiento de un circuito sincronizado bajo un reloj manual que con un reloj automático. Por esto es necesario incluir ambos relojes en la programación; una vez comprobado que el funcionamiento es correcto, se puede usar tranquilamente el reloj automático de forma exclusiva.
* Implementar un camino de LEDs puede ser complicado para un número de pistas mayor a 20, dado por el número de pines disponibles en la tarjeta; para números mayores es más sencillo utilizar una matriz.

1. **Bibliografía**

|  |  |
| --- | --- |
| [1] | Z. Hajduk, «An FPGA embedded microcontroller,» *ELSEVIER,* p. 1, 2013. |
| [2] | ELECTRONICAESTUDIO.COM, «Qué es un microcontrolador?,» ISA Soluciones, [En línea]. Available: https://www.electronicaestudio.com/que-es-un-microcontrolador/. |
| [3] | ZamuX Electronics, «MICROCONTROLADOR PIC 16F877A,» PrestaShop, [En línea]. Available: https://zamux.co/cb/ofertas/826-microcontrolador-16f877a.html. |
| [4] | uElectro, «Descripción del PIC16F877A,» blogger.com, 26 Agosto 2017. [En línea]. Available: http://electroboyss.blogspot.com/2017/08/es-un-microcontrolador-de-microchip.html. |
| [5] | H.-W. Huang, PIC Microcontroller: An Introduction to Software and Hardware Interfacing, Cengage Learning, 2005. |
| [6] | T. Agarwal, «PIC Microcontroller Architecture and Applications,» WordPress, 2016. [En línea]. Available: https://www.edgefx.in/pic-microcontroller-architecture-and-applications/. |
| [7] | Microchip, «PIC16F87XA Data Sheet,» Microchip Technology Inc.. |

[8] E. PALACIOS - F. REMIRO -. L. LÓPEZ, Microcontrolador PIC17F84 - Desarrollo de Proyectos, Ciudad de México : ALFAOMEGA GRUPO EDITOR , 2004.

[9] "Microcontroladores," 07 MAYO 2019. [Online]. Available: https://hetprostore.com/TUTORIALES/microcontrolador/.

[10] E. PÉREZ - L. MENÉNDEZ - L. FERNÁNDEZ - E. López, MICROCONTROLADORES PIC . Sistema integrado para el autoaprendizaje, BARCELONA , ESPAÑA: MARCOMBO, 2007.

1. **Enlaces**

Video del proyecto:

<https://youtu.be/ISxxai82UM0>

Repositorio del Proyecto

<https://github.com/assilva21/sistemasdigitalesII>