ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

LABORATORIO SISTEMAS DIGITALES II

PARALELO 103

MAQUINA JUGADORA DE LUDO

PROFESORA

RIOS ORELLANA SARA JUDITH

INTEGRANTES

PEREZ NOBOA BRYAN ALFONSO

SOLEDISPA CARRASCO ALDAIR ALONSO

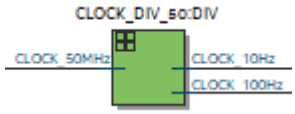
2T-2018

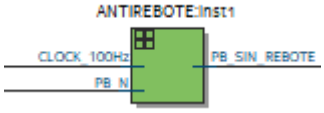
Introducción

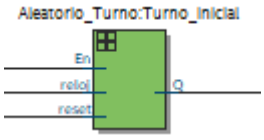
Para la realización de este proyecto se establecieron dos alternativas para poder implementarlo, la primera fue utilizar dos bloques que generen valores aleatorios el primer bloque generaría entre 0 y 1 y el segundo bloque entre 1 y 6 simulando un dado, para la segunda propuesta se estableció un solo bloque generador de aleatorio el cual serviría para seleccionar entre jugador 1 y jugador 2, además del valor del dado. Luego de juntarnos y pensar que solución sería más eficiente se decidió escoger la primera la cual tenía mejor lógica e implementación, aunque saldría más costosa.

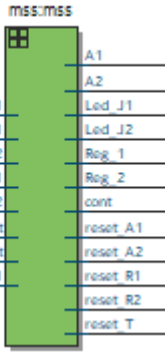
1. Capturas y descripción detallada

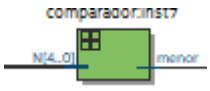
Diagrama de Bloques del Sistema Digital











Explicación:

El sistema genera un numero aleatorio entre uno y cero para validar el jugador que empieza el juego ya teniendo ente dato ese jugador debe presionar su botón y el mismo va a generar un número aleatorio entre uno y seis el cual va su registro de jugador y al tener ya su número le dará paso al siguiente jugador para que también pueda presionar su botón y obtener un número para su registro de avance, mientras vayan intercambiando los turnos y generando números al azar estos se irán sumando en sus registros de avance y el primero que llegue a treinta será el ganador que será reconocido por una señal de salida para ese jugador y con este dato se dará por terminado el juego.

2. Partición Funcional del **Sistema Digital** con su respectiva explicación a **detalle**.

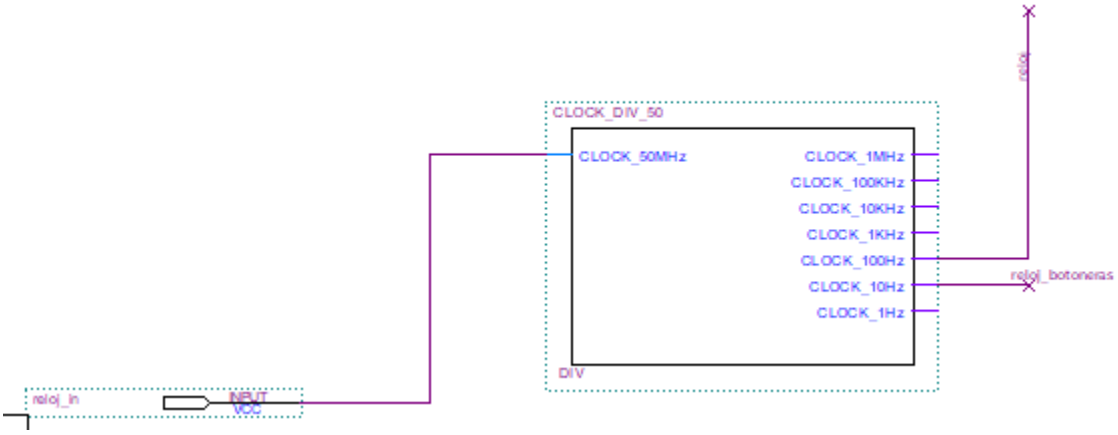


Ilustración 1: Etapa 1 de la partición funcional

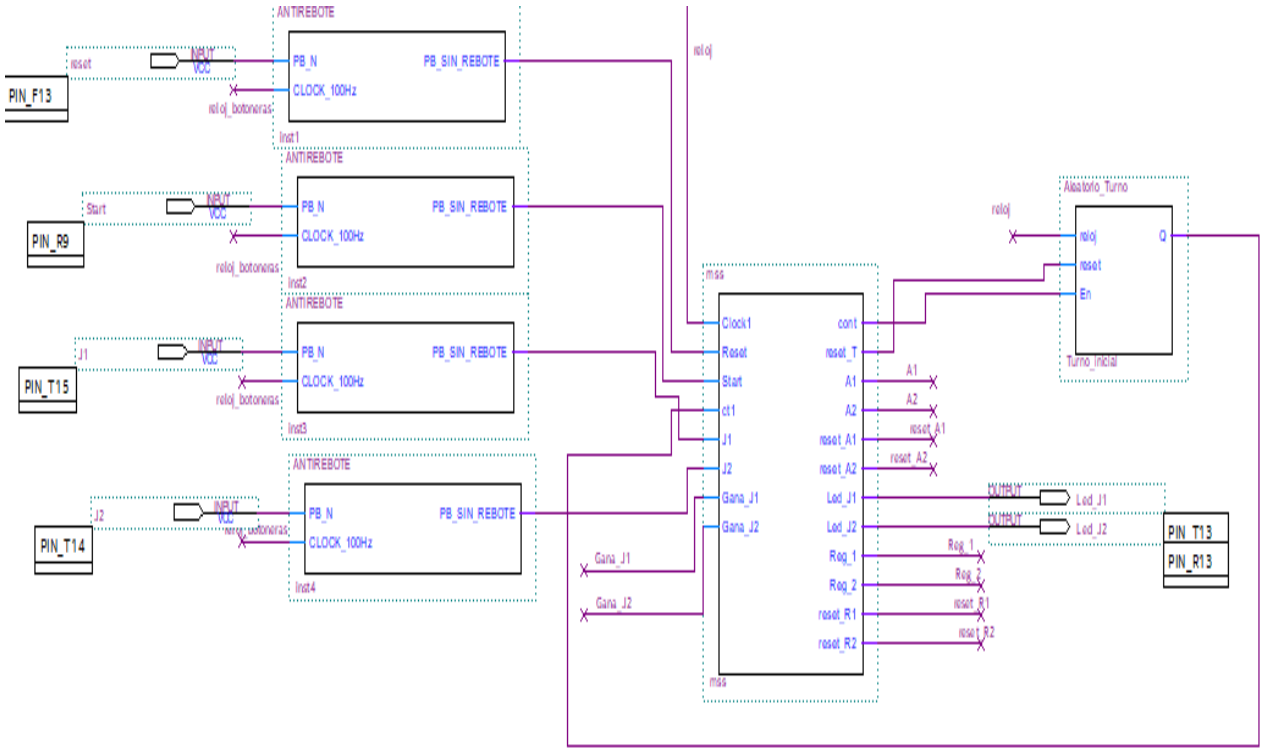


Ilustración 2: Etapa 2 de la partición funcional

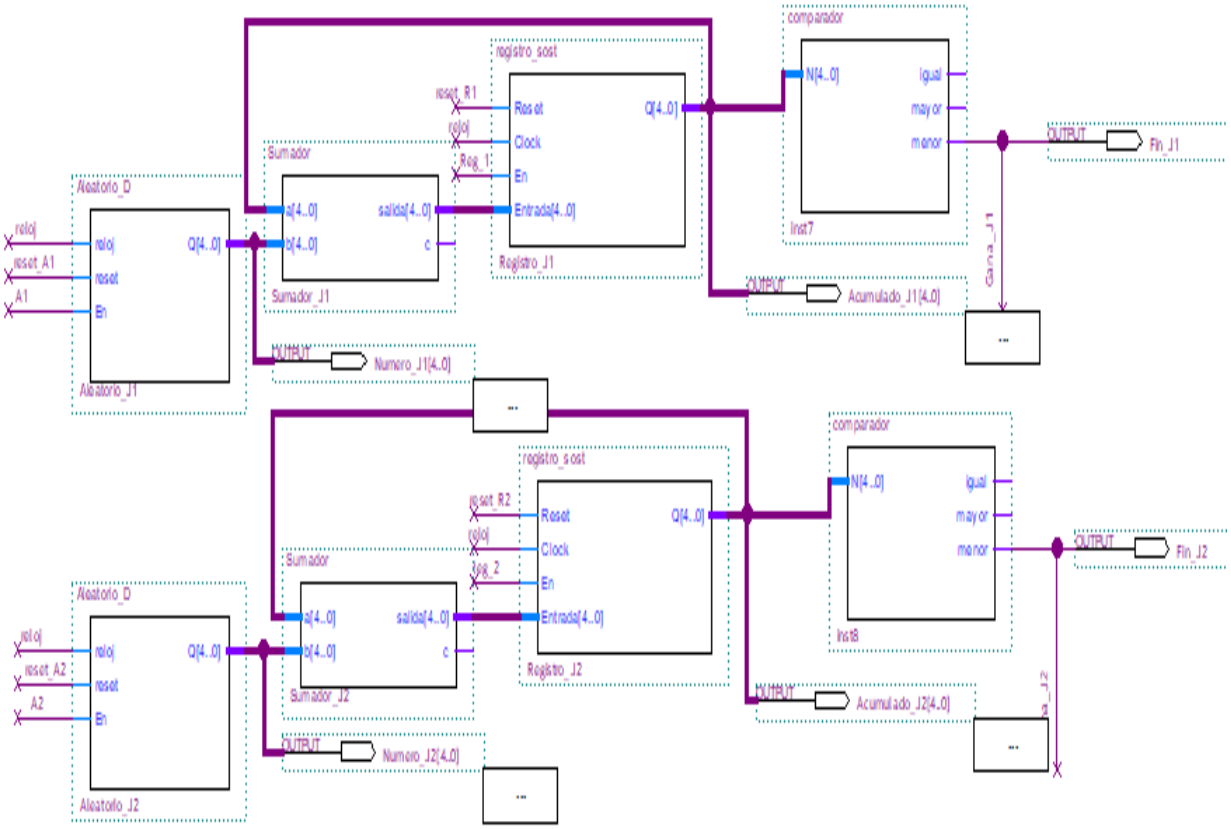


Ilustración 3: Etapa 3 de la partición funcional

Esta partición funcional consta de 3 etapas que se encuentran conectadas, la primera etapa es un bloque de reloj en la cual se generan diferentes salidas de reloj con diferentes frecuencias, para el proyecto se utilizaron 2 salidas de ese bloque las cuales fueron de 100 Hz que se usó como reloj para los demás bloques y 10 Hz que se usó para las botoneras. La segunda etapa consta de la mss la cual se encarga de controlar todo el sistema, bloques de antirebote que se usaron para tener uno solo pulso al momento de presionar una botonera y el bloque aleatorio_turno el como su nombre lo indica se usó para escoger entre jugador 1 o jugador 2. En la tercera y última etapa se encuentran los bloques que pertenecen a los jugadores, cada jugador consta con un bloque aleatorio_D el cual simuló un dado, el bloque sumador que se utilizó para sumar el nuevo valor con el del registro, además se encontraba el bloque registro_sost que se usó para almacenar los valores generados por el dado y la suma de los anteriores y por último el bloque comparador que se usó para saber si el jugador había ganado o era necesario seguir con el juego.

3. Diagrama ASM del **Controlador** en Visio o cualquier otra herramienta utilizada para elaborar diagramas de flujo, indicando claramente todas las señales de entrada y salida utilizadas (con su respectiva explicación a **detalle**).

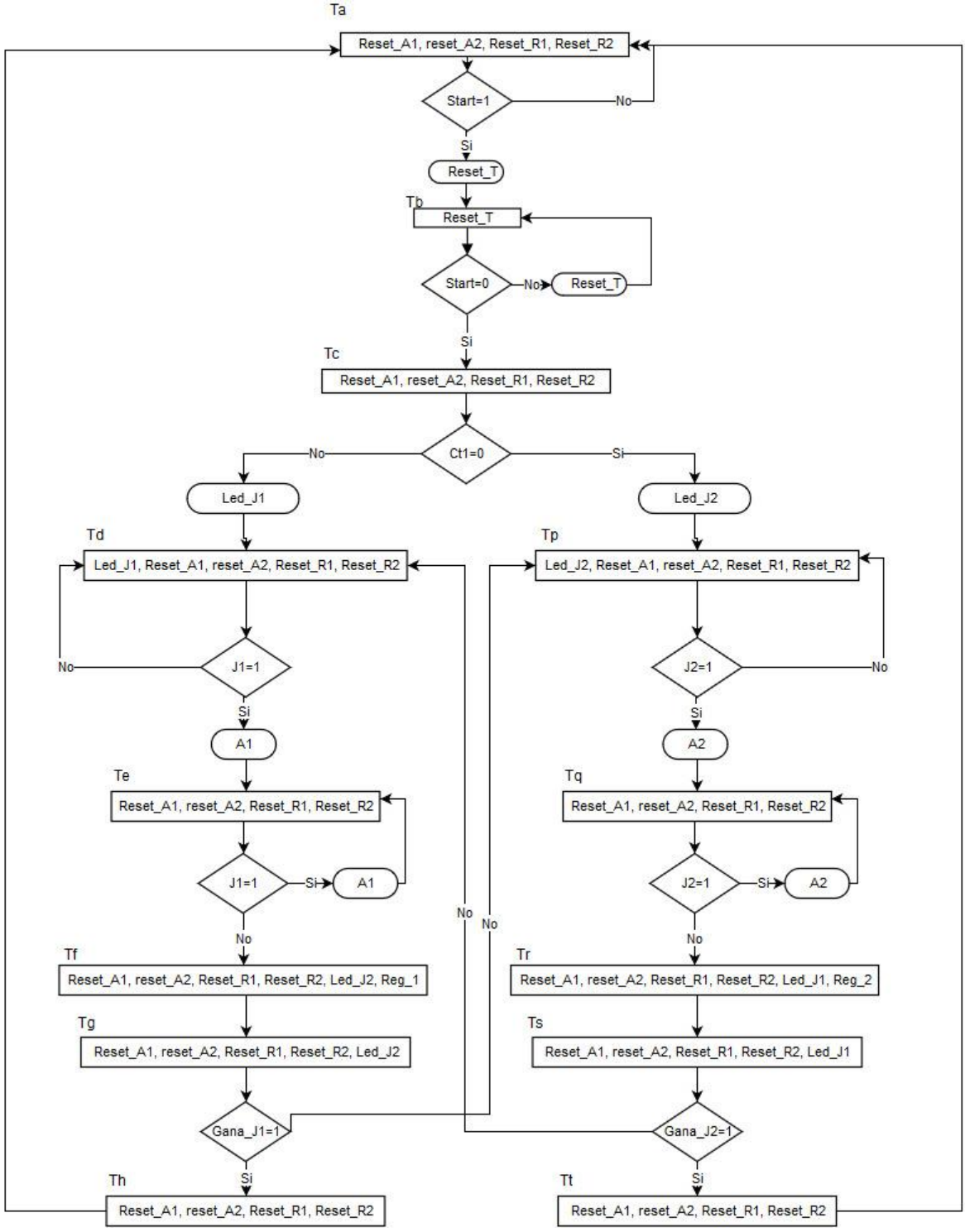


Ilustración 4: Diagrama ASM

Código VDHL controlador MSS

El código de este controlador consta de ocho entradas las cuales se dividen en Clock1, Reset, Start, ct1, ct2, J1, J2, Gana_J1, Gana_J2. Además tiene doce salidas que son cont, reset_T, A1, A2, reset_A1, reset_A2, Led_J1, Led_j2, Reg_1, Reg_2, reset_R1, reset_R2.

Este controlador tiene estados Ta, Tb, Tc, Td, Te, Tf, Tg, Th, Tp, Tq, Tr, Ts, Tt, las cuales funcionarán de la siguiente manera. En el estado Ta si el start=1 entonces pasará al estado siguiente Tb caso contrario se queda en el mismo estado. En el estado Tb si star=0 entonces pasará el siguiente estado Tc caso contrario se mantiene en el estado actual. En estado Tc si ct1=0 entonces cambiará al estado Tp caso contrario pasará al estado Td.

```
Library ieee;
use ieee.std_logic_1164.all;

Entity mss is
Port( Clock1, Reset, Start, ct1, J1, J2, Gana_J1, Gana_J2: in std_logic;
   cont, reset_T, A1, A2, reset_A1, reset_A2, Led_J1, Led_J2, Reg_1, Reg_2, reset_R1, reset_R2: out std_logic);
end mss;

Architecture arq_mss of mss is
type estado is (Ta, Tb, Tc, Td, Te, Tf, Tg, Th, Tp, Tq, Tr, Ts, Tt);
signal y : estado;
begin
   process(Reset, clock1)
   begin
      if Reset = '0' then y <= Ta;
      elsif clock1'event and clock1 = '1' then
         case y is
            when Ta =>
               if start = '1' then y<=Tb;
               else y<=Ta;
               end if;
            when Tb =>
               if start = '0' then y<=Tc;
               else y<=Tb;
               end if;
            when Tc =>
               if ct1 = '0' then y<=Tp;
               else y<=Td;
               end if;
            when Td =>
```

Ilustración 5: VDHL Mss parte 1

En el estado Td si J1=1 entonces pasará al estado Te caso contrario se queda en el estado actual. En el estado Te si J1=0 entonces pasará al estado Tf caso contrario mantendrá su propio estado actual. En el estado Tf pasará automáticamente al estado Tg, en el estado Tg cambiara a Th si Gana_J1=1 caso contrario volverá al estado Tp. En el estado Th retornará al primer estado Ta. En el estado Tp si J2=1 entonces pasa al estado Tq caso contrario se queda en el mismo estado, para el estado Tq si J2=0 entonces cambiará al estado Tr, si no es así se quedará en el mismo estado. Para el estado Tr avanzará al estado Ts automáticamente. En el estado Ts si Gana_J2=1 entonces parará al último estado Tt caso contrario cambia al estado Td.

```
    if J1 = '1' then y<=Te;
    else y<=Td;
    end if;
when Te =>
    if J1 = '0' then y<=Tf;
    else y<=Te;
    end if;
when Tf =>
    y<=Tg;
when Tg =>
    if Gana_J1 = '1' then y<=Th;
    else y<=Tp;
    end if;
when Th =>
    y<=Ta;
when Tp =>
    if J2 = '1' then y<=Tq;
    else y<=Tp;
    end if;
when Tq =>
    if J2 = '0' then y<=Tr;
    else y<=Tq;
    end if;
when Tr =>
    y<=Ts;
when Ts =>
    if Gana_J2 = '1' then y<=Tt;
    else y<=Td;
    end if;
```

Ilustración 6: VDHL Mss parte 2

En este último estado pasará al estado inicial Ta.

Para las salidas del controlador se considera lo siguiente, si se encuentra en Ta entonces cont será igual a 1 y reset_T, A1, A2, reset_R1, reset_R2 serán iguales a 0 si start=1, caso contrario entonces setet_A1, reset_A2 y reset_T será iguales a 0. En el estado Tb por defecto reset_R1 y reset_R2 serán iguales a 1 y si start=0 entonces cont=0, caso contrario cont=1. Para el estado Tc por defecto reset_R1 y reset_R2 serán iguales a 1 y reset_T=0. En el estado Td si J1=1 entonces reset_A1 y A1 serán iguales a 1 caso contrario reset_A1 será igual a 0 y por defecto los valores de reset_R1, reset_R2 serán iguales a 1, para Reg_1, Reg_2 y reset_T serán iguales a 0. Para el estado Te por defecto los valores de reset_R1, reset_R2 estarán en 1 y los Reg_1, Reg_2 serán iguales a 0, ahora si J1 es igual 0 entonces A1 es igual 0, caso contrario A1 será igual a 1. En los estados Tf yTg por defecto reset_R1, reset_R2 serán iguales a 1, Reg_1 y Reg_2 serán iguales a 0. Para el estado Th por defecto reset_R1, reset_R2 serán iguales a 1, reset_A1 y reset_A2 serán iguales a 0. En el estado Tp por defecto los valores Led_J2, reset_R1 y reset_R2 serán iguales a 1 y los de reset_T, Reg_1, Reg_2 Y Led_J1 serán iguales a cero y si J2 es igual a 1 entonces reset_A2 y A2 serán iguales a 1 caso contrario reset_A2 será igual a 0. Para el estado Tq por defecto los valores reset_R1 y reset_R2 serán iguales a 1, los de Reg_1 y Reg_2 serán iguales a 0 y si J2 es igual a 0 entonces A2 será igual a 0 caso contrario A2 será igual a 1. En el estado Tr por defecto los valores reset_R1, reset_R2 y Reg_2 serán iguales a 1 y Reg_1 será igual a 0. Para el estado Tt por defecto los valores reset_R1 y reset_R2 serán iguales a 1, los de Reg_1 y Reg_2 serán iguales a 0.

```
            when Tt =>
               y<=Ta;
         end case;
      end if;
   end process;
   process (y)
   begin
      case y is
         when Ta =>
            if start = '1' then Cont<='1';reset_T<='0';
            reset_T<='1'; A1<='0'; A2<='0'; reset_R1<='0'; reset_R2<='0';
            else reset_A1<='0'; reset_A2<='0';reset_T<='0';
            end if;
         when Tb =>
            reset_R1<='1'; reset_R2<='1';
            if start = '0' then Cont<='0';
            else Cont<='1';
            end if;
         when Tc =>
            reset_R1<='1'; reset_R2<='1';
            reset_T<='0';
         when Td =>
            reset_T<='0';
            reset_R1<='1'; reset_R2<='1';Reg_1<='0';Reg_2<='0';
            Led_J1<='1';
            Led_J2<='0';
            if J1 = '1' then reset_A1<='1'; A1<='1';
            else reset_A1<='0';
            end if;
```

Ilustración 7: VDHL Mss parte 3

```
            when Te =>
               reset_R1<='1'; reset_R2<='1';Reg_1<='0';Reg_2<='0';
               if J1 = '0' then A1<='0';
               else A1<='1';
               end if;
            when Tf =>
               reset_R1<='1'; reset_R2<='1';
               Reg_1<='1';Reg_2<='0';
            when Tg =>
               reset_R1<='1'; reset_R2<='1';
               Reg_1<='0';Reg_2<='0';
            when Th =>
               reset_R1<='1'; reset_R2<='1';
               reset_A1<='0'; reset_A2<='0';
            when Tp =>
               reset_T<='0';
               reset_R1<='1'; reset_R2<='1';
               Reg_1<='0';Reg_2<='0';
               Led_J1<='0';
               Led_J2<='1';
               if J2 = '1' then reset_A2<='1'; A2<='1';
               else reset_A2<='0';
               end if;
            when Tq =>
               reset_R1<='1'; reset_R2<='1';
               Reg_1<='0';Reg_2<='0';
               if J2 = '0' then A2<='0';
               else A2<='1';
               end if;
            when Tr =>
```

Ilustración 8: VDHL Mss parte 4

```
            reset_R1<='1'; reset_R2<='1';
            Reg_1<='0';Reg_2<='1';
        when Ts =>
            reset_R1<='1'; reset_R2<='1';
            Reg_1<='0';Reg_2<='0';
        when Tt =>
            reset_R1<='1'; reset_R2<='1';
            reset_A1<='0'; reset_A2<='0';
    end case;
  end process;
end arq_mss;
```

Ilustración 9: VDHL Mss parte 5

Diagrama de tiempo del sistema digital

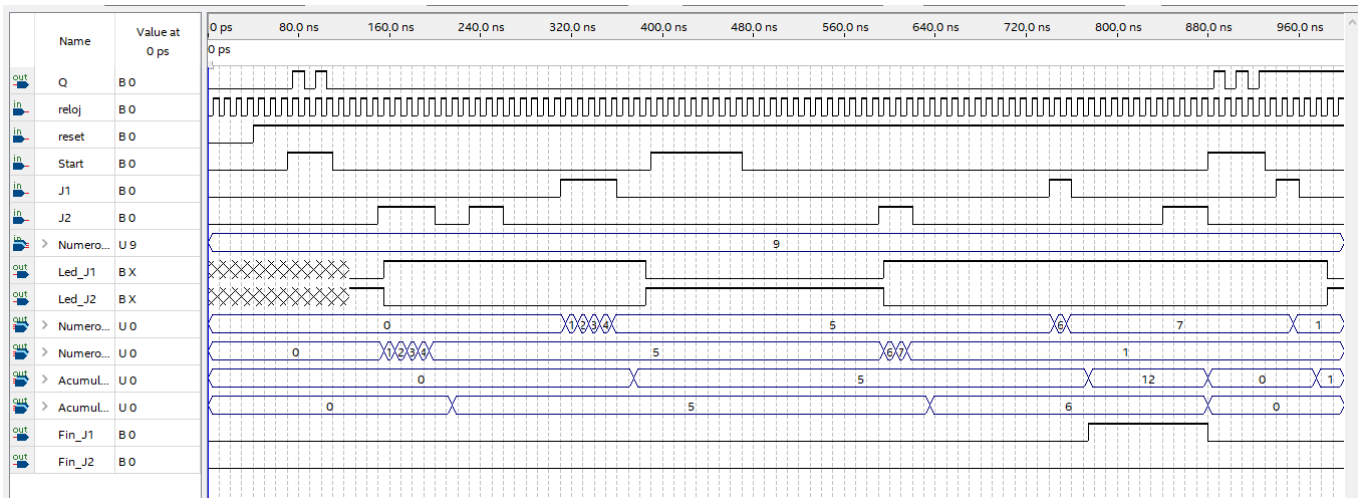


Ilustración 10: Diagrama de tiempo del sistema

Una vez que se presiona el botón start empieza a generarse un número aleatorio entre 0 y 1 hasta que se deje de presionar y ahí ya se tiene el turno del jugador que va a empezar representado por la señal Led_J1 y Led_J2. El que vaya a iniciar deberá presionar su botón propio de cada jugador que es J1 y J2 el cual solo servirá si le toca su turno caso contrario hará caso omiso de que se mantiene presionado y mientras este se encuentre presionado se va a generar un contador que cambiara ascendentemente entre 1 y 6 en forma de bucle hasta que se deje de presionar lo que va a generar un número aleatorio el cual se presentará en un display y la señal de salida del mismo es Numero_J1 y Numero_J2 estos valores se irán a un acumulador personal de cada jugador que registrará la acumulación de cada número aleatorio creado hasta que este sobrepase el valor que se establece como máximo para ganar (en el caso de esta simulación 9) y una vez que esto pase se encenderá un led llamado Fin_J1 o Fin_J2 en caso de ganar el segundo jugador.

Código VHDL por bloque

Aleatorio_Turno

Este bloque consta con cuatro entradas y una salida, las entradas son reloj captura las señales, reset permitirá resetear los valores del bloque, "En" esta señal es la habilitadora que proviene del controlador y dato_ent no se utiliza se encuentra estacionado en tierra, la salida Q es el valor que permitirá escoger si es el jugador uno o jugador dos.

```
LIBRARY IEEE;
USE  IEEE.STD_LOGIC_1164.all;
USE  IEEE.STD_LOGIC_UNSIGNED.all;

ENTITY Aleatorio_Turno IS
   PORT(reloj, reset, En : IN STD_LOGIC;
        dato_ent : IN STD_LOGIC;
        Q : OUT STD_LOGIC);
END Aleatorio_Turno;

ARCHITECTURE sol OF Aleatorio_Turno IS
SIGNAL conteo: STD_LOGIC;
BEGIN
   PROCESS(reloj,En,reset)
   BEGIN
      if reset='0' then Q<='0';
      elsif (reloj'event and reloj='1') then
         if En='1' then
            if conteo='0' then conteo<='1';
            else conteo<='0';
            end if;
         end if;
      end if;
      Q<=conteo;
   END PROCESS;
END sol;
```

Ilustración 11: VDHL del bloque Aleatorio_Turno

Comparador

Este bloque permitió comparar los valores que tienen cada uno de los jugadores en sus registros, se usó el de menor e el igual para comparar si es mayor a 27 o no, siendo el caso de que no, pasará al siguiente jugador, caso contrario ganará el jugador del estado actual.

```
LIBRARY  ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all ;

ENTITY  comparador  IS
PORT  (  N:  IN  STD_LOGIC_VECTOR(4  DOWNTO  0);
         igual,mayor,menor:  OUT  STD_LOGIC);
END  comparador;

ARCHITECTURE  solucion  OF  comparador  IS
signal A:std_logic_vector(4 downto 0);
BEGIN
   A<="11010";
   igual  <=  '1'    WHEN  A  =  N ELSE  '0';
   mayor  <=  '1'  WHEN  A  >  N  ELSE  '0';
   menor  <='1'  WHEN  A  <  N  ELSE'0';
END  solucion;
```

Ilustración 12: VDHL del bloque comparador

Aleatorio_D

Este bloque permite escoger valores entre 1 y 6 aleatoriamente, simulando el lanzamiento de un dado. Servirá para presentar un valor para el jugador del estado actual.

```
LIBRARY IEEE;
USE  IEEE.STD_LOGIC_1164.all;
USE  IEEE.STD_LOGIC_UNSIGNED.all;

ENTITY Aleatorio_D IS
   PORT(reloj, reset, En : IN STD_LOGIC;
        Q : OUT STD_LOGIC_VECTOR (4 DOWNTO 0));
END Aleatorio_D;

ARCHITECTURE sol OF Aleatorio_D IS
SIGNAL conteo: STD_LOGIC_VECTOR (4 DOWNTO 0);
signal aux: STD_LOGIC_VECTOR (0 DOWNTO 0);
BEGIN
   PROCESS(reloj,En,reset)
   BEGIN
      if reset='0' then Q<="00000";
      elsif (reloj'event and reloj='1') then
         if En='1' then
            if aux="1" then
               conteo<=conteo + '1';
               if conteo ="00110" then conteo<="00001";
               end if;
            end if;
            aux<=aux + '1';
         end if;
      end if;
      Q<=conteo;
   END PROCESS;
END sol;
```

Ilustración 13: VDHL del bloque Aleatorio_D

Registro_sost

Este bloque permitirá almacenar cada uno de los valores que son ingresados del bloque de aletorio_D.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity registro_sost is
   Port ( Reset, Clock : in std_logic;
          En : in std_logic;
          Entrada : in std_logic_vector(4 downto 0);
          Q : out std_logic_vector(4 downto 0));
end registro_sost;

Architecture sol of registro_sost is
Begin
   process (Reset, Clock)
   begin
      if Reset ='0' then Q <= "00000";
      elsif (Clock'event and Clock='1') then
         if En = '1' then
            Q <= Entrada;
         end if;
      end if;
   end process;
end sol;
```

Ilustración 14: VDHL del bloque Registro_sost

Sumador

Este bloque permitió sumar los valores que fueron generados del bloque aletorio_D y el valor actual del bloque de registro_sost.

```
library  IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_signed.all;
use  IEEE.NUMERIC_STD.all;

ENTITY  Sumador  IS
PORT  (a  :  IN  std_logic_vector(4  DOWNTO  0);
   b  :  IN  std_logic_vector(4  DOWNTO  0);
   salida  :  OUT  std_logic_vector(4  DOWNTO  0);
   c:  OUT  std_logic);
END  Sumador;

ARCHITECTURE  synth  OF  Sumador  IS
signal  suma:  std_logic_vector(4  DOWNTO  0);
BEGIN
PROCESS  (a,  b)  IS
BEGIN
suma  <=  a+b;
   salida<=  suma(4  DOWNTO  0);
   c<=  suma(4);
END  PROCESS;
END  synth;
```

Ilustración 15: VDHL del bloque Sumador

Antirebote

Este bloque permitió que no se generen una gran cantidad de valores al momento de usar las botoneras, sino que solo se genere un valor cuando se deja de presionar la botonera

```
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;
```

Ilustración 16: VDHL del bloque Antirebote

CLOCK_DIV_50

Este bloque se usó para generar valor de reloj a diferentes frecuencias, para el caso de nuestro proyecto se usaron los relojes de 100 Hz para los diferentes bloques y el de 10 Hz para las botoneras.

```
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
```

Ilustración 17: VDHL del bloque Clock_Div_50 parte 1

```
        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";
```

Ilustración 18: VDHL del bloque Clock_Div_50 parte 2

```
            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
```

Ilustración 19: VDHL del bloque Clock_Div_50 parte 3

```
            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;
```

Ilustración 20: VDHL del bloque Clock_Div_50 parte 4

Diagrama de tiempo del controlador

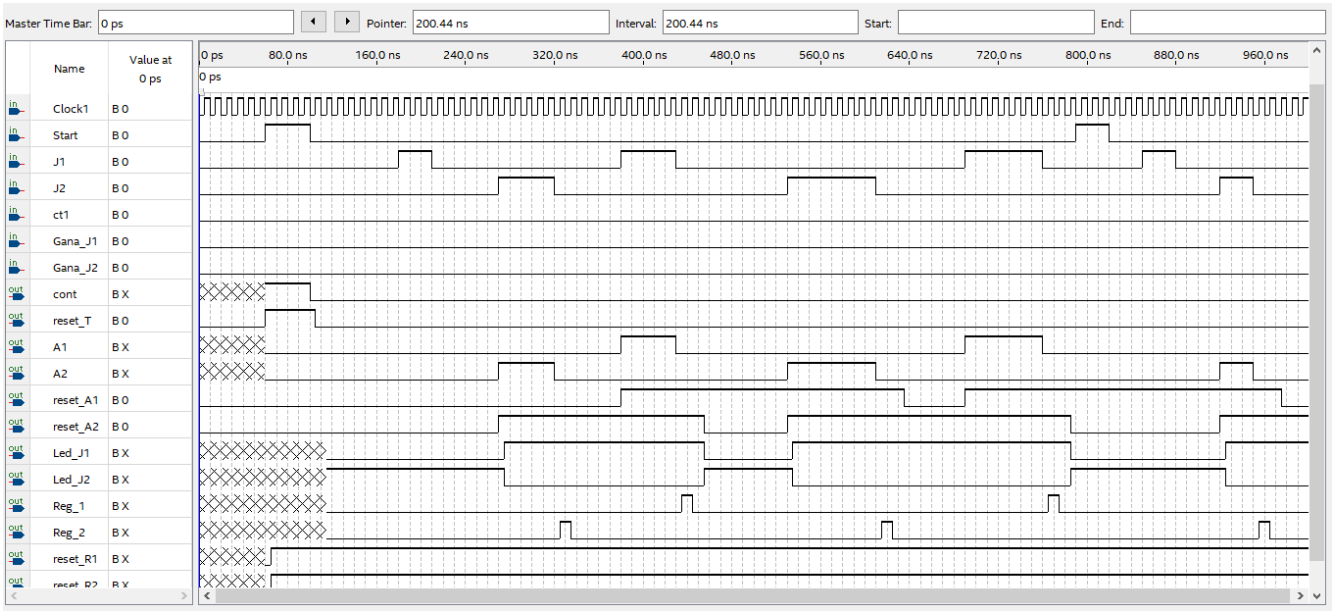


Ilustración 21: Diagrama de tiempo del controlador

En el controlador realiza las habilitaciones de acuerdo a como vaya cambiando de estado el mismo, al presionar start resetea cada una de las señales y genera la habilitación para el numero aleatorio mediante la señal reset_T el cual envía con 0 ó 1 el turno del jugador 1 ó del jugador 2 que es recibido mediante la señal ct1 que encenderá los led del turno mostrados con las señales Led_J1 y led_J2, y al presionar el botón del jugador que le toco mediante J1 ó J2 y habilita la señal para que se genere su número aleatorio mediante la señal A1 ó A2 y a su vez resetea el número que se encontraba en el display de su turno anterior mediante la señal reset_A1 ó reset_A2, ya con esto pasa a otro estado en donde habilita en registro del jugador en curso para que se acumule el resultado generado en el proceso anterior mediante las señales Reg_1 y Reg_2.

Diagrama esquemático Altium

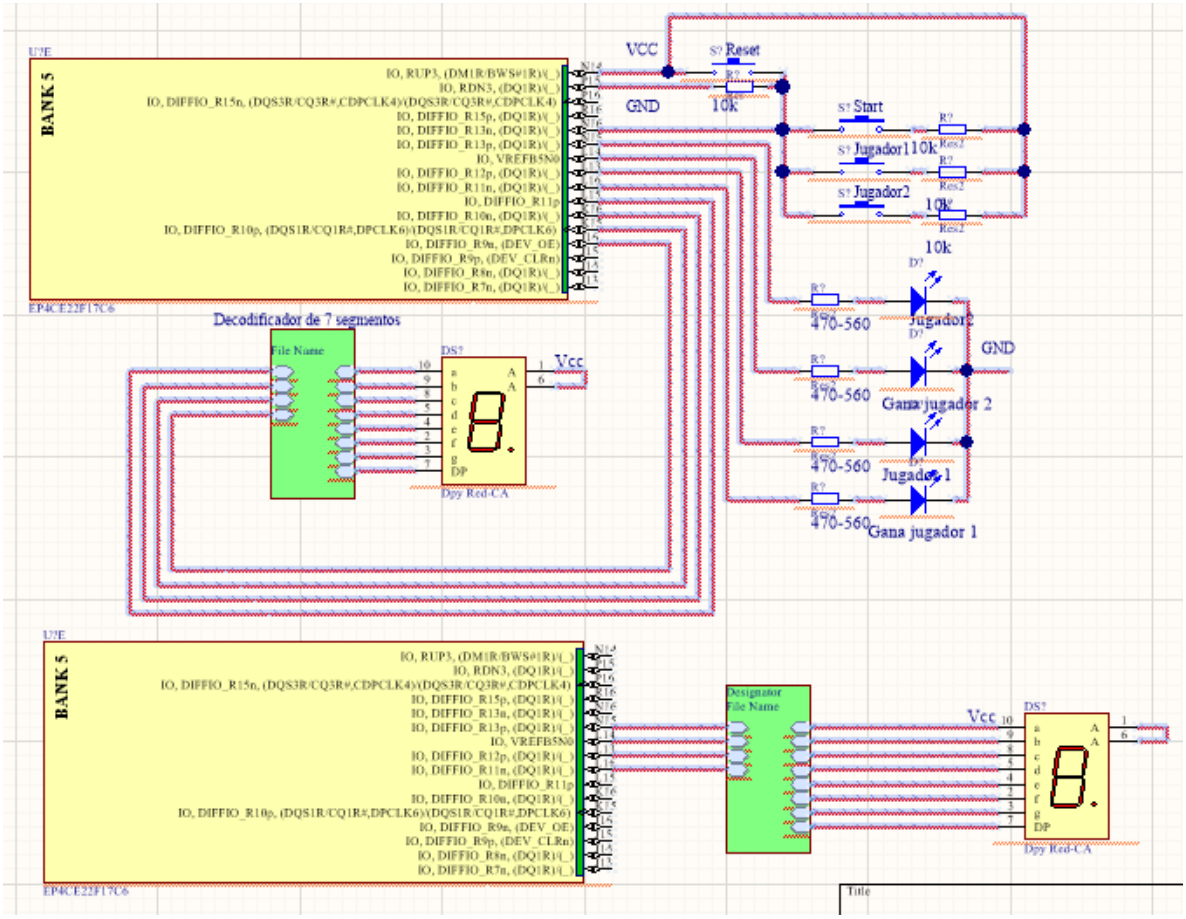


Ilustración 22: Esquemático Altium parte 1

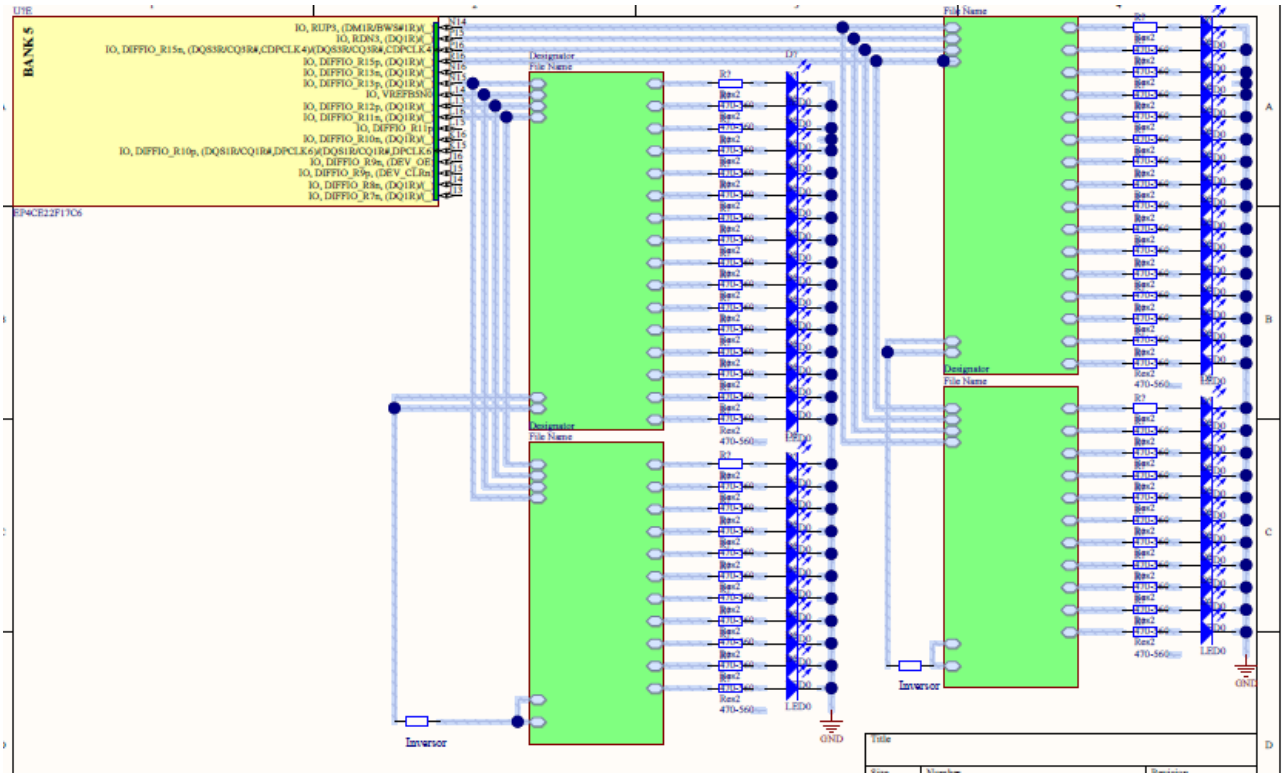


Ilustración 23: Esquemático Altium parte 2

En este diagrama esquemático se lo tuvo que dividir en dos partes debido a su contenido extenso. La primera parte consta de 2 bloques que tienen el nombre Bank 5 representan la fpga que fue instalada con una librería de altium, el primer bloque tiene conectada 4 botoneras las cuales representan el reset, start, jugador 1 y jugador 2, además de eso consta con los leds de los jugadores y los leds que representan el ganador y a su vez tiene un decodificador de 7 segmentos en el cual ingresan los valores del número a representar en el display, el segundo Bank 5 solo tiene conectado un decodificador de 7 segmentos en el cual ingresan los valores del número a representar en el display. En la segunda parte del diagrama se puede observar un solo bloque llamado Bank 5 el cual estará conectado los valor acumulados los cuales entran a u decodificador de 4 a 16 conocido como 74LS154 el cual se encuentra conectado en paralelo para poder conectar 30 leds con sus resistencias respectivas los cuales representaran los valores que tengan en el momento cada uno de los jugadores.

De la fpga se usaron en total 28 pines.

2. Link de github, donde consten todos los archivos Visio, VHD, BDF y VWF que genera el programa Quartus.

https://github.com/AldairSoledispa/Maquina-Jugadora-de-Ludo-Paralelo103

Conclusiones

- Podemos concluir que el circuito simulado no se comporta de la misma forma que el que ya se implementa con la fpga debido a que en la parte real hay variaciones por la frecuencia a la que trabajan los circuitos y la inestabilidad al presionar una botonera.

- Se logró observar que los leds que representaban los valores que tenían los jugadores se encontraban invertidos debido al uso del decodificador 74LS154 de 4 a 16 usado en paralelo con otro decodificador del mismo tipo, esto hizo que se mostrarán encendidos todos los leds y que avance apagando un led según el valor que se haya mostrado en el display de 7 segmentos.

- Podemos concluir que para el diseño real de un circuito se debe tener en cuenta el retardo de las señales al pasar por los distintos elementos del circuito por lo que el controlador debe implementar estados vacíos de ser necesarios con el objetivo de nivelar las señales y se obtenga el resultado esperado.

Recomendaciones

- Evitar el uso del decodificador 74LS154 debido a que este tipo de decodificador hace que se enciendan todos los leds de registro de los jugadores invirtiendo el resultado esperado.

- Si se desea un mejor resultado en las respuestas de tiempo de los leds de los jugadores al pasar al siguiente estado se puede hacer uso de un clock manual el cual permitirá que se realicen los cambios de estado de forma directa y sin retraso.

- Para poder evitar que se generen valores aleatorios o innecesarios al momento mantener presionada una botonera se debe usar un bloque antirebote seguido de la botonera y a su vez conectarla al controlador del sistema digital.