ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

SISTEMAS DIGITALES II

REPORTE FINAL DEL PROYECTO: "MÁQUINA DETECTORA DE REFLEJOS HUMANOS"

Integrantes:
Campoverde Rodríguez Ernesto Celín
Saquicela Loja Rommel Patricio

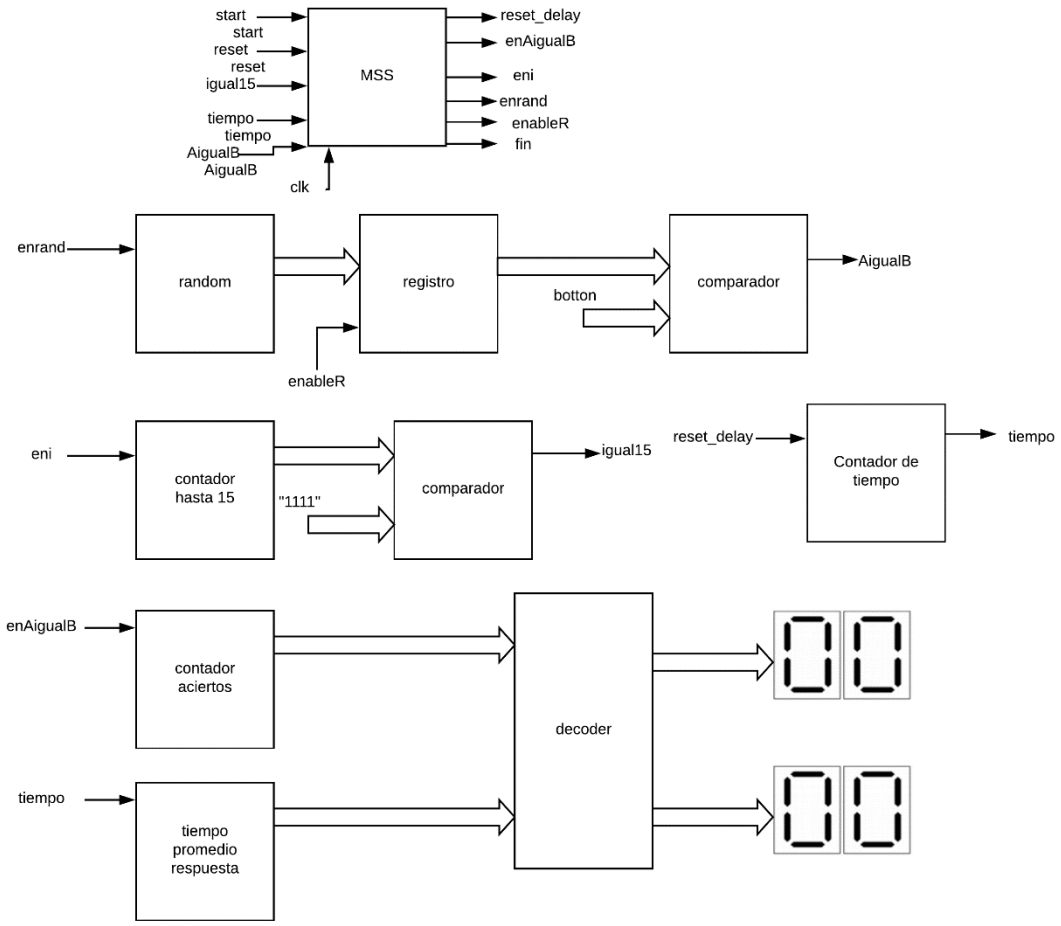
Nombre de la profesora: Ing. Sara Ríos

Paralelo de Laboratorio: 101

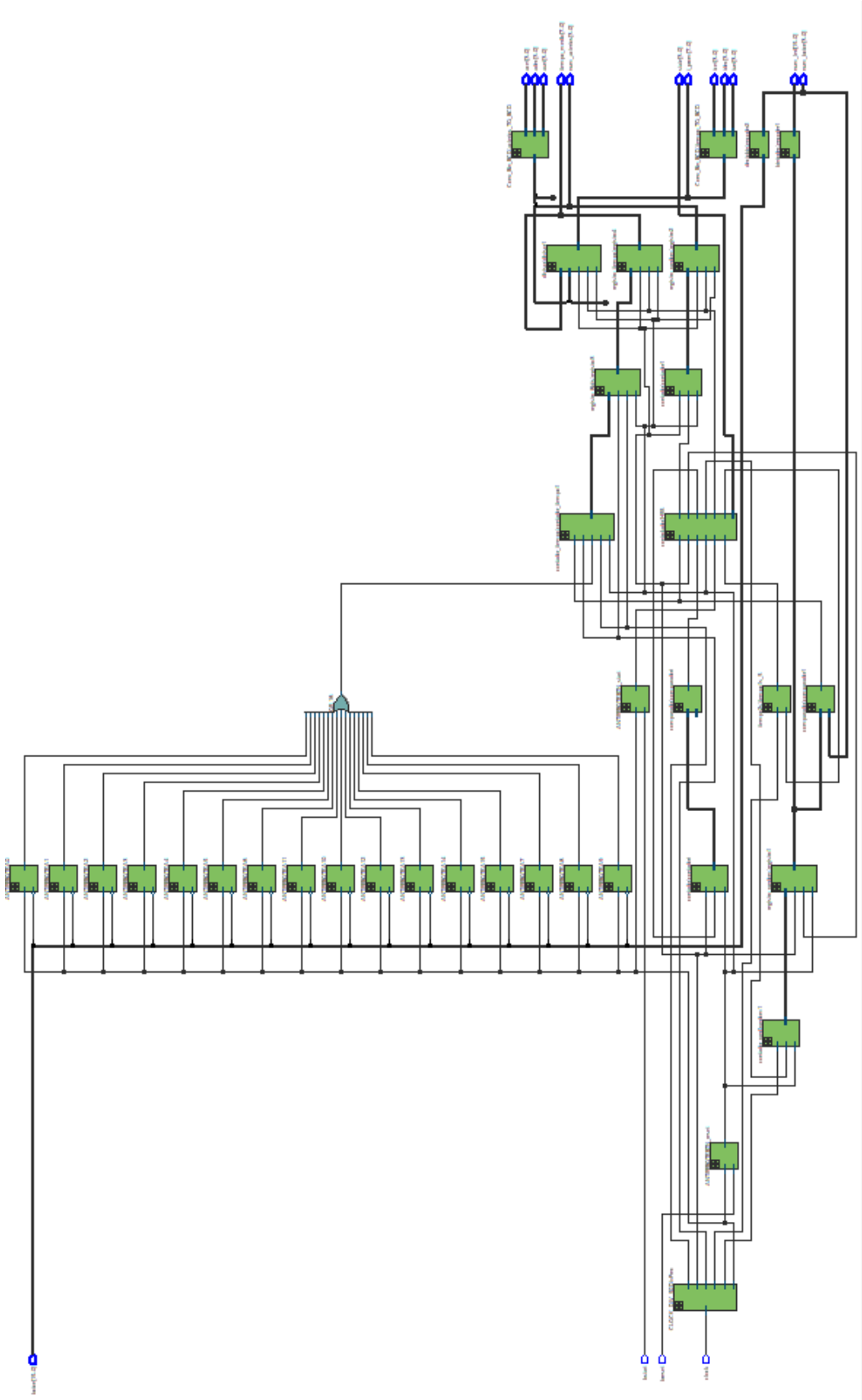
Introducción

En el presente documento se presenta el diseño e implementación de una máquina de reflejos humanos, la cual consiste en un juego de aciertos que además mide el tiempo de respuesta del jugador, estos resultados se muestran en Displays. Antes de llegar al diseño final se expuso dos posibles soluciones de diseño, estas se asemejaban en gran parte, la mayor diferencia entra las dos era el número de estados del controlador (MSS), por lo que se optó por aquella con menor estados debido a que era más compacto. Antes de implementarlo en físico se mejoró el diseño hasta llegar al presentado, los cambios además de presentar mejores resultados también redujeron costos ya que se aprovecha de mejor manera la FPGA.

Diagrama de Bloques del Sistema Digital



Partición Funcional del Sistema Digital



El Diagrama de Bloques del Sistema Digital total engloba varios componentes y bloques que interactúan entre sí procesando datos binarios con el objetivo de generar las salidas más externas: num_led, num_aciertos, a_dec, a_uni t_dec y t_uni; y la salida auxiliar para visualizar el estado actual denominada state. En el extremo del diagrama se encuentra un bloque divisor de frecuencia, el cual descompone una frecuencia en varias "subfrecuencias"; una de ellas, la de 1kHz es usada como reloj del sistema (para la MSS y la mayoría del resto de bloques).

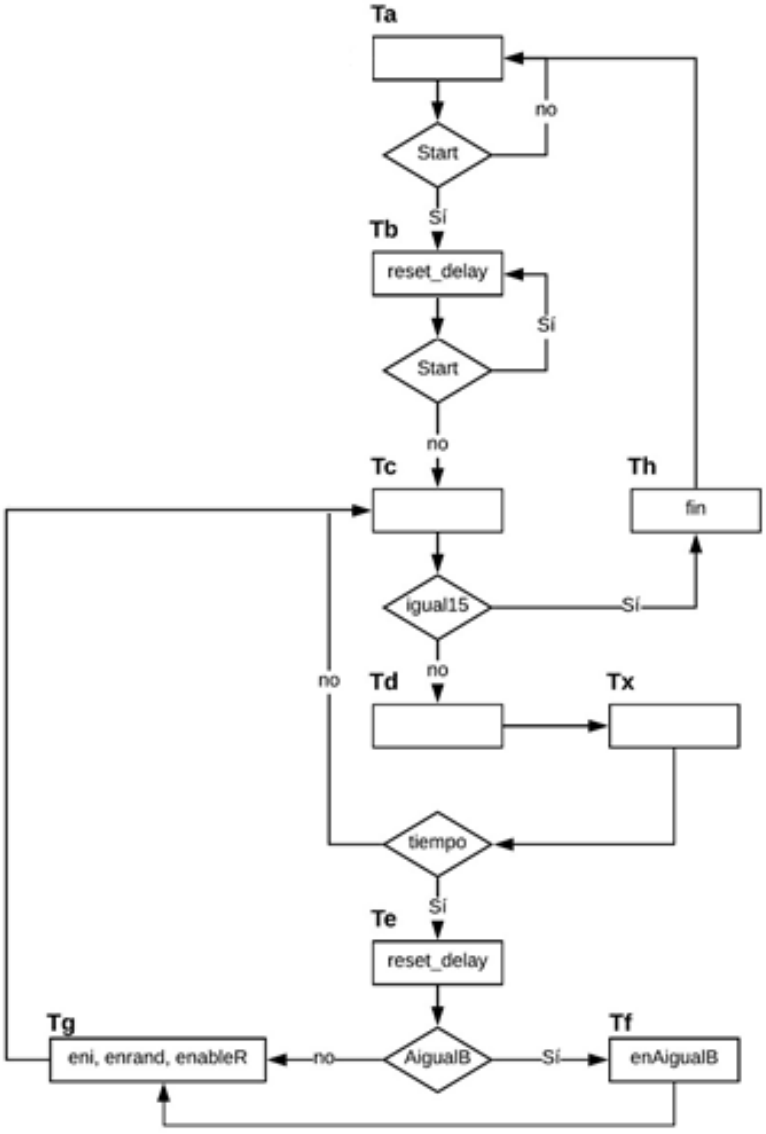
Como entradas del sistema están las señales emitidas por las botoneras reset y start; las cuales, en conjunto con la señal de sincronismo, permiten que el controlador inicie la transición entre estados. Adicionalmente, existen 15 entradas más que son tomadas en cuenta a partir de un estado determinado; dichas entradas son las de los botones que corresponden a los 15 LEDs que se encenderán aleatoriamente, permitiendo obtener la posibilidad de acertar. Cada uno de los botones está conectado a un bloque de antirebote para así evitar falsos niveles lógicos originados por fluctuaciones o ruidos externos.

Mientras que el botón reset permanezca presionado, el estado actual será Ta. Caso contrario, se evalúa el ciclo de reloj, y luego si start fue presionado y no se ha soltado, el estado actual será Tb, estado en donde se resetea un contador usado para establecer el tiempo de encendido de los LEDs. Una vez que el botón start ha sido soltado, la MSS pasa al estado Tc, que es en donde se valida si se han encendido 15 LEDs. Ya que esto no es cierto (en primera instancia), se pasa al estado Td; y luego al estado Tx. En dicho estado se valida si han transcurrido 2 segundos (se usa el bloque tiempo2s), y de no ser así, ocurre una transición a Te. Estando en Te, se efectúa la validación de si el número aleatorio guardado en su registro correspondiente es idéntico al número de LED presionado; si esto llegase a ocurrir, se pasa al estado Tf y una señal (enAigualB) parte desde el controlador hacia el contador de aciertos, que incrementará la cuenta en una unidad cada que coincida el número de LED con el valor generado de forma aleatoria. En el siguiente flanco de subida, la MSS pasa al estado Tg; este estado es de suma importancia porque es aquí donde se habilita el contador de número de LEDs a prenderse, además de que también se habilita el contador (de alta frecuencia) que generará el número aleatorio, así como el respectivo registro (registro_random) en donde se alojará dicho valor. Luego, la MSS retorna al estado Tc, y vuelve a validar si la señal proveniente del bloque comparador comparadori (bloque que compara el conteo de LEDs con el número fijo "1111", 15 en binario) es un alto. El

proceso que comprende entre el estado Tc y Tg se repite 15 veces a manera de bucle, y se acumula el conteo de aciertos si el jugador logra presionar el botón correspondiente al LED prendido aleatoriamente. Otro punto crucial es, que apenas inicia una ronda del juego de reflejos, y el usuario presiona cualquier botón, el bloque contador tiempo genera el tiempo de retardo por cada LED y lo va acumulando, para que luego dicho dato sea pasado a registros, y posteriormente dividido.

Finalmente, algunas señales son pasadas a un decodificador, por ejemplo, las unidades y decenas de aciertos.

Diagrama ASM



Código VHDL completo de la MSS

El controlador del Sistema Digital completo posee 6 señales de entrada y 5 señales de salida: *clock*, *reset*, *start*, *AigualB*, *tiempo*, *igual15* y *enableR*, *enReg1*, *enAIgualB*, *eni*, *fin*, respectivamente. La entrada clock provee sincronismo, y dicha entrada es el reloj del sistema. Por otro lado, la entrada reset permite que el sistema pase al estado inicial Ta, estado en donde se aguarda que se presione el botón start.

Una vez que se ha presionado el botón start, ocurre la transición al segundo estado Tb en el siguiente flanco de reloj de subida. La señal tiempo es otra señal de reloj (periódica) que está configurada para establecer el tiempo de encendido de los LEDs. La señal igual15 indica cuando se ha completado el lazo que permite encender 15 veces un LED de forma aleatoria, y cuando dicha señal es un alto se procede al estado Th, estado en donde se calcula el promedio de tiempo de retardo correspondiente a los aciertos.

La MSS tiene 8 estados: Ta, Tb, Tc, Td, Tx, Te, Tf, Tg y Th. Si el reset es un bajo, se procede al primer estado. Luego, si se presiona y se suelta el botón start, ocurre una transición al estado Tb. A continuación, en el estado Tc se pregunta por la señal igual15, y si es falsa, se pasa al estado Td. Después de este estado se colocó un estado auxiliar para validar el número de veces que se van a encender los LED. En el quinto estado, si la señal tiempo es un alto, se va al estado Te, y en este estado se manda la señal del contador de 2 segundos (reset_delay). En este punto se valida si el botón presionado fue el correcto, y de ser así, ocurre una transición al estado Tf. Si no, la MSS se salta el estado Tf y va al estado Tg, para en el siguiente ciclo de reloj retornar al estado Tc. En Tf se habilita al contador de aciertos. Más adelante, se pasa al estado Tg, y es aquí donde se habilita al contador de LEDs con la señal eni, se habilita al bloque aleatorio que genera el número de LED a encenderse con la señal enableR, y se habilita al registro en donde será almacenado dicho número de LED con la señal enRand. Luego se retorna al estado Tc, estado en donde se vuelve a validar si ya se han encendido 15 LEDs. De ser el caso, se pasa al último estado Th, y es en este estado donde se calcula el promedio mencionado anteriormente con el uso de la señal habilitadora fin.

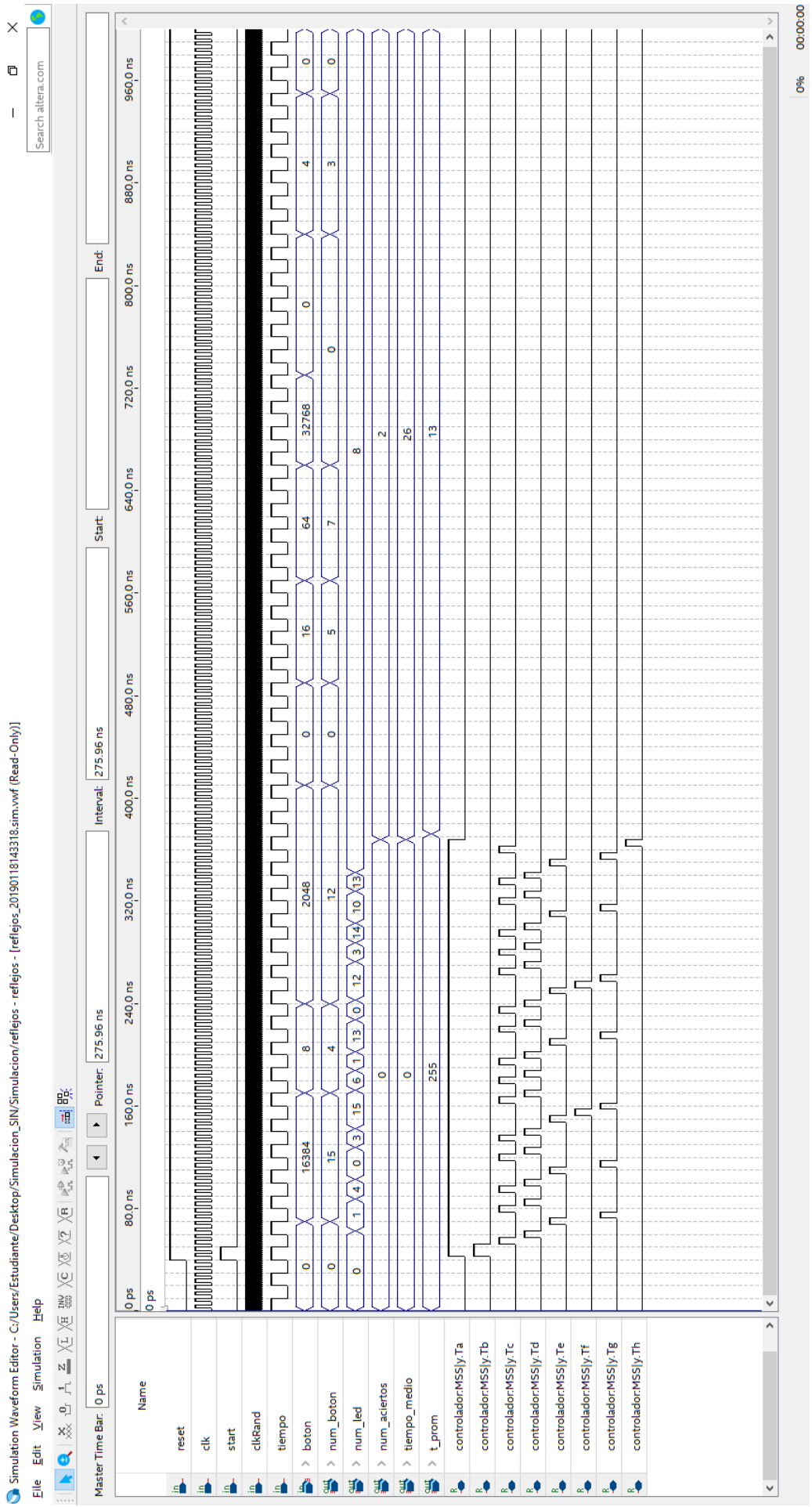
```
LIBRARY IEEE;
USE  IEEE.STD_LOGIC_1164.all;
USE  IEEE.STD_LOGIC_UNSIGNED.all;

ENTITY controlador IS
   PORT(clock,reset,start,AigualB,tiempo,igual15: IN STD_LOGIC;
        state: out std_logic_vector(3 downto 0);
       enableR,enReg1,enAigualB,eni,enrand,fin,reset_delay,reset_reg: OUT STD_LOGIC);
END controlador;

ARCHITECTURE sol OF controlador IS
Type estado is (Ta,Tb,Tc,Td,Te,Tf,Tg,Th,Tx);
SIGNAL y:estado;
BEGIN
   PROCESS(clock,reset)
   BEGIN
      if reset='0' then y<=Ta;
      elsif (clock'event and clock='1') then
         case y is
            when Ta=>   if start='1' then y<=Tb;
                        else y<=Ta;
                        end if;
            when Tb=>   if start='1' then y<=Tb;
                        else y<=Tc;
                        end if;
            when Tc=>   if igual15='1' then y<=Th;
                        else y<=Td;
                        end if;
            when Td=>   y<=Tx;
            when Tx=>   if tiempo='1' then y<=Te;
                        else y<=Tc; end if;
            when Te=>   if AigualB='1'  then y<=Tf;
                        else y<=Tg; end if;
            when Tf=>   y<=Tg;
            when Tg=>   y<=Tc;
            when Th=>   y<=Ta;
         end case;
      end if;
   END PROCESS;

PROCESS(y)
   BEGIN
      enableR<='0';enReg1<='0';enAigualB<='0';eni<='0';fin<='0';reset_delay<='1';enrand<='0';reset_reg<='0';
         case y is
            when Ta=>   state<="0001";
            when Tb=>   state<="0010"; reset_delay<='0';
            when Tc=>   state<="0011";
            when Td=>   state<="0100";
            when Tx=>   state<="0101";
            when Te=>   state<="0110";  reset_delay<='0';
            when Tf=>   enAigualB<='1'; state<="0000";
            when Tg=>   state<="1000"; eni<='1';enrand<='1';enableR<='1';
            when Th=>   fin<='1'; state<="1001";
         end case;
   END PROCESS;
END sol;
```

Diagrama de tiempo del controlador generado en VWF



En el archivo VWF generado se puede analizar que la MSS no opera hasta que la señal reset (de lógica negativa) sea un alto. Cuando la señal reset es un alto, la MSS cae en el estado inicial, esperando a que se presione el botón start. Una vez que la señal start es un alto, ocurre una latencia breve y acto seguido la MSS pasa al estado Tb. Mientras que se mantenga presionado el botón, la MSS se mantendrá en el estado Tb, si no es el caso, pasará al estado Tc. En el estado Tc se pregunta por si el número de veces que se ha encendido un LED es igual a 15, como no ocurre en primera instancia ocurre una transición al estado Td. Es en el estado Td donde se habilita el conteo de LEDs que se van encendiendo, así como se habilita el bloque generador del número aleatorio correspondiente al número de LED a prenderse y su respectivo registro de almacenamiento. Luego ocurre una transición al estado Tx, y en dicho estado, si la señal es un alto, se pasa al estado Te.

Estando en el estado Te, se valida si el botón presionado es el correcto, y si esta condición se cumple, ocurre una transición al estado Tf (aquí se habilita el contador de aciertos con la señal enAigualB). De no cumplirse la condición mencionada, es decir, si el botón presionado no es el correcto, se pasa al estado Tg, saltándose el estado Tf. En el diagrama de tiempo se puede apreciar que ocurren 3 ciertos. Del estado Tf se pasa al estado Tg, y luego se regresa al estado Tc. En el estado Tc, se vuelve a validar si la cuenta de LEDs es igual a 15, y todo el ciclo del estado Tc hasta el estado Tf se repite hasta que se cumpla esta condición. Finalmente, todo derivará en el estado Th, que es en donde se calcula el promedio de retardo de tiempo de aciertos.

Código VHDL completo de los bloques a usar

- Bloque contador

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

entity contador is
    PORT (
        clk,enable,reset    : IN  STD_LOGIC;
        salida: OUT STD_LOGIC_VECTOR(3 DOWNTO 0)
    );
end contador;

architecture sol of contador is
    signal cnt_tmp: STD_LOGIC_VECTOR(3 DOWNTO 0) := "0000";
begin
   process(clk,reset)
   begin
      if reset='0' then salida <= "0000";
      elsif(clk'event and clk='1' and enable ='1') then
            cnt_tmp <= cnt_tmp + 1;
      end if;
      salida <= cnt_tmp;
    end process;
end sol;
```

El bloque contador admite las entradas de reloj, reset, y una habilitadora. De manera genérica, si la señal reset es un alto el contador se resetea. Además, solo se realiza un incremento del contador si la señal habilitadora es un alto y si se captura un flanco de reloj de subida.

- Bloque comparador

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

entity comparador is
   Port ( A,B: in STD_LOGIC_VECTOR(3 downto 0);
          enable1: in STD_LOGIC;
      salida : OUT STD_LOGIC);
end comparador;

Architecture sol of comparador is
Begin
      salida <='1' when (A=B)and(enable1='1') else '0';

end sol;
```

Este bloque se encarga de realizar comparaciones entre números binarios; la salida será un alto si los números mencionados son iguales, en caso contrario la salida será un 0 lógico, todo lo anterior bajo la premisa de que la entrada habilitadora sea igual a un alto.

- Bloque convertidor decimal – binario

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

entity dectobin is
   Port ( Ent: in STD_LOGIC_VECTOR(15 downto 0);
      Salida : OUT STD_LOGIC_VECTOR(3 downto 0)
      );
end dectobin;

Architecture sol of dectobin is
Begin
   With Ent Select
      Salida <=   "0001" when "0000000000000001",
                  "0010" when "0000000000000010",
                  "0011" when "0000000000000100",
                  "0100" when "0000000000001000",
                  "0101" when "0000000000010000",
                  "0110" when "0000000000100000",
                  "0111" when "0000000001000000",
                  "1000" when "0000000010000000",
                  "1001" when "0000000100000000",
                  "1010" when "0000001000000000",
                  "1011" when "0000010000000000",
                  "1100" when "0000100000000000",
                  "1101" when "0001000000000000",
                  "1110" when "0010000000000000",
                  "1111" when "0100000000000000",
                  "0000" when others;
end sol;
```

El bloque se encarga de asignarle el correspondiente número binario al número decimal ingresado como entrada.

- Bloque convertidor binario - decimal

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

entity bintodec is
   Port ( Ent: in STD_LOGIC_VECTOR(3 downto 0);
      Salida : OUT STD_LOGIC_VECTOR(15 downto 0)
      );
end bintodec;

Architecture sol of bintodec is
Begin
   With Ent Select
      Salida <=   "0000000000000001" when "0001",
                  "0000000000000010" when "0010",
                  "0000000000000100" when "0011",
                  "0000000000001000" when "0100",
                  "0000000000010000" when "0101",
                  "0000000000100000" when "0110",
                  "0000000001000000" when "0111",
                  "0000000010000000" when "1000",
                  "0000000100000000" when "1001",
                  "0000001000000000" when "1010",
                  "0000010000000000" when "1011",
                  "0000100000000000" when "1100",
                  "0001000000000000" when "1101",
                  "0010000000000000" when "1110",
                  "0100000000000000" when "1111",
                  "0000000000000000" when others;
end sol;
```

El bloque realiza la tarea de convertir un número binario a su correspondiente en base 10 tomando en consideración la entrada.

- Bloque contador aleatorio

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

entity contador_rand is
    PORT (
        clk,enable,reset    : IN  STD_LOGIC;
        salida: OUT STD_LOGIC_VECTOR(3 DOWNTO 0)
    );
end contador_rand;

architecture sol of contador_rand is
    signal cnt_tmp: STD_LOGIC_VECTOR(3 DOWNTO 0) := "0000";
begin
   process(clk,reset)
   begin
      if reset='0' then salida <= "0000"; cnt_tmp <= "0000";
      elsif(clk'event and clk='1') then
            cnt_tmp <= cnt_tmp + 1;
      end if;
      if enable ='1' then
      salida <= cnt_tmp;
    end if;
    end process;
end sol;
```

Dicho bloque genera un número aleatorio entre 0 y 15 (con la cantidad de elementos por vector fijados en este caso, que son 4 bits, es editable), número que representará el número de LED a encenderse. También tiene una señal de reloj como entrada.

- Bloque registro aleatorio

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

ENTITY registro_random IS
   PORT(clock,reset,enable, enR: IN STD_LOGIC;
        Ent : IN STD_LOGIC_VECTOR(3 downto 0);
        Q : OUT STD_LOGIC_VECTOR (3 downto 0));
END registro_random;

ARCHITECTURE sol OF registro_random IS
SIGNAL temp: STD_LOGIC_VECTOR(3 downto 0):= "0000";
BEGIN
   PROCESS(clock,reset)
   BEGIN
      if reset='0' then temp <= "0000"; --Q <= "0000";
      elsif (clock'event and clock='1') then
         if(enable='1') then
            temp<=Ent;
         end if;
         if (enR='1') then
            temp<=Ent + "0101";
         end if;
      end if;
   end process;
   Q<=temp;
END sol;
```

El bloque registro_random almacena el número de LED generado previamente con el bloque contador_rand. El valor pasa al vector Q de salida solo cuando la señal habilitadora es un alto, esto es enable = ‘1’, y mientras se cumpla que se captura un flanco de reloj de subida.

- Bloque registro aleatorio 1

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

ENTITY registro_random1 IS
   PORT(clock,reset,enable, enR: IN STD_LOGIC;
        Ent : IN STD_LOGIC_VECTOR(3 downto 0);
        Q : OUT STD_LOGIC_VECTOR (3 downto 0));
END registro_random1;

ARCHITECTURE sol OF registro_random1 IS
SIGNAL temp: STD_LOGIC_VECTOR(3 downto 0):= "0000";
BEGIN
   PROCESS(clock,reset)
   BEGIN
      if reset='0' then temp <= "0000"; --Q <= "0000";
      elsif (clock'event and clock='1') then
         if(enable='1') then
            temp<=Ent + "0100";
         end if;
         if (enR='1') then
            temp<=Ent + "0101";
         end if;
      end if;
   end process;
   Q<=temp;
END sol;
```

El bloque registro_random1 almacena el número de LED generado previamente con el bloque contador_rand. A diferencia del registro_random, funciona con dos habilitadoras. El valor pasa al vector Q de salida solo cuando la señal habilitadora es un alto, esto es enable = ‘1’, y mientras se cumpla que se captura un flanco de reloj de subida. Se suman valores a la variable temporal para forzar a que los LEDs cambien el comportamiento cíclico presentado por la multiplicidad entre las frecuencias del bloque divisor, además de que se condiciona dicho valor según el valor de las entradas habilitadores para evitar conflictos por las frecuencias.

- Bloque contador tiempo

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

entity contador_tiempo is
    PORT (
        clk,enable1,enable2,AigualB,reset    : IN  STD_LOGIC;
        salida: OUT STD_LOGIC_VECTOR(7 DOWNTO 0)
    );
end contador_tiempo;

architecture sol of contador_tiempo is
    signal cnt_tmp: STD_LOGIC_VECTOR(7 DOWNTO 0) := "00000000";
begin
   process(clk,reset)
   begin
      if (reset='0') then salida <= "00000000";
      elsif(clk'event and clk='1' and enable1 ='1' and enable2 ='1' and AigualB='1') then
            cnt_tmp <= cnt_tmp + 1;
      end if;
      salida <= cnt_tmp;
    end process;
end sol;
```

Este bloque realiza la operación de calcular el retardo de tiempo que le toma al usuario presionar un botón una vez que se ha encendido un LED, todo esto bajo la premisa de que oprimió el botón correspondiente al número de LED que se prendió. La entrada habilitadora enable1 proviene del resultado de una puerta OR de 16 entradas que están conectadas a cada botonera; si se presiona cualquier botón enable1 será un alto. La segunda entrada habilitadora proviene de otra señal de reloj, la señal tiempo. Si dichas entradas habilitadoras son ‘1’, AigualB es igual a ‘1’ (número de botón y de LED iguales), y el flanco de subida de reloj de la señal clkRand, el contador se incrementará.

- Bloque registro de 8 bits

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

ENTITY registro_8bits IS
   PORT(clock,reset,enable: IN STD_LOGIC;
        Ent : IN STD_LOGIC_VECTOR(7 downto 0);
        Q : OUT STD_LOGIC_VECTOR (7 downto 0));
END registro_8bits;

ARCHITECTURE sol OF registro_8bits IS
SIGNAL temp: STD_LOGIC_VECTOR(7 downto 0):= "00000000";
BEGIN
   PROCESS(clock,reset)
   BEGIN
      if reset='0' then
      elsif (clock'event and clock='1') then
         if(enable='0') then
            temp<=Ent;
         end if;
      end if;
   end process;
   Q<=temp;
END sol;
```

El bloque de registro de 8 bits permite el almacenamiento de los 8 bits provenientes del bloque contador_tiempo únicamente cuando la señal habilitadora es un bajo; la señal enable tendrá como entrada a la seña de reloj tiempo, señal que delimita el tiempo de prendida de un LED.

- Bloque registro tiempo

```
LIBRARY IEEE;
USE  IEEE.STD_LOGIC_1164.all;
USE  IEEE.STD_LOGIC_UNSIGNED.all;
use ieee.numeric_std.all;

ENTITY registro_tiempo IS
   PORT(clock,reset,enable: IN STD_LOGIC;
        Ent : IN STD_LOGIC_VECTOR(7 downto 0);
        Q : OUT STD_LOGIC_VECTOR (7 downto 0));
END registro_tiempo;

ARCHITECTURE sol OF registro_tiempo IS

SIGNAL temp: STD_LOGIC_VECTOR(7 downto 0):= "00000000";

BEGIN
   PROCESS(clock,reset)
   BEGIN
      if reset='0' then
      elsif (clock'event and clock='1') then
         if(enable='1') then
            temp<=Ent;
         end if;
      end if;
   end process;
   Q<=temp;
END sol;
```

El bloque de registro_tiempo permite el almacenamiento del tiempo de retardo total, y tiene como entrada habilitadora la señal Fin, señal que indica que la MSS ha llegado al último estado Th.

- Bloque divisor

```
LIBRARY IEEE;
USE  IEEE.STD_LOGIC_1164.all;
USE  IEEE.STD_LOGIC_UNSIGNED.all;
use ieee.numeric_std.all;

ENTITY divisor IS
   PORT(clock,reset,enable: IN STD_LOGIC;
        A,B : IN STD_LOGIC_VECTOR(7 downto 0);
        I: out STD_LOGIC_VECTOR (7 downto 0));
END divisor;

ARCHITECTURE sol OF divisor IS

SIGNAL temp: STD_LOGIC_VECTOR(7 downto 0):= "00000000";
signal int,int2,div: integer range 0 to 255;

BEGIN
   PROCESS(clock,reset)
   BEGIN
      if reset='0' then int <= 0; int2 <= 0; div <=0; temp <= "00000000";
      elsif (clock'event and clock='1') then
               int <= to_integer(unsigned(A));
               int2 <= to_integer(unsigned(B));
      end if;
      div <= int / int2;
      I <= std_logic_vector(to_unsigned(div,8));

   end process;


END sol;
```

El bloque divisor lleva a cabo la operación de división una vez que la MSS ha llegado al último estado. Tiene una señal de entrada de reloj (clock del sistema), una entrada de reset, los vectores a dividir, y el vector de salida. Se usan señales intermedias para convertir las entradas a números enteros y poder usar el operador '/'. Finalmente, la salida se la vuelva a convertir a std_logic_vector.

- Bloque Divisor de frecuencia

```
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_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;
```

```
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;
```

El bloque divisor de frecuencia tiene como entrada la señal de sincronismo provista por la FPGA DE0-Nano, y a partir de la misma origina varias señales de reloj con frecuencias menores. Para ello es necesario efectuar varias operaciones, como lo indica el código VHDL. Dichas ‘subseñales’ son usadas en varios bloques que componen el Sistema Digital.

- Bloque componentes

```
library ieee;
use ieee.std_logic_1164.all;

package componentes is

   component divisor is
      port (clock,reset,enable: IN STD_LOGIC;
       A,B : IN STD_LOGIC_VECTOR(7 downto 0);
       I: out STD_LOGIC_VECTOR (7 downto 0));
   end component;

   component dectobin is
      port(Ent: in std_logic_vector(15 downto 0);
      Salida: out std_logic_vector(3 downto 0));
   end component;

   component bintodec is
      port(Ent: in std_logic_vector(3 downto 0);
      Salida: out std_logic_vector(15 downto 0));
   end component;

   component comparador is
   Port ( A,B: in STD_LOGIC_VECTOR(3 downto 0);
      enable1: in STD_LOGIC;
      salida : OUT STD_LOGIC);
   end component;

   component contador is
   Port ( clk,enable,reset    : IN  STD_LOGIC;
        salida: OUT STD_LOGIC_VECTOR(3 DOWNTO 0));
   end component;

   component contador_tiempo is
   Port ( clk,enable1,enable2,AigualB,reset    : IN  STD_LOGIC;
        salida: OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
   end component;

   component contador_rand is
   Port ( clk,enable,reset    : IN  STD_LOGIC;
        salida: OUT STD_LOGIC_VECTOR(3 DOWNTO 0));
   end component;
   component registro_random is
   PORT(clock,reset,enable, enR: IN STD_LOGIC;
       Ent : IN STD_LOGIC_VECTOR(3 downto 0);
       Q : OUT STD_LOGIC_VECTOR (3 downto 0));
   end component;

   component registro_random1 is
   PORT(clock,reset,enable, enR: IN STD_LOGIC;
       Ent : IN STD_LOGIC_VECTOR(3 downto 0);
       Q : OUT STD_LOGIC_VECTOR (3 downto 0));
   end component;

   component registro_8bits is
   PORT(clock,reset,enable: IN STD_LOGIC;
       Ent : IN STD_LOGIC_VECTOR(7 downto 0);
       Q : OUT STD_LOGIC_VECTOR (7 downto 0));
   end component;

   component registro_tiempo is
   PORT(clock,reset,enable: IN STD_LOGIC;
       Ent : IN STD_LOGIC_VECTOR(7 downto 0);
       Q : OUT STD_LOGIC_VECTOR (7 downto 0));
   end component;

   component decoder_a_7segmentos is
   Port( w: in std_logic_vector(3 downto 0);
         en: in std_logic;
         Q1: out std_logic_vector(6 downto 0));
   end component;

   component Conv_Bin_BCD is
    Port ( Bin : in   STD_LOGIC_VECTOR (7 downto 0);
           Cen : out  STD_LOGIC_VECTOR (3 downto 0);
           Dec : out  STD_LOGIC_VECTOR (3 downto 0);
           Uni : out  STD_LOGIC_VECTOR (3 downto 0));
   end component;

component ANTIREBOTE IS
   PORT(PB_N, CLOCK_100Hz  : IN  STD_LOGIC;
       PB_SIN_REBOTE    : OUT STD_LOGIC);
   end component;

   component 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_1KHz      :OUT  STD_LOGIC;
        CLOCK_100Hz     :OUT  STD_LOGIC;
        CLOCK_10Hz      :OUT  STD_LOGIC;
        CLOCK_1Hz       :OUT  STD_LOGIC);
    end component;

    component tiempo2s is
    PORT (
         clk2,reset     : IN  STD_LOGIC;
         salida: OUT STD_LOGIC);
     end component;

    component controlador is
    port(clock,reset,start,AigualB,tiempo,igual15: IN STD_LOGIC;
          state: out std_logic_vector(3 downto 0);
        enableR,enReg1,enAigualB,eni,enrand,fin,reset_delay,reset_reg : OUT STD_LOGIC);
    end component;
end componentes;
```

En el bloque denominado componentes se declaran todos los bloques que constituyen el Sistema Digital total, con sus respectivas señales internas y externas, siguiendo la estructura de empaquetamiento convencional.

- Bloque empaquetado

```
library ieee;
use ieee.std_logic_1164.all;
use work.componentes.all;


entity empaquetado is
   port(boton: in std_logic_vector(15 downto 0);
        clock,breset,bstart: in std_logic;
        num_aciertos: buffer std_logic_vector(3 downto 0);
        num_led: out std_logic_vector(15 downto 0);
        state: out std_logic_vector(3 downto 0);
        tiempo_medio: buffer std_logic_vector(7 downto 0);
        t_prom: buffer STD_LOGIC_VECTOR (7 downto 0);
        num_boton,tuni,tdec,tcet,auni,adec,acet: buffer std_logic_vector(3 downto 0));
end empaquetado;

architecture solucion of empaquetado is
signal aciertos,num_random,num_reg1,Qi: std_logic_vector(3 downto 0);
signal reset,start,clkRand,clk,tiempo,menor9,mayor9,OR_16,AigualB,enAigualB,enReg1,eni,igual15,enableR,fin: std_logic;
signal tiempo_reaccion,cnt_tiempo,A,B: std_logic_vector(7 downto 0);

signal clk100,clk10k,clk100k,clk10,dosseg,reset_delay,enrand,reset_reg: std_logic;
signal botton: std_logic_vector(15 downto 0);

begin
   random1: contador_rand port map(clk10k,enrand,reset,num_random);
   registro1: registro_random1 port map(clk,reset,enableR,OR_16,num_random,num_reg1);
   encoder1: bintodec port map(num_reg1,num_led);
   encoder2: dectobin port map(boton,num_boton);
   contadori: contador port map(clk,eni,reset,Qi);
   comparadori: comparador port map(Qi,"1111",'1',igual15);
   comparador1: comparador port map(num_reg1,num_boton,OR_16,AigualB);
   contador1: contador port map(clk,enAigualB,reset,aciertos);
   MSS: controlador port map(clk,reset,start,AigualB,dosseg,igual15,state,enableR,enReg1,enAigualB,eni,enrand,fin,reset_delay,reset_reg);
   registro2: registro_random port map(clk,reset,fin,'0',aciertos,num_aciertos);

   tiempo_TO_BCD: Conv_Bin_BCD port map(t_prom,tcet,tdec,tuni);
   --decoderDtuni: decoder_a_7segmentos port map(tuni,'1',Dtuni);
   --decoderDtdec: decoder_a_7segmentos port map(tdec,'1',Dtdec);
   --decoderDtcet: decoder_a_7segmentos port map(tcet,'1',Dtcet);

   aciertos_TO_BCD: Conv_Bin_BCD port map(B,acet,adec,auni);
   --decoderDauni: decoder_a_7segmentos port map(auni,'1',Dauni);
   --decoderDddec: decoder_a_7segmentos port map(adec,'1',Dadec);

   tiempo1s_S: tiempo2s port map(clk10,reset_delay,dosseg);

   OR_16 <= botton(15) OR botton(14) OR botton(13) OR botton(12) OR botton(11) OR botton(10) OR botton(9) OR botton(8) OR botton(7) OR botton(6) OR bot
   contador_tiempo1: contador_tiempo port map(clkRand,OR_16,dosseg,AigualB,reset,cnt_tiempo);
   registro3: registro_8bits port map(clkRand,reset,tiempo,cnt_tiempo,tiempo_reaccion);
```

```
    registro4: registro_tiempo port map(clk,reset,fin,tiempo_reaccion,tiempo_medio);
    A <= tiempo_medio;
    B <= "0000"&num_aciertos;
    divisor1: divisor port map(clk,reset,fin,A,B,t_prom);


    A0: ANTIREBOTE port map(not boton(0),clk100,botton(0));
    A1: ANTIREBOTE port map(not boton(1),clk100,botton(1));
    A2: ANTIREBOTE port map(not boton(2),clk100,botton(2));
    A3: ANTIREBOTE port map(not boton(3),clk100,botton(3));
    A4: ANTIREBOTE port map(not boton(4),clk100,botton(4));
    A5: ANTIREBOTE port map(not boton(5),clk100,botton(5));
    A6: ANTIREBOTE port map(not boton(6),clk100,botton(6));
    A7: ANTIREBOTE port map(not boton(7),clk100,botton(7));
    A8: ANTIREBOTE port map(not boton(8),clk100,botton(8));
    A9: ANTIREBOTE port map(not boton(9),clk100,botton(9));
    A10: ANTIREBOTE port map(not boton(10),clk100,botton(10));
    A11: ANTIREBOTE port map(not boton(11),clk100,botton(11));
    A12: ANTIREBOTE port map(not boton(12),clk100,botton(12));
    A13: ANTIREBOTE port map(not boton(13),clk100,botton(13));
    A14: ANTIREBOTE port map(not boton(14),clk100,botton(14));
    A15: ANTIREBOTE port map(not boton(15),clk100,botton(15));


    BTN_reset: ANTIREBOTE port map(breset,clk100,reset);
    BTN_start: ANTIREBOTE port map(bstart,clk100,start);


    DivFrec: CLOCK_DIV_50 port map(clock,clkRand,clk100k,clk10k,clk,clk100,clk10,tiempo);

end solucion;
```

El bloque empaquetado es el bloque de mayor jerarquía a nivel de todo el Sistema Digital. El mismo comprende todas las conexiones entre los diferentes ‘subbloques’ (usando la instrucción port map), lo cual permite la interconexión general.

- Bloque anti rebote

```
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;
```

Este bloque evita que haya interferencia por parte de ruido externo al momento de presionar una botonera estabilizando e invirtiendo la señal. Por ello fue necesario negar las señales provenientes del bloque en cuestión.

- Bloque Conv_Bin_BCD

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


entity Conv_Bin_BCD is
    Port ( Bin : in   STD_LOGIC_VECTOR (7 downto 0);
           Cen : out  STD_LOGIC_VECTOR (3 downto 0);
           Dec : out  STD_LOGIC_VECTOR (3 downto 0);
           Uni : out  STD_LOGIC_VECTOR (3 downto 0));
end Conv_Bin_BCD;

architecture Behavioral of Conv_Bin_BCD is

begin

Process(Bin)
variable Z: STD_LOGIC_VECTOR (19 downto 0);
begin

 for i in 0 to 19 loop
 Z(i) := '0';
 end loop;

 Z(10 downto 3) := Bin;


 for i in 0 to 4 loop

    if Z(11 downto 8) > 4 then
  Z(11 downto 8) := Z(11 downto 8) + 3;
  end if;

  if Z(15 downto 12) > 4 then
  Z(15 downto 12) := Z(15 downto 12) + 3;
    end if;

  Z(17 downto 1) := Z(16 downto 0);
  end  loop;



  Cen <= Z(19 downto 16);
  Dec <= Z(15 downto 12);
  Uni <= Z(11 downto 8);
end Process;
end Behavioral;
```

Este bloque permite la conversión de binario a BCD.

- Bloque tiempo2s

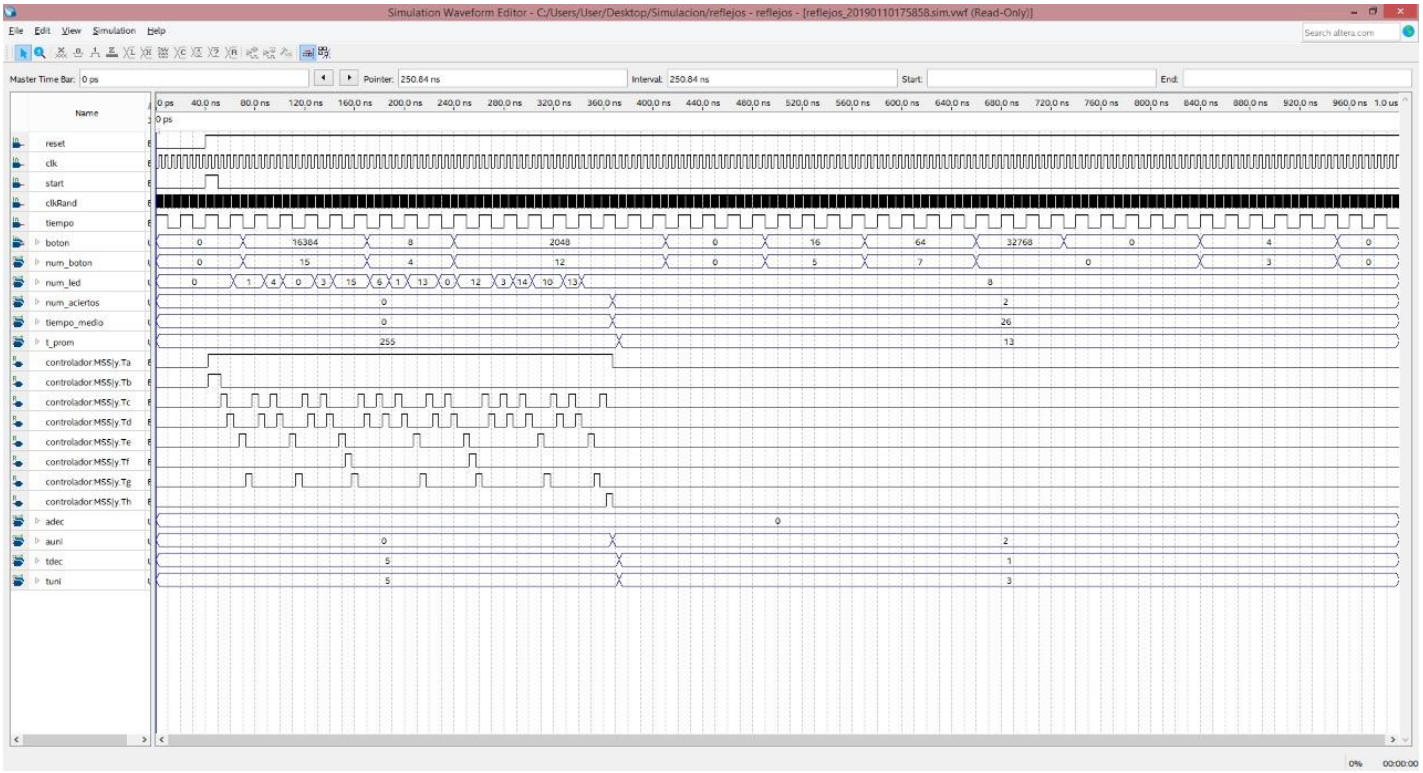
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity tiempo2s is
    PORT (
        clk2,reset    : IN  STD_LOGIC;
        salida: OUT STD_LOGIC
    );
end tiempo2s;

architecture sol of tiempo2s is
    signal cnt_tmp: STD_LOGIC_VECTOR(1 DOWNTO 0) := "00";
begin
   process(clk2,reset)
   begin
      if (reset='0') then salida <= '0';cnt_tmp <= "00";
      elsif(clk2'event and clk2='1') then
            cnt_tmp <= cnt_tmp + 1;
            if(cnt_tmp="10") then
               salida <= '1';
               cnt_tmp <= "00"; end if;
      end if;
    end process;
end sol;
```

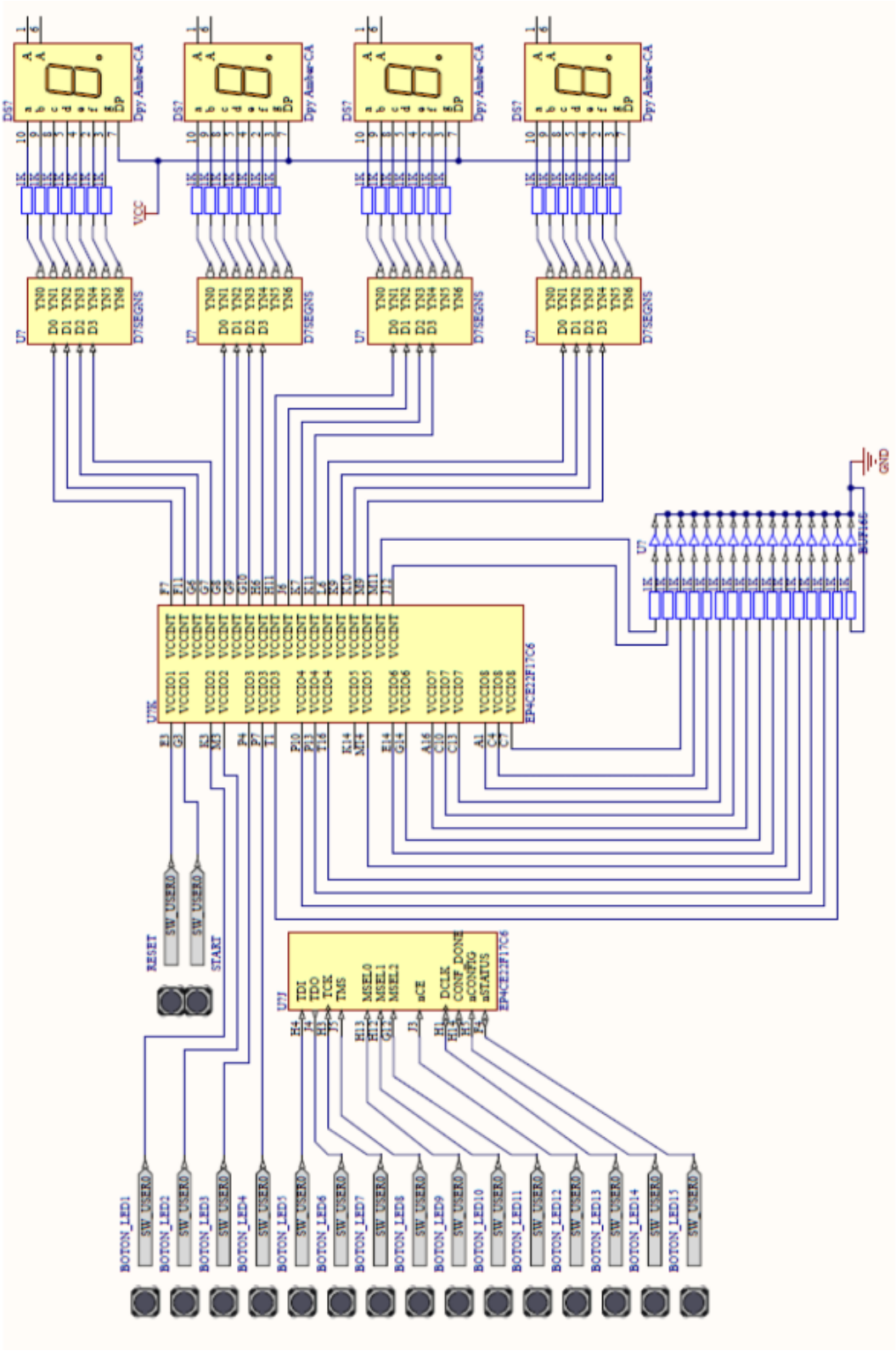
Dicho bloque provoca una latencia de 2 segundos para que los LED que se encenderán aleatoriamente estén prendidos durante dicho tiempo.

Diagrama de tiempo del Sistema Digital en VWF



En este diagrama se puede observar con mayor detalle la consolidación de todos los bloques al momento de la ejecución de la simulación, incluyendo las transiciones de los estados de la MSS. Como se detalló en la sección de los bloques del Sistema Digital total, todos los contadores, registros, convertidores, señales de reloj y demás bloques operan con señales intermedias para generar las salidas más externas: número de aciertos, tiempo medio y tiempo promedio de retardo. En el archivo VWF se puede observar que hubo 2 aciertos, y que en total el tiempo medio de retardo fue de 26 unidades de tiempo, lo cual da como resultado 13 unidades de tiempo de retardo por acierto.

Diagrama Esquemático



En el diagrama esquemático realizado en Altium Design se puede apreciar las entradas y salidas más externas del Sistema Digital: las botoneras de los LEDs que se encienden aleatoriamente (del 1 al 15), las botoneras de Start y Resetn; y los LEDs que pruebarán las capacidades de reflejos humanos del usuario, así como las 16 salidas que van dirigidas hacia los 4 decodificadores que pasan a los 2 juegos de displays que muestran el número total de aciertos y el tiempo promedio de retardo por unidad de tiempo. El diagrama esquemático muestra los pines asignados de la FPGA para los conectores JP1 y JP2. El juego de resistencias y LEDs que se observan en la parte inferior representa la tira de LEDs del proyecto.