TRABAJO 1 SISTEMAS ELECTRÓNICOS DIGITALES

VHDL Y FPGAs

CURSO 2023/24

RODRIGO MARTÍN VARGAS -55979

ÁLVARO GÓMEZ AGUDO - 55881

MARIO GÓMEZ SÁNCHEZ-CELEMÍN - 55887

GRUPO A404

ÍNDICE

1. Introducción

Una FPGA (Field Programmable Gate Array) es un circuito integrado digital programable complejo. Está formado por puertos de entrada/salida (IOB) y bloques lógicos configurables (CLB). Su interconexión y funcionalidad se programa mediante un lenguaje de descripción especializado, en nuestro caso, el VHDL. Su principal ventaja es la capacidad de reprogramarse para un trabajo específico o cambiar sus requisitos tras su fabricación.

Históricamente, las FPGA fueron inventadas en el año 1984 por Ross Freeman y Bernard Von Der Schmitt, cofundadores de la empresa Xilinx, fabricante de estas.

Tienen un amplio espectro de aplicaciones: como industria automotriz, reconocimiento de voz, controladores de dispositivos, sistemas militares, radiotransmisores/receptores gestionados por software, computación de alto rendimiento, sistemas de emulación de hardware, equipos de investigación en medicina, etc.

En nuestro caso, utilizaremos la FPGA Nexys 4 DDR Artix-7, placa diseñada para aplicaciones de formación que cuenta con arquitectura Xilinx Artix-7. Programada en Vivado con código en VHDL, un lenguaje de descripción de circuitos electrónicos digitales que utiliza distintos niveles de abstracción. El significado de las siglas VHDL es VHSIC (Very High Speed Integrated Circuits) Hardware Description Language. Esto significa que VHDL permite acelerar el proceso de diseño.

Ilustración 1: FPGA Nexys 4 DDR Artix-7

Realizaremos el trabajo de la “Clave secreta”. Se diseñará un circuito que detecta una secuencia en un orden determinado de pulsación de los pulsadores inducidos en la placa de desarrollo. Si la secuencia es correcta se encenderá un led indicándolo, y además se abrirá una puerta mediante la activación de un servomotor.

Para realizar un código en lenguaje VHDL con eficiencia y reutilizable es necesario hacerlo usando entidades o módulos que tengan funciones comunes a varios proyectos.

Se crean entidades con un fin determinado como podría ser el decodificador, el cual se utiliza en este caso para traducir el código binario de un numero a código de siete segmentos (código de los displays). Se planifican las entradas y salidas que la entidad va a necesitar, en este caso la entrada es un vector de 4 bits y la salida un vector de 7 bits.

```
ENTITY decoder IS
PORT (
entrada : IN std_logic_vector(3 DOWNTO 0);
led : OUT std_logic_vector(6 DOWNTO 0)
);
END ENTITY decoder;
```

Para que la entidad haga su trabajo correctamente es necesario añadir la arquitectura o funcionamiento, en este caso la parte de código que consigue que la traducción se haga correctamente.

```
ARCHITECTURE dataflow OF decoder IS
BEGIN
WITH entrada SELECT
led <= "0000001" WHEN "0000",
"1001111" WHEN "0001",
"0010010" WHEN "0010",
"0000110" WHEN "0011",
"1001100" WHEN "0100",
"0100100" WHEN "0101",
"0100000" WHEN "0110",
"0001111" WHEN "0111",
"0000000" WHEN "1000",
"0000100" WHEN "1001",
"1111110" WHEN others;
END ARCHITECTURE dataflow;
```

Con el fin de evitar pérdidas de tiempo innecesarias es conveniente el uso de bancos de pruebas o testbench donde simulemos y comprobemos que la salida es la esperada en todos los casos pues el generar código en este tipo de placas es muy costoso en cuanto a tiempo.

Como se ha explicado anteriormente las entidades son módulos que forman parte de o engloban a otras entidades con el fin de facilitar la creación de nuevos proyectos. Las entidades que necesitan de las funciones de otras entidades más pequeñas instancian como componentes a estas para que le realicen esas funciones menores indispensables en su tarea como pueden ser el edgedetector y el synchronizer para el FSM.

```
component EDGEDCTR is
      Port ( CLK : in STD_LOGIC;
      SYNC_IN : in STD_LOGIC;
      EDGE : out STD_LOGIC);
  end component;
  component SYNCHRNZR is
  Port ( CLK : in STD_LOGIC;
  ASYNC_IN : in STD_LOGIC;
  SYNC_OUT : out STD_LOGIC);
  end component;
```

De esta forma el código queda divido en funciones, es mucho más simple, legible y se puede reutilizar.

2. Display 7 segmentos

El principal componente de este diseño será un decodificador de BCD a siete segmentos. Un decodificador es un circuito combinacional que realiza la operación inversa a la de un codificador de datos, por eso también se puede definir un decodificador como circuito que convierte un código binario concreto en una forma sin codificar.

Los dígitos de la tarjeta son de tipo LED. Cada segmento es un LED que tiene un terminal accesible y el otro común para todos los segmentos: existen versiones de cátodo común y de ánodo común. En nuestro caso se trata de un ánodo común, por lo que para que luzca un segmento debemos aplicar al terminal no común un '0'. La placa tiene tantos dígitos que, para reducir el total de líneas, todos ellos comparten las mismas señales de control de los segmentos; esto es, todos los terminales no comunes de los segmentos A están unidos, y así hasta el segmento G. La forma de mostrar números distintos en cada dígito consiste en irlos encendiendo en secuencia haciendo que las líneas de control de los segmentos (ANx) reflejen el número correspondiente al dígito activo en ese momento, esto se conoce como multiplexación en el tiempo. Si está secuencia se ejecuta a suficiente velocidad el ojo resulta engañado y percibe todos los dígitos iluminados a la vez. Para iluminar un dígito concreto en esta tarjeta es necesario aplicar un '0' a la línea de control pertinente.

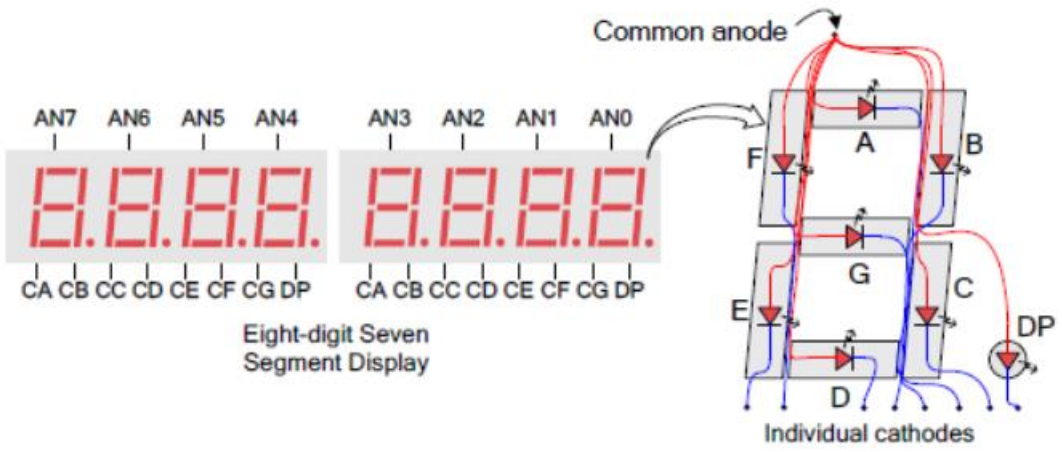

Ilustración 2: Visualizador de 8 dígitos y líneas correspondientes de la FPGA

3. Funcionamiento Edgedetector y Sincronizador

El uso de pulsadores viene acompañado de un inconveniente, el rebote o bouncing. Al ser accionados, los pulsadores tardan cierto tiempo en alcanzar un estado estable, es decir, que durante muy poco tiempo pueden cambiar entre el estado pulsado y no pulsado hast que la señal se estabiliza. Usamos el sincronizador para prevenir estos problemas de meta estabilidad y garantizar la estabilidad del sistema electrónico digital. Por su parte, el edgedetector identifica transiciones específicas en una señal digital, es decir, genera flancos de la señal a la entrada de la entidad con el fin de poder usar varios ya que el programa puede dar errores si se intenta usar varios flancos con la función rising_edge.

4. Funcionamiento servomotor con PWM

Un servomotor es un motor de corriente continua, pero en vez de conseguir un giro continuo, está diseñado para conseguir que gire un determinado ángulo en respuesta a una señal de control, y que se mantenga fijo en esa posición.

Esta señal de control es dada por los pines digitales PWM. Para controlar el servomotor se le envía pulsos cada 20 ms es decir a 50Hz. La anchura del pulso es lo que determinará el ángulo de giro, es decir lo que se conoce como PWM. Esta anchura varía según el servomotor, pero normalmente va entre 0.5 y 2.5 ms, aunque puede variar. Esto hace que el servomotor tenga un margen de operación, por lo que se puede mover entre 0º (pulso con una anchura de 0.5 ms) y un máximo, que suele ser, de 180º (pulso con una anchura de 2.5ms).

Los servomotores pequeños como el nuestro, funcionan con 5V. Disponen de 3 pines 1 rojo (5V), 1 marrón o negro (Ground) y uno blanco o amarillo (Control).

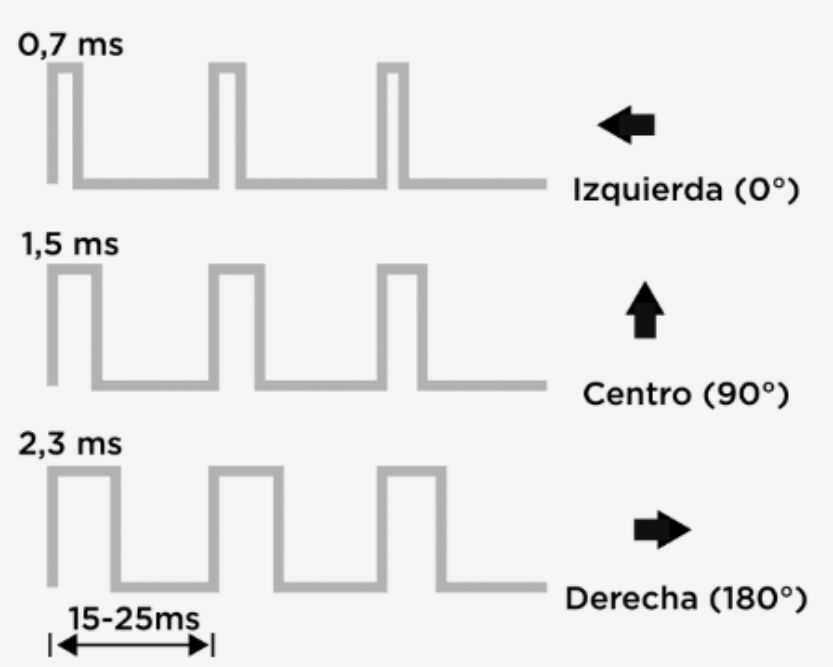

Ilustración 3: Posiciones servomotor

5. Funcionamiento leds

Utilizaremos dos leds, uno verde y otro rojo. Para su montaje dispondremos de dos resistencias de 75 ohmios para que solo circule la intensidad requerida. Su función será indicar si la contraseña es correcta o incorrecta. Instanciamos ambos leds asignando al verde la entrada signal, y al rojo la misma pero negada. (Esta señal indica que ambas contraseñas coinciden). Por ello, en caso de que signal sea igual a '1', es decir, las señales coincidan, se activara el led verde. En el resto de los casos se activará el rojo.

6. Montaje

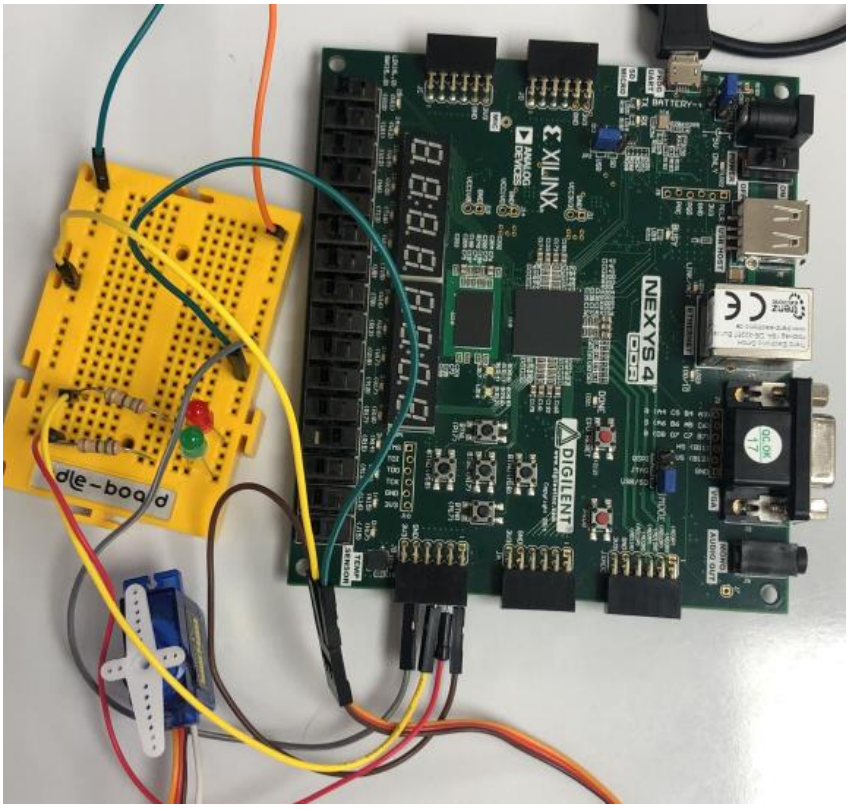

Ilustración 4: configuración placa

Ilustración 5: Servomotor accionando puerta

7. Diagrama de estados

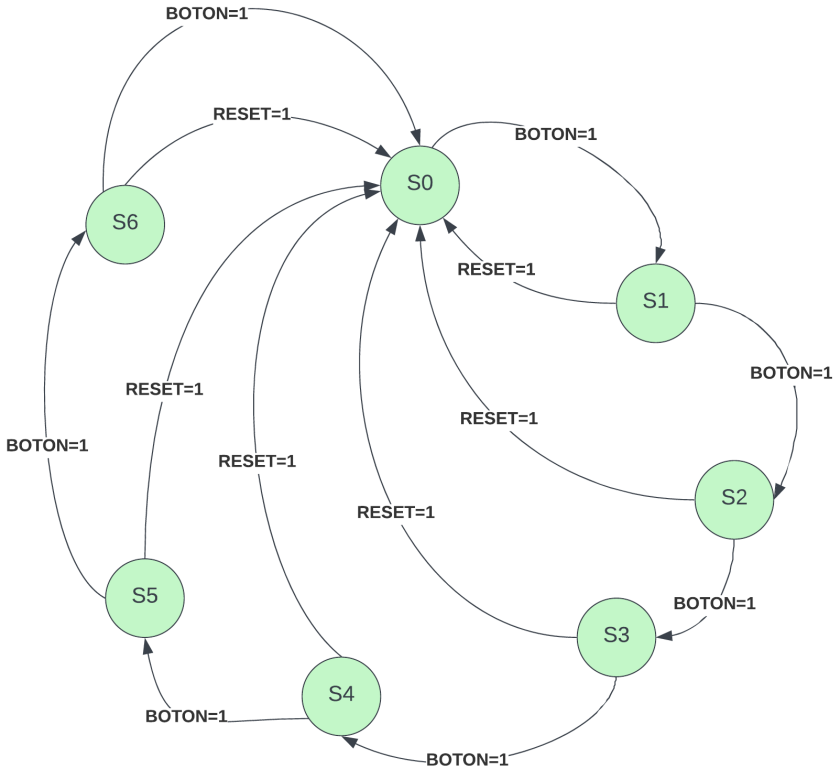

Ilustración 6: Diagrama de estados

Dentro del estado S6 tenemos dos opciones dependiendo de la salida “check”. En primer lugar, en caso de que la contraseña sea correcta, “check” será igual a “1”, se iluminará el led verde y se abrirá la puerta. En caso contrario, si no coincide la contraseña y “check” es igual a “0” se iluminará el led rojo y no se abrirá la puerta.

8. Funcionamiento del programa

El funcionamiento del código es el siguiente. Se comienza en el estado S0 o estado de reposo, para tratar de introducir la contraseña se pulsa el botón. Tras pulsar el botón se cambia al estado S1 en el cual se enciende el primer display y sobre el que podemos seleccionar el número que queremos introducir en esa posición de la contraseña. Para seleccionar el número del 0 al 9 se dispone de 4 interruptores con los que se escribe el número en binario, una vez se muestra el número deseado en el display se vuelve a pulsar el botón y se habrá guardado el número y se pasara al estado S2 con el mismo comportamiento pero con el siguiente display y posición de la contraseña. Así, hasta el estado S4 que es el estado en el que se elige el último digito de la clave. Al introducir todos los dígitos de la clave se pasa al estado S5 en el cual se muestran sobre los displays la clave correcta (displays izquierdos) y la introducida por el usuario (displays derechos). Si las claves coinciden al pulsar de nuevo el botón se pasa al estado S6 y se abrirá la puerta (gira el servomotor) y se enciende el led verde o en caso de no coincidir se enciende la luz roja y la puerta se mantiene cerrada. Para volver al estado de reposo se debe pulsar el botón estando en el estado S6 o pulsar el botón de reset en cualquiera de los estados.

9. TestBench

Para verificar el correcto funcionamiento de cada uno de los componentes es necesario probarlos, para ello se utilizan los testbench. En estos se crean estímulos para las diferentes entradas y se comprueba el resultado de la salida.

Servomotor

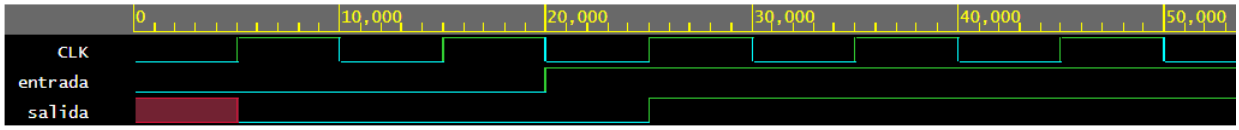

Ilustración 7: Testbench servomotor

Led, pin_led_verde, pin_led_rojo

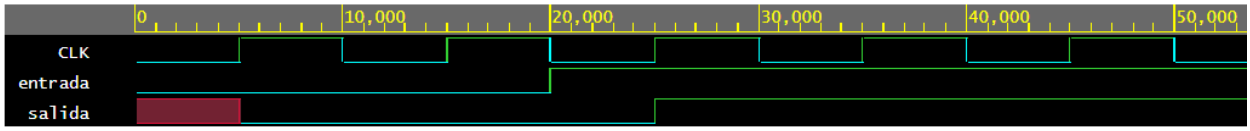

Ilustración 8: Testbench leds

Reset

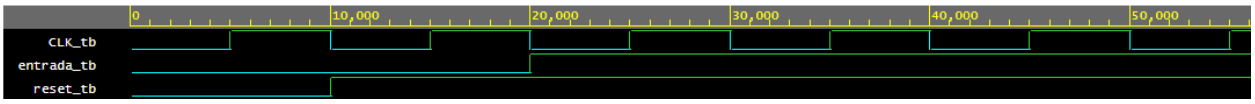

Ilustración 9: Testbench reset

Decoder

Ilustración 10: Testbench decoder

Entrada

Ilustración 11: Testbench entrada

A continuación, se desarrolla el código completo del testbench:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.Numeric_std.all;

entity top_tb is
--  Port ( );
end top_tb;

architecture Behavioral of top_tb is

  component top
    PORT (
        boton: IN std_logic;
        num : IN std_logic_vector(3 DOWNTO 0);
        reset: IN std_logic;
        CLK_top: IN std_logic;

        -- SALIDAS LEDs INTERNOS
        led_b: out std_logic;
        led_s0: out std_logic;
        led_s1: out std_logic;
        led_s2: out std_logic;
        led_s3: out std_logic;
        led_s4: out std_logic;
        led_s5: out std_logic;
        led_s6:out std_logic;
        --led_check: out std_logic;

        -- SALIDAS DISPLAYS 7 SEGMENTOS
        display: OUT std_logic_vector(3 DOWNTO 0);
        display_izq: OUT std_logic_vector(3 DOWNTO 0);
        segment : OUT std_logic_vector(6 DOWNTO 0);

        -- SALIDAS A ELEMENTOS EXTERNOS
```

```
        pwm: OUT std_logic;
        pin_led_verde: OUT std_logic;
        pin_led_rojo: OUT std_logic
      );
  END component;

COMPONENT decoder
  PORT (
    entrada : std_logic_vector(3 DOWNTO 0);
    led : std_logic_vector(6 DOWNTO 0)
    );
END COMPONENT decoder;

component EDGEDCTR is
  Port ( CLK : in STD_LOGIC;
  SYNC_IN : in STD_LOGIC;
  EDGE : out STD_LOGIC);
end component;

component SYNCHRNZR is
Port ( CLK : in STD_LOGIC;
ASYNC_IN : in STD_LOGIC;
SYNC_OUT : out STD_LOGIC);
end component;

component servomotor is
Port (
  CLK: in std_logic;
  entrada_1 : in std_logic;
  pwm: out std_logic);
end component;

component led is
Port (
  CLK: in std_logic;
```

```
      entrada_2 : in std_logic;
      led_check: out std_logic);
   end component;

   component pin_led_verde is
   Port (
      CLK: in std_logic;
      check_sig : in std_logic;
      pin_led_verde: out std_logic);
   end component;

   component pin_led_rojo is
   Port (
      CLK: in std_logic;
      check_sig : in std_logic;
      pin_led_rojo: out std_logic);
   end component;

-- SEÑALES DEL TOP PARA EL TB
 signal boton_tb: std_logic;
 signal num_tb : std_logic_vector(3 DOWNTO 0);
 signal reset_tb: std_logic;
 signal CLK_top_tb: std_logic;


 signal led_b_tb: std_logic;
 signal led_s0_tb: std_logic;
 signal led_s1_tb: std_logic;
 signal led_s2_tb: std_logic;
 signal led_s3_tb: std_logic;
 signal led_s4_tb: std_logic;
 signal led_s5_tb: std_logic;
 signal led_s6_tb: std_logic;
 signal led_check_tb: std_logic;
```

```
signal display_tb: std_logic_vector(3 DOWNTO 0);
signal display_izq_tb: std_logic_vector(3 DOWNTO 0);
signal segment_tb : std_logic_vector(6 DOWNTO 0);

signal entrada_1_tb : std_logic:='0';
signal entrada_2_tb : std_logic:='0';
signal CLK_tb: std_logic:='0';

signal pwm_tb: std_logic; --salida motor
signal check_sig: std_logic:='0';
signal pin_led_verde_tb: std_logic:='0';
signal pin_led_rojo_tb: std_logic:='0';

-- SEÑALES DEL DECODER PARA EL TB
signal entrada_tb : std_logic_vector(3 DOWNTO 0):="0000";
signal led_tb : std_logic_vector(6 DOWNTO 0);

--constantes
constant CLK_period: time := 10ns;

begin

top_tb: top port map(
        num=>num_tb,
        boton=>boton_tb,
        reset=>reset_tb,
        CLK_top=>CLK_top_tb,

        led_b=>led_b_tb,
        led_s0=>led_s0_tb,
        led_s1=>led_s1_tb,
        led_s2=>led_s2_tb,
        led_s3=>led_s3_tb,
        led_s4=>led_s4_tb,
        led_s5=>led_s5_tb,
```

```
            led_s6=>led_s6_tb,
            led_check=>led_check_tb,

            display=>display_tb,
            display_izq=>display_izq_tb,
            segment=>segment_tb,

            pin_led_verde=>pin_led_verde_tb,
            pin_led_rojo=>pin_led_rojo_tb,
            pwm=>pwm_tb);

decoder_tb: decoder port map(
            entrada=>entrada_tb,
            led=>led_tb
            );

--servo
servo_tb: servo port map(
            entrada_1=>entrada_1_tb,
            pwm=>pwm_tb,
            CLK=>CLK_tb
            );

--led
led_tb: led port map(
                                    entrada=>entrada_2_tb,
            led_check=>led_check_tb,
            CLK=>CLK_tb
           );

--led pin verde
led_tb: led_pin_verde port map(
            check_sig=>check_sig_tb,
            led_pin_verde=>led_pin_verde_tb,
            CLK=>CLK_tb
```

```
        );

--generar el reloj
gen_CLK_top: process
  begin
    CLK_top_tb<='0';
    wait for 0.5*CLK_period;
    CLK_top_tb<='1';
    wait for 0.5*CLK_period;
  end process;

Prueba_decoder:
    process begin
        report "Verificando decoder"
            severity note;

        entrada_tb<="0000";
        wait for CLK_period;
        assert led_tb<="00111111"
            report "Error en 0"
            severity failure;

        entrada_tb<="0001";
        wait for CLK_period;
        assert led_tb<="00000110"
            report "Error en 1"
            severity failure;

        entrada_tb<="0010";
        wait for CLK_period;
        assert led_tb<="01011011"
            report "Error en 2"
            severity failure;

        entrada_tb<="0011";
```

```
wait for CLK_period;
assert led_tb<="01001111"
    report "Error en 3"
    severity failure;

entrada_tb<="0100";
wait for CLK_period;
assert led_tb<="01100110"
    report "Error en 4"
    severity failure;

entrada_tb<="0101";
wait for CLK_period;
assert led_tb<="01101101"
    report "Error en 5"
    severity failure;

entrada_tb<="0110";
wait for CLK_period;
assert led_tb<="01111101"
    report "Error en 6"
    severity failure;

entrada_tb<="0111";
wait for CLK_period;
assert led_tb<="00000111"
    report "Error en 7"
    severity failure;

entrada_tb<="1000";
wait for CLK_period;
assert led_tb<="01111111"
    report "Error en 8"
    severity failure;
```

```
            report "Verificacion exitosa"
            severity note;
            wait;
        end process Prueba_decoder;


  --Test reset
process_reset: process begin
    reset_tb<='0' after 0.25*CLK_period, '1' after 0.75*CLK_period;
end process process_reset;

--Test entrada
process_entrada: process begin
    num_tb<="0101"; -- contraseña incorrecta
    wait for 2500  ns;
    num_tb<="1000"; -- contraseña correcta
    wait for 47500  ns;
end process process_entrada;

--Test boton
process_boton: process begin
    for i in 0 to 500 loop
        wait for 100 ns;
        boton_tb <= not boton_tb;
    end loop;
    wait;
end process process_boton;

--Test servo
process_servo: process begin

        entrada_1_tb<='0';
        wait for 20ns;

        entrada_1_tb<='1';
```

```
        wait for 20ns;


        wait;
end process process_servo;

--Test led
process_led: process begin

        entrada_2_tb<='0';
        wait for 20ns;

        entrada_2_tb<='1';
        wait for 20ns;

        wait;
end process process_led;

--Test pin led verde
process_pin_led_verde: process begin

        check_sig_tb<='0';
        wait for 20ns;

        check_sig_tb<='1';
        wait for 20ns;

        wait;
end process process_pin_led_verde;

--Test pin led rojo
process_pin_led_rojo: process begin

        check_sig_tb<='0';
        wait for 20ns;
```

```
        check_sig_tb<='1';
        wait for 20ns;


        wait;
    end process process_pin_led_rojo;


    end Behavioral;
```