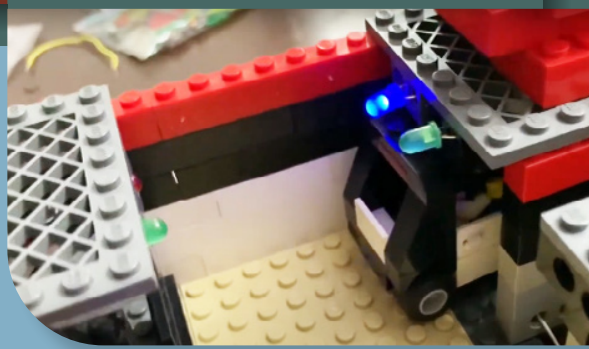
Universidad Nacional Autónoma de México

Facultad de Ingeniería

Norma Elva Chávez Rodríguez
Socorro Guevara Rodríguez
Vicente Flores Olvera
Rafael Prieto Meléndez

PRÁCTICAS PARA DISEÑO DIGITAL MODERNO



DIVISIÓN DE INGENIERÍA ELÉCTRICA

PRÁCTICAS PARA
DISEÑO DIGITAL MODERNO

Norma Elva Chávez Rodríguez
Socorro Guevara Rodríguez
Vicente Flores Olvera
Rafael Prieto Meléndez

División de Ingeniería Eléctrica
Departamento de Computación

Chávez Rodríguez, Norma Elva
Guevara Rodríguez, Socorro
Flores Olvera, Vicente
Prieto Meléndez, Rafael
Prácticas para Diseño Digital Moderno
Universidad Nacional Autónoma de México,
Facultad de Ingeniería, 2023, 91 p.

Prácticas para Diseño Digital Moderno

Primera edición electrónica
de un ejemplar (3 MB) Formato PDF
Publicado en línea en agosto de 2023

FACULTAD DE INGENIERÍA
http://www.ingenieria.unam.mx/

Hecho en México.

Unidad de Apoyo Editorial

Cuidado de la edición: Elvia Angélica Torres Rojas
Diseño, formación editorial e ilustraciones: Nismet Díaz Ferro
Fotografías: Norma Elva Chávez Rodríguez

PREFACIO

Estas prácticas están basadas en la filosofía de que los estudiantes principiantes no requieren entender a detalle el lenguaje VHDL, en cambio, deben poseer los conocimientos necesarios para modificar ejemplos y poder construir con esto un circuito básico deseado.

De esta manera aprenden la importancia de un lenguaje HDL con base en el diseño digital sin tener que aprender sus complejidades. Estas complejidades pueden estar reservadas para estudiantes más avanzados.

Los ejemplos utilizados en este tutorial fueron diseñados utilizando el lenguaje VHDL y las plataformas de desarrollo manejadas fueron ISE de la compañía Xilinx y Quartus de la compañía Altera.

Norma Elva Chávez Rodríguez
Socorro Guevara Rodríguez
Vicente Flores Olvera
Rafael Prieto Meléndez

CONTENIDO

PRÁCTICA

1

LOS OPERADORES VHDL Y LAS COMPUERTAS LÓGICAS

OBJETIVO

Aprender el funcionamiento de los operadores en lenguaje VHDL para construir diferentes tipos de expresiones mediante los cuales se pueden calcular datos.

INTRODUCCIÓN

En lenguaje VHDL existen operadores entre los que tenemos:

Operadores Lógicos: AND, OR, XOR, NOT, NAND, NOR, XNOR
Tipo Bit: Este tipo de señales solo toman los valores cero y uno
Tipo booleanos: Este tipo de señales solo toman los valores falso y verdadero
De relación: =, <, >, >=, <=, Integer, Bit y Bit_Vector
De Concatenación: &
Aritméticos: +, -, *, /
Asignación: <=

Operadores en lenguaje VHDL (primera parte)

Tipo	Características
BIT	En este tipo las señales solo toman los valores de "1" y "0"
Booleana	En este tipo las señales solo toman los valores de True y False
Std_logic	En este tipo las señales toman 9 valores, entre ellos tenemos: "1", "0", "Z" (para el 3er. estado), "-" (para los opcionales).

Operadores en lenguaje VHDL (segunda parte)

Tipo	Características
Integer	En este tipo las señales toman valores enteros. Los 1 y los 0 se escriben sin ‘’
Bit_Vector	En este tipo los valores de las señales son una cadena de unos y ceros. Ejemplo "1000"
Std_Logic_ Vector	En este tipo los valores de las señales son una cadena de los nueve valores permisibles para el tipo std_logic.
Character	Contiene todos los caracteres ISO de 8 bits, donde lo primeros 128 son los caracteres ASCII.

Modo de señales

Modo	Descripción
IN	En este modo las señales solo entran en la entidad
OUT	Las señales salen de la entidad
INOUT	Este modo se utiliza para señales bidireccionales. Se emplea con tres estados. Se puede asignar como sustituto de los tres modos anteriores, pero no se aconseja pues dificulta la comprensión del programa.

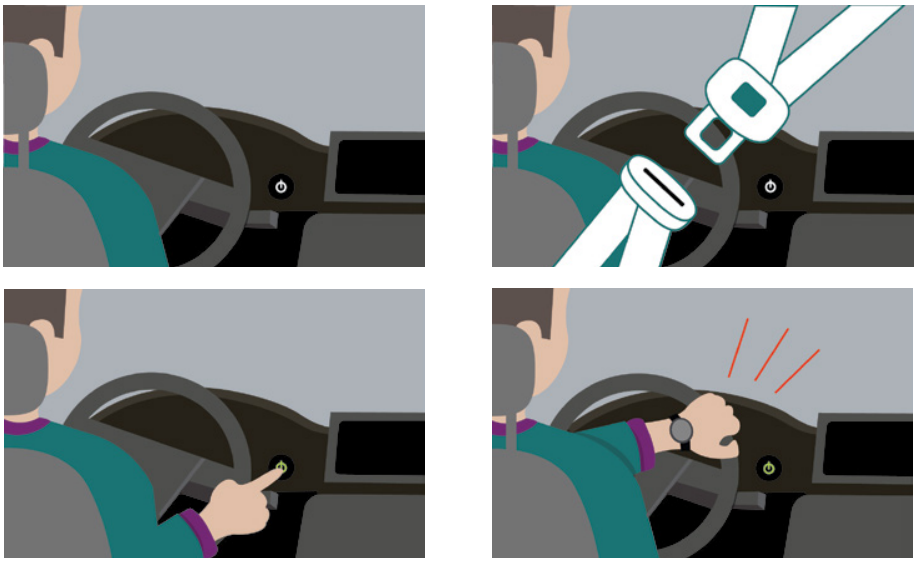
ACTIVIDADES

Diseñar y emular en una tarjeta de desarrollo un sistema de alarma para automóvil

ESPECIFICACIONES

Cuando el conductor se encuentre sentado en su asiento, no se ponga su cinturón de seguridad y tenga encendido el motor del auto, se deberá

encender un led, indicando que debe ponerse el cinturón. Los sensores para detectar si el conductor está o no en su asiento serán emulados por un interruptor llamado peso, el sensor para saber si tiene o no su cinturón puesto será emulado por un interruptor llamado cinturón y el sensor de prendido o no del motor será emulado por otro interruptor llamado encendido. La salida será emulada por un led, que se prenderá o apagará, según sea el caso.



PESO CINTURÓN ENCENDIDO

DIAGRAMA DE BLOQUES

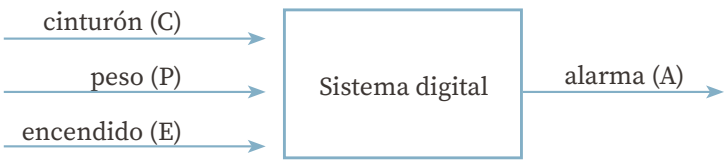


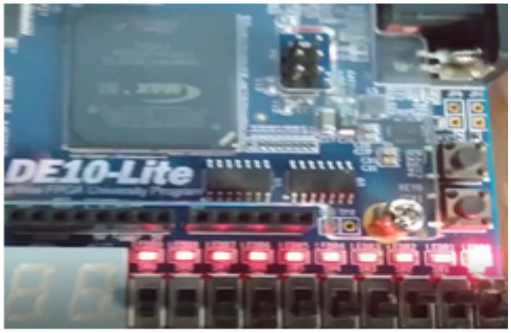
TABLA DE VERDAD Y OBTENCIÓN DE LA FUNCIÓN BOOLEANA

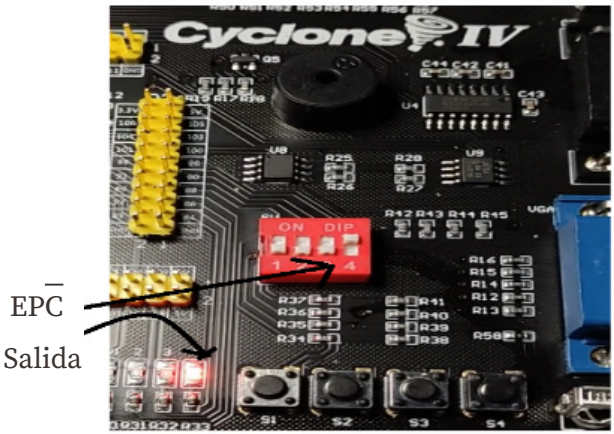
$$A = E P \overline{C}$$

Entrada			Salida
E	P	C	A
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

CÓDIGO EN VHDL

```
library IEEE;
use IEEE.std_logic_1164.ALL;
use IEEE.std_logic_arith.ALL;
use IEEE.std_logic_unsigned.ALL;
entity alarma is
    port (E : in std_logic;
          p : in std_logic;
          c : in std_logic;
          a : out std_logic);
end alarma;
architecture Behavioral of alarma is
begin
    A<= E and P and not C;
end Behavioral;
```





REVISA EL VIDEO
DE LA PRÁCTICA AQUÍ

PRÁCTICA

2

DISEÑO DE TABLAS DE VERDAD Y DECODIFICADORES BINARIOS

OBJETIVO

Aprender el manejo de la descripción por comportamiento dentro del lenguaje VHDL (Behavioral modeling), la cual consiste, como su nombre indica, en describir el comportamiento de algún sistema digital.

INTRODUCCIÓN

Hay dos maneras de representar una tabla de verdad en lenguaje VHDL, un método es usar la instrucción simultánea **with**-bus de datos de entrada-**select** y otro, utilizar las instrucciones secuenciales **process-case-when**.

ESPECIFICACIONES

En un hospital, en su sala de urgencias, se requiere tener un control de entrada. Se tienen tres tipos de pacientes: los que llegan con una súper-emergencia(S), los que llegan con una emergencia (E) y los que van tan solo a chequeo(C). Una enfermera les dará la letra que les corresponde al arribar al hospital.

El sistema debe ser de prioridad y mostrar en un display de siete segmentos quién debe entrar, de forma tal que si llega un paciente con una súper-emergencia, sin importar si al mismo tiempo llega otro tipo de paciente este debe entrar.

Cuando no exista a la entrada un paciente con una súper-emergencia, el que tiene la prioridad es el paciente que viene por una emergencia y al final entra el que viene tan solo para su chequeo. La figura muestra el diagrama de bloques de este sistema.

El manejo de este sistema se asocia a un vector de entrada de tres bits **A [2:0]**, en el cual se asigna al paciente de una súper-emergencia el valor A2, al paciente con una emergencia el valor A1 y al paciente que llegue tan solo a chequeo el valor A0.

Nota:
Para tarjetas de desarrollo con los displays de 7 segmentos conectados en paralelo, se requiere adicionar un vector de salida y código extra, los cuales se muestran en rojo y los segmentos se manejan con lógica negada.

DIAGRAMA DE BLOQUES

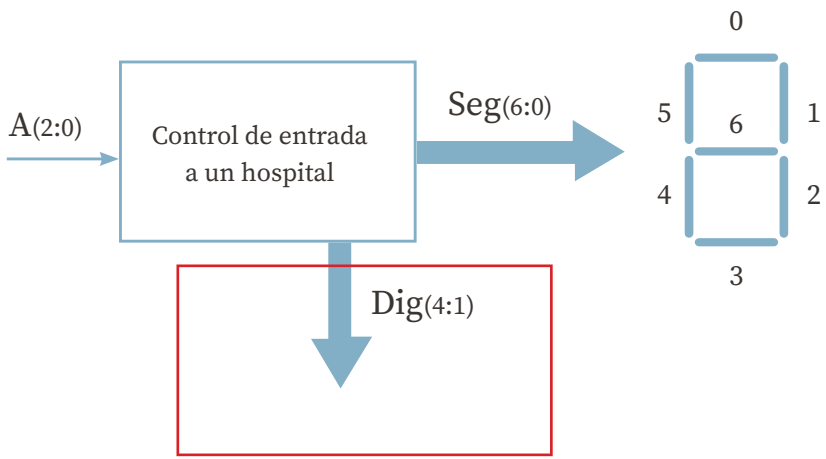


TABLA DE VERDAD

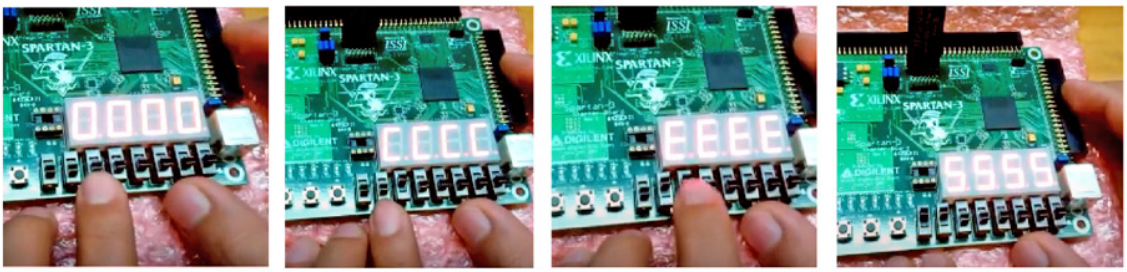
entradas	salidas	salidas
A (2:0)	Seg (6:0)	Dig (4:1)
000	1000000	1110
001	1000110	1110
010	0000110	1110
011	0000110	1110
100	0010010	1110
101	0010010	1110
110	0010010	1110
111	0010010	1110

CÓDIGO EN VHDL

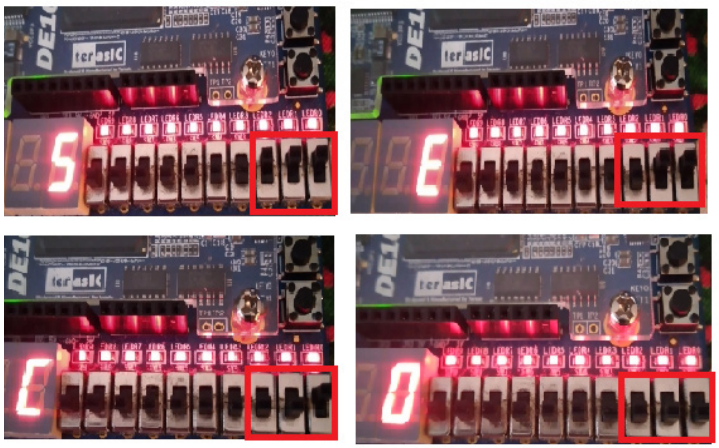
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity tabla is
    Port (A : in std_logic_vector (2 downto 0);
     Seg : out std_logic_vector (6 downto 0));
end tabla;
architecture Behavioral of tabla is
begin
    with A select
        Seg   <= "1000000" when "000",
                 "1000110" when "001",
                 "0000110" when "010",
                 "0000110" when "011",
                 "0010010" when others;
    with A select
        Dig   <= "1110" when "000",
                 "1110" when others;
end Behavioral;
```

Fotografías del sistema en tarjeta de desarrollo con displays en paralelo

Fotografías del sistema en tarjeta de desarrollo con displays



Fotografías del sistema en tarjeta de desarrollo con displays independientes



REVISA EL VIDEO
DE LA PRÁCTICA AQUÍ

PRÁCTICA

3

DISEÑO DE TABLAS DE VERDAD DENTRO DE UN PROCESO

OBJETIVO

Aprender el manejo de una tabla de verdad dentro de la instrucción process y verificar que la lista de sensibilidad está dada por las entradas en todo sistema combinacional.

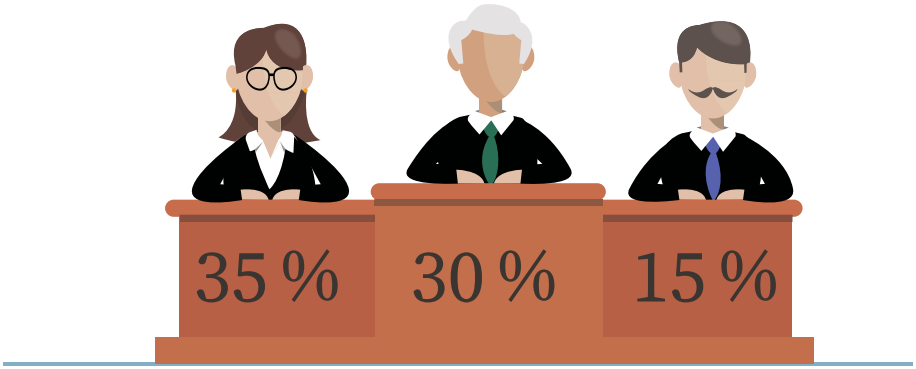
ACTIVIDADES

Diseñar el control de pago de multas que maneje un display de 7 segmentos utilizando la instrucción process en lenguaje VHDL.

ESPECIFICACIONES

Este sistema tiene por entrada la decisión de un jurado compuesto por tres personas (A2, A1, A0) cuyas opiniones valen: A2 = 35 %, A1 = 30 %, A0 = 15 %. Un uno lógico a la entrada significa que ese juez desea que el culpable pague la multa.

La multa se paga cuando el jurado quiere que sea pagada por 50 % o más. Cuando en el **display de 7** segmentos se vea la letra **P**, el infractor pagará la multa y si se ve **un cero** significa que no pagará nada. La figura 1 muestra el diagrama de bloques.



Nota:

Para tarjetas de desarrollo con los displays de 7 segmentos conectados en paralelo, se requiere adicionar un vector de salida y código extra, los cuales se muestran en rojo.

DIAGRAMA DE BLOQUES

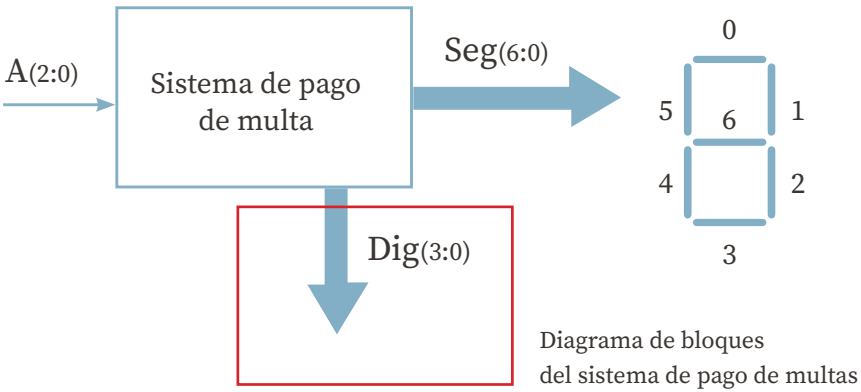


Diagrama de bloques del sistema de pago de multas

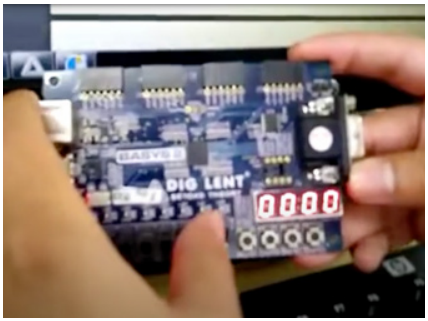
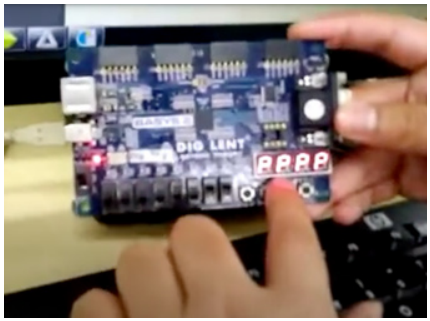
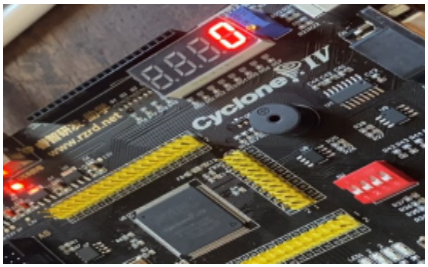
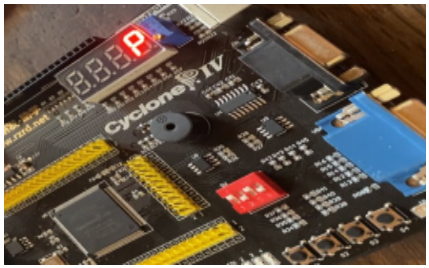
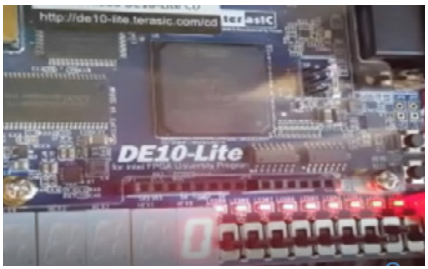
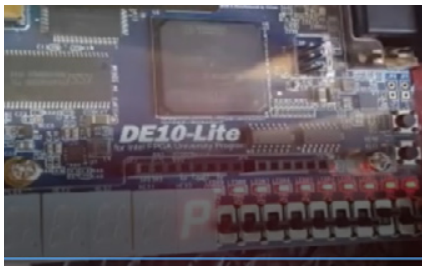
TABLA DE VERDAD

Entradas	Salidas	Salidas
A (2:0)	Seg (6:0)	Dig (4:1)
000	1000000	1110
001	1000000	1110
010	1000000	1110
011	1000000	1110
100	1000000	1110
101	0001100	1110
110	0001100	1110
111	0001100	1110

El código en VHDL de este sistema se observa a continuación, tomando la entrada como un vector A de tres bits (A_2, A_1, A_0).

CÓDIGO EN VHDL

```
library IEEE;
use IEEE.std_logic_1164.ALL;
use IEEE.std_logic_arith.ALL;
use IEEE.std_logic_unsigned.ALL;
entity Pago is
   Port (A : in  STD_LOGIC_VECTOR (2 downto 0);
         Dig : out STD_LOGIC_VECTOR (4 downto 1);
         Seg : out  STD_LOGIC_VECTOR (6 downto 0));
end Pago;
architecture Behavioral of Pago is
begin
    Process (A)
    begin
        case A is
            when "000" => Seg <= "1000000"; Dig <= "1110";
            when "001" => Seg <= "1000000"; Dig <= "1110";
            when "010" => Seg <= "1000000"; Dig <= "1110";
            when "011" => Seg <= "1000000"; Dig <= "1110";
            when "100" => Seg <= "1000000"; Dig <= "1110";
            when "101" => Seg <= "0001100"; Dig <= "1110";
            when "110" => Seg <= "0001100"; Dig <= "1110";
            when others => Seg <= "0001100"; Dig <= "1110";
        end case;
    end process;
end Behavioral;
```

Fotografías control de multas (NO PAGA)	Fotografías control de pago de multas (PAGA)
	
	
	

REVISA EL VIDEO DE LA PRÁCTICA AQUÍ

PRÁCTICA

4

DISEÑO DE COMPARADORES BINARIOS

OBJETIVO

Aprender el manejo de operadores de relación utilizando lenguaje VHDL, al comparar dos entradas binarias (A y B de n bits) para indicar la relación de igualdad o desigualdad entre ellas, por medio de "tres banderas lógicas" que corresponden a las relaciones A igual B, A mayor que B y A menor que B. También aprender a utilizar la instrucción **"IF"** dentro de un proceso.

ESPECIFICACIONES

Diseñar un circuito que compare dos números binarios (A y B), cada uno de un bit, en el cual se visualice mediante un display de 7 segmentos la letra A, cuando A es mayor que B; dos líneas paralelas, si son iguales; y la letra b, si A es menor que B.

DIAGRAMA DE BLOQUES

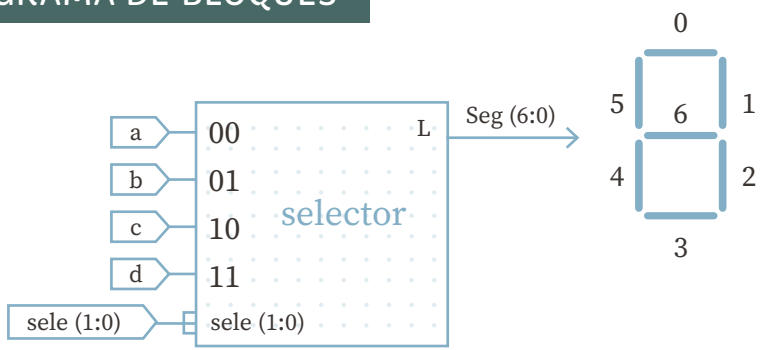
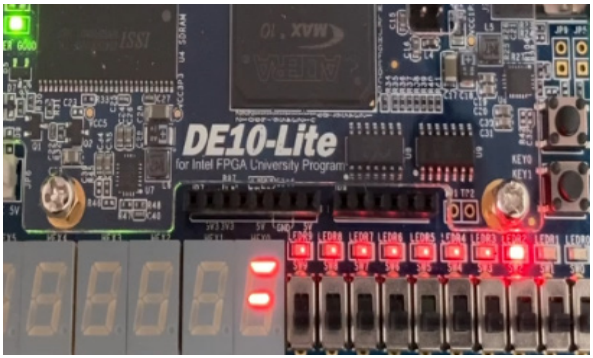


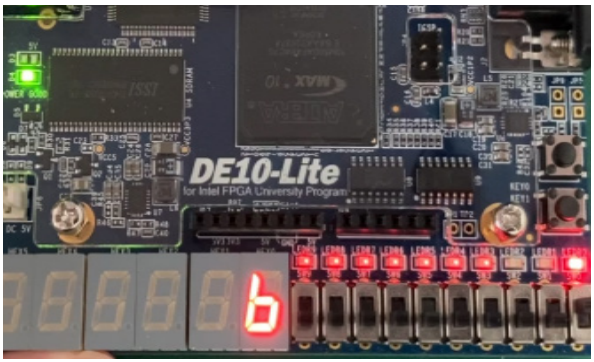
TABLA DE VERDAD

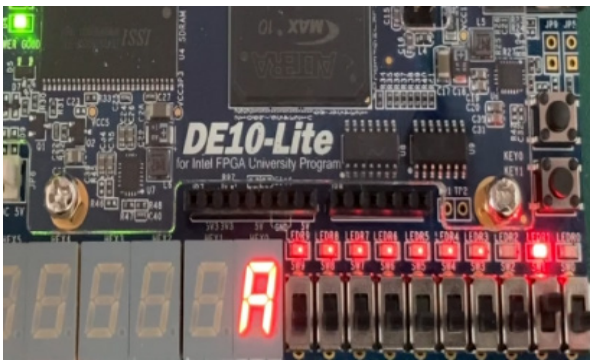
A	B	=	<	>
0	0	0111110		
0	1		0000011	
1	0			0001000
1	1	0111110		

CÓDIGO EN VHDL DEL COMPARADOR

```
library IEEE;
use IEEE.std_logic_1164.ALL;
use IEEE.std_logic_arith.ALL;
use IEEE.std_logic_unsigned.ALL;
entity comparador is
port (A: in std_logic;
      B : in std_logic;
      S : out std_logic_vector (6 downto 0));
end comparador;

architecture Behavioral of comparador is
begin
   Process (A, B)
   begin
      IF   A > B THEN
         S <= "0001000";
      ELSIF   A < B THEN
         S <= "0000011";
      ELSIF   A = B THEN
         S <= "0111110";
      end IF;
   end Process;
end Behavioral;
```







Fotografías del comparador binario

REVISA EL VIDEO DE LA PRÁCTICA AQUÍ

PRÁCTICA

5

DISEÑO DE UNA UNIDAD ARITMÉTICA LÓGICA (ALU)

OBJETIVO

Aprender el diseño y construcción de una unidad aritmética lógica que realiza operaciones aritméticas (suma, multiplicación) y operaciones lógicas (and, or).

ACTIVIDADES

Diseñar una unidad aritmética lógica.

ESPECIFICACIONES

Se requiere el diseño y construcción de un sistema digital en el que se visualice en 8 leds, las operaciones aritméticas (suma y multiplicación binarias) y las operaciones lógicas (AND y OR). Tendrá por entradas de datos dos números (A, B), cada uno de cuatro bits, y una entrada de control (C) de 2 bits para la selección de 4 distintas operaciones.

DIAGRAMA DE BLOQUES

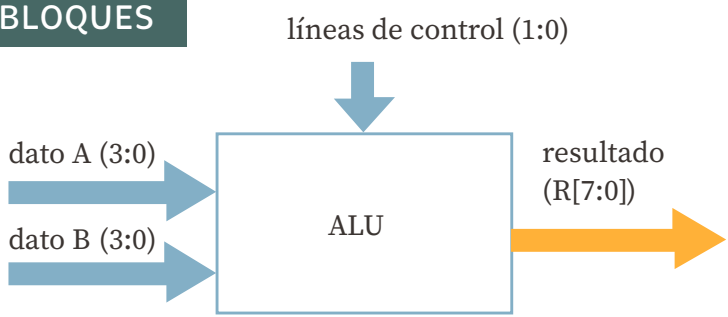


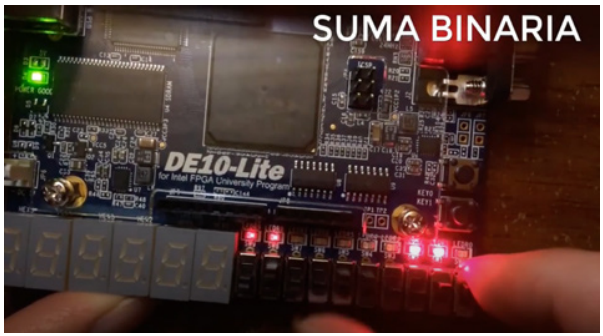
TABLA DE VERDAD

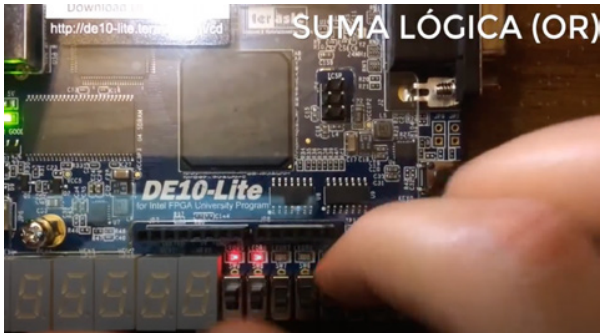
Entradas de control		Salida	Operación realizada
C_1	C_0	R	
0	0	A * B	Multiplicación binaria
0	1	A + B	Suma binaria
1	0	A AND B	Multiplicación lógica
1	1	A OR B	Suma lógica

El código en VHDL de este sistema se observa enseguida, tomando la entrada como un vector A de cuatro bits (A_3, A_2, A_1, A_0), un vector B de cuatro bits (B_3, B_2, B_1, B_0) y un vector C (C_1, C_0). Las fotografías muestran su simulación.

CÓDIGO EN VHDL

```
library IEEE;
use IEEE.std_logic_1164.ALL;
use IEEE.std_logic_arith.ALL;
use IEEE.std_logic_unsigned.ALL;
entity ALU is
   Port (A, B : in std_logic_vector (3 downto 0);
         C : in std_logic_vector (1 downto 0);
         R : out std_logic_vector (7 downto 0));
end ALU;
architecture Behavioral of ALU is
begin
   with C select
      R <=  A * B                          when "00",
            ("0000"& A) + ("0000" & B)     when "01",
            ("0000" & A) and ("0000" & B)  when "10",
            ("0000" & A) or ("0000" & B)   when others;
end Behavioral;
```





REVISA EL VIDEO
DE LA PRÁCTICA AQUÍ

PRÁCTICA

6

DISEÑO DE SELECTORES BINARIOS

OBJETIVO

Aprender el manejo del flujo de datos en un multiplexor o selector; así como, la instrucción "CASE" dentro de un proceso.

ESPECIFICACIONES

Diseñar un circuito multiplexor que tenga dos líneas de selección, emuladas por switches, para seleccionar el paso de cuatro datos binarios de un bit, emulados por push buttons. En un display de 7 segmentos se verá la letra que le toca pasar.

DIAGRAMA DE BLOQUES

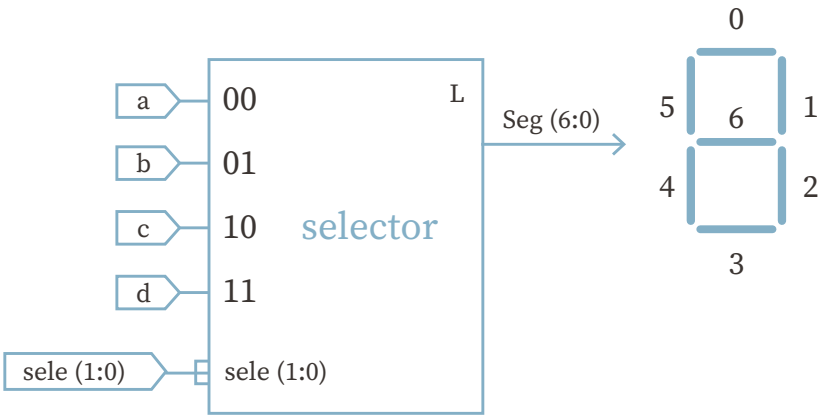
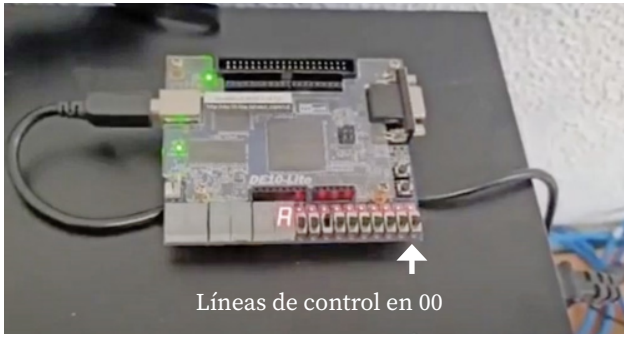


TABLA DE VERDAD

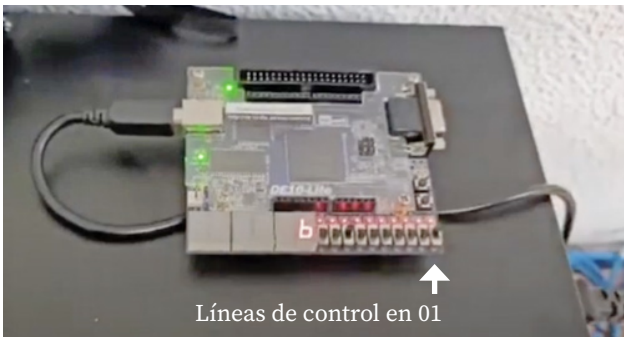
sele		L	
0	0	0001000	a
0	1	0000011	b
1	0	1000110	c
1	1	0100000	d

CÓDIGO EN VHDL

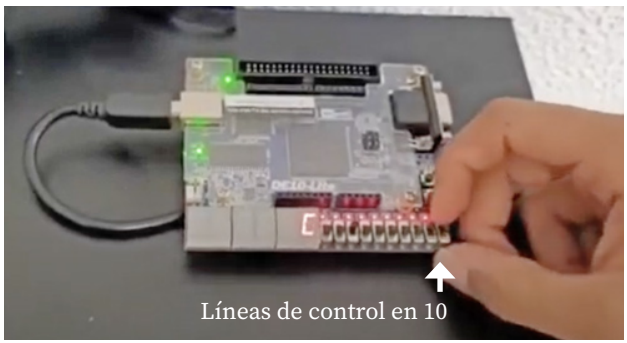
```
library IEEE;
use IEEE.std_logic_1164.ALL;
use IEEE.std_logic_arith.ALL;
use IEEE.std_logic_unsigned.ALL;
entity selector is
   Port (sele : in  STD_LOGIC_VECTOR (1 downto 0);
         L    : out  STD_LOGIC_VECTOR (6 downto 0));
end selector;

architecture Behavioral of selector is
begin
    with sele select
    L  <= "0001000" when "00",   -------   A
          "0000011" when "01",   -------   b
          "1000110" when "10",   -------   C
          "0100001"  when others;  ---  d
end Behavioral;
```



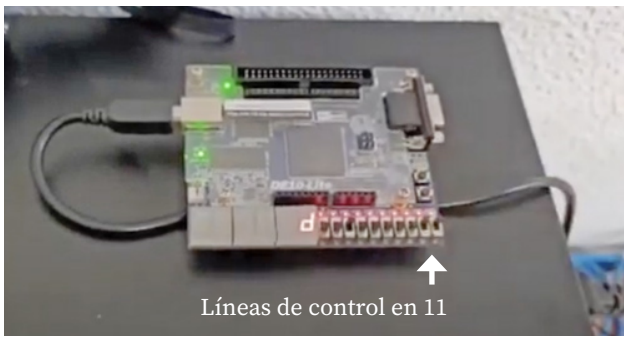
Líneas de control en 00



Líneas de control en 01



Líneas de control en 10



Líneas de control en 11

REVISA EL VIDEO
DE LA PRÁCTICA AQUÍ

PRÁCTICA

7

DISEÑO DE DIVISORES DE FRECUENCIA Y CONTADORES

OBJETIVO

Aprender a diseñar divisores de frecuencia, con el fin de visualizar cualquier sistema secuencial en tarjetas de desarrollo con FPGA, las cuales, generalmente, tienen un reloj de cristal de 50 MHz que es una frecuencia muy rápida para ser detectada por el ojo humano.

INTRODUCCIÓN

Se llama divisor de frecuencia a un dispositivo que produce a su salida una frecuencia menor que la de entrada y suele estar formado por contadores digitales.

Un contador es un circuito secuencial construido a partir de flip-flops y compuertas lógicas.

Si a un flip-flop JK todas sus entradas se le conectan a Vcc, excepto la entrada del reloj, a su salida, dicho flip-flop fluctúa entre cero y uno cada vez que el flanco activo del reloj se presente, y si se interconecta la salida del primero al reloj del segundo y la salida del segundo al reloj del tercero, y así sucesivamente, como se muestra en la figura 1, las salidas de los flip-flops fluctúan en diferentes tiempos, ya que cada uno de ellos tiene diferente entrada de reloj.

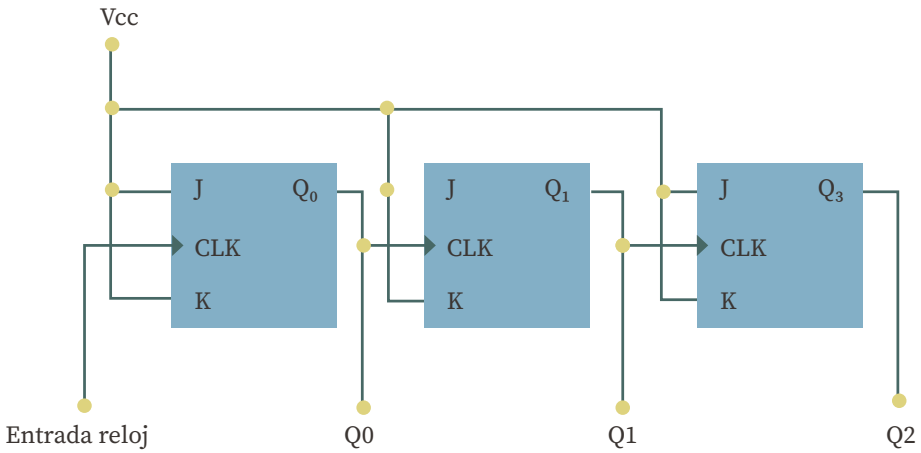


Figura 1. Flip-Flops interconectados mediante la unión de la salida de uno a la entrada de reloj del siguiente

La figura 2 muestra el diagrama de tiempos de las salidas en los flip-flops interconectados en la figura 1. En rojo se muestra el flanco activo.

Observando la figura 2 y poniendo leds a las salidas Q2, Q1, y Q0 (en ese orden), se tiene un contador descendente que cuenta del 7 al cero.

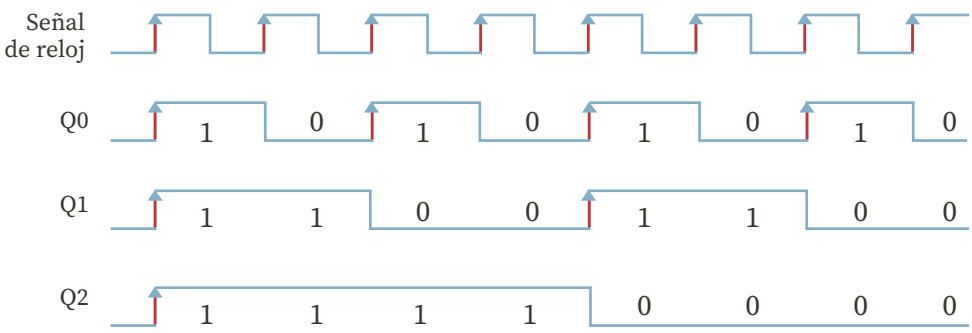


Figura 2. Diagrama de tiempos de las salidas en los flip-flops

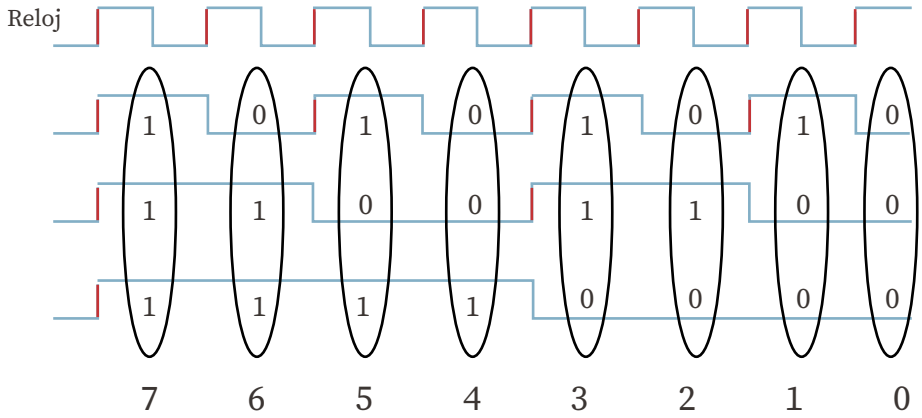


Figura 3. Contador descendente, que cuenta del 7 al cero

También se observa que, si la salida Q2 se selecciona como entrada de reloj para cualquier sistema digital, se obtiene un divisor de frecuencia respecto a la entrada del reloj original, como se observa en la figura 4.

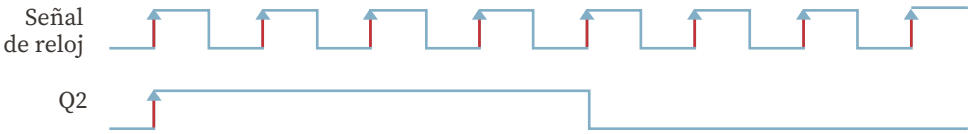


Figura 4. Divisor de frecuencia

ESPECIFICACIONES

Diseñar un contador de 28 bits, del cual únicamente se tomará en cuenta la salida del bit 26 para utilizarla como el reloj de cualquier diseño secuencial implementado en tarjetas con reloj de 50 MHz.

Esto hará que la entrada del reloj tenga un retraso de 2^{26} bits, dando un valor en hexadecimal de **48009E0** asegurando con esto un tiempo de 1.51 segundos entre cada pulso del reloj.

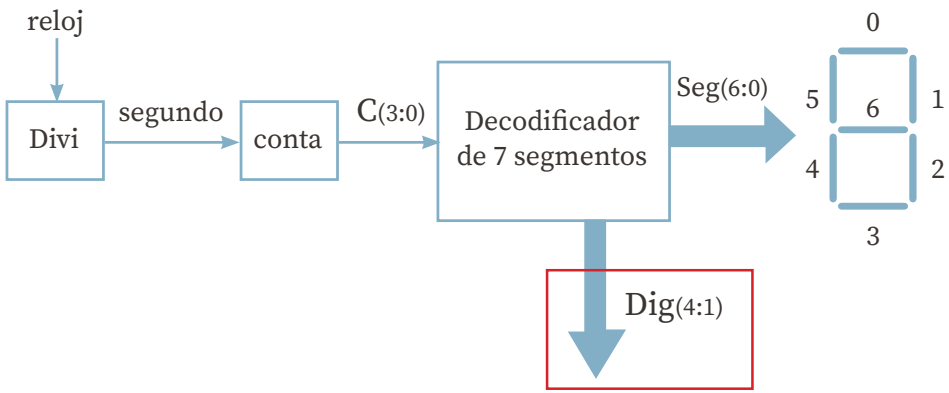
Posteriormente, diseñar un contador binario de cuatro bits que cuente del 9 al cero.

Para contar del nueve al cero se requiere de un contador de 4 bits, con el que se puede obtener la cuenta del 15 al cero, por lo que se requiere que de cada vez que se llegue al número cero se reinicie la cuenta.

Finalmente, se requiere visualizar la cuenta en un display de 7 segmentos.

Nota:
Para tarjetas de desarrollo con los displays de 7 segmentos conectados en paralelo, se requiere adicionar un vector de salida y código extra, los cuales se muestran en rojo.

DIAGRAMA DE BLOQUES



Bloque Divi
Internamente generaremos un contador de 28 bits del cual se tomará en cuenta la salida 26 para utilizarla como el reloj del contador.

Esto hará que la entrada del reloj de 50 MHz sea dividida por un factor de 2^{26}, pero considerando que el contador se reinicia al llegar al valor hexadecimal de 48009E0, se asegura con esto un tiempo de 1.51 segundos entre cada pulso del reloj. A esa salida la llamaremos Segundo.

Bloque Conta

Internamente se genera un contador de 4 bits, que se reinicia cada vez que llegue al número cero, debido a que, con 4 bits, es capaz de contar hasta el 15.

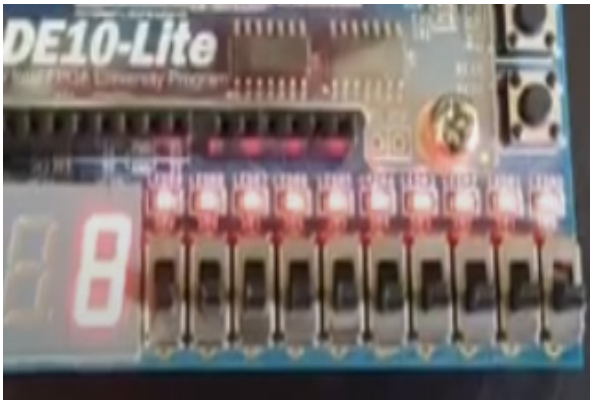
Bloque Deco

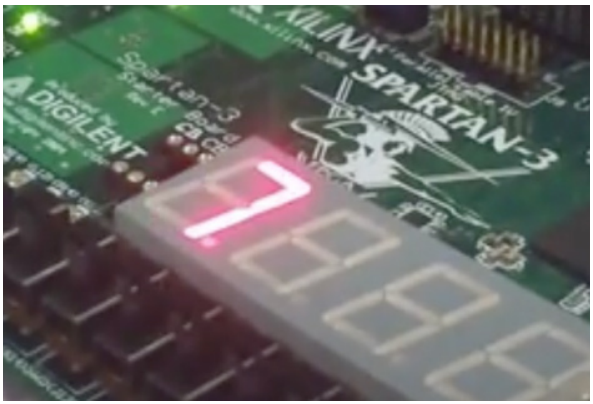
En este bloque se requiere hacer una tabla de verdad, la cual tiene por entrada un vector C de cuatro bits y a su salida requiere decodificar a 7 segmentos cada combinación ese vector.

CÓDIGO EN VHDL

```
library IEEE;
use IEEE.std_logic_1164.ALL;
use IEEE.std_logic_arith.ALL;
use IEEE.std_logic_unsigned.ALL;

entity contador is
    Port (reloj : in STD_LOGIC;
          Dig  : out STD_LOGIC_VECTOR (4 downto 1);
          Seg  : out STD_LOGIC_VECTOR (6 downto 0));
end contador;

architecture Behavioral of contador is
    signal   segundo: std_logic;
    signal C: std_logic_vector (3 downto 0);
begin
```

```
Divi: process (reloj)
variable cuenta: std_logic_vector (27 downto 0) :=  x"0000000";
   begin
      if rising_edge (reloj) then
        if cuenta = x"48009e0" then
            cuenta :=  x"0000000";
        else
            cuenta := cuenta + 1;
        end if;
      end if;
      segundo <= cuenta(24);
   end process;

 conta: process (segundo)
     variable cuenta : std_logic_vector(3 downto 0) := "1001";
 begin
     if rising_edge (segundo) then
        if cuenta = "0000" then
            cuenta := "1001";
        else
            cuenta := cuenta - 1;
        end if;
      end if;
      C <= cuenta;
 end process;

 with C select
         Seg <= "1000000" when "0000",   -- 0
                "1111001" when "0001",   -- 1
                "0100100" when "0010",   -- 2
                "0110000" when "0011",   -- 3
                "0011001" when "0100",   -- 4
                "0010010" when "0101",   -- 5
                "0000010" when "0110",   -- 6
                "1111000" when "0111",   -- 7
                "0000000" when "1000",   -- 8
                "0010000" when "1001",   -- 9
                "1000000" when others;  -- cero

 Dig <= "1110";

end Behavioral;
```





REVISA EL VIDEO
DE LA PRÁCTICA AQUÍ

PRÁCTICA

8

DISEÑO DE CONTADORES ASCENDENTE Y DESCENDENTE QUE TRABAJAN AL MISMO TIEMPO

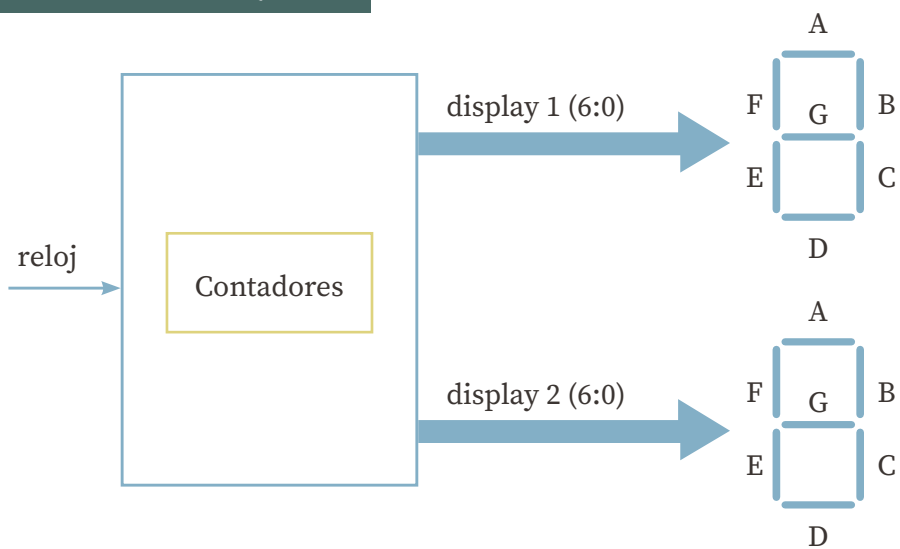
OBJETIVO

Aprender a diseñar utilizando código VHDL, contadores binarios descendentes y ascendentes, los cuales corten la cuenta en el número deseado.

ESPECIFICACIONES

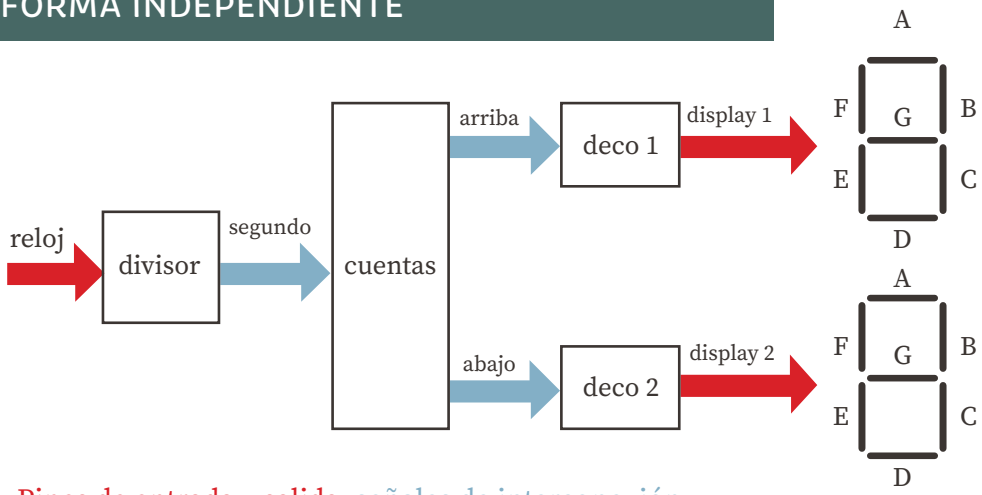
Se requiere el diseño y construcción de dos contadores que trabajen al mismo tiempo, uno que cuente del cero al nueve y vuelva a empezar, y el otro que cuente del nueve al cero y vuelva a empezar, con visualización en un display de 7 segmentos.

DIAGRAMA DE BLOQUES



Dentro del contador debemos diferenciar entre varios bloques funcionales, los cuales serán unidos mediante señales. La siguiente figura muestra los bloques funcionales del sistema “Contadores”:

BLOQUES FUNCIONALES PARA TARJETAS DE DESARROLLO CON DISPLAYS CONECTADOS EN FORMA INDEPENDIENTE



Pines de entrada y salida, señales de interconexión

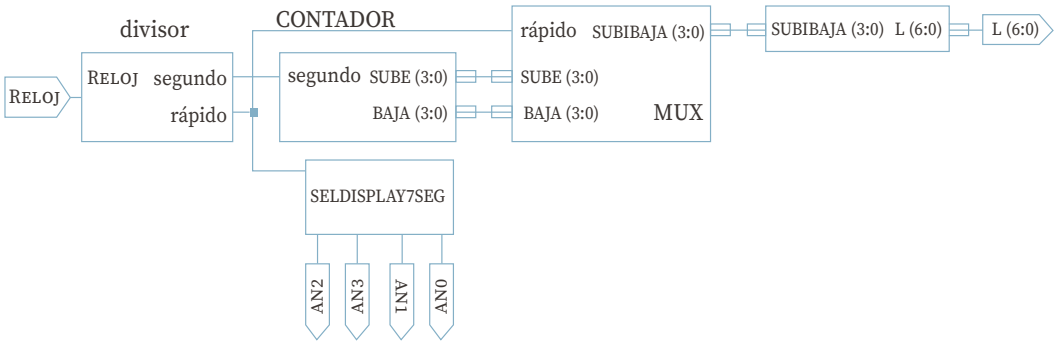
CÓDIGO EN VHDL DEL SISTEMA

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

entity contadores is
   Port (reloj : in STD_LOGIC;
         display1, display2 : out std_logic_vector (6 downto 0));
end contadores;
```

```
architecture Behavioral of contadores is
    signal segundo : std_logic;
    signal arriba : std_logic_vector(3 downto 0) :="0000";
    signal abajo:   std_logic_vector(3 downto 0):="1001";
begin

    divisor: process (reloj)
        variable cuenta: std_logic_vector(27 downto 0) :=  x"0000000";
    begin
        if rising_edge (reloj) then
            if cuenta =x"48009e0" then
                cuenta :=  x"0000000";
            else
                cuenta := cuenta+1;
            end if;
        end if;
        segundo <= cuenta(24);
    end process;

    cuentas: process (segundo)
        variable sube: std_logic_vector(3 downto 0) :=  "0000";
        variable baja : std_logic_vector(3 downto 0) :=  "1001";
    begin
        if rising_edge (segundo) then
            if sube ="1001" then
                sube :="0000";
            else
                sube := sube +1;
            end if;
            if baja ="0000" then
                baja :="1001";
            else
                baja := baja - 1;
            end if;
        end if;
        arriba <= sube;
        abajo <= baja;
    end process;
```

```
  with arriba select
    display1 <= "1000000" when "0000",   -- 0
                "1111001" when "0001",   -- 1
                "0100100" when "0010",   -- 2
                "0110000" when "0011",   -- 3
                "0011001" when "0100",   -- 4
                "0010010" when "0101",   -- 5
                "0000010" when "0110",   -- 6
                "1111000" when "0111",   -- 7
                "0000000" when "1000",   -- 8
                "0010000" when "1001",   -- 9
                "1000000" when others;  -- cero

  with abajo select
    display2 <= "1000000" when "0000",   -- 0
                "1111001" when "0001",   -- 1
                "0100100" when "0010",   -- 2
                "0110000" when "0011",   -- 3
                "0011001" when "0100",   -- 4
                "0010010" when "0101",   -- 5
                "0000010" when "0110",   -- 6
                "1111000" when "0111",   -- 7
                "0000000" when "1000",   -- 8
                "0010000" when "1001",   -- 9
                "1000000" when others;  -- cero

end behavioral;
```

BLOQUES FUNCIONALES PARA TARJETAS DE DESARROLLO CON DISPLAYS CONECTADOS EN FORMA PARALELO



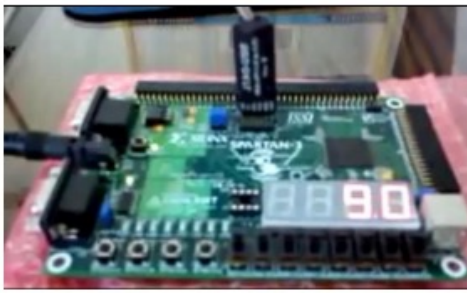
CÓDIGO EN VHDL DEL SISTEMA

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

entity contador is
   Port (RELOJ : in STD_LOGIC;
         AN0, AN1, AN2, AN3 : OUT STD_LOGIC;
         L : out std_logic_vector (6 downto 0));
end contador;

architecture behavioral of contador is
    signal segundo : std_logic;
    signal rapido : std_logic;
    signal sube: std_logic_vector(3 downto 0);
    signal baja:   std_logic_vector(3 downto 0);
    signal subibaja:   std_logic_vector(3 downto 0);
begin
```

```
divisor: process (reloj)
    variable cuenta: std_logic_vector(27 downto 0) :=  x"0000000";
begin
    if rising_edge (reloj) then
        if cuenta =x"48009E0" then
            cuenta :=  x"0000000";
        else
            cuenta := cuenta+1;
        end if;
    end if;
    segundo <= cuenta(24);
    rapido <= cuenta(10);
end process;

contador: process (segundo)
    variable arriba: std_logic_vector(3 downto 0) :=  "0000";
    variable abajo: std_logic_vector(3 downto 0) :=  "1001";
begin
    if rising_edge (segundo) then
        if arriba = "1001" then
            arriba := "0000";
        else
            arriba := arriba+1;
        end if;
        if abajo = "0000" then
            abajo := "1001";
        else
            abajo := abajo - 1;
        end if;
    end if;
    sube <= arriba;
    baja <= abajo;
end process;
```

```
    muxy: process (rapido)
    begin
        case rapido is
            when '0' =>   subibaja <= sube;
            when others =>   subibaja <= baja;
        end case;
    end process;

    seledisplay: process (rapido)
    begin
        case rapido is
            when '0' =>
                an0 <= '0';
                an1 <= '1';
                an2 <= '1';
                an3 <= '1';
            when others =>
                an0 <= '1';
                an1<= '0';
                an2<= '1';
                an3<= '1';
        end case;
    end process;

  with subibaja select
      L <= "1000000" when "0000",   -- 0
           "1111001" when "0001",   -- 1
           "0100100" when "0010",   -- 2
           "0110000" when "0011",   -- 3
           "0011001" when "0100",   -- 4
           "0010010" when "0101",   -- 5
           "0000010" when "0110",   -- 6
           "1111000" when "0111",   -- 7
           "0000000" when "1000",   -- 8
           "0010000" when "1001",   -- 9
           "1000000" when others;  -- cero

end behavioral;
```

Como podemos observar en las imágenes anteriores, el contador descendente (9-0) corresponde al primer display de 7 segmentos encendido y por consiguiente el segundo display de 7 segmentos encendido corresponde al contador ascendente (0-9).



Y después de realizar el ciclo, este contador vuelve a iniciar.

REVISA EL VIDEO
DE LA PRÁCTICA AQUÍ

PRÁCTICA

9

DISEÑO DE CRONÓMETROS DIGITALES

OBJETIVO

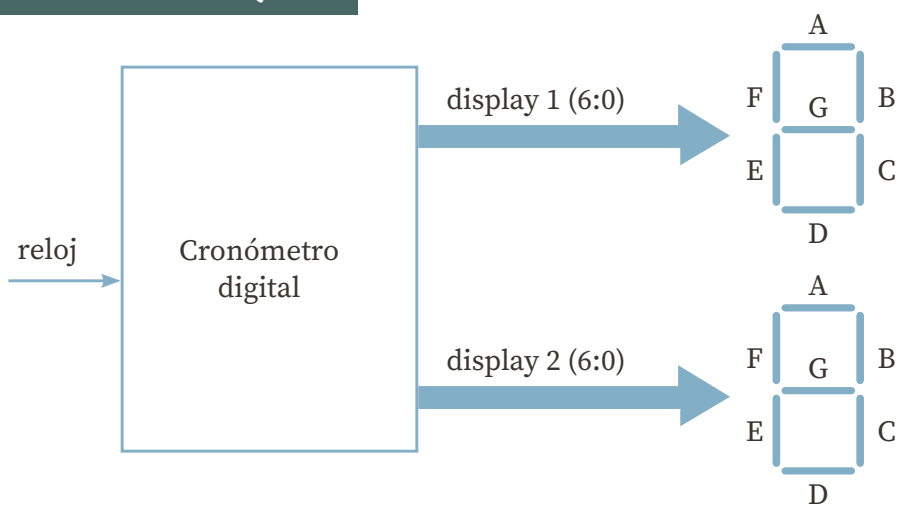
Comprender utilizando lenguaje VHDL, el funcionamiento y manejo de contadores, cuando la entrada de reloj sea una cuenta específica.

ESPECIFICACIONES

Se requieren el diseño y construcción de un cronómetro digital que en 2 displays de 7 segmentos se visualice la cuenta, empezando en 00 y al llegar a 59, se reinicie esta.

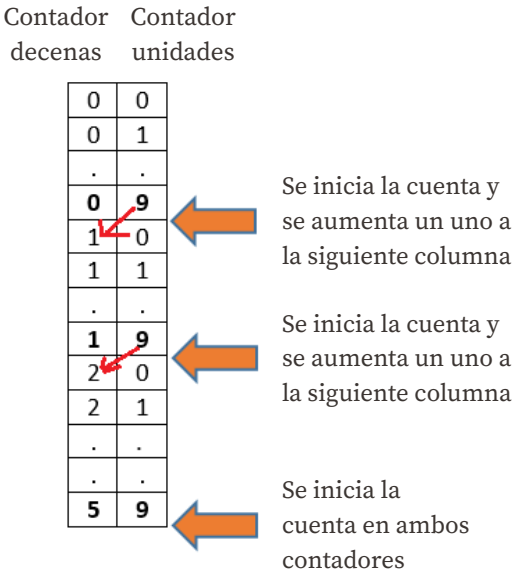
La siguiente figura muestra el diagrama del bloque de este sistema.

DIAGRAMA DE BLOQUES



ANÁLISIS DE LA CUENTA

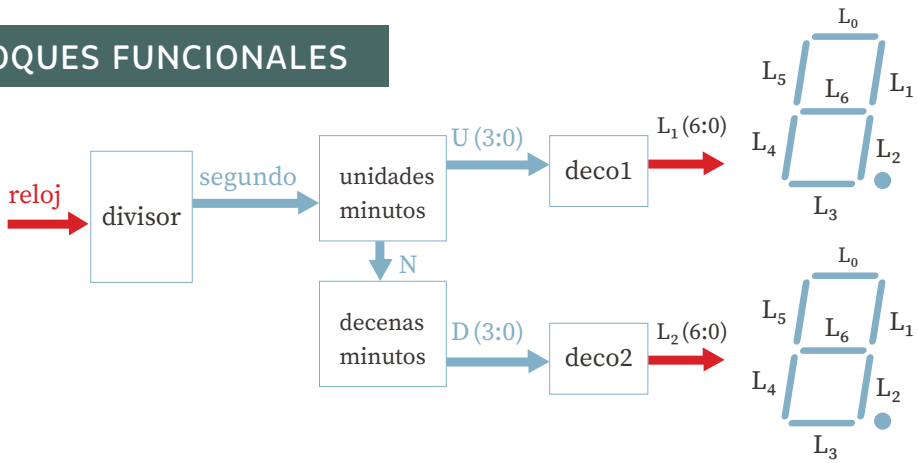
Los bloques funcionales que integran al sistema cronómetro dependen del tipo de conexión que tengan los displays de 7 segmentos.



a) **Para tarjetas con los displays de 7 segmentos conectados en forma independiente**

Dentro del cronómetro digital existen varios bloques funcionales, los cuales serán unidos mediante señales. La siguiente figura muestra los bloques funcionales del sistema "cronometro":

BLOQUES FUNCIONALES



CÓDIGO EN VHDL DEL SISTEMA

```
library IEEE;
use IEEE.std_logic_1164.ALL;
use IEEE.std_logic_arith.ALL;
use IEEE.std_logic_unsigned.ALL;

entity cronometro is
    Port (reloj : in std_logic;
          display1, display2: out std_logic_vector (6 downto 0));
end cronometro;

architecture Behavioral of cronometro is
    signal segundo, N : std_logic;
    signal U, D : std_logic_vector(3 downto 0);
begin

    Divi: process (reloj)
        variable cuenta: std_logic_vector(27 downto 0) :=  X"0000000";
    begin
        if rising_edge (reloj) then
            if cuenta =X"48009E0" then
                cuenta :=  X"0000000";
            else
                cuenta := cuenta+1;
            end if;
        end if;
        segundo <= cuenta (24);
    end process;

    Unidades: process (segundo)
        variable cuenta: std_logic_vector( 3 downto 0) :=  "0000";
    begin
        if rising_edge (segundo) then
            if cuenta ="1001" then
```

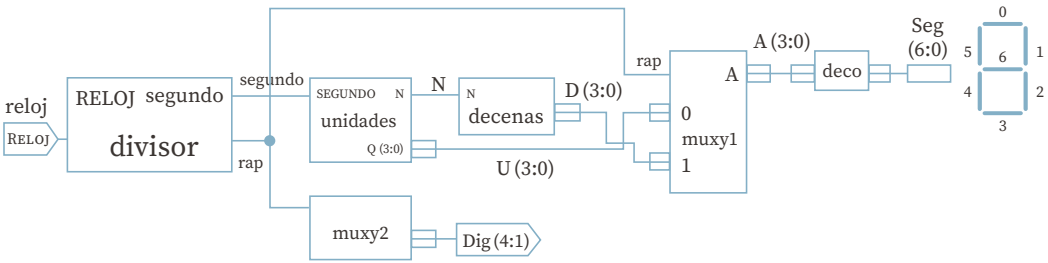
```
            cuenta :="0000";
            N <= '1';
        else
            cuenta :=  cuenta + 1;
            N <= '0';
        end if;
    end if;
    U <= cuenta;
end process;

Decenas: process (N)
    variable cuenta: std_logic_vector( 3 downto 0) :=  "0000";
begin
    if rising_edge (N) then
        if cuenta ="0101" then
            cuenta :="0000";
        else
            cuenta :=  cuenta +1;
        end if;
    end if;
    D <= cuenta;
end process;

with U select
    display1 <= "1000000" when "0000",   -- 0
                "1111001" when "0001",   -- 1
                "0100100" when "0010",   -- 2
                "0110000" when "0011",   -- 3
                "0011001" when "0100",   -- 4
                "0010010" when "0101",   -- 5
                "0000010" when "0110",   -- 6
                "1111000" when "0111",   -- 7
                "0000000" when "1000",   -- 8
                "0010000" when "1001",   -- 9
                "1000000" when others;  -- cero
```

```
    with D select
        display2 <= "1000000" when "0000",   -- 0
                    "1111001" when "0001",   -- 1
                    "0100100" when "0010",   -- 2
                    "0110000" when "0011",   -- 3
                    "0011001" when "0100",   -- 4
                    "0010010" when "0101",   -- 5
                    "1000000" when others;  --cero

end Behavioral;
```

b) Para tarjetas con los displays de 7 segmentos conectados en paralelo

Dentro del cronómetro digital existen varios bloques funcionales, los cuales serán unidos mediante señales. La siguiente figura muestra los bloques funcionales del sistema “cronometro”:

DIAGRAMA DE BLOQUES



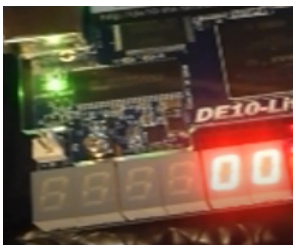
CÓDIGO EN VHDL DEL SISTEMA

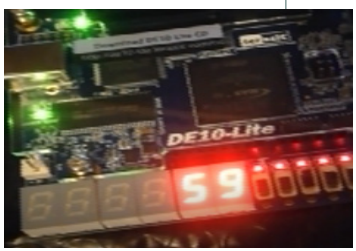
```
library IEEE;
use IEEE.std_logic_1164.ALL;
use IEEE.std_logic_arith.ALL;
use IEEE.std_logic_unsigned.ALL;

entity cronometro is
    Port (reloj : in std_logic;
          Dig : out std_logic_vector (4 downto 1);
          Seg : out std_logic_vector (6 downto 0));
    end cronometro;

architecture Behavioral of cronometro is
    signal segundo, rap, N : std_logic;
    signal U, D, A:   std_logic_vector (3 downto 0);
Begin

    Divi: process (reloj)
        variable cuenta : std_logic_vector(27 downto 0) :=  X"0000000";
    begin
        if rising_edge (reloj) then
            if cuenta =X"48009E0" then
                cuenta :=  X"0000000";
            else
                cuenta := cuenta+1;
            end if;
        end if;
        segundo <= cuenta (24);
        rap <= cuenta (10);
    end process;
```

```
unidades: process (segundo)
    variable cuenta: std_logic_vector( 3 downto 0) :=  "0000";
begin
     if rising_edge (segundo) then
        if cuenta ="1001" then
            cuenta :="0000";
            N <= '1';
        else
            cuenta :=  cuenta +1;
            N <= '0';
        end if;
    end if;
    U <= cuenta;
end process;

decenas: process (N)
    variable cuenta: std_logic_vector( 3 downto 0) :=  "0000";
begin
    if rising_edge (N) then
        if cuenta ="0101" then
            cuenta :="0000";
        else
            cuenta :=  cuenta +1;
        end if;
    end if;
    D <= cuenta;
end process;

muxy1: process (rap)
begin
    case rap is
        when '0' =>
            A <= U;
        when others =>
            A <= D;
    end case;
end process;
```

```
    muxy2: process (rap)
    begin
        case rap is
            when '0' =>
                Dig <= "1110";
            when others =>
                Dig <= "1101";
        end case;
    end process;

    with A select
        Seg <=  "1000000" when "0000",   -- 0
                "1111001" when "0001",   -- 1
                "0100100" when "0010",   -- 2
                "0110000" when "0011",   -- 3
                "0011001" when "0100",   -- 4
                "0010010" when "0101",   -- 5
                "0000010" when "0110",   -- 6
                "1111000" when "0111",   -- 7
                "0000000" when "1000",   -- 8
                "0010000" when "1001",   -- 9
                "1000000" when others;  -- cero

end Behavioral;
```

REVISA EL VIDEO DE LA PRÁCTICA AQUÍ





PRÁCTICA

10 DISEÑO DE REGISTROS DE CORRIMIENTO

OBJETIVO

Comprender el funcionamiento y manejo de los registros de corrimiento utilizando lenguaje VHDL.

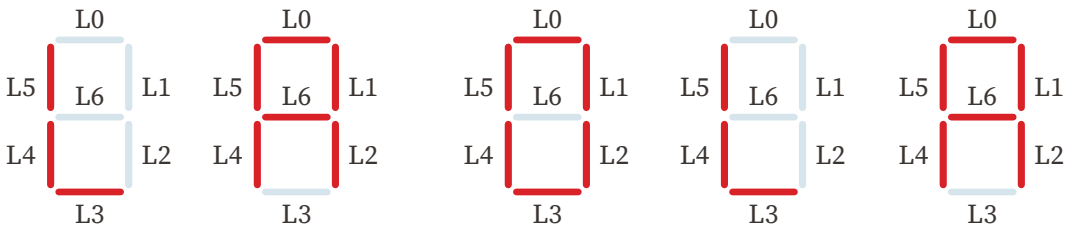
INTRODUCCIÓN

Un registro de corrimiento es un conjunto de flip-flops conectados de tal forma que los números binarios almacenados en él son desplazados con cada pulso de reloj aplicado.

ESPECIFICACIONES

Se requiere el diseño y construcción de un sistema digital que despliegue cualquier mensaje que se vea recorrer en el display de 7 segmentos.

1. Se requiere seleccionar el mensaje que se verá en los displays de 7 segmentos, por ejemplo: **LA OLA**
2. Se analizan los ceros y unos que se requieren para visualizar el mensaje.



L6	L5	L4	L3	L2	L1	L0
1	0	0	0	1	1	1

L6	L5	L4	L3	L2	L1	L0
0	0	0	1	0	0	0

L6	L5	L4	L3	L2	L1	L0
1	0	0	0	0	0	0

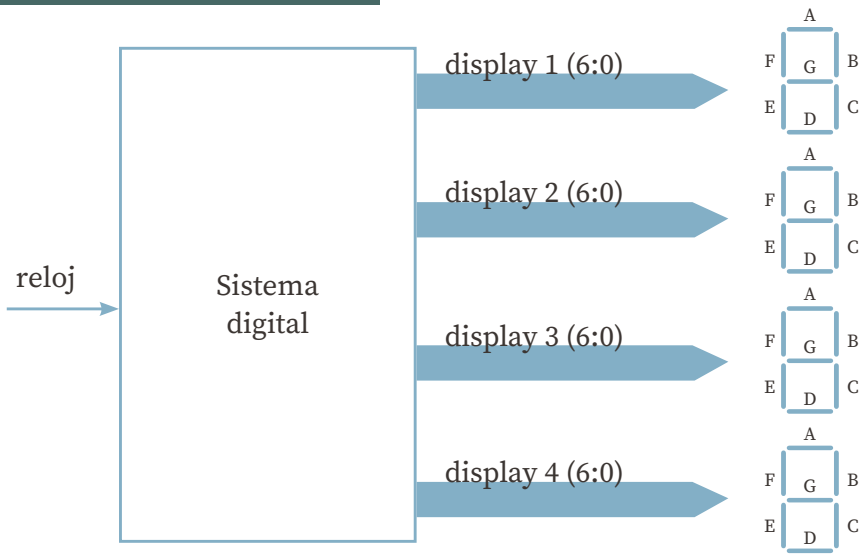
L6	L5	L4	L3	L2	L1	L0
1	0	0	0	1	1	1

L6	L5	L4	L3	L2	L1	L0
0	0	0	1	0	0	0

3. Se analiza el número de registros requerido para cada carácter (letras y espacios).

Para este caso, se requieren de 8 registros (del R_0 al R_7), el número 7 en binario es 111, por lo que se requiere un **contador de 3 bits.** Con el fin de que el mensaje salga de carácter en carácter.

DIAGRAMA DE BLOQUES

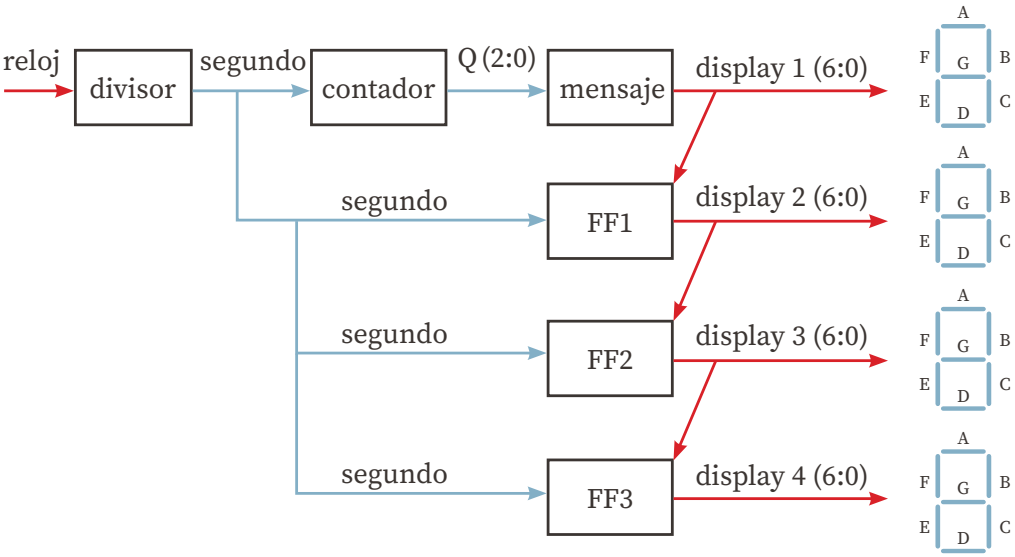


Análisis de los caracteres utilizados

Dentro del sistema digital se tienen varios bloques funcionales, los cuales serán unidos **mediante señales.** Los bloques funcionales que integran al sistema dependen del tipo de conexión que tengan los displays de 7 segmentos.

a) Para tarjetas con los displays de 7 segmentos conectados en forma independiente

BLOQUES FUNCIONALES



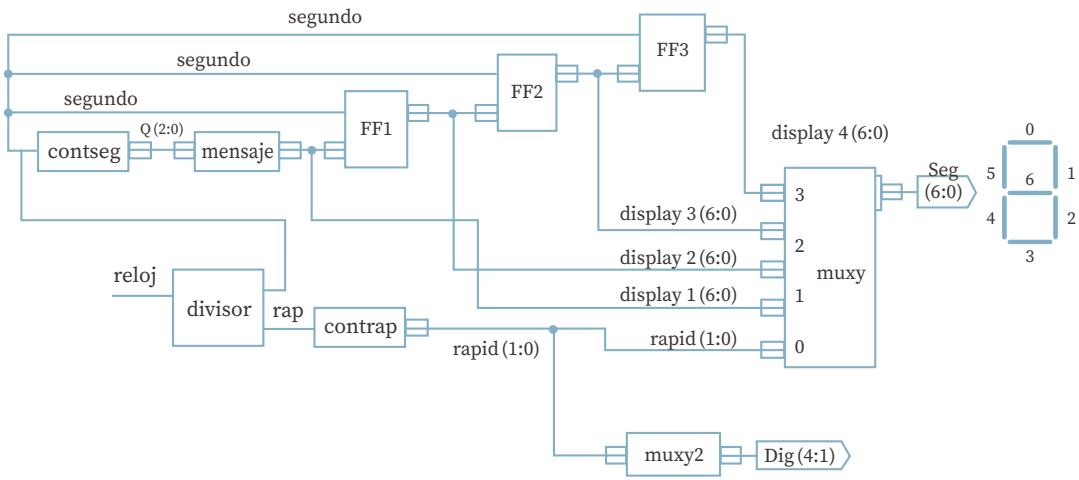
CÓDIGO EN VHDL DEL SISTEMA

```
library IEEE;
use IEEE.std_logic_1164.ALL;
use IEEE.std_logic_arith.ALL;
use IEEE.std_logic_unsigned.ALL;

entity corre is
    Port ( reloj : in STD_LOGIC;
    display1, display2, display3, display4 : inout std_logic_vector (6 downto 0));
end corre;

architecture Behavioral of corre is
    signal segundo : std_logic;
    signal Q : std_logic_vector (2 downto 0);
begin
```

```
divisor : process (reloj)
    variable cuenta: std_logic_vector(27 downto 0) :=  X"0000000";
begin
    if rising_edge (reloj) then
        if cuenta = X"48009E0" then
            cuenta :=  X"0000000";
        else
            cuenta := cuenta+1;
        end if;
    end if;
    segundo <= cuenta (22);
end process;

contador : process (segundo)
    variable cuenta: std_logic_vector( 2 downto 0) := "000";
begin
    if rising_edge (segundo) then
        cuenta :=  cuenta + 1 ;
    end if;
    Q <= cuenta;
end process;

with Q select  --------mensaje
    display1 <= "1000111" when "000",    -- L
                "0001000" when "001",    -- A
                "1000000" when "011",    -- O
                "1000111" when "100",    -- L
                "0001000" when "101",    -- A
                "1111111" when others;  -- espacios

FF1 : process (segundo)
begin
    if rising_edge (segundo) then
        display2 <= display1;
    end if;
end process;
```

```
    FF2 : process (segundo)
    begin
        if rising_edge (segundo) then
            display3 <= display2;
        end if;
    end process;

    FF3 : process (segundo)
    begin
        if rising_edge (segundo) then
            display4 <= display3;
        end if;
    end process;

end Behavioral;
```

b) Para tarjetas con los displays de 7 segmentos conectados en paralelo

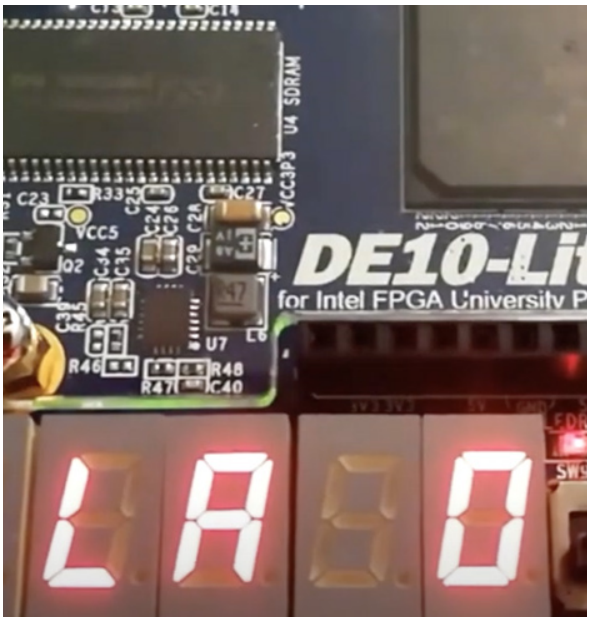


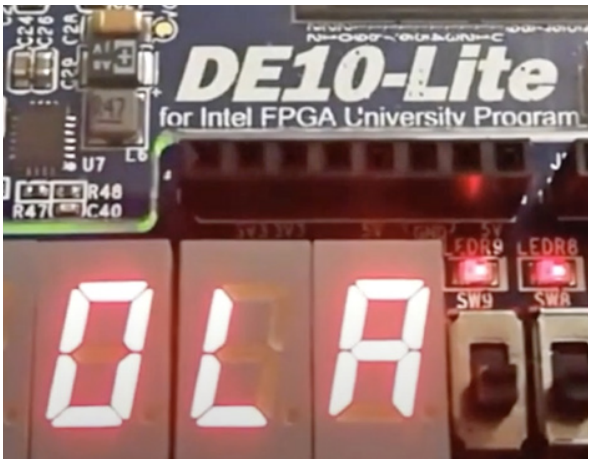
CÓDIGO EN VHDL DEL SISTEMA ISE

```
library IEEE;
use IEEE.std_logic_1164.ALL;
use IEEE.std_logic_arith.ALL;
use IEEE.std_logic_unsigned.ALL;

entity corre is
    Port (reloj : in std_logic;
          Seg : out std_logic_vector (6 downto 0);
          Dig : out std_logic_vector (3 downto 0));
end corre;

architecture Behavioral of corre is
    signal segundo, rap : std_logic;
    signal Q: std_logic_vector (2 downto 0);
    signal rapid: std_logic_vector (1 downto 0);
    signal display1, display2, display3, display4 : std_logic_vector (6 downto 0);
begin

    divisor: process (reloj)
        variable cuenta: std_logic_vector (27 downto 0) := X"0000000";
    begin
        if rising_edge (reloj) then
            if cuenta = X"48009E0" then    -- es el tiempo para 1.51 seg
                cuenta := X"0000000";
            else
                cuenta := cuenta+1;
            end if;
        end if;
        segundo <= cuenta (22);
        rap <= cuenta (10);
    end process;
```

```
conseg :  process (segundo)
    variable cuenta: std_logic_vector (2 downto 0):= "000";
begin
    if rising_edge (segundo) then
        cuenta := cuenta+1;
    end if;
    Q <= cuenta;
end process;

with Q select  --------mensaje
    display1 <= "1000111" when "000",    -- L
                "0001000" when "001",    -- A
                "1000000" when "011",    -- O
                "1000111" when "100",    -- L
                "0001000" when "101",    -- A
                "1111111" when others;  -- espacios

FF1 : process (segundo)
begin
    if rising_edge (segundo) then
        display2 <= display1;
    end if;
end process;

FF2 : process (segundo)
begin
    if rising_edge (segundo) then
         display3 <= display2;
     end if;
end process;
```

```
    FF3 : process (segundo)
    begin
        if rising_edge (segundo) then
            display4 <= display3;
        end if;
    end process;

    contrap: process (rap)
        variable cuenta: std_logic_vector (1 downto 0) := "00";
    begin
        if rising_edge (rap) then
            cuenta := cuenta+1;
        end if;
        rapid <= cuenta;
    end process;

    with rapid select ----- muxy2
        Dig <= "1110" when "00",     -- 1
               "1101" when "01",     -- 2
               "1011" when "10",     -- 3
               "0111" when others; -- 4

    Muxy: process (rapid)
    begin
        case rapid is
            when "11" =>    Seg <= display4;
            when "10" =>    Seg <= display3;
            when "01" =>    Seg <= display2;
            when others => Seg <= display1;
        end case;
    end process;

end Behavioral;
```





REVISA EL VIDEO
DE LA PRÁCTICA AQUÍ

PRÁCTICA

11

DISEÑO DE CARTAS ASM

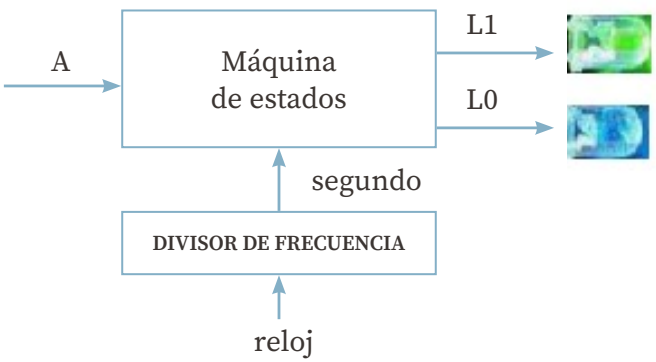
OBJETIVO

Comprender mediante el uso del lenguaje VHDL, el funcionamiento y manejo de cartas ASM para la creación de máquinas de estados, las cuales controlan sistemas secuenciales con gran diversidad de salidas. Aprender a crear sus propias bibliotecas.

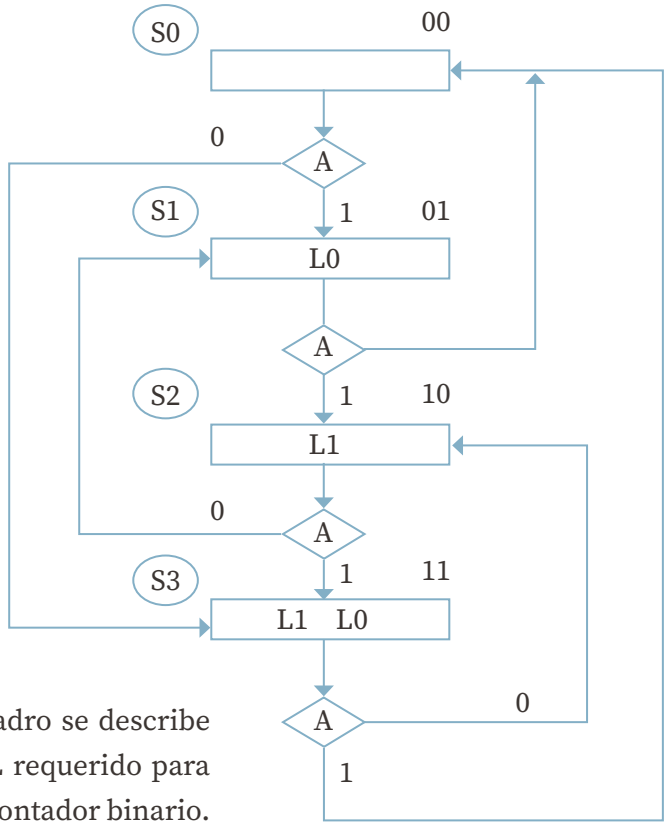
ESPECIFICACIONES

Diseñar y construir un contador que cuente en forma ascendente/descendente los números binarios de cero a tres; cuando el valor de la variable A sea uno, la cuenta se efectuará de forma ascendente, y si el valor de la variable A es cero, la cuenta se efectuará en forma descendente. El cambio de cuenta de ascendente a descendente y viceversa se puede efectuar en cualquier estado.

DIAGRAMA DE BLOQUES



CARTA ASM



En el siguiente cuadro se describe el código en VHDL requerido para la entidad de este contador binario.

CÓDIGO EN VHDL

```
library IEEE;
use IEEE.std_logic_1164.ALL;
use IEEE.std_logic_arith.ALL;
use IEEE.std_logic_unsigned.ALL;

entity ASM is
    port (reloj : in std_logic;
          A : in std_logic;
          L: out std_logic_vector (1 downto 0));
end ASM;
```

En el siguiente cuadro se observa el código en VHDL requerido para la arquitectura de este contador binario.

CÓDIGO EN VHDL

```
architecture Behavioral of ASM is
    type estados is (s0,s1,s2,s3);
    signal epresente, esiguiente : estados;
    signal segundo : std_logic;
begin

    divisor: process (reloj)
        variable cuenta: std_logic_vector (27 downto 0):=X"0000000";
    begin
        if rising_edge (reloj) then
            if cuenta = X"48009E0" then   --es el tiempo para 1.51 seg
                cuenta := X"0000000";
            else
                cuenta := cuenta+1;
            end if;
        end if;
        segundo <= cuenta (24);
    end process;

    MdE1: process (segundo)      --- transición de un estado a otro
    begin
        if rising_edge (segundo) then
            epresente <= esiguiente;
        end if;
    end process;
```

```
MdE2: process (epresente, A)  -- contenido en cada estado
begin
    case epresente is
        when s0 =>
            L <= "00";
            if A='1' then
                esiguiente<= s1;
            else
                esiguiente <= s3;
            end if;
        when s1 =>
            L <= "01";
            if A='1' then
                esiguiente<= s2;
            else
                esiguiente<= s0;
            end if;
        when s2 =>
            L <= "10";
            if A='1' then
                esiguiente<= s3;
            else
                esiguiente <= s1;
            end if;
        when s3 =>
            L <= "11";
            if A='1' then
                esiguiente<= s0;
            else
                esiguiente<= s2;
            end if;
    end case;
end process;

end Behavioral;
```

PRÁCTICA

12 DISEÑO DE UN RELOJ DIGITAL

OBJETIVO

Aprender el manejo de la descripción por comportamiento de varios bloques funcionales dentro del lenguaje VHDL, la cual consiste, como su nombre indica, en describir el comportamiento de algún sistema digital.

ESPECIFICACIONES

Se requiere el diseño y construcción de un reloj digital que cuente con 4 displays, los dos primeros son para visualizar las horas y los siguientes dos, para los minutos. Cada vez que se llegue a 23 horas con 59 minutos, se volverá a empezar la cuenta. La siguiente figura muestra el diagrama del bloque de este sistema.

ANÁLISIS DE LA CUENTA

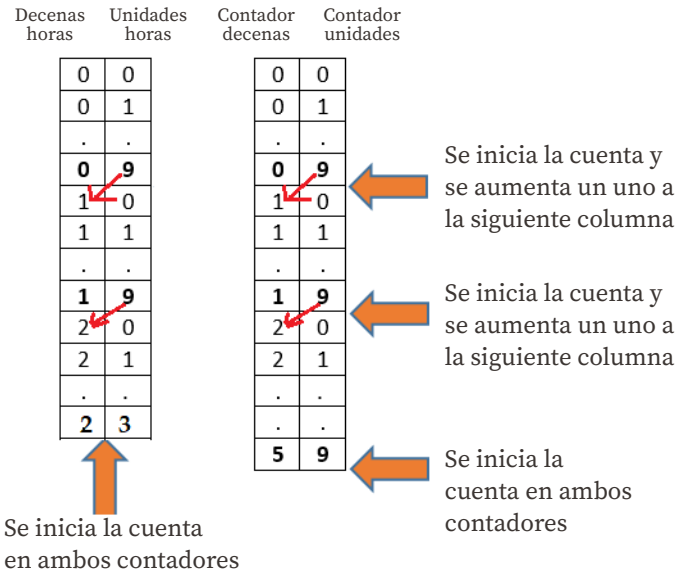
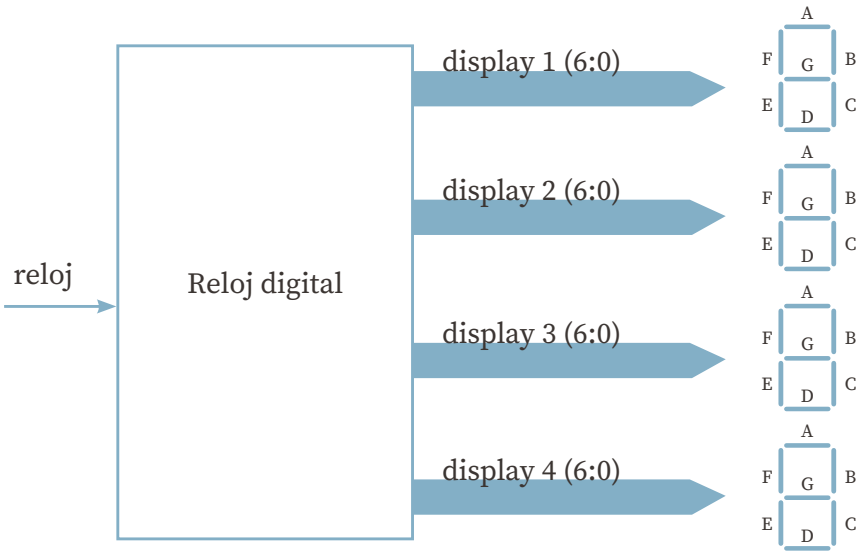


DIAGRAMA DE BLOQUE

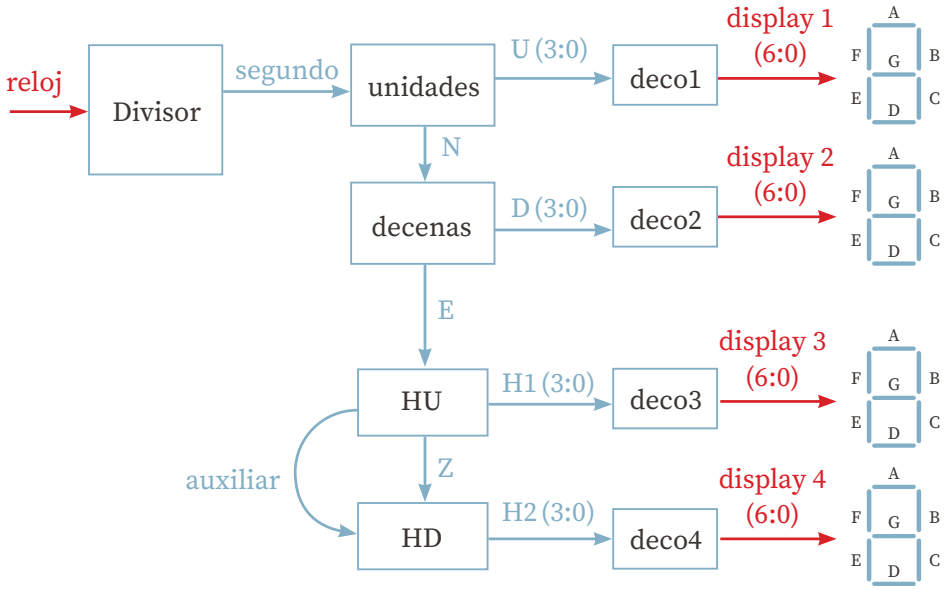


Dentro del reloj digital existen varios bloques funcionales, los cuales serán unidos mediante señales.

Existen dos tipos de tarjetas de desarrollo, unas tienen unidos los displays de 7 segmentos en forma independiente y, otras, en paralelo.

a) **Para tarjetas de desarrollo con displays de 7 segmentos conectados en forma independiente, el diagrama de bloques será el siguiente: las líneas que unen entradas y salidas se encuentran en color rojo, mientras que las señales están en color azul.**

BLOQUES FUNCIONALES



Pines de entrada y salida, y señales de interconexión

CÓDIGO VHDL DEL SISTEMA

```
library IEEE;
use IEEE.std_logic_1164.ALL;
use IEEE.std_logic_arith.ALL;
use IEEE.std_logic_unsigned.ALL;

entity relojdigital is
    Port (reloj : in std_logic;
        display1, display2, display3, display4 : out std_logic_vector (6 downto 0));
end relojdigital;
```

```
architecture Behavioral of relojdigital is
    signal segundo, N, E, auxiliar, Z: STD_LOGIC;
    signal U,D,H1,H2: std_logic_vector(3 downto 0):="0000";
Begin

    Divisor : process (reloj)
        variable cuenta: std_logic_vector(27 downto 0) :=  X"0000000";
    begin
        if rising_edge (reloj) then
            if cuenta =X"48009E0" then
                cuenta :=  X"0000000";
            else
                cuenta := cuenta+1;
            end if;
        end if;
        segundo <= cuenta (22);
    end process;

    Unidades: process (segundo)
        variable cuenta: std_logic_vector(3 downto 0) :=  "0000";
    begin
        if rising_edge (segundo) then
            if cuenta ="1001" then
                cuenta :="0000";
                N <= '1';
            else
                cuenta := cuenta +1;
                N <= '0';
            end if;
        end if;
        U <= cuenta;
    end process;
```

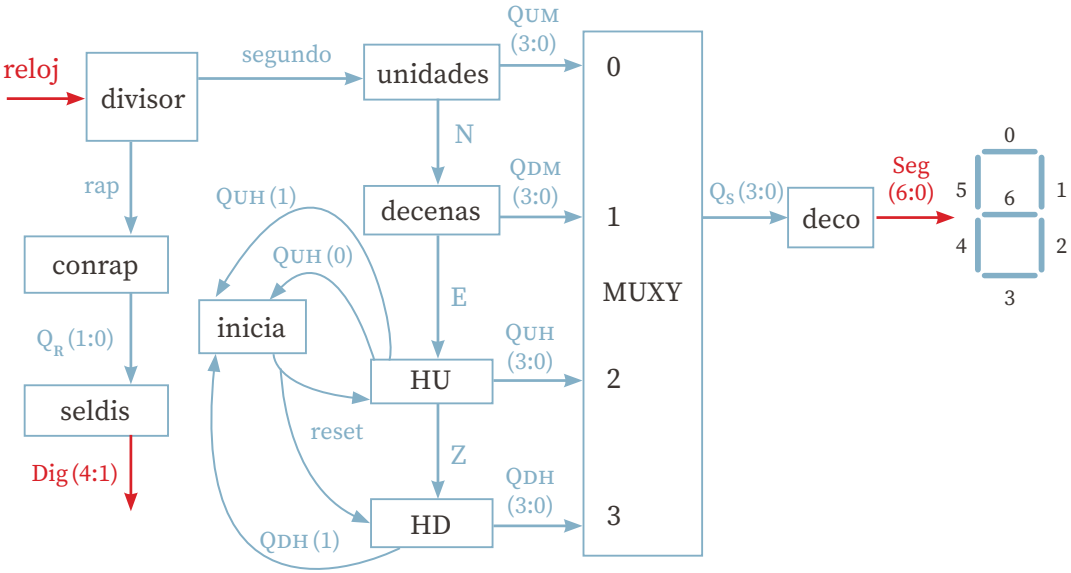
```
Decenas: process (N)
    variable cuenta: std_logic_vector( 3 downto 0) :=  "0000";
begin
    if rising_edge (N) then
        if cuenta ="0101" then
            cuenta :="0000";
            E <= '1';
        else
            cuenta :=  cuenta +1;
            E <= '0';
        end if;
    end if;
    D <= cuenta;
end process;

HU: process (E)
    variable cuenta: std_logic_vector(3 downto 0) :=  "0000";
begin
    if rising_edge (E) then
        if cuenta ="1001" then
            cuenta := "0000";
            Z <= '1';
            AUX <= '0';
        elsif H1 = "0011" AND H2 = "0010" then
            cuenta := "0000";
            AUX <= '1';
            Z <= '1';
        else
            cuenta := cuenta + 1;
            Z <= '0';
            AUX <= '0';
        end if;
    end if;
    H1 <= cuenta;
end process;
```

```
HD: process (Z)
   variable cuenta: std_logic_vector( 3 downto 0) :=  "0000";
begin
   if rising_edge (Z) then
      if  AUX = '1' then
         cuenta := "0000";
      else
         cuenta := cuenta + 1;
      end if;
   end if;
   H2 <= cuenta;
end process;

with U select
   display1 <= "1000000" when "0000",   -- 0
               "1111001" when "0001",   -- 1
               "0100100" when "0010",   -- 2
               "0110000" when "0011",   -- 3
               "0011001" when "0100",   -- 4
               "0010010" when "0101",   -- 5
               "0000010" when "0110",   -- 6
               "1111000" when "0111",   -- 7
               "0000000" when "1000",   -- 8
               "0010000" when "1001",   -- 9
               "1000000" when others;  -- cero

with D select
   display2 <= "1000000" when "0000",   -- 0
               "1111001" when "0001",   -- 1
               "0100100" when "0010",   -- 2
               "0110000" when "0011",   -- 3
               "0011001" when "0100",   -- 4
               "0010010" when "0101",   -- 5
               "0000010" when "0110",   -- 6
               "1111000" when "0111",   -- 7
               "0000000" when "1000",   -- 8
               "0010000" when "1001",   -- 9
               "1000000" when others;  -- cero
```

```
    with H1 select
        display3 <= "1000000" when "0000",   -- 0
                    "1111001" when "0001",   -- 1
                    "0100100" when "0010",   -- 2
                    "0110000" when "0011",   -- 3
                    "0011001" when "0100",   -- 4
                    "0010010" when "0101",   -- 5
                    "0000010" when "0110",   -- 6
                    "1111000" when "0111",   -- 7
                    "0000000" when "1000",   -- 8
                    "0010000" when "1001",   -- 9
                    "1000000" when others;  -- cero

    with H2 select
        display4 <= "1000000" when "0000",   -- 0
                    "1111001" when "0001",   -- 1
                    "0100100" when "0010",   -- 2
                    "0110000" when "0011",   -- 3
                    "0011001" when "0100",   -- 4
                    "0010010" when "0101",   -- 5
                    "0000010" when "0110",   -- 6
                    "1111000" when "0111",   -- 7
                    "0000000" when "1000",   -- 8
                    "0010000" when "1001",   -- 9
                    "1000000" when others;  -- cero

end Behavioral;
```

b) Para tarjetas de desarrollo con displays de 7 segmentos conectados en paralelo, el diagrama de bloques será el siguiente: las líneas que unen entradas y salidas se encuentran en color rojo, mientras que las señales están en color azul.

BLOQUES FUNCIONALES



CÓDIGO EN VHDL DEL SISTEMA

```
library IEEE;
use IEEE.std_logic_1164.ALL;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity relojdigital is
    Port (reloj : in STD_LOGIC;
          Dig : out std_logic_vector (4 downto 1);
          Seg : out std_logic_vector (6 downto 0));
end relojdigital;

architecture Behavioral of relojdigital is
    signal segundo, rap, N, E, Z, reset : std_logic;
    signal Qs, Qum, Qdm, Quh, Qdh: std_logic_vector (3 downto 0);
    signal QR: std_logic_vector(1 downto 0);
begin
```

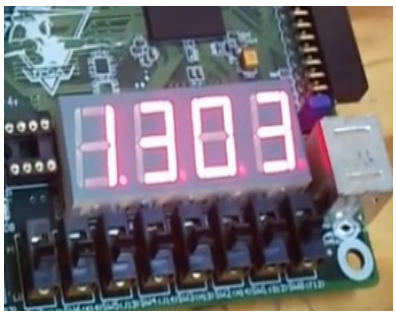
```
divisor: process (reloj)
    variable cuenta: std_logic_vector(27 downto 0) :=  X"0000000";
begin
    if rising_edge (reloj) then
        if cuenta = x"48009e0" then
            cuenta := x"0000000";
        else
            cuenta := cuenta + 1;
        end if;
    end if;
segundo <= cuenta(22);
rap <= cuenta(10);
end process;

unidades: process (segundo)
    variable cuenta: std_logic_vector(3 downto 0) := "0000";
begin
    if rising_edge (segundo) then
        if cuenta ="1001" then
            cuenta :="0000";
            N <= '1';
        else
            cuenta := cuenta +1;
            N <= '0';
        end if;
    end if;
    Qum <= cuenta;
end process;

decenas: process (N)
    variable cuenta: std_logic_vector(3 downto 0) := "0000";
begin
    if rising_edge (N) then
        if cuenta = "0101" then
            cuenta := "0000";
            E <= '1';
```

```
        else
            cuenta := cuenta +1;
            E <= '0';
        end if;
    end if;
    Qdm <= cuenta;
end process;

HoraU:  Process(E, reset)
    variable cuenta: std_logic_vector(3 downto 0):="0000";
begin
    if rising_edge(E) then
        if cuenta="1001" then
            cuenta := "0000";
            Z <= '1';
        else
            cuenta := cuenta + 1;
            Z <= '0';
        end if;
    end if;
    if reset = '1' then
        cuenta := "0000";
    end if;
    Quh <= cuenta;
    Quh(0) <= cuenta(0);
    Quh(1) <= cuenta(1);
end Process;

HoraD: Process (Z, reset)
    variable cuenta: std_logic_vector(3 downto 0):="0000";
begin
    if rising_edge(Z) then
        if cuenta="0010" then
            cuenta := "0000";
```

```
        else
            cuenta := cuenta + 1;
        end if;
    end if;
    if reset = '1' then
        cuenta := "0000";
    end if;
    Qdh <= cuenta;
end Process;

inicia: process (Quh(0), Quh(1), Qdh(1))
begin
    reset <= (Quh(0) and Quh(1) and Qdh(1));
end process;

conrap: process (rap)
    variable cuenta: std_logic_vector (1 downto 0) := "00";
begin
    if rising_edge (rap) then
        cuenta := cuenta +1;
    end if;
    QR <= cuenta;
end process;

muxy: process (QR)
begin
    if QR = "00" then
        QS<= Qum;
    elsif QR = "01" then
        QS<= Qdm;
    elsif QR = "10" then
        QS<= Quh;
    elsif QR = "11" then
        QS<= Qdh;
    end if;
end process;
```

```
seldis:  process (QR)
begin
    case QR is
        when "00" =>
            Dig<= "1110";
        when "01" =>
            Dig<= "1101";
        when "10" =>
            Dig<= "1011";
        when others =>
            Dig<= "0111";
    end case;
end process;

with  Qs select
    Seg <= "1000000" when "0000",   -- 0
           "1111001" when "0001",   -- 1
           "0100100" when "0010",   -- 2
           "0110000" when "0011",   -- 3
           "0011001" when "0100",   -- 4
           "0010010" when "0101",   -- 5
           "0000010" when "0110",   -- 6
           "1111000" when "0111",   -- 7
           "0000000" when "1000",   -- 8
           "0010000" when "1001",   -- 9
           "1000000" when others;  -- cero

end Behavioral;
```

REVISA EL VIDEO DE LA PRÁCTICA AQUÍ



PRÁCTICA

13 DISEÑO Y CONSTRUCCIÓN DE DOS SISTEMAS QUE TRABAJAN AL MISMO TIEMPO

OBJETIVO

Diseñar dos sistemas que hacen muy distintas tareas y que trabajan al mismo tiempo dentro de un mismo FPGA, utilizando código VHDL.

ESPECIFICACIONES

Diseño y construcción de 2 sistemas que trabajen al mismo tiempo.

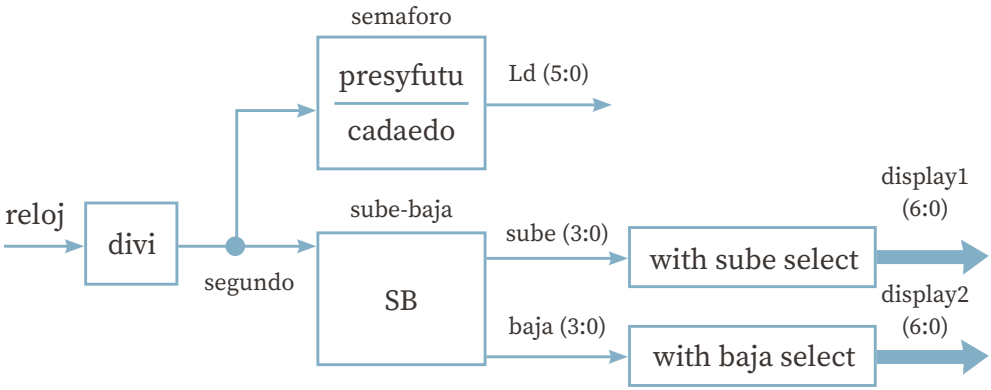
El sistema 1 requiere el diseño y construcción de dos contadores que trabajen al mismo tiempo, uno que cuente de cero a nueve y vuelva a empezar, y el otro que cuente de nueve al cero y vuelva a empezar, utilizando dos displays de 7 segmentos.

El sistema 2 es un control de semáforos en dos avenidas. Las restricciones de los semáforos son las siguientes: la luz roja debe durar 60 segundos, la luz verde 45 segundos y la luz amarilla 15 segundos. Las luces en los semáforos norte y sur se prenderán y apagarán al mismo tiempo, Las luces en los semáforos este y oeste se prenderán y apagarán al mismo tiempo.

Dentro del proyecto, existen varios bloques funcionales, los cuales serán unidos mediante señales. Existen dos tipos de tarjetas de desarrollo, unas tienen unidos los displays de 7 segmentos en forma independiente y otras, en paralelo.

a) **Para tarjetas de desarrollo con displays de 7 segmentos conectados en forma independiente, el diagrama de bloques será el siguiente:**

DIAGRAMA DE BLOQUES

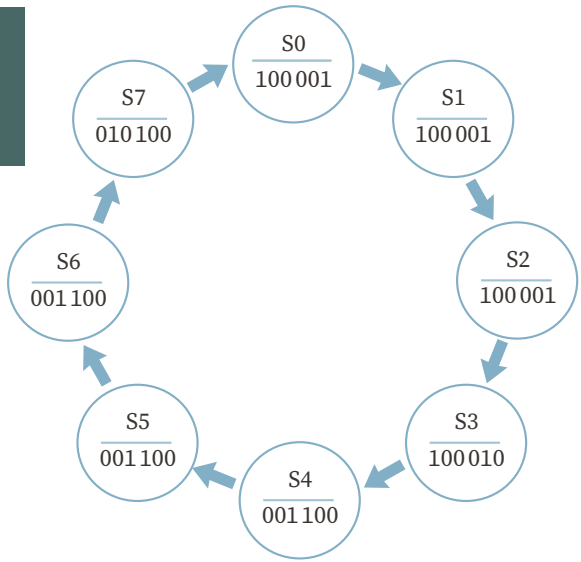


CÓDIGO EN VHDL DE LAS BIBLIOTECAS Y ENTIDAD

```
library IEEE;
use IEEE.std_logic_1164.ALL;
use IEEE.std_logic_arith.ALL;
use IEEE.std_logic_unsigned.ALL;

entity ASM is
    Port (reloj: in std_logic;
          Ld: out std_logic_vector (5 downto 0);
          displaay1: out std_logic_vector (6 downto 0);
          displaay2: out std_logic_vector (6 downto 0));
end ASM;
```

DIAGRAMA DE ESTADOS DEL SISTEMA 2 (SEMÁFOROS)

PS: Rns, Ans, Vns,
Reo, Aeo, Veo



CÓDIGO EN VHDL DE LA ARQUITECTURA

```
architecture Behavioral of ASM is
    signal segundo : std_logic;
    signal sube, baja : std_logic_vector (3 downto 0);
    type ESTADOS is (s7,s6,s5,s4,s3,s2,s1,s0);
    signal epresente, esiguiente : ESTADOS;
begin

    Divi: process (reloj)
        variable cuenta: std_logic_vector (27 downto 0):=X"0000000";
    begin
        if rising_edge (reloj) then
            if cuenta = X"48009E0" then
                cuenta := X"0000000";
            else
                cuenta := cuenta + 1;
            end if;
        end if;
        segundo <= cuenta (24);
    end process;
```

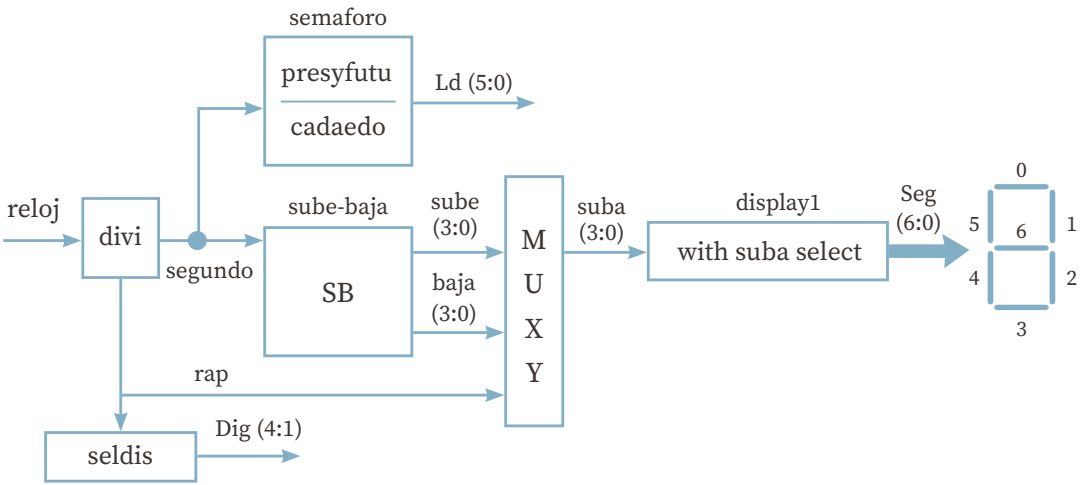
```
SB: process (segundo)
    variable arriba: std_logic_vector(3 downto 0):="0000";
    variable abajo:   std_logic_vector(3 downto 0):="1001";
begin
    if rising_edge (segundo) then
        if arriba ="1001" then
            arriba := "0000";
        else
            arriba := arriba + 1;
        end if;
        if abajo = "0000" then
            abajo := "1001";
        else
            abajo := abajo - 1;
        end if;
    end if;
    sube <= arriba;
    baja <= abajo;
end process;

with sube select
    display1 <= "1000000" when "0000",   -- 0
                "1111001" when "0001",   -- 1
                "0100100" when "0010",   -- 2
                "0110000" when "0011",   -- 3
                "0011001" when "0100",   -- 4
                "0010010" when "0101",   -- 5
                "0000010" when "0110",   -- 6
                "1111000" when "0111",   -- 7
                "0000000" when "1000",   -- 8
                "0010000" when "1001",   -- 9
                "1000000" when others;  -- cero
```

```
with baja select
    display2 <= "1000000" when "0000",   -- 0
                "1111001" when "0001",   -- 1
                "0100100" when "0010",   -- 2
                "0110000" when "0011",   -- 3
                "0011001" when "0100",   -- 4
                "0010010" when "0101",   -- 5
                "0000010" when "0110",   -- 6
                "1111000" when "0111",   -- 7
                "0000000" when "1000",   -- 8
                "0010000" when "1001",   -- 9
                "1000000" when others;  -- cero

presyfutu: process (segundo)
begin
    if rising_edge (segundo) then
        epresente <= esiguiente;
    end if;
end process;

cadaedo: process (epresente)
begin
    case epresente is
        when s0 =>
            Ld <= "100001";
            esiguiente <= s1;
        when s1 =>
            Ld <= "100001";
            esiguiente<= s2;
        when s2 =>
            Ld <= "100001";
            esiguiente <= s3;
        when s3 =>
            Ld <= "100010";
            esiguiente <= s4;
```

```
            when s4 =>
                Ld <= "001100";
                esiguiente<= s5;
            when s5 =>
                Ld <= "001100";
                esiguiente <= s6;
            when s6 =>
                Ld <= "001100";
                esiguiente <= s7;
            when others =>
                Ld <= "010100";
                esiguiente <= s0;
        end case;
    end process;

end Behavioral;
```

b) Para tarjetas de desarrollo con displays de 7 segmentos conectados en paralelo, el diagrama de bloques será el siguiente:

DIAGRAMA DE BLOQUES

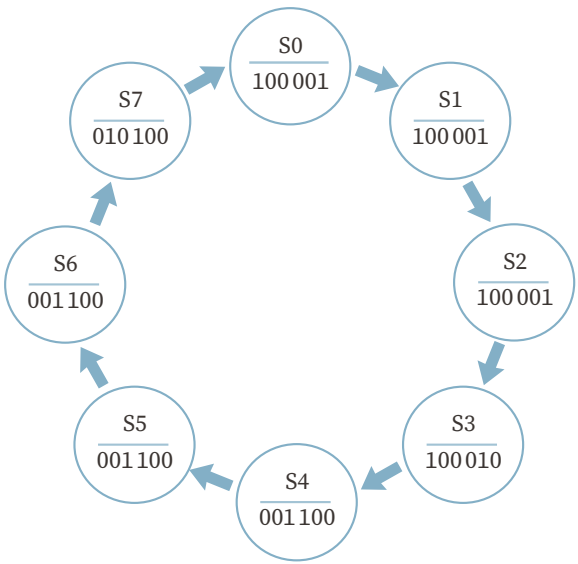


CÓDIGO EN VHDL DE LAS BIBLIOTECAS Y ENTIDAD

```
library IEEE;
use IEEE.std_logic_1164.ALL;
use IEEE.std_logic_arith.ALL;
use IEEE.std_logic_unsigned.ALL;

entity ASM is
    Port (reloj: in std_logic;
          Ld: out std_logic_vector (5 downto 0);
          Seg: out std_logic_vector (6 downto 0);
          Dig: out std_logic_vector (4 downto 1));
end ASM;
```

DIAGRAMA DE ESTADOS DEL SISTEMA 2 (SEMÁFOROS)

PS: Rns, Ans, Vns,
Reo, Aeo, Veo

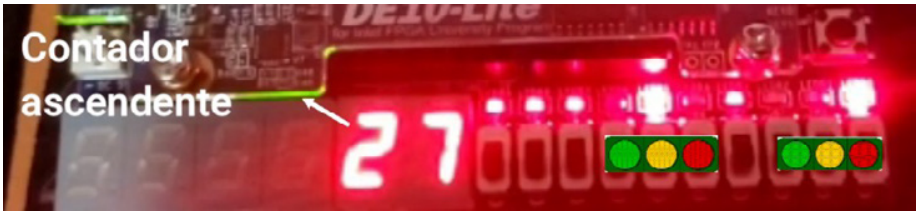


CÓDIGO EN VHDL DE LA ARQUITECTURA

```
architecture Behavioral of ASM is
    signal segundo, rap : std_logic;
    signal sube, baja, suba : std_logic_vector (3 downto 0);
    type ESTADOS is (s7,s6,s5,s4,s3,s2,s1,s0);
    signal epresente, esiguiente : ESTADOS;
begin

    Divi: process (reloj)
        variable cuenta: std_logic_vector (27 downto 0) := X"0000000";
    begin
        if rising_edge (reloj) then
            if cuenta = X"48009E0" then
                cuenta := X"0000000";
             else
                cuenta := cuenta+1;
            end if;
        end if;
        segundo <= cuenta (24);
    end process;

    SB: process (segundo)
        variable arriba: std_logic_vector(3 downto 0):="0000";
        variable abajo:   std_logic_vector(3 downto 0):="1001";
    begin
        if rising_edge (segundo) then
            if arriba ="1001" then
                arriba := "0000";
            else
                arriba := arriba + 1;
            end if;
            if abajo = "0000" then
                abajo := "1001";
```

```
        else
            abajo := abajo - 1;
        end if;
    end if;
    sube <= arriba;
    baja <= abajo;
end process;

with suba select
    Seg <= "1000000" when "0000", --- 0
           "1111001" when "0001",  -- 1
           "0100100" when "0010",   --2
           "0110000" when "0011",   --3
           "0011001" when "0100",   --4
           "0010010" when "0101",   --5
           "0000010" when "0110",   --6
           "1111000" when "0111",   --7
           "0000000" when "1000",   --8
           "0010000" when "1001",   --9
           "1000000" when others;  -- cero

presyfutu: process (segundo)
begin
    if rising_edge (segundo) then
        epresente <= esiguiente;
    end if;
end process;

cadaedo: process (epresente)
begin
    case epresente is
        when s0 =>
            Ld <= "100001";
            esiguiente <= s1;
        when s1 =>
            Ld <= "100001";
            esiguiente<= s2;
```

```
            when s2 =>
                Ld <= "100001";
                esiguiente <= s3;
            when s3 =>
                Ld <= "100010";
                Esiguiente <= s4;
            when s4 =>
                Ld <= "001100";
                esiguiente<= s5;
            when s5 =>
                Ld <= "001100";
                esiguiente <= s6;
            when s6 =>
                Ld <= "001100";
                esiguiente <= s7;
            when others =>
                Ld <= "010100";
                esiguiente <= s0;
        end case;
    end process;

    seldis: process (rap)
    begin
        case rap is
            when '0' =>
                Dig <= "1110";
            when others =>
                Dig <= "1101";
            end case;
    end process;

end Behavioral;
```



REVISA EL VIDEO
DE LA PRÁCTICA AQUÍ

PRÁCTICA

14 DISEÑO Y CONSTRUCCIÓN DE VARIOS MENSAJES QUE UTILIZAN LOS MISMOS RECURSOS

OBJETIVO

Aprender a diseñar y construir, mediante el uso del código VHDL, varios mensajes con visualización en displays de siete segmentos, utilizando la misma tarjeta.

ESPECIFICACIONES

Diseñar un sistema capaz de mostrar un mensaje en corrimiento en 4 displays de 7 segmentos, con la posibilidad que un dipswitch pueda cambiar el mensaje que se muestra en dichos displays.

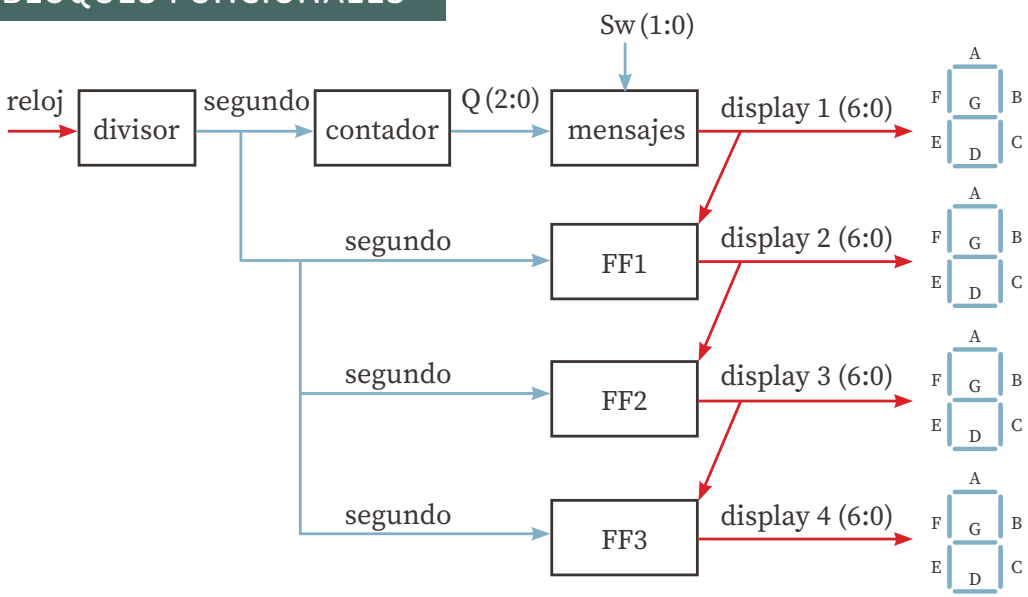
El sistema será utilizado por un profesor que requiere entregar su calificación a un grupo de alumnos, pero debido a medidas sanitarias, tendrá que pasar a cada alumno por separado. Por lo anterior, instalaron cuatro displays de siete segmentos en la puerta, para que los alumnos pudieran reconocer el nombre de pila de la persona que tiene que entrar a recoger su calificación. El profesor podrá seleccionar el nombre de pila desde su escritorio accionando un dipswitch.

Nombres de pila de alumnos que van a entrar:

- Jeni
- Hanna
- Susy
- Sebas
- Fer
- Dan

a) **Para tarjetas con los displays de 7 segmentos conectados en forma independiente**

BLOQUES FUNCIONALES



CÓDIGO EN VHDL DEL SISTEMA

```
library IEEE;
use IEEE.std_logic_1164.ALL;
use IEEE.std_logic_arith.ALL;
use IEEE.std_logic_unsigned.ALL;

entity corre is
    Port (reloj : in STD_LOGIC;
          sw : in STD_LOGIC_vector (1 downto 0);
          display1, display2, display3, display4 : inout std_logic_vector (6 downto 0));
end corre;
```

```
architecture Behavioral of corre is
    signal segundo : std_logic;
    signal Q : std_logic_vector (2 downto 0);
begin

    divisor : process (reloj)
       variable cuenta: std_logic_vector(27 downto 0) :=  X"0000000";
    begin
       if rising_edge (reloj) then
          if cuenta =X"48009E0" then
             cuenta :=  X"0000000";
          else
             cuenta := cuenta + 1;
          end if;
       end if;
       segundo <= cuenta (22);
    end process;

    contador : process (segundo)
       variable cuenta: std_logic_vector( 2 downto 0) := "000";
    begin
       if rising_edge (segundo) then
          cuenta :=  cuenta + 1 ;
       end if;
       Q <= cuenta;
    end process;

   mensajes : Process(sw)
    begin
       case sw is
          when "110" => --- JENI mensaje 1
             if Q = "000" then
                display1 <= "1100001"; --- J
             elsif Q = "001" then
                display1 <= "0000110"; --- E
             elsif Q = "010" then
                display1 <= "1001000"; --- N
```

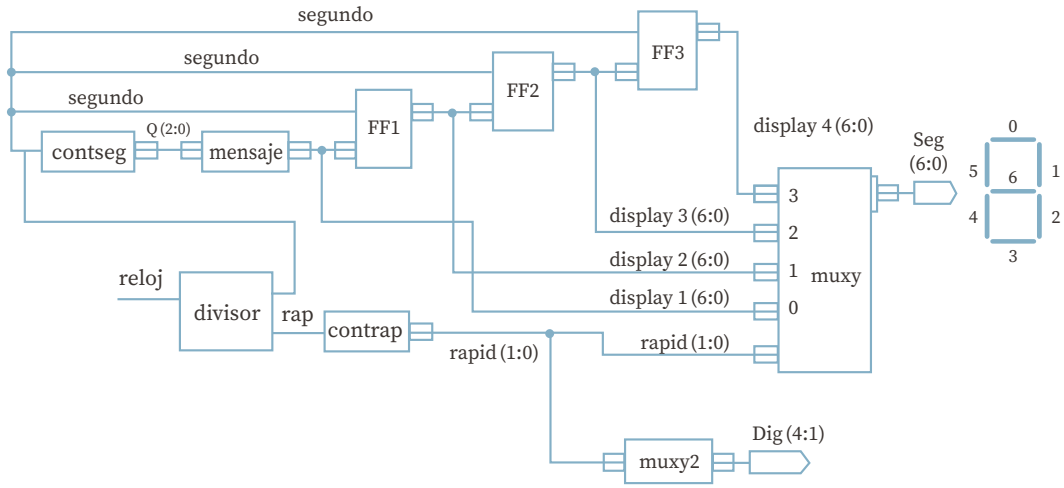
```
        elsif Q = "011" then
            display1 <= "1001111"; --- I
        else
            display1 <= "1111111"; --- espacios
        end if;
    when "101" => --- SEbAS
        if Q = "000" then
            display1 <= "0010010"; --- S
        elsif Q = "001" then
            display1 <= "0000110"; --- E
        elsif Q = "010" then
            display1 <= "0000011"; --- b
        elsif Q = "011" then
            display1 <= "0001000"; --- A
        elsif Q = "100" then
            display1 <= "0010010"; --- S
        else
            display1 <= "1111111"; --- espacios
        end if;
    when "100" => -- HANNA
        if Q = "000" then
            display1 <= "0001001"; --- H
        elsif Q = "001" then
            display1 <= "0001000"; --- A
        elsif Q = "010" then
            display1 <= "1001000"; --- N
        elsif Q = "011" then
            display1 <= "1001000"; --- N
        elsif Q = "100" then
            display1 <= "0001000"; --- A
        else
            display1 <= "1111111"; --- espacios
        end if;
    when "011" => -- FEr
        if Q = "000" then
            display1 <= "0001110"; --F
        elsif Q = "001" then
            display1 <= "0000110"; --E
```

```
            elsif Q = "010" then
                display1 <= "0101111"; --r
            else
                display1 <= "1111111"; --espacios
            end if;
        when "010" => --SUSY
            if Q = "000" then
                display1 <= "0010010"; --- S
            elsif Q = "001" then
                display1 <= "1000001"; -- U
            elsif Q = "010" then
                display1 <= "0010010"; --- S
            elsif Q = "011" then
                display1 <= "0010001"; --- y
            else
                display1 <= "1111111"; --- espacios
            end if;
        when "001" => -- DAN
            if Q = "000" then
                display1 <= "0100001"; --- D
            elsif Q = "001" then
                display1 <= "0001000"; --- A
            elsif Q = "010" then
                display1 <= "1001000"; --- N
            else
                display1 <= "1111111"; --- espacios
            end if;
        when others => ---- VACIO
            display1 <= "1111111"; --- espacios
    end case;
end process;

FF1 : process (segundo)
begin
    if rising_edge (segundo) then
        display2 <= display1;
    end if;
end process;
```

```
    FF2 : process (segundo)
    begin
        if rising_edge (segundo) then
            display3 <= display2;
        end if;
    end process;

    FF3 : process (segundo)
    begin
        if rising_edge (segundo) then
            display4 <= display3;
        end if;
    end process;

end Behavioral;
```

b) Para tarjetas con los displays de 7 segmentos conectados en paralelo

DIAGRAMA DE BLOQUES



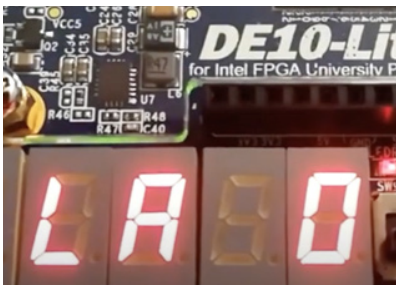
CÓDIGO EN VHDL DEL SISTEMA ISE

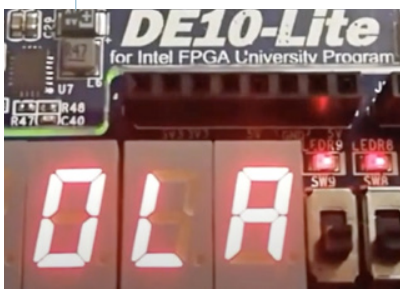
```
library IEEE;
use IEEE.std_logic_1164.ALL;
use IEEE.std_logic_arith.ALL;
use IEEE.std_logic_unsigned.ALL;

entity corre is
    Port (reloj : in std_logic;
          Seg : out std_logic_vector (6 downto 0);
          Dig : out std_logic_vector (3 downto 0));
end corre;

architecture Behavioral of corre is
    signal segundo, rap : std_logic;
    signal Q : std_logic_vector(2 downto 0);
    signal rapid : std_logic_vector(1 downto 0);
    signal display1, display2, display3, display4 : std_logic_vector(6 downto 0);
begin

    divisor : process (reloj)
        variable cuenta: std_logic_vector(27 downto 0) := X"0000000";
    begin
        if rising_edge (reloj) then
            if cuenta = X"48009E0" then   -- es el tiempo para 1.51 seg
                cuenta := X"0000000";
            else
                cuenta := cuenta + 1;
            end if;
        end if;
        segundo <= cuenta(22);
        rap <= cuenta(10);
    end process;
```

```
conseg:  process (segundo)
    variable cuenta: std_logic_vector(2 downto 0) := "000";
begin
    if rising_edge (segundo) then
        cuenta := cuenta + 1;
    end if;
    Q <= cuenta;
end process;

with Q select   -------- mensaje
    display1 <= "1000111" when "000",   -- L
                "0001000" when "001",   -- A
                "1000000" when "011",   -- O
                "1000111" when "100",   -- L
                "0001000" when "101",   -- A
                "1111111" when others;  -- espacios

FF1: process (segundo)
begin
    if rising_edge (segundo) then
        display2 <= display1;
    end if;
end process;

FF2: process (segundo)
begin
    if rising_edge (segundo) then
        display3 <= display2;
    end if;
end process;

FF3: process (segundo)
begin
    if rising_edge (segundo) then
        display4 <= display3;
    end if;
end process;
```

```
    contrap: process (rap)
        variable cuenta: std_logic_vector(1 downto 0) := "00";
    begin
        if rising_edge (rap) then
            cuenta := cuenta + 1;
        end if;
        rapid <= cuenta;
    end process;

    with rapid select   ----- muxy2
        Dig <= "1110" when "00",   -- 1
               "1101" when "01",   -- 2
               "1011" when "10",   -- 3
               "0111"when others; -- 4

    Muxy: process (rapid)
    begin
        case rapid is
            when "11" =>
                Seg <= display4;
            when "10" =>
                Seg <= display3;
            when "01" =>
                Seg <= display2;
            when others =>
                Seg <= display1;
        end case;
    end process;

end Behavioral;
```

REVISA EL VIDEO
DE LA PRÁCTICA AQUÍ





UNIDAD DE APOYO EDITORIAL

Prácticas para Diseño Digital Moderno
Se publicó la primera edición electrónica de un ejemplar
(3 MB) en formato PDF en agosto de 2023,
en el repositorio de la Facultad de Ingeniería, UNAM,
Ciudad Universitaria, Ciudad de México. C.P. 04510

El diseño estuvo a cargo de la Unidad de Apoyo Editorial
de la Facultad de Ingeniería. Las familias tipográficas
utilizadas fueron Brevia para titulares
y Source Serif Pro para texto.