Práctica 2

Simón Sánchez Rúa

Juan Camilo Arteaga Ibarra

Grupo 8

José Edinson Aedo Cobo

Universidad de Antioquia

Facultad de ingeniería

Medellín

2022

Objetivos:

- Entender y manejar el software Vivado para el diseño de circuitos digitales mediante el uso de VHDL.
- A partir de funciones booleanas diseñar, simular e implementar circuitos digitales usando FPGAS.

Materiales:

- Software Vivado.
- FPGAS.

Descripción:

Realizar la implementación de diferentes circuitos lógicos mediante el software Vivado y haciendo uso del lenguaje de programación VHDL. Posteriormente sintetizar, simular e implementar el diseño. Para ello se debe considerar los conocimientos obtenidos a partir de las sesiones teóricas en el desarrollo a partir de diferentes métodos para plantear las funciones booleanas propuestas para la práctica de laboratorio.

Práctica No. 2: diseño e implementación de circuitos combinacionales

Objetivos:

- Familiarizarse con la metodología de diseño de circuitos digitales.
- Diseñar circuitos lógicos combinacionales usando diferentes técnicas de modelado con lenguajes de descripción de hardware (VHDL) y el uso de Test Bench para su verificación por simulación.
- Adquirir habilidades con la plataforma Vivado para el diseño, simulación e implementación de sistemas digitales usando FPGAS.

Descripción:

En la primera parte, se realizará un tutorial con el fin de aprender el ciclo de diseño con la plataforma Vivado, considerando el diseño con VHDL, las síntesis, la simulación y la implementación el circuito en la FPGA.

En la segunda parte, cada Grupo continuará trabajando con la función combinacional de 4 entradas utilizada en la práctica 1. Se debe partir de la función minimizada con el método de Quine McCluskey. La función se implementará en una FPGA a partir de su diseño realizado en VHDL. Se debe diseñar la función de tres formas, considerando los tres estilos de modelado: estructural, flujo de datos y comportamental. Se debe verificar su funcionamiento mediante el diseño de un Test Bench que permita la simulación lógica de forma exhaustiva. Finalmente se debe implementar el circuito en la FPGA ARTIX-7 (Xilinx Artix-7 XC7A35T-ICPG236C) y verificar su funcionamiento con pruebas en la tarjeta Basys3 (si se dispone de la tarjeta).

En la tercera parte, se hará el modelado de una Unidad Lógica y Aritmética (ALU) de 4 bits y un circuito que contiene la ALU. Las ALU realizará 4 funciones asignadas a cada grupo, según el anexo. Finalmente se debe simular la ALU y verificar su funcionamiento con pruebas en la tarjeta Basys3 (si se dispone de la tarjeta).

Procedimiento:

1. La parte 1 fue un tutorial para comprender y empezar el desarrollo en el software Vivado, haciendo uso del lenguaje VHDL.

2. **Estilos de modelado de funciones combinacionales en VHDL.**

Diseñe la función que le fue asignada en la practica 1 de 4 entradas usando los tres estilos de descripción:

- Estructural.
- Flujo de datos.
- Comportamental.

Para esto siga el siguiente procedimiento:

7. Genere un nuevo proyecto en el Vivado y modele su circuito combinacional de cinco entradas (que usó en la práctica No.1) de tres formas usando los tres estilos de modelado y cree un componente para cada uno (use los mismos puertos de entrada/salida, para los tres componentes); utilice un nombre de entidad y arquitectura diferente de acuerdo con el estilo implementado.

2. Cree un nuevo proyecto, integrando los tres componentes diseñados en el ítem anterior, usando los mismos puertos de entrada para todos y una salida asociada a cada uno. Diseñe un Test Bench exhaustivo para probar las tres implementaciones simultáneamente, con salida por Consola.

3. Simule los tres circuitos.

4. Sintetice el diseño e impleméntelo en la FPGA. (siga el procedimiento de i-vi del literal e la primera parte).

5. Verifique que se cumple tabla de verdad para los tres componentes diseñados observando las señales que entrega el simulador (Waveform). Note que el circuito tiene 5 entradas y 3 salidas (una para cada función implementada en un estilo diferente). Verifique los tiempos de atraso para cada componente realizando una simulación de timing.

6. Realice una comparación de los esquemáticos de los circuitos sintetizados para cada componente (estilo). ¿Nota alguna diferencia? Compare también el número de celdas de la FPGA utilizadas en la implementación de cada componente.

Si se dispone de tarjeta, realice el siguiente procedimiento:

7. Verificar en la tarjeta el circuito diseñado. Programe la tarjeta y use 8 suiches SW para las entradas y tres leds para las salidas. Y verifique la tabla de verdad.

3. Modelado de una implementación del circuito a implementar

1. Modelo en VHDL cada uno de los componentes del circuito que se muestra en la figura siguiente. La ALU tiene datos de entrada y salida de 4 bits y un carry de salida para las operaciones aritméticas. En el Anexo, se asignan las operaciones que debe realizar ALU por Grupo.

2. Construya un Test Bench exhaustivo. Con salida por Consola.

3. Realice simulaciones funcionales y de timing. Para esto sintetice e implemente el circuito utilizando el Vivado.

4. Establezca los tiempos de atraso del circuito a partir de las simulaciones de timing.

Si se dispone de tarjeta, realice el siguiente procedimiento:

5. Verificar en la tarjeta el circuito diseñado. Programe la tarjeta y use suiches SW para las entradas, un led para la salida Cout y el Display de 7 segmentos (active un solo display). Verifique cada función de la ALU.

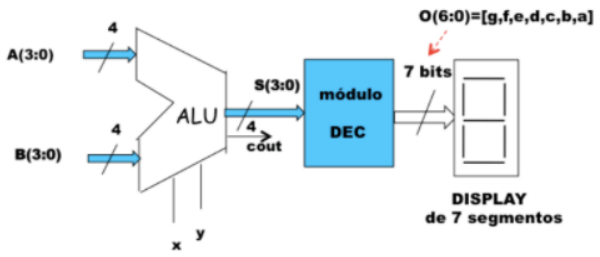

Tabla de verdad
Circuito DEC

S(3:0)	a b c d e f g
0000	0 0 0 0 0 0 1
0001	1 0 0 1 1 1 1
0010	0 0 1 0 0 1 0
0011	0 0 0 0 1 1 0
0100	1 0 0 1 1 0 0
0101	0 1 0 0 1 0 0
0110	1 1 0 0 0 0 0
0111	0 0 0 1 1 1 0
1000	0 0 0 0 0 0 0
1001	0 0 0 0 1 0 0
1010	1 1 1 1 1 1 0
1011	0 1 1 0 0 0 0
1100	1 1 0 1 1 0 1
1101	1 1 1 1 0 0 1
1110	0 1 1 0 0 1 1
1111	1 0 1 1 0 1 1

Figure 1: circuito a implementar.

Desarrollo:

Parte 2.

Se crea el proyecto con dos archivos .vhd, uno para la función y otro para el Test Bench. Luego, se añade el archivo Basys3_Master.xdc que fue otorgado para la realización de la práctica. Y por último se define la FPGA a utilizar, la cual es la Basys 3 Artix-7 (xc7a35ticpg236-1L), que es en la que se va a correr el diseño una vez esté completo.

Para realizar el código, se toma la función:

Grupo	Función (Maxterm)	Don't care
8	5, 6, 7, 8, 11, 12, 13	9, 10

Tabla 1: función del grupo.

$$(\bar{y} + w + \bar{z}) \cdot (x + \bar{y} + \bar{w}) \cdot (\bar{x} + y) \cdot (\bar{x} + w)$$

	x	y	w	z	Z1
M0	0	0	0	0	1
M1	0	0	0	1	1
M2	0	0	1	0	1
M3	0	0	1	1	1
M4	0	1	0	0	1
M5	0	1	0	1	0
M6	0	1	1	0	0
M7	0	1	1	1	0
M8	1	0	0	0	0
M9	1	0	0	1	x
M10	1	0	1	0	x
M11	1	0	1	1	0
M12	1	1	0	0	0
M13	1	1	0	1	0
M14	1	1	1	0	1
M15	1	1	1	1	1

Tabla 2: tabla de verdad de la función.

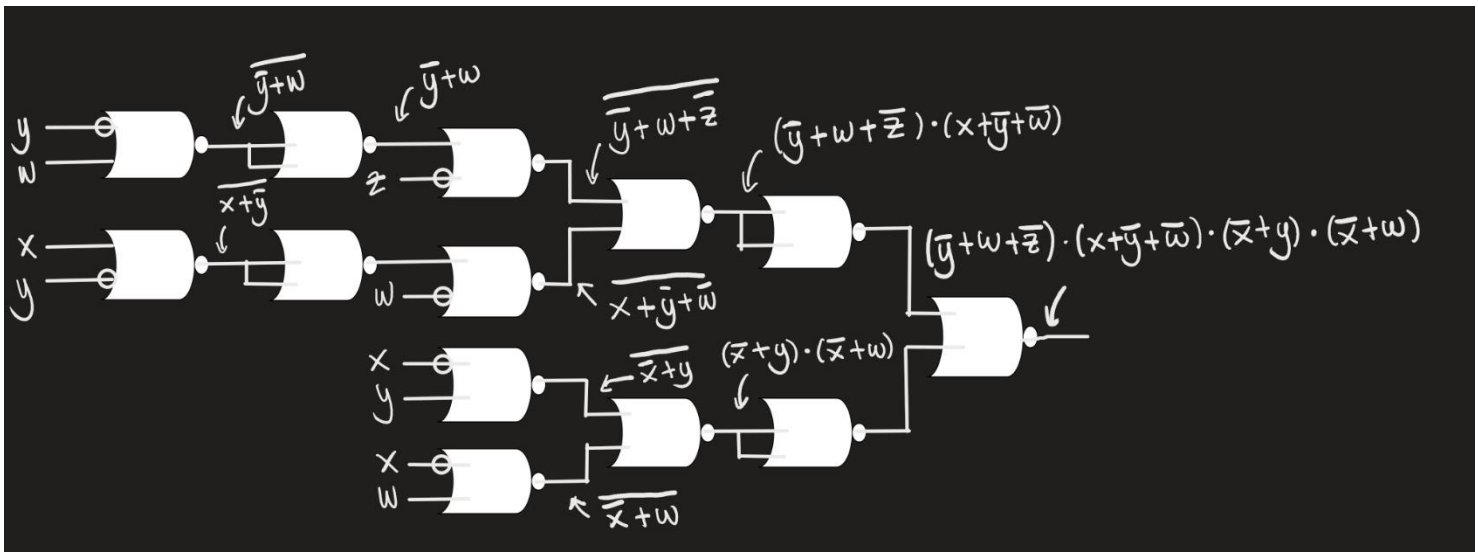

Figure 2: red de compuertas NOR de la función.

A partir de lo descrito se procede a realizar los modelos de la función para la implementación, teniendo en cuenta las características del lenguaje VHDL para diseñar la arquitectura del archivo .vhd, añadiendo las librerías, definiendo la entidad y luego dentro de la arquitectura algunas señales y componentes para los modelos.

Flujo de datos.

Tomando la figura 2 como base para el modelo de flujo de datos y habiendo definido previamente los vectores a utilizar en el proceso, se compone el modelo de la siguiente forma:

```
--Start(Data flow)


        pr(0) <= not(input(2)) nor input(1);
        pr(1) <= input(3) nor not(input(2));
        pr(2) <= not(pr(0));
        pr(3) <= not(pr(1));
        pr(4) <= pr(2) nor not(input(0));
        pr(5) <= pr(3) nor not(input(1));
        pr(6) <= pr(4) nor pr(5);
        pr(7) <= not pr(6);
        pr(8) <= not(input(3)) nor input(2);
        pr(9) <= not(input(3)) nor input(1);
        pr(10) <= pr(8) nor pr(9);
        pr(11) <= not pr(10);
        pr(12) <= pr(7) nor pr(11);
        output(0) <= pr(12);
--End(Data Flow)
```

Figure 3: modelo de flujo de datos.

Estructural.

Se crean los archivos .vhd de los componentes, definiendo su arquitectura y posteriormente en el archivo principal se definen como componentes.

```
--Inverter for the structural model
component opinv port
( a : in std_logic;
  c : out std_logic
);
end component;

--NORr the structural model
component opnor port
( inp : in std_logic_vector(1 downto 0);
  sout : out std_logic
);
end component;
```

Figure 4: componentes del modelo estructural.

Luego, se realiza escribe el modelo partiendo de la red de compuertas NOR (figura 2).

```
--Start(Structural)
  --NOT X
  not_1 : opinv port map(
      a => input(3),
      c => st(0)
  );

  --NOT Y
  not_2 : opinv port map(
      a => input(2),
      c => st(1)
  );

  --NOT W
  not_3 : opinv port map(
      a => input(1),
      c => st(2)
  );

  --NOT Z
  not_4 : opinv port map(
      a => input(0),
      c => st(3)
  );

  --NORs
  nor_1 : opnor port map(
      inp(0) => st(1),
      inp(1) => input(1),
      sout => st(4)
  );

  nor_2 : opnor port map(
      inp(0) => input(3),
      inp(1) => st(1),
      sout => st(5)
  );

  nor_3 : opnor port map(
      inp(0) => st(4),
      inp(1) => st(4),
      sout => st(6)
  );

  nor_4 : opnor port map(
      inp(0) => st(5),
      inp(1) => st(5),
      sout => st(7)
  );

  nor_5 : opnor port map(
      inp(0) => st(6),
      inp(1) => st(3),
      sout => st(8)
  );

  nor_6 : opnor port map(
      inp(0) => st(7),
      inp(1) => st(2),
      sout => st(9)
  );

  nor_7 : opnor port map(
      inp(0) => st(8),
      inp(1) => st(9),
      sout => st(10)
  );

  nor_8 : opnor port map(
      inp(0) => st(10),
      inp(1) => st(10),
      sout => st(11)
  );

  nor_9 : opnor port map(
      inp(0) => st(0),
      inp(1) => input(2),
      sout => st(12)
  );

  nor_10 : opnor port map(
      inp(0) => st(0),
      inp(1) => input(1),
      sout => st(13)
  );

  nor_11 : opnor port map(
      inp(0) => st(12),
      inp(1) => st(13),
      sout => st(14)
  );

  nor_12 : opnor port map(
      inp(0) => st(14),
      inp(1) => st(14),
      sout => st(15)
  );

  nor_13 : opnor port map(
      inp(0) => st(11),
      inp(1) => st(15),
      sout => output(1)
  );
--END(Structural)
```

Figure 5: modelo estructural.

Comportamental.

Para el modelo comportamental se parte de la tabla de verdad (tabla 2) y se establecen los casos para las salidas que son “1” y la salida “0” para los demás de los casos.

```
 -- | "0010" | "0011" | "0100" | "1110" | "1111"
 --Start(Comportamental)
process(input)
    begin
    case input is
        when "0000"  =>
        outp <= '1';
        when "0001" =>
        outp <= '1';
        when "0010" =>
        outp <= '1';
        when "0011" =>
        outp <= '1';
        when "0100" =>
        outp <= '1';
        when "1110" =>
        outp <= '1';
        when "1111" =>
        outp <= '1';
        when others =>
        outp <= '0';
    end case;
end process;
 --END(Comportamental)
output(2) <= outp;
```

Figure 6: modelo comportamental.

Archivo Basys.

En el archivo .xdc se definen los switches y los LED que se van a activar para la implementación del diseño en el software Vivado

```
## Switches
set_property PACKAGE_PIN V17 [get_ports {input[0]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {input[0]}]
set_property PACKAGE_PIN V16 [get_ports {input[1]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {input[1]}]
set_property PACKAGE_PIN W16 [get_ports {input[2]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {input[2]}]
set_property PACKAGE_PIN W17 [get_ports {input[3]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {input[3]}]
```

Figure 7: switches a utilizar.

```
## LEDs
set_property PACKAGE_PIN U16 [get_ports {output[0]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {output[0]}]
set_property PACKAGE_PIN E19 [get_ports {output[1]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {output[1]}]
set_property PACKAGE_PIN U19 [get_ports {output[2]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {output[2]}]
```

Figure 8: LEDs a encender.

Test Bench.

Por último, se realiza el Test Bench del diseño para realizar las pruebas y verificar el correcto funcionamiento del código. Para ello se incluyen las librerías, se define la entidad sin entradas y sin salidas, ya que es una simulación. A continuación, se define la arquitectura, en la cual se define una componente del diseño principal, las señales que servirán de ayuda para la obtención de datos, una función de proceso para la obtención de la salida esperada, la conexión entre los vectores de entrada y salida del Test Bench y la función principal, y para finalizar se escribe el código del proceso para comparar las respuestas obtenidas y escribir en la consola la respuesta esperada y la resultante en el archivo .vhd principal.

```
library UNISIM;
use UNISIM.VComponents.all;

Entity fun_tb Is
end fun_tb;

Architecture behavior of fun_tb Is
    Component fun
    port (
        input : in STD_LOGIC_VECTOR(3 downto 0);
        output : out STD_LOGIC_VECTOR(2 downto 0)
        );
    End Component;

    Signal inp : STD_LOGIC_VECTOR(3 downto 0); --:="0000"
    Signal out_out : STD_LOGIC_VECTOR(2 downto 0); --:="000"
    Signal out_exp_out : STD_LOGIC;
    Signal inpf : STD_LOGIC_VECTOR(3 downto 0):="0000";

    procedure expected_out (
        inp_in : in std_logic_vector(3 downto 0);
        out_expected : out std_logic
    ) is

    Variable out_expected_int : std_logic;
```

Figure 9: Test Becnh (1).

```
    Variable out_expected_int : std_logic;

    begin
        out_expected_int := (not(inp_in(2)) or inp_in(1)or
        out_expected := out_expected_int;
    end expected_out;

begin
    uut:  fun PORT MAP (
            input => inp,
            output => out_out
         );

    process
        variable s: line;
        variable i : integer := 0;
        variable count : integer := 0;
        variable proc_out : STD_LOGIC;
```

Figure 10: Test Bench (2).

```
    process
        variable s: line;
        variable i : integer := 0;
        variable count : integer := 0;
        variable proc_out : STD_LOGIC;

        begin
        for i in 0 to 15 loop
          count := count + 1;

          wait for 50 ns;
          inp <= inpf;
          wait for 10 ns;
          expected_out(inp, proc_out);
          out_exp_out <= proc_out;A

--        -- If the outputs match, then announce it to the simulator console.
          if (out_out(0) = proc_out and out_out(1) = proc_out and out_out(2) =
                write (s, string'("   LED output MATCHED at ")); write (s, coun
                writeline (output, s);
          else
              write(s, string'("LED output mis-matched at ")); write (s, count)
              writeline (output, s);
          end if;
          inpf<=inpf+1;
        end loop;

      end process;
end behavior;
```

Figure 11: Test Bench (3).

Para observar el proyecto completo entrar a la carpeta Part2 del repositorio en GitHub.

Habiendo terminado el diseño completo, se sintetiza el proyecto, se corre la síntesis, se conecta la FPGA, se corre la implementación, se crea el archivo Bitstream y se abre, todo esto con el fin de utilizar la FPGA para realizar el proceso de verificación de un buen funcionamiento, utilizando switches que deben permitir encender los LEDs si se tiene la entrada para permitirlo.

Figure 12: FPGA a utilizar.

Parte **3.**

Al igual que para la parte anterior, se inicializa el proyecto de la misma forma, ya que se deben crear los mismos archivos y se va a utilizar la misa FPGA.

Para esta parte se utilizará el circuito y la tabla de verdad de la figura 1. Las operaciones de la ALU están definidas por grupo,

8	11	s <= A xor B
	10	s <= B – 2 (0 si B<2)
	01	s<= A + B
	00	s<= A nor B

Figure 13: operaciones de la ALU.

Diseño principal.

En la arquitectura, dentro de una función process, se plantean condicionales para los valores de X y Y que dictaminan y realizan la operación propuesta para las entradas del sistema, luego, se compara la salida de este proceso con los números del 0 al 15 en binario y se le asigna un valor a un vector, dicho valor está parametrizado por la tabla de verdad de la figura 1.

```
architecture Behavioral of ALU is

    --Signal outputSF : std_logic_vector(4 downto 0);
    begin
        process(inputXY, outputSF)

            begin
            outputSF(4) <= '0';
                if inputXY = "11" then
                    outputSF(3 downto 0) <= inputA(3 downto 0) xor inputB(3 downto 0);
                elsif inputXY = "10" then
                    if inputB < 2 then
                        outputSF <= "00000";
                    else
                        outputSF <= inputB-2;
                    end if;
                elsif inputXY = "01" then
                    outputSF <= inputA + inputB;

                else
                    outputSF(3 downto 0) <= inputA(3 downto 0) nor inputB(3 downto 0);
                end if;
                case outputSF(3 downto 0) is
                when "0000" =>
                output7 <= "0000001";
                when "0001" =>
                output7 <= "1001111";
                when "0010" =>
                output7 <= "0010010";
                when "0011" =>
                output7 <= "0000110";
```

Figure 14: parte de la arquitectura del diseño.

Archivo Basys.

El archivo .xdc tendrá 10 switches definidos de la FPGA, para usar como las entradas A, B, X y Y.

```
## Switches
set_property PACKAGE_PIN V17 [get_ports {inputA[0]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {inputA[0]}]
set_property PACKAGE_PIN V16 [get_ports {inputA[1]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {inputA[1]}]
set_property PACKAGE_PIN W16 [get_ports {inputA[2]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {inputA[2]}]
set_property PACKAGE_PIN W17 [get_ports {inputA[3]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {inputA[3]}]
#set_property PACKAGE_PIN W15 [get_ports {input[4]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {input[4]}]
set_property PACKAGE_PIN V15 [get_ports {inputB[0]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {inputB[0]}]
set_property PACKAGE_PIN W14 [get_ports {inputB[1]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {inputB[1]}]
set_property PACKAGE_PIN W13 [get_ports {inputB[2]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {inputB[2]}]
set_property PACKAGE_PIN V2 [get_ports {inputB[3]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {inputB[3]}]
#set_property PACKAGE_PIN T3 [get_ports {sw[9]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {sw[9]}]
set_property PACKAGE_PIN T2 [get_ports {inputXY[0]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {inputXY[0]}]
set_property PACKAGE_PIN R3 [get_ports {inputXY[1]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {inputXY[1]}]
```

Figure 15: switches a utilizar.

También se define el LED para el carry de salida, resultante del proceso de la ALU.

```
## LEDs
set_property PACKAGE_PIN U16 [get_ports {outputSF[4]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {outputSF[4]}]
```

Figure 16: LED del carry de salida.

Y por último se establecen los pines del 7 segmentos.

```
##7 segment display
set_property PACKAGE_PIN W7 [get_ports {output7[6]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {output7[6]}]
set_property PACKAGE_PIN W6 [get_ports {output7[5]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {output7[5]}]
set_property PACKAGE_PIN U8 [get_ports {output7[4]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {output7[4]}]
set_property PACKAGE_PIN V8 [get_ports {output7[3]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {output7[3]}]
set_property PACKAGE_PIN U5 [get_ports {output7[2]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {output7[2]}]
set_property PACKAGE_PIN V5 [get_ports {output7[1]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {output7[1]}]
set_property PACKAGE_PIN U7 [get_ports {output7[0]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {output7[0]}]
```

Figure 17: pines a utilizar en el 7 segmentos.

Test Bench.

Tal como para el Test Bench de la parte 2, se inicia con los mismos parámetros, teniendo en cuenta que el circuito a simular es diferente.

```
library UNISIM;
use UNISIM.VComponents.all;

Entity ALU_tb Is
end ALU_tb;

Architecture behavior of ALU_tb Is
    Component ALU
    port (
          inputA : in STD_LOGIC_VECTOR(4 downto 0);
          inputB : in STD_LOGIC_VECTOR(4 downto 0);
          inputXY : in STD_LOGIC_VECTOR(1 downto 0);
          outputSF : inout STD_LOGIC_VECTOR(4 downto 0);
          output7 : out STD_LOGIC_VECTOR(6 downto 0)
        );
    End Component;

    Signal switchA : STD_LOGIC_VECTOR(4 downto 0) ;
    Signal switchB : STD_LOGIC_VECTOR(4 downto 0) ;
    Signal switchXY : STD_LOGIC_VECTOR(1 downto 0) ;
    Signal outputProcess : STD_LOGIC_VECTOR(4 downto 0);
    Signal led_out : STD_LOGIC_VECTOR(6 downto 0);
    --Signal led_exp_out : STD_LOGIC_VECTOR(6 downto 0);
```

Figure 18: algunas partes del Test Bench (1).

En este caso, el procedure contendrá el mismo desarrollo de la ALU, con la finalidad de ser utilizado para comparar posteriormente con los resultados del diseño principal.

```
procedure expected_led (
    inputAR : in STD_LOGIC_VECTOR(4 downto 0);
    inputBR : in STD_LOGIC_VECTOR(4 downto 0);
    inputXYR : in STD_LOGIC_VECTOR(1 downto 0);
    outputS : inout std_logic_vector(4 downto 0);
    led_expected : out std_logic_vector(6 downto 0)
) is

begin

        outputS(4) := '0';
            if inputXYR = "11" then
                outputS(3 downto 0) := inputAR(3 downto 0) xor inputBR(3 downto 0);
            elsif inputXYR = "10" then
                if inputBR < 2 then
                    outputS := "00000";
                else
                    outputS := inputBR-2;
                end if;
            elsif inputXYR = "01" then
                outputS := inputAR + inputBR;

            else
                outputS(3 downto 0) := inputAR(3 downto 0) nor inputBR(3 downto 0);
            end if;
    case outputS(3 downto 0) is
            when "0000" =>
            led_expected := "0000001";
            when "0001" =>
            led_expected := "1001111";
            when "0010" =>
            led_expected := "0010010";
            when "0011" =>
            led_expected := "0000110";
            when "0100" =>
            led_expected := "1001100";
```

Figure 19: algunas partes del Test Bench (2).

A continuación, se establece el PORT MAP y se prosigue con el diseño de la impresión en consola de la salida de la simulación y del diseño principal.

```
begin
    uut:  ALU PORT MAP (
            inputA => switchA,
            inputB => switchB,
            inputXY => switchXY,
            output7 => led_out,
            outputSF => outputProcess
         );

    process

        variable s : line;
        variable i : integer := 0;
        variable i2 : integer := 0;
        variable i3 : integer := 0;
        variable countA : integer := 0;
        variable countB : integer := 0;
        variable countXY : integer := 0;
        variable proc_out : STD_LOGIC_VECTOR(6 downto 0);
        variable outputs : std_logic_vector(4 downto 0):= "00000";


    begin
        switchA <= count_int_A;
        switchB <= count_int_B;
        switchXY <= count_int_XY;
        for i in 0 to 15 loop
          countA := countA + 1;

          wait for 60 ns;
          for i2 in 0 to 15 loop
              countB := countB +1;
              wait for 50 ns;
```

```
        for i3 in 0 to 3 loop
            countXY := countXY + 1;
            wait for 10 ns;
            expected_led (switchA,switchB,switchXY,
            write(s, string'("A: "));write (s, swit
            writeline (output, s);
            write(s, string'(" Expected Out S: "));
            writeline (output, s);
            switchXY <= switchXY + 1;
        end loop;

        write(s, string'("Ended loop "));write(s, c
        writeline(output, s);
        write(s, string'("------------------------
        writeline(output, s);
    switchB <= switchB + 1;
    end loop;
    switchB <= count_int_B;
    switchA <= switchA + 1;
end loop;
```

Figure 20: algunas partes del Test Bench (3).

Si se desea ver el proyecto completo entrar a la carpeta Part3 del repositorio en GitHub.

Para finalizar, se siguen los últimos pasos mencionados para la parte 2, con la diferencia de que se utilizarán switches para encender un LED y un 7 segmentos según la entrada ingresada. Todo esto en la misma FPGA.

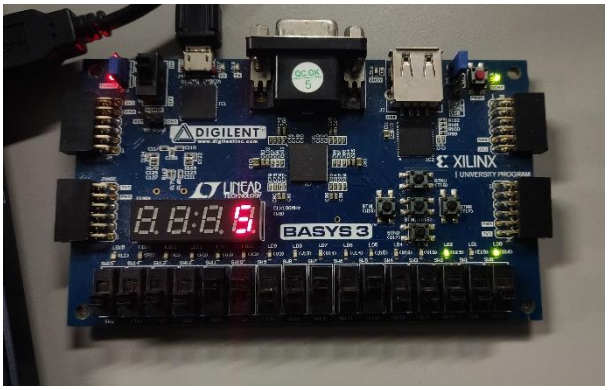

Figure 21: FPGA programada.

Conclusiones:

- Los valores obtenidos en la FPGA y el Test Bench concuerdan para los dos diseños de circuitos, queriendo decir que el resultado obtenido es correcto.

- Se logró satisfactoriamente usar los 3 tipos de modelados para la función booleana asignada para la parte 2, de manera que se obtuvo conocimiento de cómo realizar el diseño de una misma función de diferentes maneras.

- Se afianzó el conocimiento obtenido durante las sesiones de clase al aplicarlo en el uso del software Vivado, realizando el código en VHDL para los circuitos propuestos en la práctica.

- Se comprendió el uso de una FPGA como hardware para la solución de problemas.