INSTITUTO POLITÉCNICO NACIONAL

Escuela Superior de Computo

Práctica no. 1 ALU con Operaciones Lógicas, Aritméticas y Corrimiento

Arquitectura de Computadoras

Autores:
Hernández Vergara, Eduardo
Rojas Cruz, José Ángel
Alcantara Covarrubias, Erik
Martínez Alquicira, Mariana

Profesor:
Pastrana Fernández Carlos Jesús

15 de Octubre de 2022

Índice

1. Introdución

1.1. Objetivo

El alumno realizará una ALU,la que incluirá registros de corrimientos, lógicos y aritméticos la cual podrá realizar las operaciones con datos de 'n' bits especificados, tendrá una terminal que sirva para el control de carga.

1.2. Introducción teórica

1.2.1. Unidad Aritmetica Lógica (ALU)

La ALU es la parte de la computadora que realiza las operaciones lógicas y aritmeticas de los datos. Todos los demás elementos de los sistemas de computadoras (Control Unit, Registros, memoria, entrada/salida) son algunos de los datos que se le entregan a la ALU para que los procese y después los devuelva. Entonces, en un sentido, hemos alcanzado la esencia de una computadora cuando consideramos una ALU.

Una ALU y, en realidad todos los componentes electronicos en una computadora estan basados en el uso de simples dispositivos lógicos digitales simples que pueden guardar digitos binarios y realizar simples operaciones lógicas boleanas.

En la siguiente figura se muestra en términos generales, como esta interconectada la ALU con el resto del procesador. Los datos son presentados en los registros de la ALU, y los resultados de la operación se guarda en los registros. Estos registros son almacenados temporalmente mientras el procesador este conectado por rutas de señal al ALU. Al ALU también se le pueden configurar banderas como el resultado de una operación. Por ejemplo, la bandera de overflow se configura en 1 si el resultado excede el tamaño del registro en donde va a ser almacenado. Los valores de estas banderas también son almacenadas en los registros dentro del procesador. La unidad de control nos provee señalesque controlan la operación del ALU y el movimiento de datos de entrada y salida del ALU.

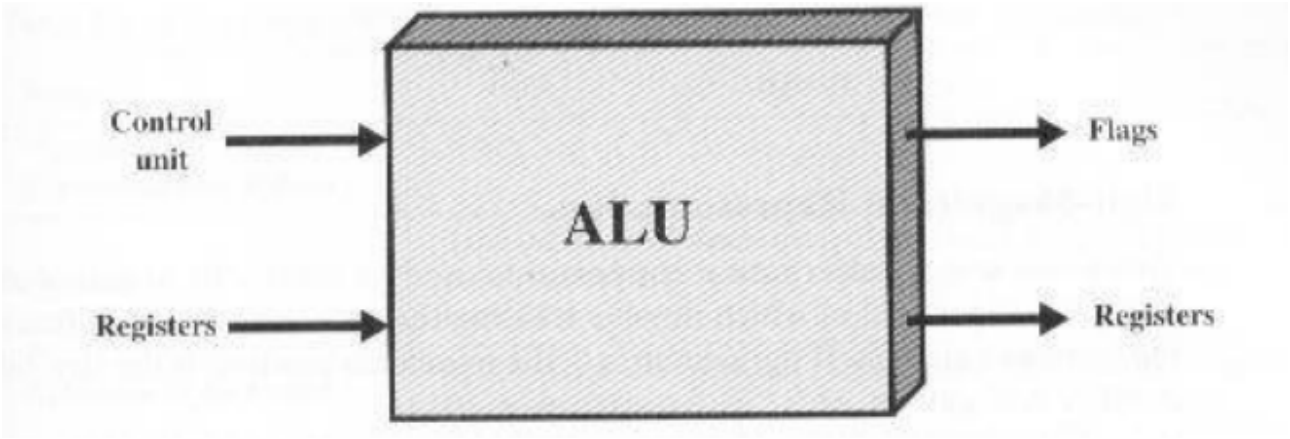

Figura 2: Entradas y Salidas del ALU

1.2.2. ¿Qué es VHDL?

VHDL es un lenguaje de diseño de software destinado a usarse en todas las fases del diseño de sistemas digitales. VHDL viene de VHSIC (Very High Speed Integrated Circuits) lenguaje descriptor de hardware. Su desarrollo empezó en 1983 con un contrato por parte del departamento de defenza de los Estados Unidos de Ámerica, y se volvió un estándar de la IEEE en 1987.

1.2.3. ¿Qué es una FPGA?

Una FPGA (Field Programmable Gate Array) es un complejo circuito integrado digital programable compuesto por bloques lógicos configurables (CLB) y puertos de entrada/salida (IOB), cuya interconexión y funcionalidad puede ser programada mediante un lenguaje de descripción especializado.

Su principal característica y ventaja es que pueden reprogramarse para un trabajo específico o cambiar sus requisitos después de haberse fabricado. Esto también implica que en muchos casos se pueden hacer cambios físicos sin hacer modificaciones costosas en la placa que lo soporta.

Básicamente, una FPGA es un conjunto de múltiples circuitos (lógicos y de otros tipos) dispuestos matricialmente, cuyas interconexiones son programables por el usuario para la aplicación requerida. En una FPGA se programa su hardware, a diferencia de los microcontroladores / microprocesadores, en los que solo existe un hardware fijo y se programa su software (firmware).

Históricamente, las FPGA fueron inventadas en el año 1984 por Ross Freeman y Bernard Von Der Schmitt, cofundadores de la empresa Xilinx, fabricante de las mismas. Como resultado de numerosas evoluciones, la compañía produjo la primera familia de dispositivos lógicos programables por el usuario, de propósito general.

Las FPGA, además de contener puertas lógicas AND y OR, tienen memoria RAM, controladores de reloj, etc., por lo que son muy apropiadas para el diseño de sistemas embebidos con microprocesador. La compañía Xilinx ha evolucionado dicha tecnología hasta convertirla en un nuevo concepto a tener en cuenta en ciertos entornos de trabajo.

1.3. Material y Equipo Empleado

- 1 FPGA
- 1 Tarjeta de evaluación de pruebas ó Protoboard microswitches con resistencias 10 KoHms, 20 leds con resistencias 330 oHms, 1 display de 4 digitos multiplexado de ánodo común.
- 1 eliminador de celular de 5v con cable mini usb.
- Quartus Prime Lite

2. Desarrollo Experimental

2.1. Objetivo Específico

1. Desarrollar en la FPGA un programa en VHDL de una ALU, en cuál se puedan seleccionar tres tipos de operaciones, Lógicas, Aritméticas y Shifters, estás son:

   - Lógicas (A y B de 10 bits):
     a) Negación o Complemento a 1 de A.
     b) Complemento a 2 de A.
     c) AND entre A y B.
     d) OR entre A y B.
   - Shifters (A de 10 bits):
     a) LSL (Logical Shift Left).
     b) ASR. (Arithmetic Shift Right).
   - Aritméticas (A y B)
     a) Suma 1 byte c/ carry out.
     b) Resta 1 byte.
     c) Multiplicación de 5 bits

2. Con los siguientes especificaciones:

   - Realizar el programa en VHDL que incluya todas las operaciones básicas (Lógicas, Shifters, Aritméticas) del tamaño de dato correcto como se solicita en la descripción.
   - Recuerde que todas las operaciones anteriores deben funcionar como una unidad, para este inciso se deben programar todas las opciones por componentes o multi procesos.
   - Para las opciones de corrimientos de debe incluir un clock para sincronizar los desplazamientos.
   - Todas las entradas deben ser consideradas dependiendo de la operación a realizar, estas deberán ser introducidas por medio de microswitches.
   - La asignación de pines se debe de hacer de acuerdo a la disponibilidad en las terminales de salida en la FPGA.
   - La salida del bloque aritmético se tendrá que ser desplegada en el display de 7 seg. 4 digitos y a su vez la salida de los shifters y la unidad lógica se mostrara únicamente mediante Leds.

3. Se manejara el mismo programa del inciso anterior pero ahora todas las operaciones tanto Aritméticas, shifters o lógicas se mostraran mediante mensaje de texto pregrabado en memoria ROM indexada en VHDL para el texto identificador de la operación por ejemplo: SunnA, rEstA, nnulti, And, or, not, Co1, Co2, ror, roL, LSL,LSren tipo ventana deslizante display de 4 dígitos de tipo ánodo común de 7 seg. El cual se multiplexará para que se despliegue 5 segundos antes de mostrar el resultado de la operación. Cada una de las letras será almacenada en un solo byte, y se multiplexará similar a los números pero con ventana deslizante que significa que el mensaje se desplaza para mostrar todas las letras durante 5 segundos de forma automática antes de mostrar el resultado.

3. Código

Listing 1: Selector Operaciones

```
LIBRARY IEEE;

USE IEEE.STD_LOGIC_1164.ALL;

ENTITY selectorOp IS
  PORT(
    a,b,c,d : IN STD_LOGIC_VECTOR(9 DOWNTO 0);
    sel : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
    sal, salLed : OUT STD_LOGIC_VECTOR(9 DOWNTO 0)
  );
END ENTITY;

ARCHITECTURE bhr OF selectorOp IS
BEGIN

  PROCESS(a,b,c,d,sel)
  BEGIN
    CASE sel IS
      WHEN "00" =>
        sal <= a;
        salLed <= "0000000000";
      WHEN "01" =>
        sal <= "0000000000";
        salLed <= b;
      WHEN "10" =>
        sal <= "0000000000";
        salLed <= c;
      WHEN "11" =>
        sal <= d;
        salLed <= d;
    END CASE;
  END PROCESS;
END ARCHITECTURE;
```

Listing 2: Full adder

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY Fadder IS PORT (
    A, B, Cin : IN STD_LOGIC;
    S, Cout : OUT STD_LOGIC
);
END Fadder;
--El nombre del archivo .vhdl tiene q ser el mismo que de la entidad
--Corregi, las operaciones logicas que hacen cada uno
ARCHITECTURE FullA OF Fadder IS
BEGIN
    S <= Cin XOR (A XOR B);
    Cout <= (A AND B) OR (B AND Cin) OR (A AND Cin);
END FullA;
```

Listing 3: Full adder fw

```
LIBRARY IEEE;
```

```
2  USE IEEE.STD_LOGIC_1164.ALL;
3
4  ENTITY Full_Adder_wF IS PORT (
5      operation : IN STD_LOGIC;
6      Var1, Var2 : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
7      Res : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
8      CarryF, OverflowF, ZeroF, SumF, CoutF : OUT STD_LOGIC
9  );
10 END Full_Adder_wF;
11
12 ARCHITECTURE Adder_wF OF Full_Adder_wF IS
13
14     SIGNAL C : STD_LOGIC_VECTOR (7 DOWNTO 0);
15     SIGNAL B_Prime : STD_LOGIC_VECTOR (7 DOWNTO 0);
16     SIGNAL S_Prime : STD_LOGIC_VECTOR (7 DOWNTO 0);
17
18     COMPONENT Fadder IS PORT (
19         A, B, Cin : IN STD_LOGIC;
20         S, Cout : OUT STD_LOGIC
21         );
22     END COMPONENT;
23
24 BEGIN
25
26     B_Prime(0) <= Var2(0) XOR operation;
27     B_Prime(1) <= Var2(1) XOR operation;
28     B_Prime(2) <= Var2(2) XOR operation;
29     B_Prime(3) <= Var2(3) XOR operation;
30     B_Prime(4) <= Var2(4) XOR operation;
31     B_Prime(5) <= Var2(5) XOR operation;
32     B_Prime(6) <= Var2(6) XOR operation;
33     B_Prime(7) <= Var2(7) XOR operation;
34
35    --Que krajos hace S_Prime
36     Sum1 : Fadder PORT MAP(Var1(0), B_Prime(0), operation, S_Prime(0), C
           (0));
37     Sum2 : Fadder PORT MAP(Var1(1), B_Prime(1), C(0), S_Prime(1), C(1));
38     Sum3 : Fadder PORT MAP(Var1(2), B_Prime(2), C(1), S_Prime(2), C(2));
39     Sum4 : Fadder PORT MAP(Var1(3), B_Prime(3), C(2), S_Prime(3), C(3));
40     Sum5 : Fadder PORT MAP(Var1(4), B_Prime(4), C(3), S_Prime(4), C(4));
41     Sum6 : Fadder PORT MAP(Var1(5), B_Prime(5), C(4), S_Prime(5), C(5));
42     Sum7 : Fadder PORT MAP(Var1(6), B_Prime(6), C(5), S_Prime(6), C(6));
43     Sum8 : Fadder PORT MAP(Var1(7), B_Prime(7), C(6), S_Prime(7), C(7));
44
45     Res <= S_Prime;
46     CarryF <= operation XOR C(7);
47     OverflowF <= C(6) XOR C(7);
48     ZeroF <= NOT(S_Prime(0) OR S_Prime(1) OR S_Prime(2) OR S_Prime(3) OR
           S_Prime(4) OR S_Prime(5) OR S_Prime(6) OR S_Prime(7));
49     SumF <= S_Prime(7);
50     CoutF <= C(7);
51
52 END Adder_wF;
```

Listing 4: Full adder 5

```
1  LIBRARY IEEE;
2  USE IEEE.STD_LOGIC_1164.ALL;
3
4  ENTITY Full_Adder_Five IS PORT (
```

```
    selector : IN STD_LOGIC;
    A, B : IN STD_LOGIC_VECTOR(4 DOWNTO 0);
    S : OUT STD_LOGIC_VECTOR(4 DOWNTO 0);
    Carry, Overflow, Zero, Sum, Cout : OUT STD_LOGIC
);
END Full_Adder_Five;

ARCHITECTURE Adder_F OF Full_Adder_Five IS

    SIGNAL C : STD_LOGIC_VECTOR (4 DOWNTO 0);
    SIGNAL B_Prime : STD_LOGIC_VECTOR (4 DOWNTO 0);
    SIGNAL S_Prime : STD_LOGIC_VECTOR (4 DOWNTO 0);

    COMPONENT Fadder IS PORT (
        A, B, Cin : IN STD_LOGIC;
        S, Cout : OUT STD_LOGIC);
    END COMPONENT;

BEGIN

    B_Prime(0) <= B(0) XOR selector;
    B_Prime(1) <= B(1) XOR selector;
    B_Prime(2) <= B(2) XOR selector;
    B_Prime(3) <= B(3) XOR selector;
    B_Prime(4) <= B(4) XOR selector;

    Sum1 : Fadder PORT MAP(A(0), B_Prime(0), selector, S_Prime(0), C(0));
    Sum2 : Fadder PORT MAP(A(1), B_Prime(1), C(0), S_Prime(1), C(1));
    Sum3 : Fadder PORT MAP(A(2), B_Prime(2), C(1), S_Prime(2), C(2));
    Sum4 : Fadder PORT MAP(A(3), B_Prime(3), C(2), S_Prime(3), C(3));
    Sum5 : Fadder PORT MAP(A(4), B_Prime(4), C(3), S_Prime(4), C(4));

    S <= S_Prime;
    Carry <= selector XOR C(4);
    Overflow <= C(3) XOR C(4);
    Zero <= NOT(S_Prime(0) OR S_Prime(1) OR S_Prime(2) OR S_Prime(3) OR
       S_Prime(4));
    Sum <= S_Prime(4);
    Cout <= C(4);

END Adder_F;
```

Listing 5: Unidad Aritmetica

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.numeric_std.ALL;
--nombre de las varriables cambiaselos por si acaso, de los selectores
--aparte de checar bien que va a entrar y salir, porq luego quien sabe
--de donde obtienes los datos, por ejemplo el selector de los full adders
ENTITY UAritmetica IS
  PORT (
    selector : IN STD_LOGIC_VECTOR (1 DOWNTO 0);
    A, B : IN STD_LOGIC_VECTOR(9 DOWNTO 0);
    S : OUT STD_LOGIC_VECTOR(9 DOWNTO 0);
    SalidaMux: out std_logic_vector(3 downto 0);
    Carry, Overflow, Zero, Sum, Cout : OUT STD_LOGIC
  );
END ENTITY UAritmetica;

```

```
17 --- Suma(1 byte), Resta(1 byte) y Multiplicacion(5 bits)
18
19 ARCHITECTURE Aritmetica OF UAritmetica IS
20
21   SIGNAL A_Prime, B_Prime : STD_LOGIC_VECTOR(4 DOWNTO 0);
22   SIGNAL A_Temp, B_Temp : STD_LOGIC_VECTOR(7 DOWNTO 0);
23   SIGNAL S_Prime : STD_LOGIC_VECTOR(7 DOWNTO 0);
24   SIGNAL S_Prime_2 : STD_LOGIC_VECTOR(7 DOWNTO 0);
25   SIGNAL S_Temp : STD_LOGIC_VECTOR(9 DOWNTO 0);
26   SIGNAL Carry1, Overflow1, Zero1, Sum1, Cout1, Carry2, Overflow2, Zero2,
       Sum2, Cout2, Carry3, Overflow3, Zero3, Sum3, Cout3 : STD_LOGIC;
27
28   COMPONENT Full_Adder_wF IS PORT (
29     operation : IN STD_LOGIC;
30     Var1, Var2 : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
31     Res : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
32     CarryF, OverflowF, ZeroF, SumF, CoutF : OUT STD_LOGIC
33     );
34   END COMPONENT;
35
36 --Falta este
37   COMPONENT Multiplicador IS PORT (
38     ope : IN STD_LOGIC;
39     VarA, VarB : IN STD_LOGIC_VECTOR(4 DOWNTO 0);
40     Salida : OUT STD_LOGIC_VECTOR(9 DOWNTO 0);
41     carryF, overflowF, zeroF, sumF : OUT STD_LOGIC
42     );
43   END COMPONENT;
44
45 BEGIN
46
47   Cout3 <= '0';
48
49   A_Temp <= A(7 DOWNTO 0);
50   B_Temp <= B(7 DOWNTO 0);
51
52   A_Prime <= A(4 DOWNTO 0);
53   B_Prime <= B(4 DOWNTO 0);
54
55   Suma : Full_Adder_wF PORT MAP('0', A_Temp, B_Temp, S_Prime, Carry1,
       Overflow1, Zero1, Sum1, Cout1);
56   Res : Full_Adder_wF PORT MAP('1', A_Temp, B_Temp, S_Prime_2, Carry2,
       Overflow2, Zero2, Sum2, Cout2);
57   Mul : Multiplicador PORT MAP('0', A_Prime, B_Prime, S_Temp, Carry3,
       Overflow3, Zero3, Sum3);
58
59   PROCESS (selector, A, B) IS
60   BEGIN
61     CASE selector IS
62       WHEN "00" =>
63         --Suma
64         S <= "00" & S_Prime;
65         Carry <= Carry1;
66         Overflow <= Overflow1;
67         Zero <= Zero1;
68         Sum <= Sum1;
69         Cout <= Cout1;
70         SalidaMux <= "0001";
71       WHEN "01" =>
72         --Resta
```

```
73          S <= "00" & S_Prime_2;
74          Carry <= Carry2;
75          Overflow <= Overflow2;
76          Zero <= Zero2;
77          Sum <= Sum2;
78          Cout <= Cout2;
79          SalidaMux <= "0010";
80        WHEN "10" =>
81          --Multiplicacion
82          S <= S_Temp;
83          Carry <= Carry3;
84          Overflow <= Overflow3;
85          Zero <= Zero3;
86          Sum <= Sum3;
87          Cout <= Cout3;
88          SalidaMux <= "0100";
89        WHEN "11" =>
90          --Dividir
91          S <= "0000000000";
92          Carry <= '0';
93          Overflow <= '0';
94          Zero <= '0';
95          Sum <= '0';
96          Cout <= '0';
97          SalidaMux <= "1000";
98        WHEN OTHERS =>
99          S <= "0000000000";
100         Carry <= '0';
101         Overflow <= '0';
102         Zero <= '0';
103         Sum <= '0';
104         Cout <= '0';
105         SalidaMux <= "0000";
106     END CASE;
107   END PROCESS;
108 END ARCHITECTURE Aritmetica;
```

Listing 6: Lógicas

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.ALL;
3  USE ieee.std_logic_arith.ALL;
4  USE ieee.std_logic_unsigned.ALL;
5  ENTITY Logicas IS
6    PORT (
7      a, b : IN STD_LOGIC_VECTOR(9 DOWNTO 0);
8      cntrl : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
9      clk : IN STD_LOGIC;
10     salida : OUT STD_LOGIC_VECTOR(9 DOWNTO 0);
11     sf : OUT STD_LOGIC
12   );
13 END ENTITY Logicas;
14
15 ARCHITECTURE ArqLogicas OF Logicas IS
16   SIGNAL aux : STD_LOGIC_VECTOR(10 DOWNTO 0);
17 BEGIN
18   OperacionesLogicas : PROCESS (cntrl, a, b, clk)
19   BEGIN
20     IF rising_edge(clk) THEN
21       IF cntrl = "00" THEN --Operacion AND
```

```
22          salida <= a AND b;
23          sf <= '0';
24        ELSIF cntrl = "01" THEN --Operacion OR
25          salida <= a OR b;
26          sf <= '0';
27        ELSIF cntrl = "10" THEN -- Complemento a 1
28          salida <= NOT a;
29          sf <= '0';
30        ELSIF cntrl = "11" THEN -- Complemento a 2
31          aux <= NOT('0' & a) + 1;
32          salida <= aux(9 DOWNTO 0);
33          sf <= aux(10);
34        END IF;
35      END IF;
36    END PROCESS OperacionesLogicas;
37  END ARCHITECTURE ArqLogicas;
```

Listing 7: Letras

```
1  LIBRARY IEEE;
2  USE IEEE.STD_LOGIC_1164.ALL;
3
4  ENTITY letras IS
5    PORT(
6      addr : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
7      word : OUT STD_LOGIC_VECTOR(6 DOWNTO 0)
8    );
9  END ENTITY;
10
11 ARCHITECTURE bhr OF letras IS
12
13   SUBTYPE textos IS STD_LOGIC_VECTOR(6 DOWNTO 0);
14   TYPE textos_pantalla IS ARRAY (11 DOWNTO 0) OF textos;
15
16   CONSTANT memory : textos_pantalla := (
17     0 => "0010010",
18     1 => "1000001",
19     2 => "0101011",
20     3 => "0001000",
21     4 => "0101111",
22     5 => "0000110",
23     6 => "0000111",
24     7 => "1000111",
25     8 => "1001111",
26     9 => "0100001",
27     10 => "0100011",
28     11 => "1000110"
29     );
30 BEGIN
31   PROCESS(addr)
32   BEGIN
33     CASE addr IS
34       WHEN "0000" => word <= memory(0);
35       WHEN "0001" => word <= memory(1);
36       WHEN "0010" => word <= memory(2);
37       WHEN "0011" => word <= memory(3);
38       WHEN "0100" => word <= memory(4);
39       WHEN "0101" => word <= memory(5);
40       WHEN "0110" => word <= memory(6);
41       WHEN "0111" => word <= memory(7);
```

```
42        WHEN "1000" => word <= memory(8);
43        WHEN "1001" => word <= memory(9);
44        WHEN "1010" => word <= memory(10);
45        WHEN "1011" => word <= memory(11);
46        WHEN OTHERS => word <= "1000000";
47      END CASE;
48    END PROCESS;
49 END ARCHITECTURE;
```

Listing 8: Display

```
1  LIBRARY IEEE;
2
3  USE IEEE.STD_LOGIC_1164.ALL;
4  USE IEEE.STD_LOGIC_UNSIGNED;
5  USE IEEE.NUMERIC_STD;
6
7  ENTITY display IS
8    PORT(
9      en : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
10     operacion : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
11     clk, trigger : IN STD_LOGIC;
12     d1,d2,d3,d4 : OUT STD_LOGIC;
13     a,b,c,d,e,f,g : OUT STD_LOGIC;
14     led : OUT STD_LOGIC
15   );
16 END ENTITY;
17
18 ARCHITECTURE bhr OF display IS
19   SIGNAL sel : INTEGER RANGE 0 TO 3 := 0;
20   SIGNAL var : INTEGER RANGE 0 TO 9999999 := 0;
21   SIGNAL seg : INTEGER RANGE 0 TO 15 := 0;
22   SIGNAL contador : INTEGER RANGE 0 TO 100 := 0;
23   SIGNAL aux : STD_LOGIC_VECTOR(3 DOWNTO 0);
24   SIGNAL selector : STD_LOGIC_VECTOR(3 DOWNTO 0);
25   SIGNAL msg, message : STD_LOGIC_VECTOR(19 DOWNTO 0);
26   SIGNAL disp, letter : STD_LOGIC_VECTOR(6 DOWNTO 0) := "0000000";
27   SIGNAL ledsito : STD_LOGIC := '0';
28   SIGNAL disparador : STD_LOGIC := trigger;
29   --SIGNAL disparador : STD_LOGIC := '1';
30   SIGNAL letrita : STD_LOGIC_VECTOR(3 DOWNTO 0) := "0000";
31   SIGNAL l1,l2,l3,l4 : STD_LOGIC_VECTOR(3 DOWNTO 0);
32   SIGNAL op : STD_LOGIC_VECTOR(3 DOWNTO 0):= "0000";
33
34   COMPONENT letras IS
35     PORT(
36       addr : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
37       word : OUT STD_LOGIC_VECTOR(6 DOWNTO 0)
38     );
39   END COMPONENT;
40
41   COMPONENT texto IS
42     PORT(
43       address : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
44       palabra : OUT STD_LOGIC_VECTOR(19 DOWNTO 0)
45     );
46   END COMPONENT;
47 BEGIN
48
49   desplegar: PROCESS(clk)
```

```
50  BEGIN
51    IF(RISING_EDGE(clk)) THEN
52      IF disparador = '1' THEN
53        -- Asignar 1 sola vez el valor de la letra
54        IF var = 0 THEN
55          message <= msg;
56        END IF;
57        IF seg < 15 THEN
58          -- Ventana deslizante operacion
59          l1 <= message(7 DOWNTO 4);
60          l2 <= message(11 DOWNTO 8);
61          l3 <= message(15 DOWNTO 12);
62          l4 <= message(19 DOWNTO 16);
63          IF (contador = 100) THEN
64            contador <= 0;
65            message <= message(15 DOWNTO 0) & l4;
66            var <= var + 1;
67            seg <= seg + 1;
68          END IF;
69          IF sel = 3 THEN
70            sel <= 0;
71          END IF;
72        ELSE
73          -- Salida numerica y obtener la entrada de la UA
74          ledsito <= '1';
75          l4 <= en(15 DOWNTO 12);
76          l3 <= en(11 DOWNTO 8);
77          l2 <= en(7 DOWNTO 4);
78          l1 <= en(3 DOWNTO 0);
79        END IF;
80        -- Colocar digito en el display correspondiente
81        CASE sel IS
82          WHEN 0 => selector <= "1000";aux <= l4;
83          WHEN 1 => selector <= "0100";aux <= l3;
84          WHEN 2 => selector <= "0010";aux <= l2;
85          WHEN 3 => selector <= "0001";aux <= l1;
86        END CASE;
87        -- Aumentar contadores
88        sel <= sel + 1;
89        contador <= contador + 1;
90      ELSE
91        aux <= "1111";
92      END IF;
93    END IF;
94    letrita <= aux;
95    IF ledsito = '0' THEN
96      disp <= letter;
97    ELSE
98      CASE letrita IS
99        -- Binario a BCD
100       WHEN "0000" => disp <= "1000000";
101       WHEN "0001" => disp <= "1111001";
102       WHEN "0010" => disp <= "0100100";
103       WHEN "0011" => disp <= "0110000";
104       WHEN "0100" => disp <= "0011001";
105       WHEN "0101" => disp <= "0010010";
106       WHEN "0110" => disp <= "0000010";
107       WHEN "0111" => disp <= "1111000";
108       WHEN "1000" => disp <= "0000000";
109       WHEN "1001" => disp <= "0010000";
```

```
110         WHEN "1111" => disp <= "1111111";
111         WHEN OTHERS => disp <= "0111111";
112       END CASE;
113     END IF;
114   END PROCESS desplegar;
115
116
117   selOperacion : PROCESS(operacion)
118   BEGIN
119     CASE operacion IS
120       WHEN "000000000001" => op <= "0000";
121       WHEN "000000000010" => op <= "0001";
122       WHEN "000000000100" => op <= "0010";
123       WHEN "000000001000" => op <= "0011";
124       WHEN "000000010000" => op <= "0100";
125       WHEN "000000100000" => op <= "0101";
126       WHEN "000001000000" => op <= "0110";
127       WHEN "000010000000" => op <= "0111";
128       WHEN "000100000000" => op <= "1000";
129       WHEN "001000000000" => op <= "1001";
130       WHEN "010000000000" => op <= "1010";
131       WHEN "100000000000" => op <= "1011";
132       WHEN OTHERS => op <= "1111";
133     END CASE;
134   END PROCESS selOPeracion;
135   d4 <= selector(3);
136   d3 <= selector(2);
137   d2 <= selector(1);
138   d1 <= selector(0);
139   a <= disp(0);
140   b <= disp(1);
141   c <= disp(2);
142   d <= disp(3);
143   e <= disp(4);
144   f <= disp(5);
145   g <= disp(6);
146
147   mensaje: texto PORT MAP(op, msg);
148   letra: letras PORT MAP(letrita, letter);
149 END ARCHITECTURE;
```

Listing 9: Display

```
1  LIBRARY IEEE;
2
3  USE IEEE.STD_LOGIC_1164.ALL;
4
5  ENTITY texto IS
6    PORT(
7      address : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
8      palabra : OUT STD_LOGIC_VECTOR(19 DOWNTO 0)
9    );
10 END ENTITY;
11
12 ARCHITECTURE bhr OF texto IS
13
14 SUBTYPE textos IS STD_LOGIC_VECTOR(19 DOWNTO 0);
15   TYPE textos_pantalla IS ARRAY (11 DOWNTO 0) OF textos;
16
17   CONSTANT memory : textos_pantalla := (
```

```
18      0 => "00000001001000100011",--suma
19      1 => "01000101000001100011",--resta
20      --2 => "001000100001011101101000",--multi
21      2 => "00100010000101110110",--mult
22      3 => "00110010100111111111",--and
23      4 => "10100100111111111111",--or
24      5 => "00101010011011111111",--not
25      6 => "10111010111111111111",--c01
26      7 => "10110011110011111111",--cA2
27      8 => "01001010010011111111",--ror
28      9 => "01001010011111111111",--rol
29      10 => "01110000011111111111",--LSL
30      11 => "0111000001001111111"--LSr
31      );
32
33  BEGIN
34    PROCESS(address)
35    BEGIN
36      CASE address IS
37        WHEN "0000" => palabra <= memory(0); --suma
38        WHEN "0001" => palabra <= memory(1); --suma
39        WHEN "0010" => palabra <= memory(2); --suma
40        WHEN "0011" => palabra <= memory(3); --suma
41        WHEN "0100" => palabra <= memory(4); --suma
42        WHEN "0101" => palabra <= memory(5); --suma
43        WHEN "0110" => palabra <= memory(6); --suma
44        WHEN "0111" => palabra <= memory(7); --suma
45        WHEN "1000" => palabra <= memory(8); --suma
46        WHEN "1001" => palabra <= memory(9); --suma
47        WHEN "1010" => palabra <= memory(10); --suma
48        WHEN "1011" => palabra <= memory(11); --suma
49        WHEN OTHERS => palabra <= "11111111111111111111"; --suma
50      END CASE;
51    END PROCESS;
52  END ARCHITECTURE;
```

Listing 10: Barrel Shifter

```
1  LIBRARY IEEE;
2  USE IEEE.std_logic_1164.ALL;
3  USE IEEE.numeric_std.ALL;
4  use IEEE.std_logic_unsigned.all;
5  ENTITY barrelShifter IS
6      PORT (
7          a : IN STD_LOGIC_VECTOR(9 DOWNTO 0);
8          cntrl, clk, iniciar : IN STD_LOGIC;
9          salShifters : OUT STD_LOGIC_VECTOR(9 DOWNTO 0);
10         salPrub : OUT STD_LOGIC
11     );
12 END ENTITY barrelShifter;
13
14 ARCHITECTURE shifters OF prueba IS
15     SIGNAL aux : STD_LOGIC_VECTOR(9 DOWNTO 0);
16     signal i: INTEGER RANGE 0 TO 9 := 0;
17 BEGIN
18     --Las opciones del shifter
19     Shifter : PROCESS (clk, cntrl)
20     BEGIN
21         IF iniciar = '0' THEN
22             aux <= "0000000000";
```

```
        ELSIF (rising_edge(clk)) THEN
            IF (cntrl = '0') THEN
                --LSL
                aux(9 downto 1) <= aux(8 DOWNTO 0);
                aux(0) <= a(i);
                i <= i + 1;
            ELSE
                aux <= aux(9) & aux(9 DOWNTO 1); --ASR
                aux(9) <= a(i);
                i <= i + 1;
            END IF;
            salShifters <= aux;
        END IF;
    END PROCESS Shifter;
    salPrub <= clk;
END ARCHITECTURE shifters;
```

Listing 11: Divisor de frecuencia

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
entity divisorFrecuencia is
    port (
        clk, iniciar: in STD_LOGIC;
        salidaLC : OUT STD_LOGIC
    );
end entity divisorFrecuencia;
architecture divFre of divisorFrecuencia is
    SIGNAL contador : INTEGER RANGE 0 TO 9999999 := 0;
    SIGNAL salidMed : std_logic;
begin

    DivisorFrecuencia : PROCESS (clk, iniciar)
    BEGIN
        IF iniciar = '0' THEN
            salidMed <= '0';
            contador <= 0;
        ELSIF rising_edge(clk) THEN
            IF contador = 9999999 THEN
                contador <= 0;
                salidMed <= NOT salidMed;
            ELSE
                contador <= contador + 1;
            END IF;
        END IF;
    END PROCESS DivisorFrecuencia;
    salidaLC <= salidMed;
end architecture divFre;
```

4. Conclusiones

4.1. Conclusiones Generales

El circuito presentado en la practica fue elaborado deacuerdo a los requerimientos solicitados, se trabajo de forma eficiente en la implementacion de las operaciones que se tenian que realizar ademas de ir aplicando diversos conceptos teoricos sobre las partes de un computador; ademas del uso de las matematicas para la elaboracion de un algoritmo que permitiece obtener los resultados esperados a partir de ciertas entradas de datos. Con esto podemos observar la importancia de los ALU como parte fundamental de los procesadores que se utilizan actualmente y por ello como futuros ingenieros debemos conocer su aplicacion tanto teorica como practica dando un enfoque hacia la problematica que buscamos resolver. Y al final podemos decir que: es poco, pero es trabajo honesto.

4.2. Conclusiones Eduardo Hernández Vergara

En el desarrollo de la práctica pudimos aprender/recordar algunos de los elementos más básicos de la programación en VHDL, esto con el objetivo de desarrollar nuestra ALU básica/intermedia, en lo personal hubo cosas complicadas al momento de intentar implementarlas a VHDL y programarlo en la FPGA, como fue el caso de la ventana deslizante, y la unidad aritmetica, la ventana deslizante fue especialmente retadora por los ciclos de reloj, ya que fue complicado sincronizar procesos adecuadamente para que saliera la ventana deslizante, mientras que la unidad Aritmetica simplemente no salia, en cuanto a las opciones de shifters y las operaciónes lógicas fueron relativamente sencillas, en el caso de los Shifters el unico problema era hacer que ciclara solo, mientras que las operaciones lógicas no fueron desafio alguno. Sin embargo estos obstaculos fueron superados para finalmente dar solución a la práctica.

4.3. Conclusiones José Ángel Rojas Cruz

En el desarrollo de la práctica, me di cuenta de la importancia que tiene conocer a fondo el funcionamiento de los componentes que forman parte de la unidad aritmética de la CPU, debido a que era importante conocerlo para saber cómo es que se manejarían los distintos componentes de la ALU y las salidas esperadas de cada uno de ellos, para así poder replicarlos en VHDL con la ayuda de la FPGA. Lo anterior se logró comprender al unir todos los elementos de la ALU en uno solo, desde la unidad aritmético-lógica hasta las operaciones de barrel shifters, teniendo en cuenta que en algunos casos tenían distintas formas de presentar las salidas, por lo que se debía de entender cómo lograr hacer este cambio y aplicarlo correctamente. Por último, la implementación de la ventana deslizante represento un reto por lo menos lógico debido a que se tuvieron que usar algunas herramientas que no se habían utilizado antes en programas desarrollados anteriormente, como consultar una memoria ROM y aplicar distintos eventos con solo una frecuencia de reloj. Creo que esta práctica hace que me dé cuenta de que necesito un estudio más profundo de las herramientas que nos brinda el lenguaje VHDL para poder facilitar y en algunos casos solucionar, problemas que se me presenten más adelante.

4.4. Conclusiones Erik Alcantara Covarrubias

Gracias a esta practica nos dimos cuenta de la gran importancia de el uso de los diferentes componentes de una CPU, la unidad de control, los multiplexores, y la ALU fueron muy complicados de realizar, pero gracias a ellos pudimos hacer operaciones matematicas y lograr decisiones en base a datos obtenidos de forma automatica

4.5. Conclusiones Mariana Martínez Alquicira

El desarrollo de esta práctica nos sirvió para recordar algunos conceptos manejados en la anterior materia, al igual que poner en práctica conocimientos que adquirimos en las clases, como el entendimiento de las partes de la computadora y que parte tiene control de qué. Con respecto a la programación, la parte que más se nos complico fue la parte aritmética, ya que nos percatamos que al ponerlo en marcha, obteníamos resultados erróneos cuando las cifras eran iguales. La parte lógica fue más sencilla ya que únicamente para el complemento 1, obtenemos la negación, para el complemento 2 usamos la misma negación del complemento 1 y le sumamos uno, por ultimo las operaciones AND y OR. Y por parte de las operaciones Shifters fue un tanto complicado pero no presentó más inconvenientes y la parte con más trabajo fue la parte del display. Al final el circuito pudo cumplir el objetivo de la práctica, el cual implicaba la elaboración de una ALU en la cual se pueden hacer operaciones aritméticas, lógicas y Shifters, con datos obtenidos de forma aleatoria.

5. Referencias

Referencias

[1] Stallings, W. (2003). *Computer Organization and Architecture*. Pearson Education International.

[2] Jasinski, R. (2015). *Efective Coding with VHDL: principles and best practice*. The MIT Press.

[3] Akka Technologies. (n.d.). *FPGA: qué es y cuáles son las características de este componente. [online]*. Available at: https://www.akka-technologies.com/fpga/ [Accessed 16 Oct. 2022]..