UNIVERSIDAD DEL NORTE
DEPARTAMENTO DE INGENIERÍAS ELÉCTRICA Y ELECTRÓNICA
Asignatura: Lógica Digital. Docente: Carlos Cárdenas.

Lab 2: Diseño de ALU (unidad aritmética lógica)

Grupo 05: Barcenas Santiago, Bermudez Laura, Gutierrez Alex
{jsbarcenas, bermudesl, delahozfa} @uninorte.edu.co
28 de octubre de 2020

***Abstract*- - This report shows the detailed analysis for the second laboratory practice regarding the implementation of a 8-bit ALU capable of performing logical and arithmetic operations such as: AND, OR, addition, subtraction, and comparison in complement to 2, in this time, the design made was only simulated in logisim and Quartus.**

key words

Sumador lógico, restador lógico, comparador lógico, Bit, ALU.

I. INTRODUCCIÓN

Hoy en día es más frecuente encontrar uso de la tecnología para diferentes aplicaciones. Dicha tecnología casi siempre está compuesta de elementos o dispositivos electrónicos que realizan cálculos u operaciones. El objetivo del laboratorio es diseñar una unidad especializada capaz de realizar dichas operaciones o bien conocida como ALU. En específico una ALU de 8 bits que permite realizar 5 diferentes operaciones lógicas y aritméticas como son, OR, AND, suma, resta y comparación. De igual forma es necesario trabajar con ellas de forma independiente, usando ya sea circuitos con compuertas lógicas como también con multiplexores. Para llevar a cabo este laboratorio los circuitos fueron descritos y simulados en logisim y en lenguaje VHDL por medio del software Quartus. Fue necesario implementar los circuitos de sumador, restador, comparador, conversor entre otros y luego exportarlos como librerías en logisim para poder obtener el circuito completo de la ALU de 4 bits. Adicionalmente estos diferentes sistemas de MSI (codificadores, decodificadores, multiplexores, demultiplexores, sumadores y comparadores) permiten desarrollar funciones más complejas para simplificar ciertas áreas del circuito lógico en comparación con el uso de compuertas básicas[1].En este laboratorio, se desea diseñar un circuito que realice diversas operaciones usando números ingresados de forma binaria, mostrándoles a través de displays 7 segmentos que muestran dichos números en forma decimal, pasando a ser operados según la selección indicada en el ALU de 8 Bits, para luego mostrar un resultado decodificador de binario a decimal en un display como el descrito anteriormente.

III. MARCO TEÓRICO

Para la realización de este laboratorio, se utilizaron los siguientes conceptos: ALU, comparador lógico, I. ALU (Arithmetic logic unit) Una ALU se compone básicamente de dos registros de entrada, un registro de salida, un registro de estado (Overflow) y un registro de control o selectoras. La selección de cada una de las operaciones que puede realizar la ALU es posible gracias al código de las selectoras. (figura 1)

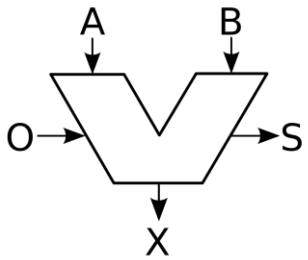


Figura 1.1: Unidad aritmética lógica

II. Un circuito **comparador**, como su nombre lo indica, compara dos entradas binarias para indicar

la relación de igualdad o desigualdad entre ellas. (Figura 1.2)

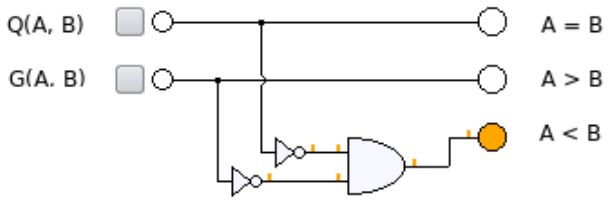


Figura 1.2: Comparador

III. **Bit** es el acrónimo de binary digit, un bit es un dígito del sistema de numeración binario. El bit es la unidad mínima de información empleada en la informática en cualquier dispositivo digital. (figura 1.3)

Figura 1.3: Bit

IV. Logisim: es un simulador lógico que permite diseñar y simular circuitos electrónicos digitales mediante una interfaz gráfica de usuario.

V. Quartus: es un programa para el análisis y síntesis de los circuitos realizados en VHDL.

III. DESARROLLO

El primer paso para realizar el laboratorio fue diseñar una ALU. Su diseño se implementa tanto en logisim como en quartus fue necesario hacer el montaje de otros circuitos combinacionales que realizarán las operaciones para así importarlos como librerías y poderlos implementar dentro de la ALU.

Cuando se introduce un código binario, el sumador y el restador cumplen con la función de sumar o restar los números decimales producidos por los binarios introducidos. Por otro lado, la función del comparador es, como su nombre lo dice, comparar los decimales producidos por el código binario. En el caso del circuito final incorporado en este laboratorio, comparador es el que hace que se encienda un LED cuando se cumple que B>A. Por otro lado, se encenderá el LED llamado Overflow cuando alguna de las operaciones no se pueda realizar.

La representación estándar por suma de productos para el comparador, sumador y restador son las siguientes:

A. Comparador

• Si A>B entonces X = AB'

• Si A=0 entonces Y = A'B' + AB

• Si B>A entonces Z = A'B

B. Sumador

• Para la salida X entonces X = A'B'C + A'BC' + AB'C' + ABC

• Para la salida Y entonces Y = BC + AC + AB

C. Restador

• Para la salida X entonces X = A'B'C A'BC' + AB'C' + ABC

• Para la salida Y entonces Y = A'C + A'B + BC

Para realizar la ALU se hicieron otros circuitos, que luego se importaron como librerías para implementar el circuito final de la ALU. Primero se identificó las operaciones lógicas que se usarían para llevar a cabo el circuito mediante la tabla de verdad.

C_in	A	B	C_out	Suma
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Figura 2: Sumador de 1 Bit, tabla de verdad

En segunda instancia se tuvo en cuenta que:

-El diseño para un sumador de 1 Bit, debía cumplir con las condiciones iniciales planteadas en el laboratorio, el resultado del diseño del circuito se logra apreciar en la figura 2 y 2.1 respectivamente

a partir de la tabla de verdad como se muestra a continuación:

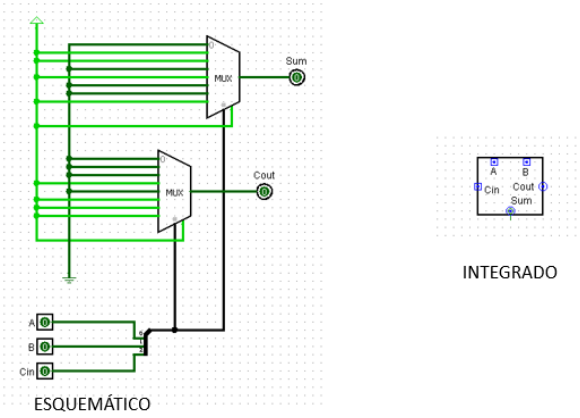


Figura 2: Sumador de 1 Bit

-EL diseño para un sumador de 8 Bits: en este caso diseñamos un circuito que nos realizará la operación suma de 8 Bits como se muestra a continuación:

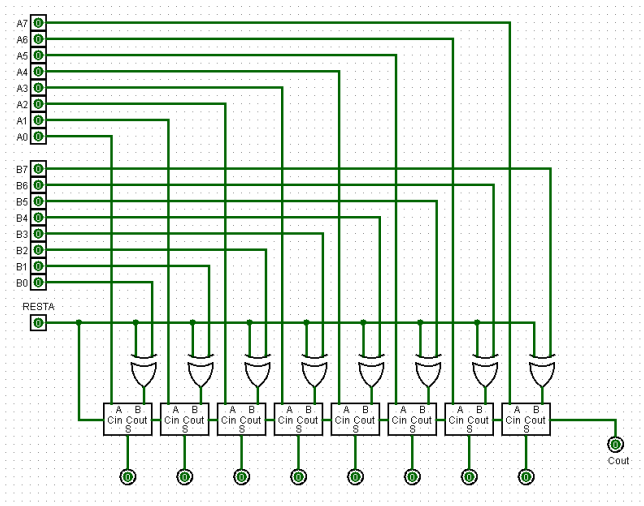


Figura 3: Sumador de 8 Bits

Restador de 8 Bits: tuvimos que diseñar un restador de 8 Bits, para ello usamos el sumador de 8 Bits previamente implementado y le agregamos una entrada "resta" que al habilitarla nos trabaja como tal, esto se ilustra en la figura 4 y podemos observar como la entrada "resta" está activa.

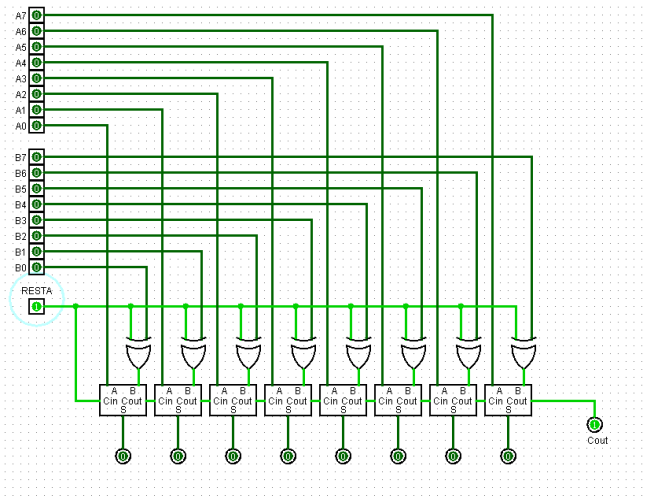


Figura 4: restador de 8 Bits

Comparador de 8 bits: diseñamos un circuito para comparar las entradas y obtendremos la salida según corresponda la igualdad (A=B, A<B o A>B), esto lo podemos verificar en la figura 5.

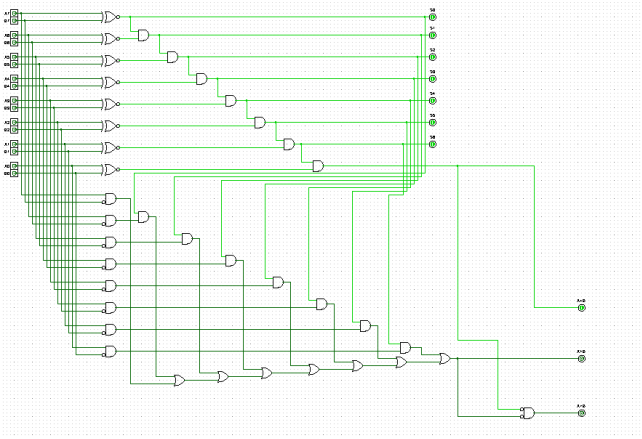


Figura 5: Comparador de 8 bits

Decodificador de displays 7 segmentos: para poder habilitar los displays de 7 segmentos, tuvimos que diseñar un decodificador de para cada dígito, para ello utilizamos la información de la tabla de verdad del decodificador [2]. Los decodificadores integrados de cada dígito se encuentran en la figura 6.

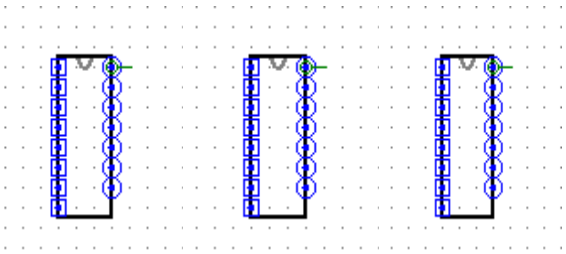


Figura 6: decodificadores

Luego de tener todos nuestros circuitos prediseñados los usamos en la creación de nuestra ALU, para ello utilizamos la tabla selectora de operación ilustrada en la figura 7, luego de eso utilizamos un multiplexor para habilitar las operaciones y dependiendo de la configuración se habilita el resultado de la operación requerida.

Línea de Control F	Operación a Realizar
00X	AND
010	OR
011	SUMA
1X0	COMPARACION
1X1	RESTA (A-B)

Figura 7: tabla selectora de operación

La implementación de nuestra ALU se encuentra en la figura 8.

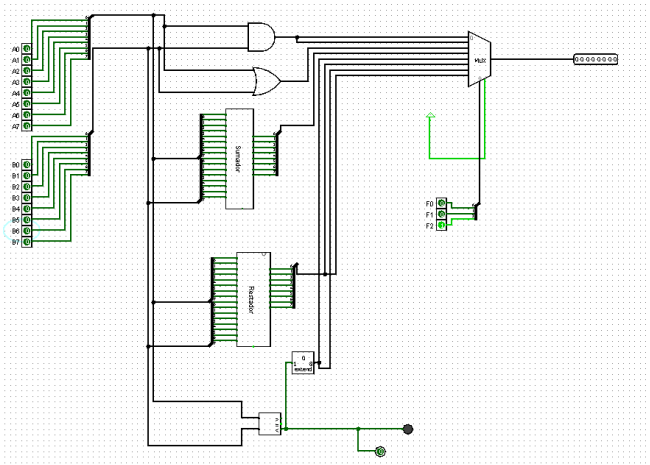


Figura 8: ALU

Al tener nuestra ALU funcionando de manera correcta le conectamos los displays para visualizar los valores ingresados en binario de forma decimal, esto lo observamos en la figura 9, además en la figura 10 se nos ilustra cómo debe ser conectado el display de 7 segmento.

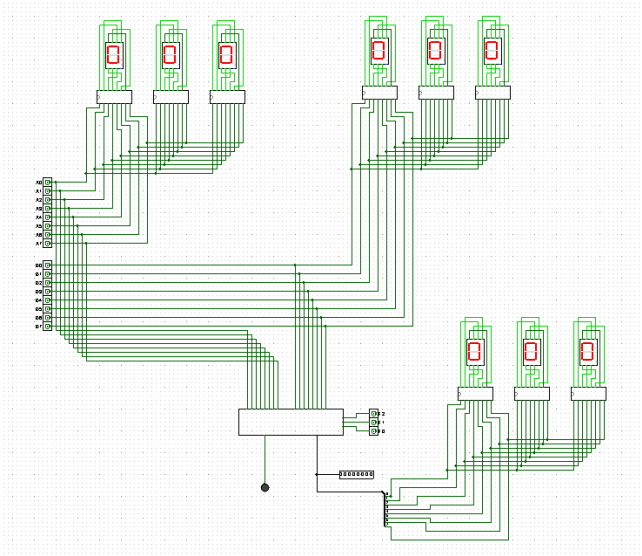
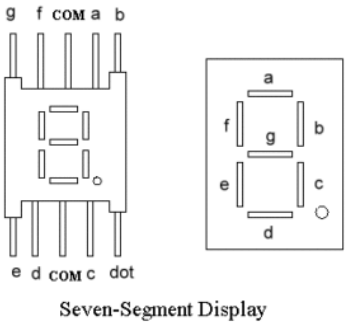


Figura 9: ALU con displays

IV. VHDL



Seven-Segment Display

Figura 10. Display 7 segmentos con letras.

se colocaron compuertas OR para el segundo display ya que ahí solo se necesitaba un 0 o un 1. Para un 0 debían estar activos ABCDEF y para un 1 solo BC. De esta manera, la OR hace que se conecten B y C para que siempre estén encendidos y que cuando se necesite un 1 se apaguen los demás y solo queden encendidos B y C.

Los siguientes códigos fueron implementados en VHDL:

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

entity OR_8bit is port(

    A,B   :  in std_logic_vector(7 downto 0);
    R     :  OUT std_logic_vector(7 downto 0);
);


end OR_8bit;


architecture arq of OR_8bit is

begin
R <= A or B;
end arq;
```

Figura 11: Código de compuerta OR

```vhdl
ibrary ieee;
use ieee.std_logic_1164.all;

entity AND_8bit is port(

    A,B   :  in std_logic_vector(7 downto 0);
    R     :  OUT std_logic_vector(7 downto 0);
);


end AND_8bit;


architecture arq of AND_8bit is

begin
R <= A and B;
end arq;
```

Figura 12: Código de compuerta AND.

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

entity sum1bit is port(

    A,B,Cin  : in std_logic;
    sum, Cout: out std_logic

);
end entity;

architecture rtl of sum1bit is
-- signals
   signal AxB : std_logic;
begin
AxB <= A xor B;

sum <= AxB xor Cin;
Cout <= (A and B) or (AxB and Cin);

end architecture;
```

Figura 13: Código compuerta sumadora de 1 bit.

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

entity sum8bit is port(

   A,B   : in std_logic_vector(7 downto 0);
   Cin   : in std_logic;
   sum   : out std_logic_vector(7 downto 0);
   Cout  : out std_logic

);
end entity;

architecture rtl of sum8bit is
-- list components

component sum1bit is port(
   A,B,Cin  : in std_logic;
   sum, Cout: out std_logic
);
end component;
-- Signals

   signal c : std_logic_vector(6 downto 0);
begin

u0: sum1bit port map(A(0),B(0),Cin,sum(0),c(0));
u1: sum1bit port map(A(1),B(1),C(0),sum(1),c(1));
u2: sum1bit port map(A(2),B(2),C(1),sum(2),c(2));
u3: sum1bit port map(A(3),B(3),C(2),sum(3),c(3));
u4: sum1bit port map(A(4),B(4),C(3),sum(4),c(4));
u5: sum1bit port map(A(5),B(5),C(4),sum(5),c(5));
u6: sum1bit port map(A(6),B(6),C(5),sum(6),c(6));
u7: sum1bit port map(A(7),B(7),C(6),sum(7),Cout);


end architecture;
```

Figura 14: Código compuerta sumadora de 8 bits.

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

entity Alu8bit is port(

   A,B   : in std_logic;
   sel   : in std_logic_vector(1 downto 0);
   R     : out std_logic
);
end entity;

architecture rtl of Alu8bit is
-- Component List

component sum8bit is port(
   A,B,Cin  : in std_logic;
   sum, Cout: out std_logic
);
end component;


component AND_8bit is port(
   A,B   :  in std_logic_vector(7 downto 0);
   R     :  OUT std_logic_vector(7 downto 0);

   );
end component;


component OR_8bit is port(
   A,B   :  in std_logic_vector(7 downto 0);
   R     :  OUT std_logic_vector(7 downto 0);

   );
end component;


component comparador is port(
   A : in std_logic_vector(7 downto 0);
   B : in std_logic_vector(7 downto 0);
   menor : out std_logic;
   igual : out std_logic;
```

Figura 15: Código ALU 8 bits parte 1.

```
component comparador is port(
    A : in std_logic_vector(7 downto 0);
    B : in std_logic_vector(7 downto 0);
    menor : out std_logic;
    igual : out std_logic;
    mayor:  out std_logic

    );
end component;

component mux_8_1bit is port(
    A : in std_logic_vector(7 downto 0);
    Sel : in std_logic_vector(2 downto 0);
    B : out std_logic
    );

end component;


-- signals
    signal Cin_signal,Cout_signal: std_logic;
    signal vector_mux : std_logic_vector(7 downto 0);


begin

vector_mux(0) <= AND_8bit;
vector_mux(1)  <= OR_8bit;
vector_mux(2) <= sum8bit ;
vector_mux(3)  <= comparador;
Cin_signal <= '0';
sumador: sum8bit port map(A,B,Cin_signal,vector_mux(2),Cout_
Mux: mux_8_1bit port map(vector_mux,sel,R);

end architecture;
```

Figura 16. Código ALU parte 2.

V. ANÁLISIS DE RESULTADOS

Se logró diseñar el circuito deseado con las especificaciones de la guía del laboratorio en Logisim. Se implementó una ALU complemento a 2, con la función de sumador, restador, comparador y también con las compuertas AND y OR. El circuito final permitió ingresar los valores de A, B y F de manera binaria y se logró visualizar en los display de 7 segmentos los valores ingresados. Además, los dos leds que se observan en el circuito de la figura 15 prendían, el de overflow cuando la operación supera el valor máximo o mínimo permitido y el de comparación si se realiza la operación de comparación y B > A. En la parte del código en VHDL se utilizó un código con descripción estructural, lo que quiere decir que se desarrollaron los códigos correspondientes a cada operación que se deseaba hacer, como también para el ALU que se encargaría de realizar y seleccionar dichas operaciones. Estos códigos fueron suministrados por el profesor en clase, pero también fue necesario realizar algunos por nuestra cuenta para obtener el resultado deseado. Luego de desarrollar los códigos, se presentaron algunos errores al compilar, que posteriormente serán corregidos, pues eran errores al digitar algún símbolo en el código. También hubo algunas complicaciones porque el software no funcionaba correctamente al principio por errores en los pasos de instalación, y esto entorpece el desarrollo del trabajo. Finalmente, se introdujeron los códigos comportamentales en el código estructural con ayuda de las librerías de trabajo de Quartus. Al principio se generan 2 variables de 8 bits en código binario, que son recibidas por el decodificador para el display 7 segmentos, que se encarga de reflejar en dicho display el número en forma decimal, posteriormente las mismas variables ingresan en la ALU, el cual tiene otra entrada de 3 Bits en la cual se ingresa el código de la operación deseada, posteriormente se decodifica el resultado de dicha operación y es mostrada en forma decimal en otro display de 7 segmentos.

VI. CONCLUSIONES

Finalmente, se puede concluir que las Unidades Aritmética Lógicas permiten simplificar y optimizar los circuitos lógicos que usan operaciones más complejas. cabe resaltar que para diseñar el circuito hubo que realizar arreglos para la ALU de 1 bit y con estas conformar la de 8 Bits que se encargaría de hacer todas las operaciones requeridas por el problema. De igual forma fue necesario realizar un decodificador para el display de 7 segmentos usando un demultiplexor. A pesar de que las ALU simplifican el número de componentes necesarios para el montaje, requieren un planteamiento un poco más complejo del problema a la hora de desarrollar el código en VHDL y el esquema lógico.

VII. REFERENCIAS

[1] D. William, R. Curtis, Digital Design: A Systems Approach. Cambridge University Press, 2020.

[2] Display LED 7 Segments. [2] Recuperado de:

https://saber.patagoniatec.com/2016/07/display-led-7-segmentos/

[3] Visualizador de siete segmentos. [Figura 10]. Recuperado de: https://es.wikipedia.org/wiki/Visualizador_de_siete_segmentos