Instituto Tecnológico de Buenos Aires

Electrónica III

Trabajo Práctico I

Grupo 3:

De Leon, Nicolás
Leg. 57232

Lago, Valentina
Leg. 57249

Bualó, Santiago
Leg. 57557

Silva, Gonzalo
Leg. 56089

Vigón, Tomás
Leg. 57327

Profesor:

Kevin Dewald

Presentado: 5 de septiembre de 2018

Índice

1 Ejercicio 1

1.1 Introducción

Uno de los problemas de la era digital que se han enfrentado, es el hecho de representar números, puesto que para ello, solo se tiene una cantidad finita de dígitos. Para esto, se contemplaron dos solucienes: escribir los numeros en formato punto fijo, o en formato punto flotante, tanto para números no signados como signados. En este ejercicio, hablaremos del punto fijo.

El punto fijo es una forma de representación numérica que consiste en previamente establecer la cantidad de dígitos que se desean para la parte entera, y establecer otra cantidad de dígitos para la parte fraccionaria (más conocida como la que "está detras de la coma"). Como los dígitos pueden ser establecidos de manera arbitraria, existen parámetros para definirlos, a saber: resolución, rango y exactitud.

La resolución consiste en determinar la magnitud más pequeña que es posible de representar con la cantidad de dígitos elegidos. El rango referencia la cantidad neta que es posible de representar con los dígitos elegidos, ya que el valor sale de la resta entre el número más grande representable menos el más pequeño representable. Por último, la exactitud consiste en el máximo error que se comete entre un número real y su representación.

1.2 Objetivo

El objetivo del presente ejercicio es realizar un programa en el cual, dandole una cantidad de cifras de la parte entera, la cantidad de cifras de la parte fraccionaria y la condición de signado o no signado, devuelva la resolución y el rango del sistema conformado.

1.3 Funcionamiento del programa

El programa fue realizado en el lenguaje C. Se le pasan por la línea de parámetros los siguientes valores: signado/no signado, cantidad de bits de la parte entera, cantidad de bits de la parte fraccionara. En primera instancia, el programa evalúa que la cantidad de valores ingresados sea la correcta. Luego, extrae los valores para poder trabajar con ellos y verifica que éstos ingresados sean validos, a saber: el valor signado/no signado acepta el valor 0 como no signado y 1 o '-' como signado; los otros valores sólo es posible ingresar números.

Luego, procede a hacer las cuentas para devolver los parámetros estipulados, se basa en trabajar con potencias de 2 tanto negativas como positivas, y según si el parametro es signado o no signado.Por último, se imprimen en pantalla los valores calculados.

Para finalizar, destacamos que se realizó una función propia para hacer la operación 2^n un poco mas sencilla.

2 Ejercicio 2

Dada la siguiente expresión en maxtérminos

$$f(d,c,b,a) = \prod (M_0, M_1, M_5, M_7, M_8, M_{10}, M_{14}, M_{15})$$

se la reescribió en función de mintérminos para realizar las simplificaciones

$$f(d,c,b,a) = \sum (m_2, m_3, m_4, m_6, m_9, m_{11}, m_{12}, m_{13})$$

2.1 Álgebra booleana

La función expandida tiene la forma

$$f(d,c,b,a) = (\bar{d}.\bar{c}.b.\bar{a} + \bar{d}.\bar{c}.b.a) + (\bar{d}.c.\bar{b}.\bar{a} + \bar{d}.c.b.\bar{a}) + (d.\bar{c}.\bar{b}.a + d.\bar{c}.b.a) + (d.c.\bar{b}.\bar{a} + d.c.\bar{b}.a)$$

Se agrupó en paréntesis para aplicar la propiedad distributiva $12.a$ del libro sobre esos términos

$$f(d,c,b,a) = \bar{d}.\bar{c}.b.(\bar{a} + a) + \bar{d}.c.\bar{a}.(\bar{b} + b) + d.\bar{c}.a.(\bar{b} + b) + d.c.\bar{b}.(\bar{a} + a)$$

Luego, utilizando la propiedad $8b$ también del libro

$$f(d,c,b,a) = \bar{d}.\bar{c}.b + \bar{d}.c.\bar{a} + d.\bar{c}.a + d.c.\bar{b}$$

Donde la función quedó expresada en su forma más simplificada.

2.2 Mapas de Karnaugh

Los mapas de Karnaugh son otra herramienta para la simplificación de funciones y se puede usar tanto con la expresión en mintérminos como con maxtérminos. A continuación se muestra el mapa de Karnaugh para este ejercicio.

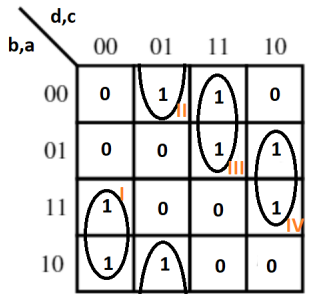

Figura 1: Mapa de Karnaugh

Se agruparon los unos ya que estos representan los mintérminos y por grupo únicamente se pueden tomar 2^n unos; mientras más grande sea la agrupación, mayor es la simplificación.

Las simplificaciones son por grupo, la variable que se modifica no es tenida en cuenta, las que están asociadas al cero se niegan y las que están asociadas al uno se expresan normalmente. Las expresiones obtenidas de cada grupo son:

I	II	III	IV
$d.\bar{c}.b$	$d.c.\bar{a}$	$d.c.\bar{b}$	$d.\bar{c}.a$

Cuadro 1: Simplificación

Finalmente sumándolos se llega al resultado final.

2.3 Circuito resultante con compuertas AND, OR, NOT

Teniendo en cuenta la expresión más simplificada se armó el siguiente circuito

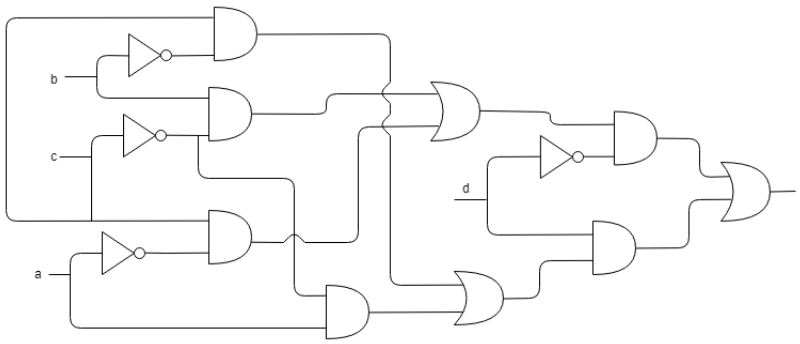

Figura 2: Circuito simplificado

Se utilizaron 13 componentes y 22 entradas, por lo tanto, el costo del circuito toma el valor de 35.

2.4 Circuito solamente con compuertas NOR

De la misma manera que la mencionada en el punto anterior se obtuvo el siguiente circuito

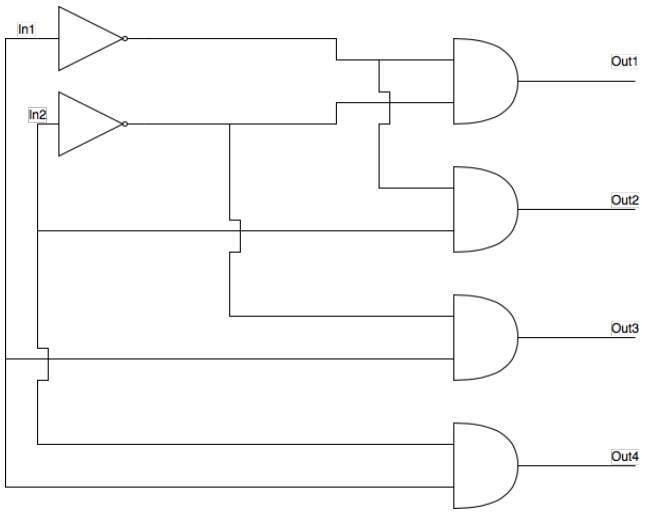

Figure 4: Decoder de dos entradas

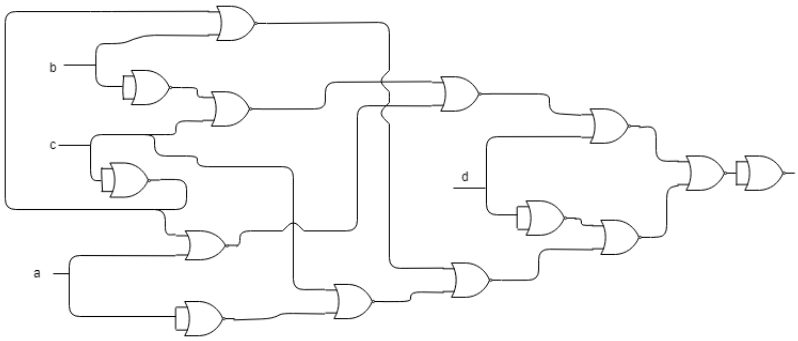

Figura 3: Circuito simplificado

El circuito contiene 14 componentes y 28 entradas por lo que el costo es de 42.

Ejercicio 3

Se desean realiazar dos modulos en Verilog, un Decoder de dos entradas y un Mux de cuatro entradas. Ademas de esos dos modulos se implemento un Encoder.

El programa que se realizo en Verilog, fue hecho con if statments, esto quiere decir que este, es ajeno a las compuertas logicas y a su distribucion necesarias para poner en funcionamiento el circuito. A continuacion, se insertaran figuras de estos tres circuitos, para poder entender el funcionamiento de estos.

Este es un decoder como el que se realizo en el programa, tiene dos entradas y cuatro salidas y su tabla de verdad es:

Input 1	Input 2	Output 1	Output 2	Output 3	Output 4
0	0	**1**	**0**	**0**	**0**
0	1	**0**	**1**	**0**	**0**
1	0	**0**	**0**	**1**	**0**
1	1	**0**	**0**	**0**	**1**

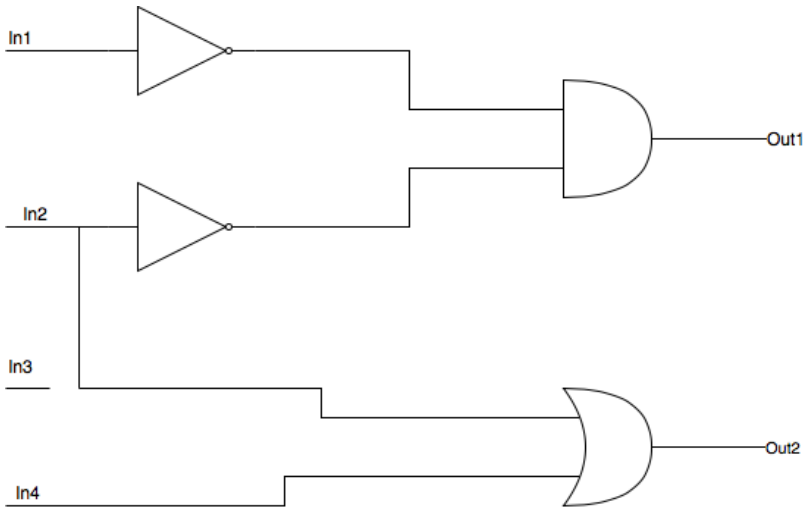

Figure 6: Encoder de 4 entradas

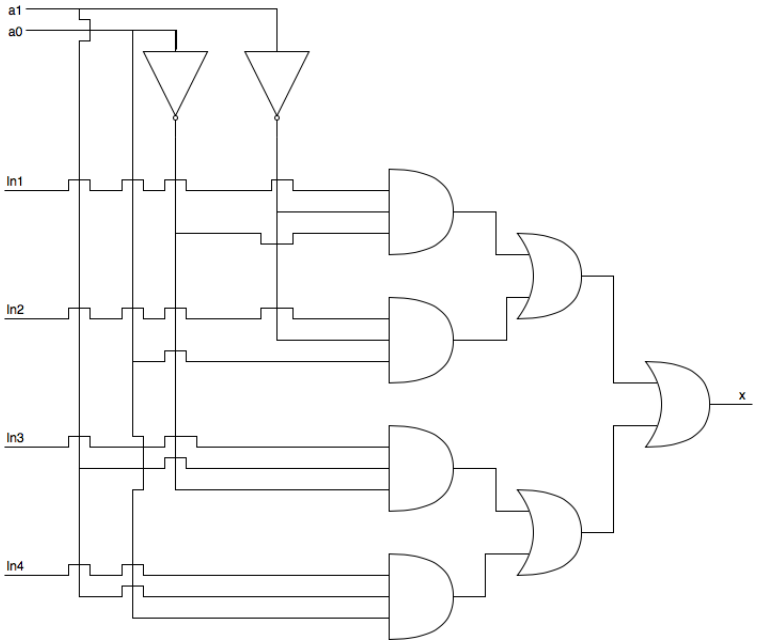

Figure 5: Mux de 4 entradas

Este es un Mux como el que se realizo en el progama, tiene cuatro entradas, una salida y dos select lines. La tabla de verdad es:

a1	a0	x
0	0	**In1**
0	1	**In2**
1	0	**In3**
1	1	**In4**

Este es un encoder como el que se realizo en Verilog, es de 4 entradas y dos salidas. La tabla de verdad es:

In1	In2	In3	In4	Out1	Out2
1	0	0	0	**0**	**0**
0	1	0	0	**0**	**1**
0	0	1	0	**1**	**0**
0	0	0	1	**1**	**1**

Como se puede ver en la figura 3, una de las entradas no esta conectada al circuito, pero esto no modifica la salida, ya que puede ser 1 en un solo caso, en el cual todas las demas entradas son 0, en el resto de los casos, In3 es siempre 0.

Ejercicio 4

Se desea realizar un circuito que convierta un número binario de 4 bits en su complemento a dos.

- Exprese el valor de cada bit de salida en función de los mintérminos de los bits de entrada.
- Exprese el valor de cada bit de salida en forma simplificada.
- Dibuje el circuito lógico resultante utilizando compuertas AND, OR, NOT.
- Implemente el circuito resultante en Verilog.

Para obtener el complemento a dos de un numero binario se invierte el valor de cada una de sus cifras, es decir, se realiza el complemento a uno, y luego se le suma uno al número resultante de la inversión.

Su utilidad principal se encuentra en las operaciones matemáticas con números binarios. En particular, la resta de números binarios se facilita enormemente utilizando el complemento a dos: la resta de dos números binarios puede obtenerse sumando al minuendo el complemento a dos del sustraendo.

Además, llamaremos mintérmino m_i a aquel que se forma multiplicando (AND lógico) todas las variables, negando aquellas que valen 0 en la combinación para la cual queremos que el mintérmino valga 1. Para N variables booleanas, existen 2^N mintérmino, uno para cada posible combinación de ellas.

A continuación se realiza la tabla de verdad para un numero binario de 4 bits. Donde a cada número binario ($x_1x_2x_3x_4$) le corresponde su respectivo complemento a dos a la salida ($f_1f_2f_3f_4$).

x_1	x_2	x_3	x_4	f_1	f_2	f_3	f_4	m_i
0	0	0	0	0	0	0	0	m_0
0	0	0	1	1	1	1	1	m_1
0	0	1	0	1	1	1	0	m_2
0	0	1	1	1	1	0	1	m_3
0	1	0	0	1	1	0	0	m_4
0	1	0	1	1	0	1	1	m_5
0	1	1	0	1	0	1	0	m_6
0	1	1	1	1	0	0	1	m_7
1	0	0	0	1	0	0	0	m_8
1	0	0	1	0	1	1	1	m_9
1	0	1	0	0	1	1	0	m_{10}
1	0	1	1	0	1	0	1	m_{11}
1	1	0	0	0	1	0	0	m_{12}
1	1	0	1	0	0	1	1	m_{13}
1	1	1	0	0	0	1	0	m_{14}
1	1	1	1	0	0	0	1	m_{15}

Table 2: Complemento a dos ($f_1f_2f_3f_4$) del bit de entrada ($x_1x_2x_3x_4$)

Se expresa la salida como función de los mintérminos. Para expresar la función en mintérminos tomamos donde la función sea 1 y unimos los mintérminos con sumas:

$$f_1(m_i) = m_1 + m_2 + m_3 + m_4 + m_5 + m_6 + m_7 + m_8$$

$$f_2(m_i) = m_1 + m_2 + m_3 + m_4 + m_9 + m_{10} + m_{11} + m_{12}$$

$$f_3(m_i) = m_1 + m_2 + m_5 + m_6 + m_9 + m_{10} + m_{13} + m_{14}$$

$$f_4(m_i) = m_1 + m_3 + m_5 + m_7 + m_9 + m_{11} + m_{13} + m_{15}$$

Se reemplaza por los valores de entrada,

$$f_1(x_1, x_2, x_3, x_4) = \overline{x_1}\overline{x_2}\overline{x_3}x_4 + \overline{x_1}\overline{x_2}x_3\overline{x_4} + \overline{x_1}\overline{x_2}x_3x_4 + \overline{x_1}x_2\overline{x_3}\overline{x_4} + \overline{x_1}x_2\overline{x_3}x_4 + \overline{x_1}x_2x_3\overline{x_4} + \overline{x_1}x_2x_3x_4 + x_1\overline{x_2}\overline{x_3}\overline{x_4}$$
$$f_2(x_1, x_2, x_3, x_4) = \overline{x_1}\overline{x_2}\overline{x_3}x_4 + \overline{x_1}\overline{x_2}x_3\overline{x_4} + \overline{x_1}\overline{x_2}x_3x_4 + \overline{x_1}x_2\overline{x_3}\overline{x_4} + x_1\overline{x_2}\overline{x_3}x_4 + x_1\overline{x_2}x_3\overline{x_4} + x_1\overline{x_2}x_3x_4 + x_1x_2\overline{x_3}\overline{x_4}$$
$$f_3(x_1, x_2, x_3, x_4) = \overline{x_1}\overline{x_2}\overline{x_3}x_4 + \overline{x_1}\overline{x_2}x_3\overline{x_4} + \overline{x_1}x_2\overline{x_3}x_4 + \overline{x_1}x_2x_3\overline{x_4} + x_1\overline{x_2}\overline{x_3}x_4 + x_1\overline{x_2}x_3\overline{x_4} + x_1x_2\overline{x_3}x_4 + x_1x_2x_3\overline{x_4}$$
$$f_4(x_1, x_2, x_3, x_4) = \overline{x_1}\overline{x_2}\overline{x_3}x_4 + \overline{x_1}\overline{x_2}x_3x_4 + \overline{x_1}x_2\overline{x_3}x_4 + \overline{x_1}x_2x_3x_4 + x_1\overline{x_2}\overline{x_3}x_4 + x_1\overline{x_2}x_3x_4 + x_1x_2\overline{x_3}x_4 + x_1x_2x_3x_4$$

Finalmente, utilizando mapas de Karnaugh, se simplifican las ecuaciones. Los mapas de Karnaugh reducen la necesidad de hacer cálculos extensos para la simplificación de expresiones booleanas.

El mapa de Karnaugh consiste en una representación bidimensional de la tabla de verdad de la función a simplificar. Puesto que la tabla de verdad de una función de N variables posee 2^N filas, el mapa K correspondiente debe poseer también 2^N cuadrados. Las variables de la expresión son ordenadas en función de su peso y siguiendo el código Gray, de manera que sólo una de las variables varía entre celdas adyacentes. La transferencia de los términos de la tabla de verdad al mapa de Karnaugh se realiza de forma directa, albergando un 0 ó un 1, dependiendo del valor que toma la función en cada fila.

En la Figura 1 podemos observar el mapa de Karnaugh de f_1 con cuatro conjuntos encerrados por distintos colores:

• La función del conjunto amarillo se produce porque se observa que ninguna x_i varia durante su conjunto. Como x_2, x_3 y x_4 no varian pero valen 0, entonces obtenemos como función del conjunto $x_1\overline{x_2}\overline{x_3}\overline{x_4}$.

• La función del conjunto rojo se produce al observar que x_1 y x_2 no varian. Sin embargo, como $x_1$es cero, obtenemos como función del conjunto $\overline{x_1}x_2$.

• En el caso del conjunto verde, x_1 y x_3 no varian pero x_1 es cero, por lo que resulta $\overline{x_1}x_3$.

• Finalmente, en el conjunto azul, x_1 y x_4 no varian pero x_1 es cero, por lo que resulta $\overline{x_1}x_4$.

De esta manera,

$$f_1(x_1, x_2, x_3, x_4) = x_1\overline{x_2}\overline{x_3}\overline{x_4} + \overline{x_1}x_2 + \overline{x_1}x_3 + \overline{x_1}x_4 = x_1\overline{x_2}\overline{x_3}\overline{x_4} + \overline{x_1}(x_2 + x_3 + x_4)$$

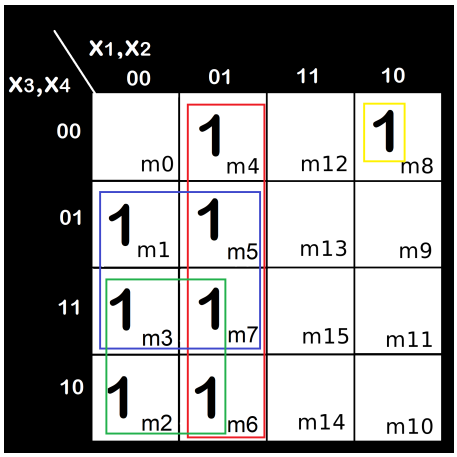

Figure 7: Mapa de Karnaugh $f_1(x_1, x_2, x_3, x_4)$

El análisis para los siguientes casos se realiza de la misma manera, pero no será detallado. Solo se procederá a mostrar las funciones simplificadas y sus respectivos mapas de Karnaugh.

Para f_2,

$$f_2(x_1, x_2, x_3, x_4) = x_2\overline{x_3}\overline{x_4} + \overline{x_2}x_3 + \overline{x_2}x_4 = x_2\overline{x_3}\overline{x_4} + \overline{x_2}(x_3 + x_4)$$

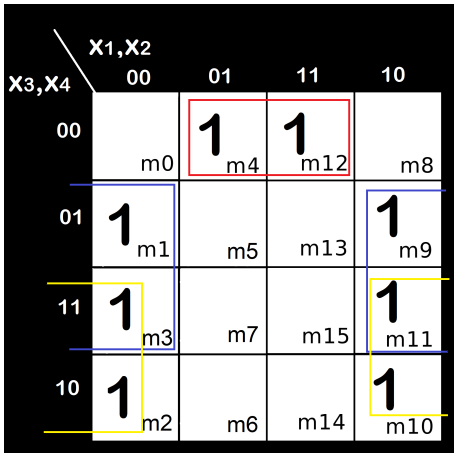

Figure 8: Mapa de Karnaugh $f_2(x_1, x_2, x_3, x_4)$

Para f_3,

$$f_3(x_1, x_2, x_3, x_4) = \overline{x_3}x_4 + x_3\overline{x_4}$$

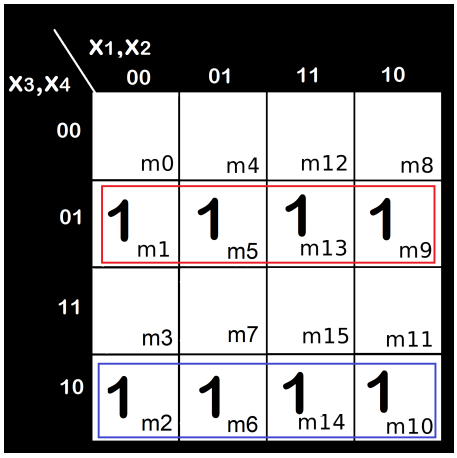

Figure 9: Mapa de Karnaugh $f_3(x_1, x_2, x_3, x_4)$

Para f_4,

$$f_4(x_1, x_2, x_3, x_4) = x_4$$

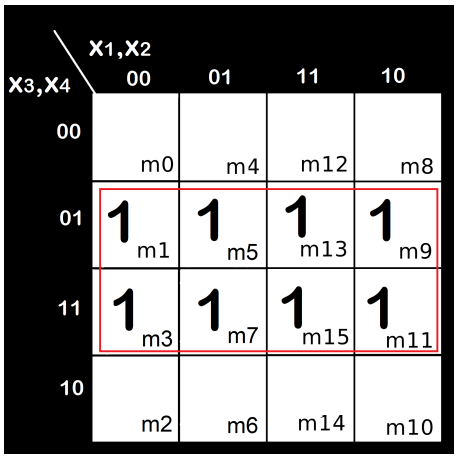

Figure 10: Mapa de Karnaugh $f_4(x_1, x_2, x_3, x_4)$

El circuito lógico resultante se muestra en la Figura 5, donde los Xi corresponden a la entrada y los Fi a la salida del circuito. Respetando la consigna, solo se utilizaron compuestas NOT, AND y OR.

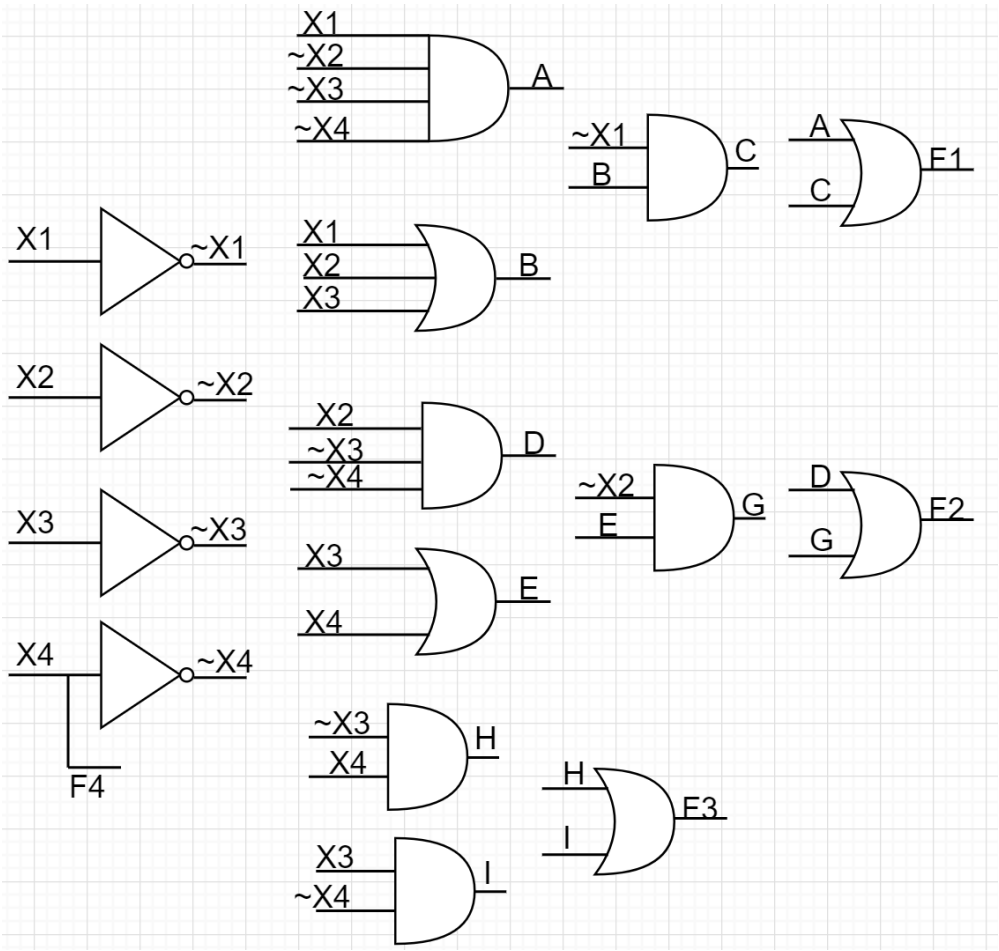

Figure 11: Circuito resultante realizado en draw.io

3 Ejercicio 5

Para la implementación del sumador de 2 numeros en BCD de un digito se observaron las diferencias entre la representación binaria común y la BCD para numeros del 0 al 19.

Decimal	Binario	BCD
0	0000 0000	0000 0000
1	0000 0001	0000 0001
2	0000 0010	0000 0010
3	0000 0011	0000 0011
4	0000 0100	0000 0100
5	0000 0101	0000 0101
6	0000 0110	0000 0110
7	0000 0111	0000 0111
8	0000 1000	0000 1000
9	0000 1001	0000 1001
10	0000 1010	0001 0000
11	0000 1011	0001 0001
12	0000 1100	0001 0010
13	0000 1101	0000 0011
14	0000 1110	0001 0100
15	0000 1111	0001 0101
16	0001 0000	0001 0110
17	0001 0001	0001 0111
18	0001 0010	0001 1000
19	0001 0011	00001 1001

Figura 12: Comparativo Binario BCD

Se observó un exeso de 6 para todos los numeros mayores a 9, es decir, uno podria hacer la suma binaria y el resultado de 0 a 9 daría el resultado esperado en BCD pero para resultados de 10 a 19 es necesario un factor de correxión de +6. Para decidir si es mayor o menor a 9 se puede hacer un diagrama logico en el cual si hay carry en la suma de los primeros cuatro bits, o esta encendido el MSB con el bit anterior, o el MSB con el segundo se puede afirmar que es mayor a 9. Se esquematizó el siguiente bloque comparador:

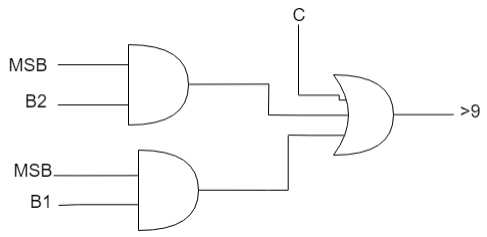

Figura 13: Comparador mayor a 9

Se llevo a cabo el diseño de un modulo “Simple Adder” el cual debería poder hacer la suma con carry de 2 bits y devolver el resultado y el carry de salida, para que luego bajo una conexión entre 4 de estos podamos obtener la suma en binario de dos numeros de 4 bits y un carry out de la suma total.

a	b	Ci	Result	Co
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Figura 14: Tabla Logica del Simple Adder

Ci\ab	**00**	**01**	**11**	**10**
0	0	1	0	1
1	1	0	1	0

Figura 15: Mapa de Karnau para Result

$Result = m_1 + m_2 + m_4 + m_7 = \bar{a}\bar{b}c + \bar{a}b\bar{c} + a\bar{b}\bar{c} + abc = \bar{a}(\bar{b}c + b\bar{c}) + a(\bar{b}\bar{c} + bc) = \bar{a}(\bar{b}c + b\bar{c}) + a(\bar{bc + b\bar{c}})$, recordando que $XOR = A\bar{B} + B\bar{A}$.

Calculo aux: $a(\bar{b}\bar{c} + bc) = a(\bar{b}\bar{\bar{c}} + bc) = a + (\bar{b}\bar{\bar{c}} + bc) = a + (\bar{\bar{b}\bar{c}}\bar{bc}) = a + ((b + \bar{c})(\bar{b} + \bar{c})) = a(b\bar{c} + c\bar{b})$.

$Co = Cib + Cia + ab = ab + Ci(a + b)$.

Ci\ab	**00**	**01**	**11**	**10**
0	0	0	1	0
1	0	1	1	1

Figura 16: Mapa de Karnau para Co

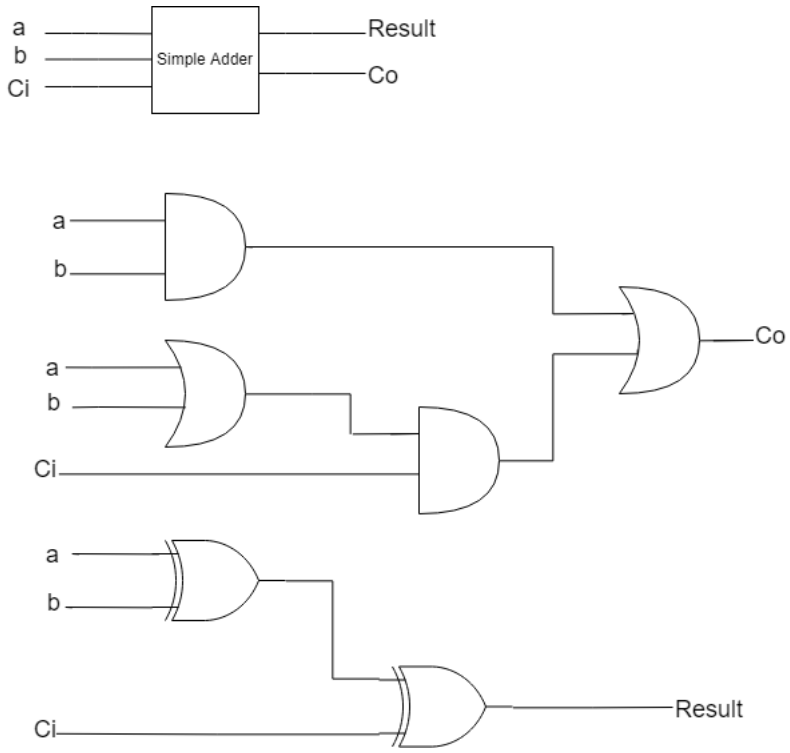

Figura 17: Modulo Simple Adder

Habiendo diseñado el módulo de Simple Adder se encadenaron 4 de estos para formar efectivamente el modulo Bit4Adder donde obtenemos un resultado final, un carry out y un overflow.

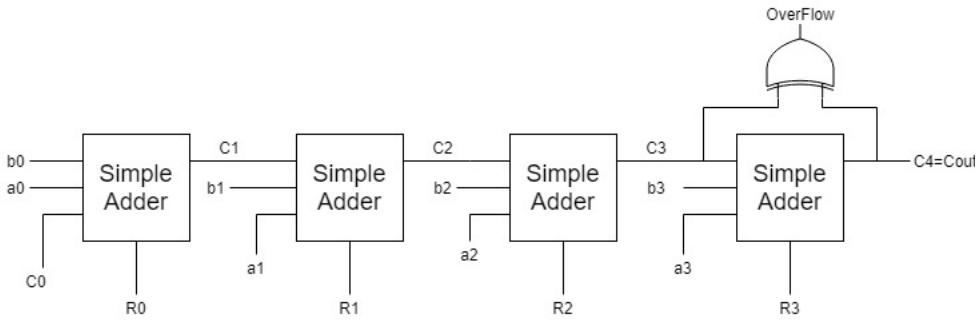

Figura 18: Bit4Adder

Con los módulos armados anteriormente podemos hacer nuestra suma BCD donde nuestro comparador nos termina sumando 0110 a nuestra respuesta de la suma original si excede 1001. De esta manera obtenemos un numero de 8 bits representando hasta 19 en BCD.

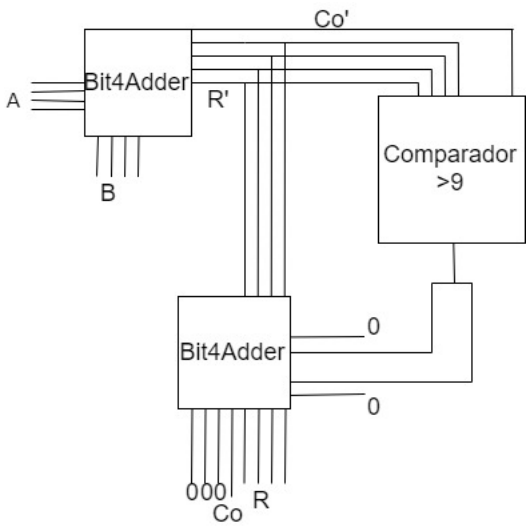

Figura 19: Bit4Adder

4 ALU

Para el diseño de la ALU se reutilizaron los módulos vistos anteriormente para hacer el complemento a 2 y la suma de 4 bits.

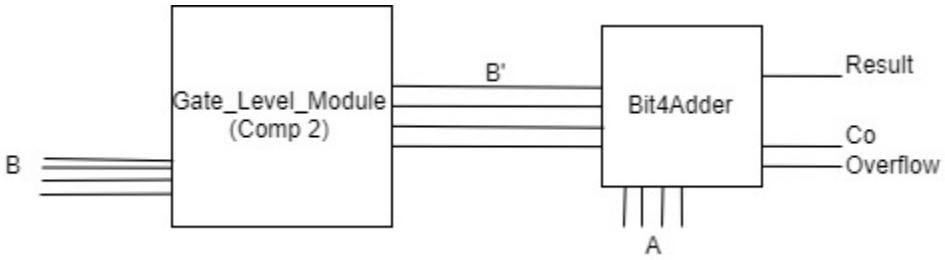

Figura 20: Modulo Res4Bit

El diseño de la ALU esta basado en un operador de 3bits el cual define que operación y que CCR se muestra a la salida como se vera en el diagrama lógico a continuación. Se definieron las siguientes operaciones:

operador=3'b001⟶Suma
operador=3'b010⟶Resta
operador=3'b011⟶ShiftLeft
operador=3'b100⟶Complemento a 2
operador=3'b101⟶Negado
operador=3'b110⟶And
operador=3'b101⟶Or
operador=3'b111⟶Xor

Se definieron las operaciones ShiftLeft, Complemento a 2 y negación para el primer número ingresado a la ALU.

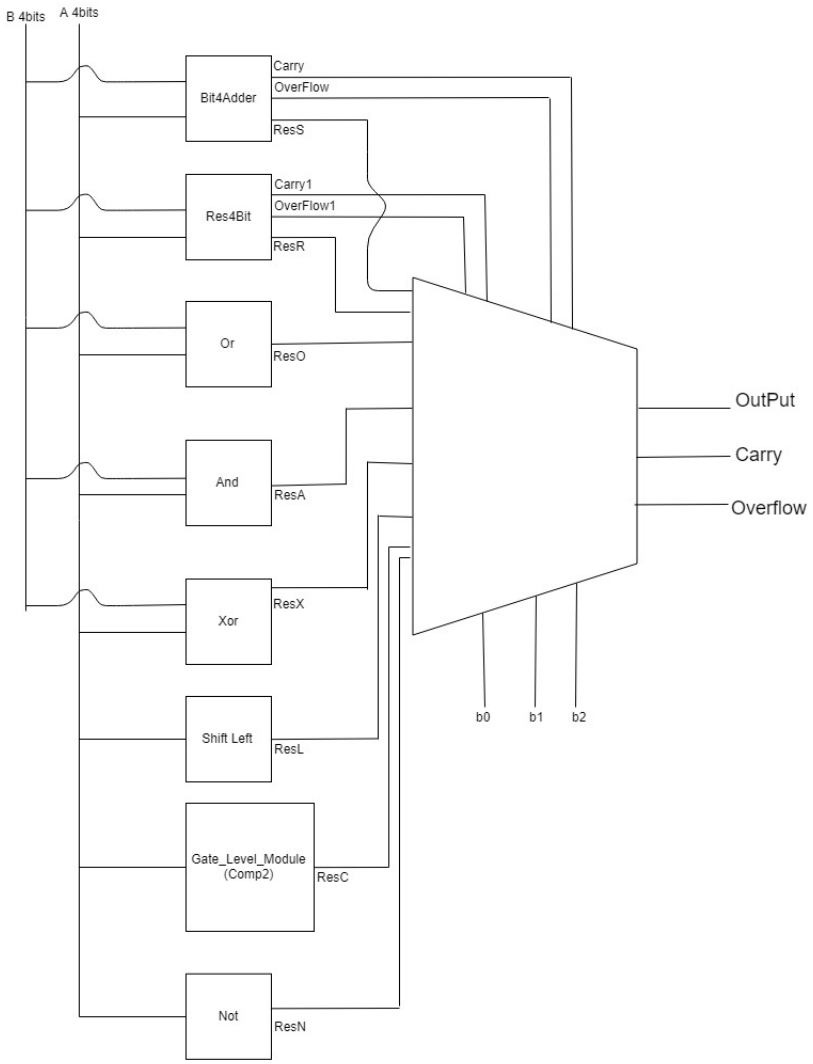

Figura 21: ALU

b0	b1	b2	Output	Carry	Overflow
0	0	0	ResS	Carry	Overflow
0	0	1	ResR	Carry1	Overflow1
0	1	0	ResL	0	0
0	1	1	ResC	0	0
1	0	0	ResN	0	0
1	0	1	ResA	0	0
1	1	0	ResO	0	0
1	1	1	ResX	0	0

Figura 22: Tabla Lógica de la ALU