Instituto Tecnológico de Buenos Aires

Electrónica III

Trabajo Práctico N°1

Grupo 2

Integrantes:
Santiago Arribere, 59169
Gonzalo Davidov, 59117
R. Nicolás Trozzo, 59434
Matías Francois, 59828
Pablo Scheinfeld, 59065

Profesores:
Kevin Dewald
Pablo Enrique Wundes
Miguel Pablo Aguirre

4 de septiembre de 2019

Abstract
El presente informe tiene como objetivo trabajar sobre diferentes conceptos relacionados con la electrónica digital, tales como la lógica combinacional y el álgebra booleana, para así poder aplicar lo aprendido en los ejercicios planteados por la cátedra. A partir de estos conceptos se analizan los diversos casos de circuitos lógicos para su posterior implementación en el lenguaje Verilog. De esta manera la persona que lea este informe podrá familiarizarse con la resolución de circuitos

Contenido

1. Ejercicio 1

1.1. Análisis

Este ejercicio consiste en implementar un programa el cual dada cierta convención de punto fijo devuelva el rango y la resolución del mismo.
La convención se recibe al momento de ejecutar el programa en el siguiente orden: Signado, Bits parte entera y Bits parte fraccionaria, a lo largo del presente informe se hará referencia a los mismos como: 's', 'a' y 'b' respectivamente.
Inicialmente para el cálculo de la resolución se puede observar que la misma viene dada por el bit menos significativo, por lo tanto responde a la expresión 1, no importando si se está trabajando con un número signado o no signado.

$$Res = 2^{-b} \tag{1}$$

Luego para hallar el rango se calcula la diferencia entre el mayor número que es posible representar y el menor, de esta forma se obtienen las expresiones 2 y 3, para números signados y no signados, respectivamente. Con esto se puede evidenciar que el rango tampoco depende de la convención de signo que se proponga utilizar, por lo tanto 's' solo debe ser validado para el correcto funcionamiento del programa.

$$Ran = (Max - Min) = \sum_{i=-b}^{a-1} 2^i - 0 = \sum_{i=-b}^{a-1} 2^i \tag{2}$$

$$Ran = (Max - Min) = \sum_{i=-b}^{a-2} 2^i - (-2^{a-1}) = \sum_{i=-b}^{a-1} 2^i \tag{3}$$

1.2. Elección del lenguaje

Inicialmente, al ser un programa aparentemente sencillo, se había optado por utilizar el lenguaje 'C', de esta forma se procedió a realizar el algoritmo necesario para la validación, el mismo comprueba que 's' sea '1' o '0', e inicialmente se comprobó solamente que 'a' y 'b' fuesen números enteros, y que ambos sean distintos de '0' simultáneamente. Luego de la validación de datos, se desarrolló el algoritmo del calculo del rango y la resolución, pero al momento de tener que imprimir la respuesta en pantalla no se pudo obtener el formato deseado debido a la forma de representar números con decimales, debido a que el lenguaje no ajusta dinámicamente la forma de representar este tipo de números, sino que en lugar de eso muestra una cantidad especificada de cifras significativas del valor.
Por lo tanto para poder solucionar esta limitación de decidió cambiar al lenguaje 'C++', con el cual la forma de representar en pantalla los números era la requerida para responder con lo solicitado.
Otra limitación que se encontró durante la resolución de este apartado fue el límite numérico que era posible procesar, ya que, al utilizar el tipo de dato *float*, la resolución propia del mismo es de 2^{-126}, por lo tanto 'b'=126 es el límite superior que el programa puede recibir como valor para la parte fraccionaria. Y como limitación para la parte entera se tiene 'a'=24, ya que para valores mayores del mismo, la precisión del número representado deja de ser exacta y comienza a tomar aproximaciones a números múltiplos de potencias de 2.

2. Ejercicio 2

El ejercicio 2 se basa resolver dos problemas, el primero, un ejercicio de 5 variables, dado en función de mintérminos y el segundo, de 4 variables, en función de maxtérminos.

2.1. Primera parte

$$f(a,b,c,d,e) = \sum m(0,2,4,7,8,10,12,16,18,20,23,24,25,26,27,28)$$

2.1.1. Resolución mediante algebra booleana

La expresión obtenida por los minterminos dados es la siguiente:

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

$$a.\overline{b}.\overline{c}.d.\overline{e} + a.\overline{b}.c.\overline{d}.\overline{e} + a.\overline{b}.c.d.e + a.b.\overline{c}.\overline{d}.\overline{e} + a.b.\overline{c}.\overline{d}.e + a.b.\overline{c}.d.\overline{e} + a.b.\overline{c}.d.e + a.b.c.\overline{d}.\overline{e}$$

Se puede ampliar la expresión, con la finalidad de luego reducirla significativamente a partir de las simplificaciónes del álgebra booleana :

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

$$a.\overline{b}.\overline{c}.d.\overline{e} + a.\overline{b}.c.\overline{d}.\overline{e} + a.\overline{b}.c.d.e + a.b.\overline{c}.\overline{d}.\overline{e} + a.b.\overline{c}.\overline{d}.e + a.b.\overline{c}.d.\overline{e} + a.b.\overline{c}.d.e + a.b.c.\overline{d}.\overline{e} + a.\overline{b}.\overline{c}.\overline{d}.\overline{e} + a.b.\overline{c}.\overline{d}.\overline{e} +$$

$$\overline{a}.\overline{b}.\overline{c}.\overline{d}.\overline{e} + a.b.\overline{c}.d.\overline{e} + \overline{a}.b.\overline{c}.\overline{d}.\overline{e} + a.b.\overline{c}.\overline{d}.\overline{e}$$

De aquí utilizando la propiedad de absorción

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

$$a.\overline{b}.\overline{c}.d.\overline{e} + a.\overline{b}.c.\overline{d}.\overline{e} + a.\overline{b}.c.d.e + a.b.\overline{c}.\overline{d}.\overline{e} + a.b.\overline{c}.\overline{d}.e + a.b.\overline{c}.d.\overline{e} + a.b.\overline{c}.d.e + a.b.c.\overline{d}.\overline{e} + a.\overline{b}.\overline{c}.\overline{d}.\overline{e} + a.b.\overline{c}.\overline{d}.\overline{e} +$$

$$\overline{a}.\overline{b}.\overline{c}.\overline{d}.\overline{e} + a.b.\overline{c}.d.\overline{e} + \overline{a}.b.\overline{c}.\overline{d}.\overline{e} + a.b.\overline{c}.\overline{d}.\overline{e}$$

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

$$a.\overline{c}.\overline{d}.\overline{e}$$

Así llegamos a:

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

$$a.\overline{c}.\overline{c}.\overline{e}$$

Luego de esto, nuevamente aplicaremos absorcion de la siguiente manera:

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

$$a.\overline{c}.\overline{d}.\overline{e}$$

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

Entonces hallamos la siguiente expresión:

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

Para luego realizar una ultima absorción:

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

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

De esta manera se obtiene la expresión deseada:

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

2.1.2. Resolución mediante mapas de Karnaugh

a = 0

de \ bc	00	01	11	10
00	1	1	1	1
01	0	0	0	0
11	0	1	0	0
10	1	0	0	1

a = 1

de \ bc	00	01	11	10
00	1	1	1	1
01	0	0	0	1
11	0	1	0	1
10	1	0	0	1

De aquí, mediante la separación por grupos marcada previamente en los mapas de Karnaugh, se observa de qué variables depende cada grupo y si estas están negadas o no, llegando a la siguiente expresión:

$$f(A,B,C,D,E) = \overline{C}.\overline{E} + \overline{D}.\overline{E} + \overline{B}.C.D.E + A.B.\overline{C}$$

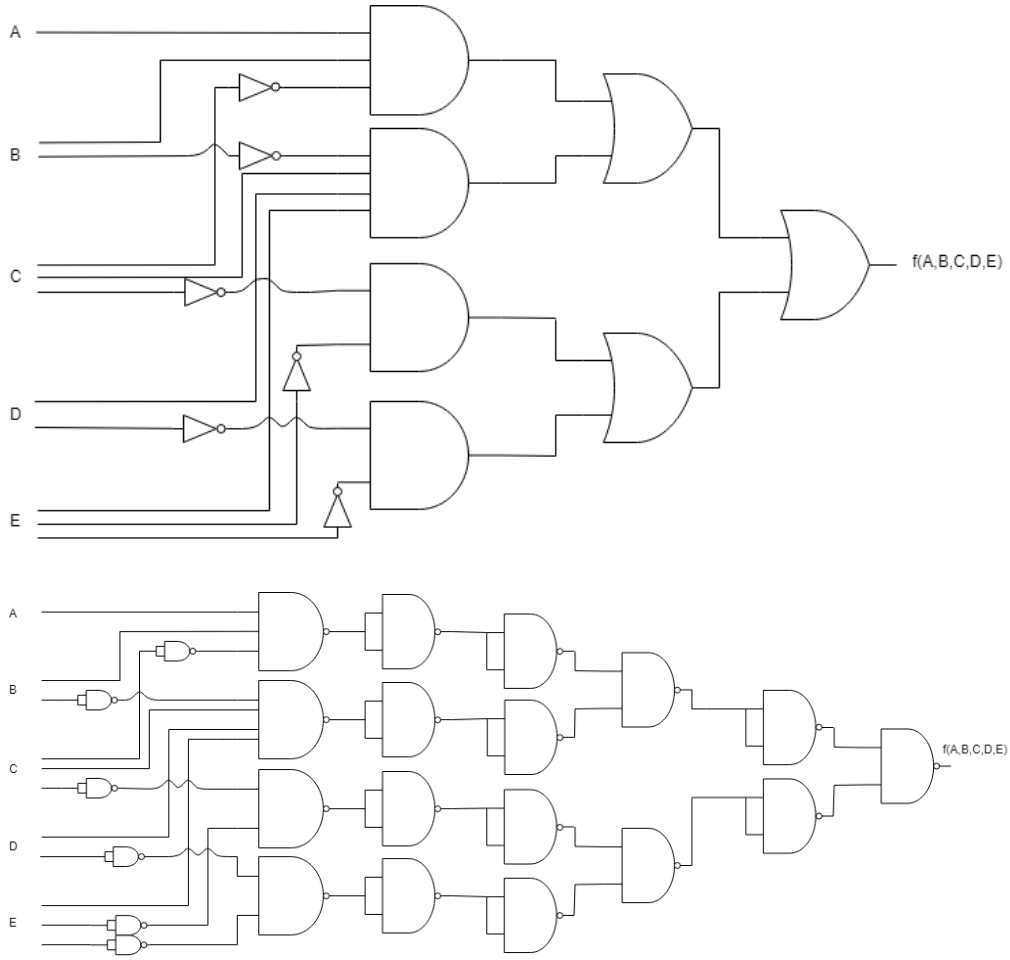
Como era de esperarse, el resultado obtenido mediante mapas de Karnaugh es igual a la simplificación obtenida mediante álgebra booleana.

2.1.3. Representación con compuertas AND, OR y NOT

La representación de la expresión obtenida previamente mediante compuertas lógicas AND, OR y NOT se puede ver a continuación.

2.1.4. Representación con compuertas NAND

La representación de la expresión encontrada previamente mediante compuertas lógicas NAND se puede ver a continuación.



2.2. Segunda parte

$$f(d,c,b,a) = \prod (M0, M2, M4, M7, M8, M10, M12)$$

2.2.1. Resolución mediante algebra booleana

La expresión obtenida por los maxtérminos dados es la siguiente:

$$f(d,c,b,a) = (d+c+b+a)\cdot(d+c+\overline{b}+a)\cdot(d+\overline{c}+b+a)\cdot(d+\overline{c}+\overline{b}+\overline{a})\cdot(\overline{d}+c+b+a)\cdot(\overline{d}+c+\overline{b}+a)\cdot(\overline{d}+\overline{c}+b+a)$$

De aquí utilizando la propiedad de absorción podremos reducir la expresión a:

$$f(d,c,b,a) = (d+c+b+a)\cdot(d+c+\overline{b}+a)\cdot(d+\overline{c}+b+a)\cdot(d+\overline{c}+\overline{b}+\overline{a})\cdot(\overline{d}+c+b+a)\cdot(\overline{d}+c+\overline{b}+a)\cdot(\overline{d}+\overline{c}+b+a)$$

$$f(d,c,b,a) = (d+c+a)\cdot(\overline{c}+b+a)\cdot(d+\overline{c}+\overline{b}+\overline{a})\cdot(\overline{d}+c+a)$$

De esta forma, aprovechando el hecho de que x.x=x lo que haremos será traer los términos: $(d+c+b+a)y(\overline{d}+c+b+a)$ de la primer linea de donde partimos como productos, de esta forma obtendremos la siguiente expresión:

$$f(d,c,b,a) = (d+c+a)\cdot(\overline{c}+b+a)\cdot(d+\overline{c}+\overline{b}+\overline{a})\cdot(\overline{d}+c+a)\cdot(d+c+b+a)\cdot(\overline{d}+c+b+a)$$

Ahora utilizando nuevamiente la propiedad de absorción tendremos:

$$f(d,c,b,a) = (d+c+a)\cdot(\overline{c}+b+a)\cdot(d+\overline{c}+\overline{b}+\overline{a})\cdot(\overline{d}+c+a)\cdot(d+c+b+a)\cdot(\overline{d}+c+b+a)$$

$$f(d,c,b,a) = (c+a)\cdot(\overline{c}+b+a)\cdot(d+\overline{c}+\overline{b}+\overline{a})\cdot(c+b+a)$$

Nuevamente realizamos una última absorción:

$$f(d,c,b,a) = (c+a)\cdot(\overline{c}+b+a)\cdot(d+\overline{c}+\overline{b}+\overline{a})\cdot(c+b+a)$$

$$f(d,c,b,a) = (c+a)\cdot(b+a)\cdot(d+\overline{c}+\overline{b}+\overline{a})$$

Finalmente, la expresión hallada es:

$$f(d,c,b,a) = (c+a)\cdot(b+a)\cdot(d+\overline{c}+\overline{b}+\overline{a})$$

2.2.2. Resolución mediante mapas de Karnaugh

ba \ dc	00	01	11	10
00	0	0	0	0
01	1	1	1	1
11	1	0	1	1
10	0	1	1	0

De aquí, mediante la separación por grupos marcada previamente en los mapas de Karnaugh, se observa de cuáles variables depende cada grupo y si estas están negadas o no, llegando a la siguiente expresión:

$$f(d,c,b,a) = (c+a)\cdot(b+a)\cdot(d+\overline{c}+\overline{b}+\overline{a})$$

Como era de esperarse, el resultado obtenido mediante mapas de Karnaugh es igual a la simplificación obtenida mediante álgebra booleana.

2.2.3. Representación con compuertas AND, OR y NOT

La representación de la expresión obtenida previamente mediante compuertas lógicas AND, OR y NOT se puede ver a continuación, en la figura 1

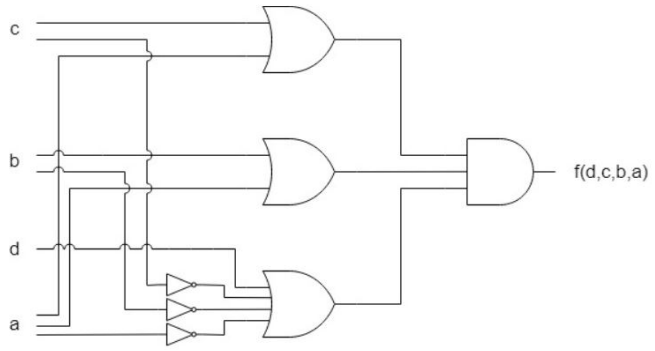


Figura 1: Circuito con compuertas AND, OR y NOT

2.2.4. Representación con compuertas NAND

La representación de la expresión obtenida previamente mediante compuertas lógicas NAND se puede ver a continuación en la figura 2

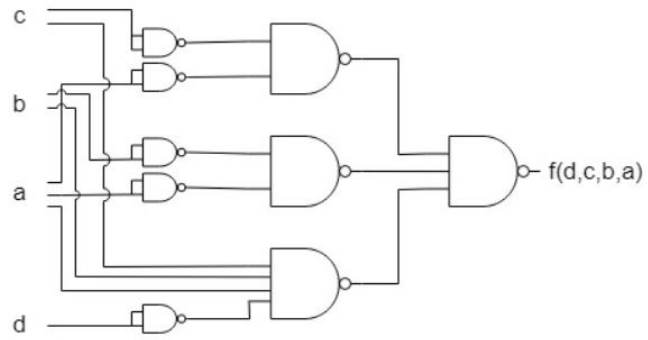


Figura 2: Circuito con compuertas NAND

3. Ejercicio 3

El ejercicio consta en crear mediante el uso del lenguaje Verilog un encoder de cuatro entradas y dos salidas y un demux de cuatro salidas con dos *select lines*.
Se tiene como consideración que ante cualquier entrada no esperada la salida sea alta impedancia, es decir, Z.
A continuación se muestran las tablas de verdad:

S1	S2	In	Out0	Out1	Out2	Out3
0	0	0	0	0	0	0
0	0	1	0	0	0	1
0	1	0	0	0	0	0
0	1	1	0	0	1	0
1	0	0	0	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	0	0
1	1	1	1	0	0	0

Tabla 1: Tabla de verdad de el demux de 4 salidas

In1	In2	In3	In4	Out0	Out1
1	0	0	0	0	0
0	1	0	0	0	1
0	0	1	0	1	0
0	0	0	1	1	1

Tabla 2: Tabla de verdad de el encoder de 4 entradas

4. Ejercicio 4

4.1. Expresión en función de los mintérminos

Con el fin de convertir un número binario de 4 bits en su equivalente en código de Gray, se plantearon todas las posibilidades en la siguiente tabla de verdad, donde Y_0, Y_1, Y_2 e Y_3 representan las correspondientes salidas y A, B, C y D las entradas del sistema. Ambas se encuentran mostradas en forma ordenada, tomando Y_0 como el bit más significativo y Y_3 como el bit menos significativo. De la misma forma, para las entradas, A representa el bit más significativo y D el menos significativo.

A	B	C	D	Y_0	Y_1	Y_2	Y_3
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	0	1	1	1
0	1	1	0	0	1	0	1
0	1	1	1	0	1	0	0
1	0	0	0	1	1	0	0
1	0	0	1	1	1	0	1
1	0	1	0	1	1	1	1
1	0	1	1	1	1	1	0
1	1	0	0	1	0	1	0
1	1	0	1	1	0	1	1
1	1	1	0	1	0	0	1
1	1	1	1	1	0	0	0

Tabla 3: Tabla de verdad

A partir del análisis de la tabla 3 fue posible expresar el valor de cada bit de salida en función de los mintérminos de los bits de entrada. Estas expresiones se encuentran formuladas a continuación:

$$Y_0(A,B,C,D) = A.\overline{B}.\overline{C}.\overline{D} + A.\overline{B}.\overline{C}.D + A.\overline{B}.C.\overline{D} + A.\overline{B}.C.D + A.B.\overline{C}.\overline{D} + A.B.\overline{C}.D + A.B.C.\overline{D} + A.B.C.D$$

$$Y_1(A,B,C,D) = \overline{A}.B.\overline{C}.\overline{D} + \overline{A}.B.\overline{C}.D + \overline{A}.B.C.\overline{D} + \overline{A}.B.C.D + A.\overline{B}.\overline{C}.\overline{D} + A.\overline{B}.\overline{C}.D + A.\overline{B}.C.\overline{D} + A.\overline{B}.C.D$$

$$Y_2(A,B,C,D) = \overline{A}.\overline{B}.C.\overline{D} + \overline{A}.\overline{B}.C.D + \overline{A}.B.\overline{C}.\overline{D} + \overline{A}.B.\overline{C}.D + A.\overline{B}.C.\overline{D} + A.\overline{B}.C.D + A.B.\overline{C}.\overline{D} + A.B.\overline{C}.D$$

$$Y_3(A,B,C,D) = \overline{A}.\overline{B}.\overline{C}.D + \overline{A}.\overline{B}.C.\overline{D} + \overline{A}.B.\overline{C}.D + \overline{A}.B.C.\overline{D} + A.\overline{B}.\overline{C}.D + A.\overline{B}.C.\overline{D} + A.B.\overline{C}.D + A.B.C.\overline{D}$$

4.2. Desarrollo de los mapas de Karnaugh de las salidas

Para los siguientes mapas se utilizó la misma convención que para la tabla de verdad. De esta forma el orden es A-B-C-D, siendo A el bit más significativo y D el bit menos significativo.

Mapa de Karnaugh para Y_0:

CD \ AB	00	01	11	10
00	0	0	1	1
01	0	0	1	1
11	0	0	1	1
10	0	0	1	1

Mediante la observación del mapa y teniendo en cuenta que para el conjunto marcado en rojo tanto B, como C y como D cambian de valor (entre 0 y 1) se pudo concluir que la expresión más simplificada para Y_0 es:

$$Y_0 = A \tag{4}$$

Mapa de Karnaugh para Y_1:

CD \ AB	00	01	11	10
00	0	1	0	1
01	0	1	0	1
11	0	1	0	1
10	0	1	0	1

En este caso, al realizar un análisis del mapa se pudieron apreciar dos grupos (marcados en magenta y amarillo). Dentro del primer conjunto (magenta) sucede un cambio en los valores de C y D, manteniéndose A en 0 y B en 1. Con respecto al segundo grupo (amarillo) C y D nuevamente toman distintos valores, mientras que A se mantiene en 1 y B en 0. Como conclusión, la expresión más simplificada para Y_1 es:

$$Y_1 = \overline{A}.B + A.\overline{B} \tag{5}$$

Mapa de Karnaugh para Y_2:

CD \ AB	00	01	11	10
00	0	1	1	0
01	0	1	1	0
11	1	0	0	1
10	1	0	0	1

Con el fin de obtener la expresión más simplificada para Y_2 se distinguieron 2 grupos en el mapa. Para el caso del grupo azul, B se mantiene con el valor 1 y C con el valor 0 mientras que A y D cambian. Por otra parte, examinando el conjunto gris, se cumple que B vale 0 y C vale 1, siendo las únicas variables que no cambian su valor. De esta forma, la expresión queda:

$$Y_2 = B.\overline{C} + \overline{B}.C \tag{6}$$

Mapa de Karnaugh para Y_3:

CD \ AB	00	01	11	10
00	0	0	0	0
01	1	1	1	1
11	0	0	0	0
10	1	1	1	1

Por último, tienendo en cuenta que: dentro del grupo marrón C vale 0 y D, 1; en el grupo verde, C es constantemente 1 y D, 0 -siendo las únicas variables que mantienen su valor en cada caso-, para Y_3 la expresión más simplificada es:

$$Y_3 = \overline{C}.D + C.\overline{D} \tag{7}$$

Por otra parte, analizando todas las expresiones se puede apreciar que, excluyendo la entrada A y la salida Y_0, el resto de las salidas son el resultado de ingresar 2 entradas a una compuerta XOR. Este hecho se puede pensar como dicha operación (XOR) es la que se realiza entre cada bit binario de la entrada y su bit inmediatamente anterior (más significativo) para formar cada bit del número expresado por código de Gray - exceptuando el primer bit ya que toman el mismo valor independientemente del resto-.

4.3. Representación por medio de un circuito lógico

A partir de las expresiones 4, 5, 6, y 7, se pudo plantear el comportamiento a través de un circuito lógico. Para ello, se tomaron compuertas del tipo AND, OR y NOT. A continuación se expone dicho esquema.

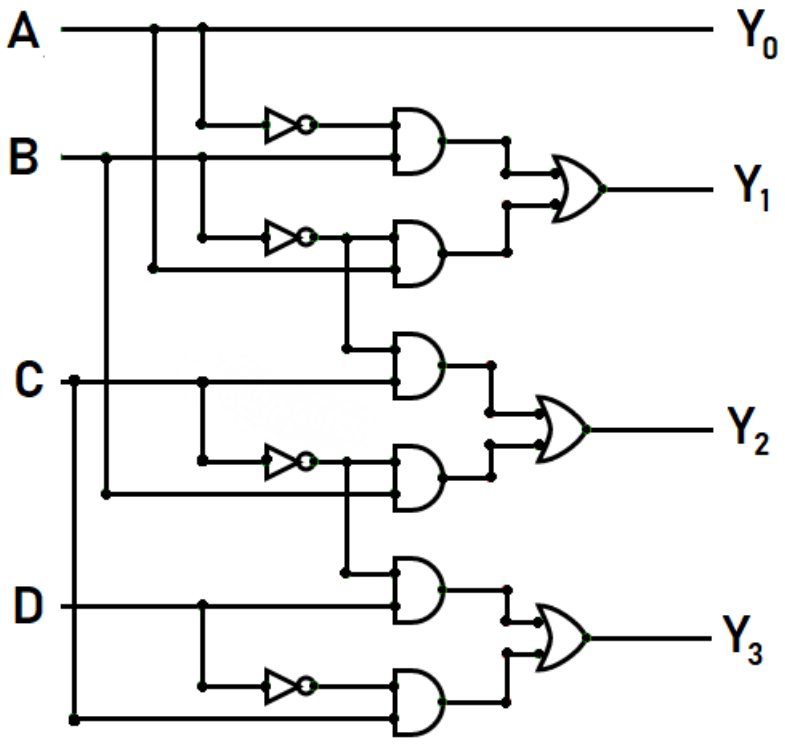


Figura 3: Representación de las expresiones mediante un circuito lógico

El circuito esquematizado en la figura 3 es el resultado de expresiones dispuestas en la forma de suma de productos. Esto se puede observar en el diagrama debido a que, excepto en Y_0 su valor es directamente equivalente al de A, las salidas parten inmediatamente después de una compuerta OR cuyas entradas siempre son las salidas de dos compuertas AND.

4.4. Implementación en Verilog

En la presente sección se explicita como fue el razonamiento lógico llevado a cabo para la implementación del circuito en Verilog. Para realizar dicha tarea, se decidió desarrollar un archivo con el módulo principal del circuito y otro archivo con un banco de pruebas.
Con respecto al archivo encargado del funcionamiento, lo primero que se realizó fue observar las expresiones de las salidas del circuito para comprender de que manera relacionarlas con la entrada. Al ser 4 entradas y 4 salidas donde cada una representa un nibble, en Verilog se utilizaron arreglos de 4 bits para trabajar sobre ellas. Se define un arreglo como input y luego el otro como output. Por esta razón, se debe tener en cuenta que de las salidas de la convención utilizada anteriormente, Y_0 será el bit 3 del arreglo de salida y Y_3 será el bit 0.
A continuación, y teniendo en cuenta que Y_0 toma el valor de A y que ambos son el bit más significativo de la salida y entrada respectivamente, se igual el bit 3 del arreglo de salida al bit 3 del arreglo de entrada.
Como ya fue mencionado, el resto de las salidas pueden ser entendidas como el resultado de una compuerta XOR entre el bit de entrada en la misma posición que el de salida y el de entrada en una posición anterior (más significativo). Este razonamiento se lleva a cabo con las 3 expresiones restantes, siendo estas muy simples al hacer uso de la operación $'\wedge'$ (XOR) de Verilog. Así, por ejemplo, el bit 1 de salida (Y_2) depende de un XOR entre el bit 1 y 2 de entrada (B y C).
En relación al banco de pruebas, se utilizó otro archivo .v donde a una entrada de 4 bits se analiza la salida resultante. Este proceso se realiza en el banco para todos los numeros binarios de 4 bits a la entrada y observando si la salida es su correspondiente código de Gray. El resultado fue exitoso en todos los casos.

5. Ejercicio 5

El último ejercicio requiere la implementación en Verilog de un módulo multiplicador de dos números de un dígito en formato BCD, expresando a la salida el resultado como un número de dos dígitos en el mismo formato.
El módulo tiene 3 partes principales: la validación de la entrada, la multiplicación en sí, y la conversión de binario a BCD del resultado de la multiplicación. A continuación se explican consideraciones generales del código y las etapas de la implementación.

5.1. Consideraciones generales

Se detallan las consideraciones que se tuvieron en cuenta a la hora de programar para lograr una mayor calidad del código. Hay un módulo principal llamado 'BCDMultiplier', que recibe los números BCD a multiplicar en dos arreglos de 4 bits, devuelve el resultado de la multiplicación en formato BCD en un arreglo de 8 bits, y además devuelve 2 bits más de validación que son explicados más adelante. Todos los arreglos de bits fueron definidos siguiendo la conveción Big Endian ([0:n]), en vez de Little Endian ([n:0]), para tener el bit más significativo en la primera posición, ya que se consideró más intuitivo de esa forma.

5.2. Validación de la entrada

Antes de proceder a realizar la multiplicación, es necesario validar que cada nybble recibido corresponda a un número BCD válido, es decir, que corresponda a un número menor a 10. Para poder determinar la validez de un número recibido, en la Tabla 4 se muestra la tabla de verdad de la validación, considerando al número como $X = x_1x_2x_3x_4$ donde se puede observar que la salida toma el valor 1, que se toma como error, si y sólo si el valor de la entrada supera el valor de 9, que es el límite para la representación en un dígito BCD.

N	x_1	x_2	x_3	x_4	y = $x_1x_2x_3x_4 > b1001$
0	0	0	0	0	0
1	0	0	0	1	0
2	0	0	1	0	0
3	0	0	1	1	0
4	0	1	0	0	0
5	0	1	0	1	0
6	0	1	1	0	0
7	0	1	1	1	0
8	1	0	0	0	0
9	1	0	0	1	0
10	1	0	1	0	1
11	1	0	1	1	1
12	1	1	0	0	1
13	1	1	0	1	1
14	1	1	1	0	1
15	1	1	1	1	1

Tabla 4: Tabla de verdad de la validación de la entrada

De la tabla se obtiene el siguiente mapa de Karnaugh:

x_1x_2 \ x_3x_4	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	1	1	1	1
10	0	0	1	1

Resolviendo para los mintérminos, se llega a la siguiente suma de productos:

$$y = x_1 \cdot x_2 + x_1 \cdot x_3 \tag{8}$$

Por lo tanto, para validar los números de entrada, basta implementar la Ecuación 8.
A la hora de la implementación en Verilog, se decidió realizar un módulo específico que realice la validación, mediante la expresión lógica obtenida anteriormente. Esta validación se realiza para ambos números de entrada, y el módulo principal tiene dos bits de output para indicar la validez de cada parámetro de entrada.

5.3. Multiplicación binaria y conversión a BCD

Una vez validada la entrada, se debe realizar la multiplicación en binario. Para esto se hace uso del operador '*' de Verilog, que arroja el resultado de la multiplicación binaria. Teniendo el resultado en binario de la multiplicación, hay múltiples formas de pasarlo a BCD. En primer lugar se había decidido implementar el algoritmo DoubleDabble, pero la implementación no era acorde a la filosofía de Verilog, ya que había que realizar muchas iteraciones. Dado que realizar una tabla de verdad involucraba más de 50 casos, se utilizó el operador % (módulo) de Verilog, para obtener las decenas y las unidades.

5.4. Banco de pruebas

El banco de pruebas prueba 6 casos que se consideran representativos de la totalidad de entradas que puede tener el módulo. En primer lugar, se prueba si una multiplicación por 0 arroja 0 como resultado. Los siguientes dos casos son multiplicaciones válidas con resultado distinto de 0. Por último, hay 3 casos inválidos, en los que se puede observar que el módulo distingue cuál de sus entradas es inválida.
Para que el código del banco de pruebas no fuera repetitivo, se utilizaron tasks de Verilog para las salidas hacia el usuario.

6. Conclusión

A modo de cierre, a lo largo del informe se pudieron desarrollar las temáticas propuestas y cumplir los objetivos planteados. Fue posible utilizar las herramientas fundamentales de la lógica combinacional, es decir, tablas de verdad y mapas de Karnaugh, para la implementación circuital gráfica o programada. En cuanto al lenguaje Verilog, se tuvo una primera aproximación a los lenguajes de programación funcionales, mediante la implementación de módulos. Adicionalmente, se desarrollaron bancos de pruebas para verificar el correcto funcionamiento de los módulos. Por último, para el trabajo fue utilizada la herramienta GitHub, muy utilizada en el ámbito profesional para llevar un control de versiones de los avances.