

INSTITUTO TECNOLÓGICO DE BUENOS AIRES

22.12 - ELECTRÓNICA III

Trabajo Práctico N°1

Grupo 4

BERTACHINI, Germán	58750
DIEGUEZ, Manuel	56273
GALDEMAN, Agustín	59827
LAGUINGUE, Juan Martín	57430

Profesores:

DEWALD, Kevin

WUNDES, Pablo



PRESENTADO EL 5 DE SEPTIEMBRE DE 2019

Índice

1. Cálculo de Resolución y rango con convención de punto fijo	2
2. Algebra booleana y compuertas lógicas	2
2.1. Primera Expresión: suma de productos	2
2.1.1. Simplificación mediante álgebra booleana	2
2.1.2. Simplificación mediante mapas de Karnaugh	4
2.1.3. Implementación mediante compuertas AND, OR y NOT . .	5
2.1.4. Implementación mediante compuertas NAND	6
2.2. Segunda expresión: producto de sumas	6
2.2.1. Simplificación mediante álgebra booleana	6
2.2.2. Simplificacioón mediante mapas de Karnaugh	7
2.2.3. Implementación mediante compuertas AND, OR y NOT . .	7
2.2.4. Implementación mediante compuertas NAND	8
3. Implementación de módulos en verilog	8
3.1. Demultiplexor de 4 salidas	8
3.2. Codificador de 4 entradas	10
4. Ejercicio 4 - Conversor a codigo de Gray	11
5. Ejercicio 5	13

1. Cálculo de Resolución y rango con convención de punto fijo

El sistema binario es un sistema de numeración en el que los números se representan utilizando únicamente dos cifras. Es fundamental para la lógica computacional, debido a la simpleza y naturalidad con el que puede aplicarse al funcionamiento de una computadora o una máquina digital en general. El código binario permite representar números enteros, racionales e incluso signados. Existen diferentes maneras de representar el signo mediante el sistema binario, pero la más usada es por el complemento a dos del número. Existen además distintas convenciones para representar un número racional en binario. En este ejercicio se usará una de ellas: la convención de números en punto o coma fija. En ella se trabaja con un número fijo de bits y se acuerda dejar una cantidad determinada de bits para trabajar la parte fraccionaria del número.

En este ejercicio se escribió un programa al cual se le ingresa si el número es signado o no, cuantos bits de parte entera y cuantos bits de parte fraccionaria hay. Con esos datos se devuelven dos características del número: su resolución, o la cantidad más pequeña representable con la convención ingresada, y su rango, el número más grande representable menos el más pequeño.

El programa se escribió en Python 3.7 y no se requieren de paquetes externos para su funcionamiento.

2. Algebra booleana y compuertas lógicas

En el álgebra booleana se conoce como término canónico de una función lógica a todo producto o suma en la cual aparecen todas las variables en su forma directa o inversa. Cualquier función lógica puede expresarse de forma canónica utilizando los conceptos de min y maxtérminos. El primero, se refiere a todas las filas para las cuales la función lógica es igual a uno en la tabla de verdad correspondiente, mientras que el segundo se corresponde con todos los valores para los cuales la función es igual a cero.

En el siguiente ejercicio se simplificarán dos funciones lógicas aplicando ambos conceptos. Luego se utilizarán mapas de Karnaugh para llegar a la misma expresión. A continuación se dibujará el circuito lógico resultante utilizando compuertas AND, OR y NOT, y por último, se usarán compuertas NAND exclusivamente.

2.1. Primera Expresión: suma de productos

2.1.1. Simplificación mediante álgebra booleana

La expresión de la cual se parte está dada por la siguiente fórmula:

$$f(e, d, c, b, a) = \sum m(0, 2, 4, 7, 8, 10, 12, 16, 18, 20, 23, 24, 25, 26, 27, 28) \quad (1)$$

La ecuación 1 nos indica cuales entradas en la tabla de verdad son iguales a uno. A partir de ella podemos plantear la siguiente ecuación, la cual se trabajará para llegar a su respectiva expresión canónica.

$$\begin{aligned} & \bar{e} \cdot \bar{d} \cdot \bar{c} \cdot \bar{b} \cdot \bar{a} + \bar{e} \cdot \bar{d} \cdot \bar{c} \cdot b \cdot \bar{a} + \bar{e} \cdot \bar{d} \cdot c \cdot \bar{b} \cdot \bar{a} + \bar{e} \cdot \bar{d} \cdot c \cdot b \cdot a + \bar{e} \cdot d \cdot \bar{c} \cdot \bar{b} \cdot \bar{a} + \bar{e} \cdot d \cdot \bar{c} \cdot b \cdot \bar{a} + \bar{e} \cdot d \cdot c \cdot \bar{b} \cdot \bar{a} + \bar{e} \cdot d \cdot c \cdot b \cdot a + \\ & e \cdot \bar{d} \cdot \bar{c} \cdot \bar{b} \cdot \bar{a} + e \cdot \bar{d} \cdot \bar{c} \cdot b \cdot \bar{a} + e \cdot \bar{d} \cdot c \cdot \bar{b} \cdot \bar{a} + e \cdot \bar{d} \cdot c \cdot b \cdot a + e \cdot d \cdot \bar{c} \cdot \bar{b} \cdot \bar{a} + e \cdot d \cdot \bar{c} \cdot b \cdot \bar{a} + e \cdot d \cdot c \cdot \bar{b} \cdot \bar{a} + e \cdot d \cdot c \cdot b \cdot a \end{aligned} \quad (2)$$

La expresión anterior puede parecer complicada en un principio, pero esta ordenada según cada mintermino correspondiente. Para comenzar el proceso de simplificación, gracias a la ley de idempotencia se puede sumar términos ya presentes en la ecuación anterior. So procederá a duplicar los minterminos 2, 18 y 26 y a ordenar la fórmula para facilitar la simplificación.

$$\bar{e} \cdot \bar{d} \cdot \bar{c} \cdot \bar{b} \cdot \bar{a} + \bar{e} \cdot \bar{d} \cdot \bar{c} \cdot b \cdot \bar{a} + e \cdot \bar{d} \cdot \bar{c} \cdot \bar{b} \cdot \bar{a} + e \cdot \bar{d} \cdot \bar{c} \cdot b \cdot \bar{a} \quad (3)$$

$$\bar{e} \cdot d \cdot \bar{c} \cdot b \cdot \bar{a} + e \cdot d \cdot \bar{c} \cdot b \cdot \bar{a} + \bar{e} \cdot \bar{d} \cdot \bar{c} \cdot b \cdot \bar{a} + e \cdot \bar{d} \cdot \bar{c} \cdot b \cdot \bar{a} \quad (4)$$

$$\bar{e} \cdot \bar{d} \cdot c \cdot \bar{b} \cdot \bar{a} + \bar{e} \cdot d \cdot c \cdot \bar{b} \cdot \bar{a} + e \cdot \bar{d} \cdot c \cdot \bar{b} \cdot \bar{a} + e \cdot d \cdot c \cdot \bar{b} \cdot \bar{a} \quad (5)$$

$$e \cdot d \cdot \bar{c} \cdot \bar{b} \cdot \bar{a} + e \cdot d \cdot \bar{c} \cdot b \cdot \bar{a} + e \cdot d \cdot c \cdot \bar{b} \cdot \bar{a} + e \cdot d \cdot c \cdot b \cdot \bar{a} \quad (6)$$

$$e \cdot \bar{d} \cdot c \cdot b \cdot a + \bar{e} \cdot \bar{d} \cdot c \cdot b \cdot a \quad (7)$$

Las sumas de productos anteriores pueden trabajarse si se factorizan por los términos apropiados:

$$\bar{d} \cdot \bar{c} \cdot \bar{a} \cdot (\bar{e} \cdot \bar{b} + \bar{e} \cdot b + e \cdot \bar{b} + e \cdot b) \quad (8)$$

$$\bar{c} \cdot b \cdot \bar{a} \cdot (\bar{e} \cdot b + e \cdot d + \bar{e} \cdot \bar{d} + e \cdot \bar{d}) \quad (9)$$

$$c \cdot \bar{b} \cdot \bar{a} \cdot (\bar{e} \cdot \bar{d} + \bar{e} \cdot d + e \cdot \bar{d} + e \cdot d) \quad (10)$$

$$e \cdot d \cdot c \cdot (\bar{b} \cdot \bar{a} + \bar{b} \cdot a + b \cdot \bar{a} + b \cdot a) \quad (11)$$

$$\bar{d} \cdot c \cdot b \cdot a \cdot (\bar{e} + e) \quad (12)$$

Luego aplicando la propiedad de combinación en todos los productos:

$$\bar{d} \cdot \bar{c} \cdot \bar{a} \cdot (\bar{e} \cdot \bar{b} + \bar{e} \cdot b + e \cdot \bar{b} + e \cdot b) \quad (13)$$

$$\bar{c} \cdot b \cdot \bar{a} \cdot (\bar{e} \cdot b + e \cdot d + \bar{e} \cdot \bar{d} + e \cdot \bar{d}) \quad (14)$$

$$c \cdot \bar{b} \cdot \bar{a} \cdot (\bar{e} \cdot \bar{d} + \bar{e} \cdot d + e \cdot \bar{d} + e \cdot d) \quad (15)$$

$$e \cdot d \cdot c \cdot (\bar{b} \cdot \bar{a} + \bar{b} \cdot a + b \cdot \bar{a} + b \cdot a) \quad (16)$$

$$\bar{d} \cdot c \cdot b \cdot a \cdot (\bar{e} + e) \quad (17)$$

En 13, 14, 15 y 16 se puede aplicar la propiedad de combinación, mientras que en 12 se utiliza la ley del complemento:

$$\bar{d} \cdot \bar{c} \cdot \bar{a} \cdot (\bar{e} + e) \quad (18)$$

$$\bar{c} \cdot b \cdot \bar{a} \cdot (d + \bar{d}) \quad (19)$$

$$c \cdot \bar{b} \cdot \bar{a} \cdot (\bar{e} + e) \quad (20)$$

$$e \cdot d \cdot c \cdot (\bar{b} + b) \quad (21)$$

$$\bar{d} \cdot c \cdot b \cdot a \quad (22)$$

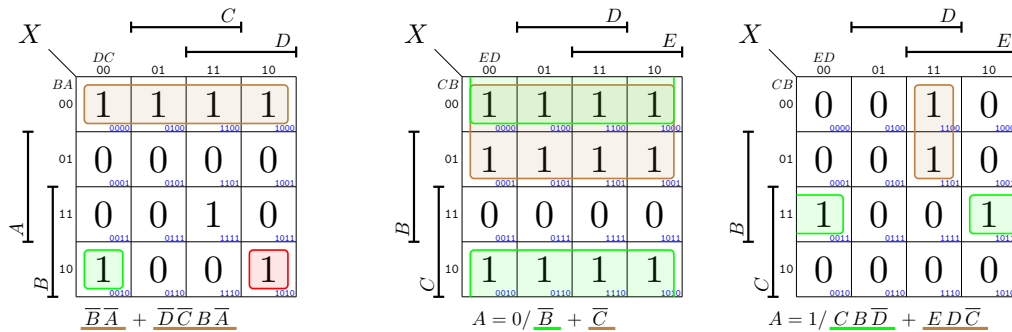
Sumando 18, 19, 20, 21 y 22:

$$\bar{d} \cdot \bar{c} \cdot \bar{a} \cdot (\bar{e} + e) + \bar{c} \cdot b \cdot \bar{a} \cdot (d + \bar{d}) + c \cdot \bar{b} \cdot \bar{a} \cdot (\bar{e} + e) + e \cdot d \cdot c \cdot (\bar{b} + b) + \bar{d} \cdot c \cdot b \cdot a \quad (23)$$

Finalmente se aplica nuevamente la ley del complemento en los cuatro primeros productos y se llega a la expresión final:

$$\boxed{\bar{d} \cdot \bar{c} \cdot \bar{a} + \bar{c} \cdot b \cdot \bar{a} + c \cdot \bar{b} \cdot \bar{a} + e \cdot d \cdot c + \bar{d} \cdot c \cdot b \cdot a} \quad (24)$$

2.1.2. Simplificación mediante mapas de Karnaugh



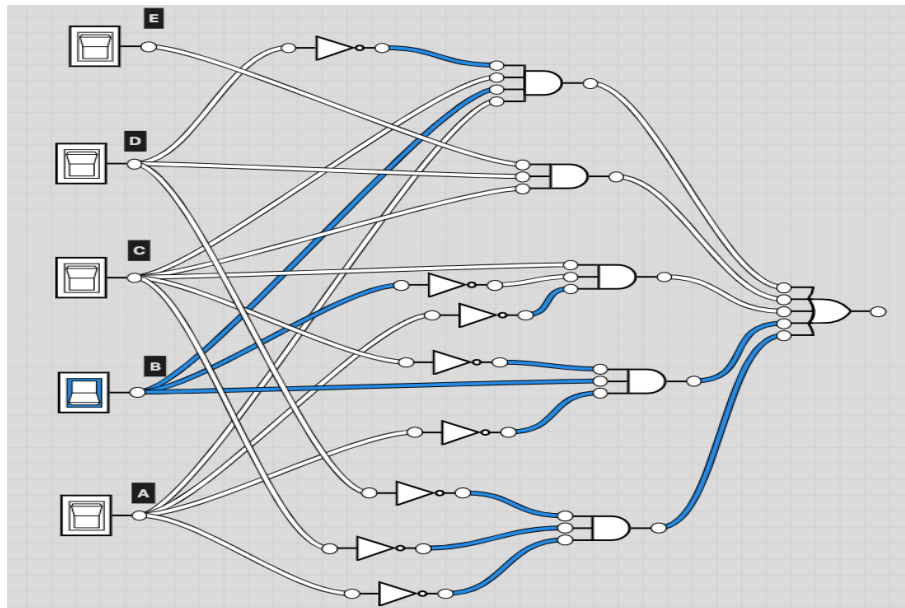
(0,2,8,10,16,18,24,26)	$\bar{C}\bar{E}$
(0,4,8,12,16,20,24,28)	$\bar{D}\bar{E}$
(24,25,26,27)	$AB\bar{C}$
(7,23)	$\bar{B}CDE$

	$\bar{D}\bar{E}$	$\bar{D}E$	$D\bar{E}$	DE
$\bar{A}\bar{B}\bar{C}$	1	0	0	1
$\bar{A}\bar{B}C$	1	0	1	0
$\bar{A}B\bar{C}$	1	0	0	0
$\bar{A}BC$	1	0	0	1
$A\bar{B}\bar{C}$	1	0	0	1
$A\bar{B}C$	1	0	1	0
$AB\bar{C}$	1	0	0	0
ABC	1	1	1	1

	$\bar{D}\bar{E}$	$\bar{D}E$	$D\bar{E}$	DE
$\bar{A}\bar{B}\bar{C}$	0	1	3	2
$\bar{A}\bar{B}C$	4	5	7	6
$\bar{A}B\bar{C}$	12	13	15	14
$\bar{A}BC$	8	9	11	10
$A\bar{B}\bar{C}$	16	17	19	18
$A\bar{B}C$	20	21	23	22
$AB\bar{C}$	28	29	31	30
ABC	24	25	27	26

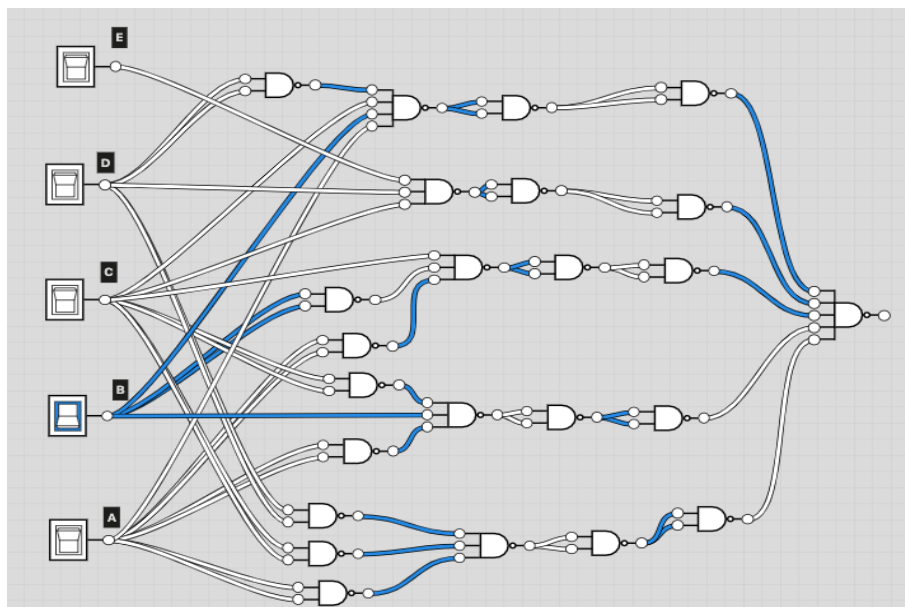
2.1.3. Implementación mediante compuertas AND, OR y NOT

La expresión obtenida en 24 puede ser implementada mediante compuertas lógicas fácilmente:



2.1.4. Implementación mediante compuertas NAND

A su vez el circuito anterior implementado mediante compuertas nand queda de la siguiente manera:



2.2. Segunda expresión: producto de sumas

2.2.1. Simplificación mediante álgebra booleana

Para la segunda parte del ejercicio, se comienza a partir del siguiente producto de sumas:

$$f(d, c, b, a) = \prod M_0, M_2, M_4, M_7, M_8, M_{10}, M_{12} \quad (25)$$

A a partir de 25 se define la siguiente expresion:

$$(d+c+b+a) \cdot (d+c+\bar{b}+a) \cdot (d+\bar{c}+b+a) \cdot (d+\bar{c}+\bar{b}+\bar{a}) \cdot (\bar{d}+c+b+a) \cdot (\bar{d}+c+\bar{b}+a) \cdot (\bar{d}+\bar{c}+b+a) \quad (26)$$

En este caso se usa nuevamente la propiedad de idempotencia, en este caso para un producto, para duplicar el mintermino número 8 y 0 y así facilitar el trabajo algebraico. Luego se separan los productos en tres grupos para su posterior factorización:

$$(d+c+b+a) \cdot (d+c+\bar{b}+a) \cdot (\bar{d}+c+b+a) \cdot (\bar{d}+c+\bar{b}+a) \quad (27)$$

$$(d+c+b+a) \cdot (d+\bar{c}+b+a) \cdot (\bar{d}+c+b+a) \cdot (\bar{d}+\bar{c}+b+a) \quad (28)$$

$$(d+\bar{c}+\bar{b}+\bar{a}) \quad (29)$$

Tanto en 27 como en 28 es posible usar la propiedad de combianción para el producto entre los primeros dos y los últimos dos términos para simplificar las expresiones. 29 esta expresado en forma canónica.

$$(d+c+a) \cdot (\bar{d}+c+a) \quad (30)$$

$$(d+b+a) \cdot (\bar{d}+b+a) \quad (31)$$

Luego se utiliza la propiedad de combianción para el producto nuevamente, en ambas ecuaciones:

$$(c+a) \quad (32)$$

$$(b+a) \quad (33)$$

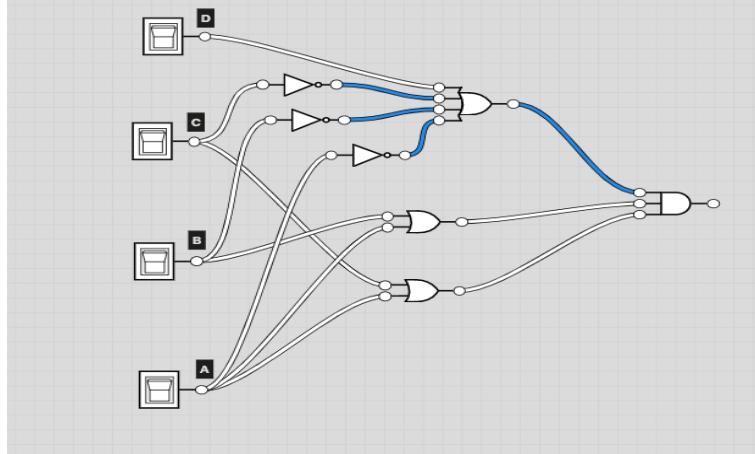
Por último se unen 32, 33 y 29 para llegar a una expresión final:

$$\boxed{(c+a) \cdot (b+a) \cdot (d+\bar{c}+\bar{b}+\bar{a})} \quad (34)$$

2.2.2. Simplificacioón mediante mapas de Karnaugh

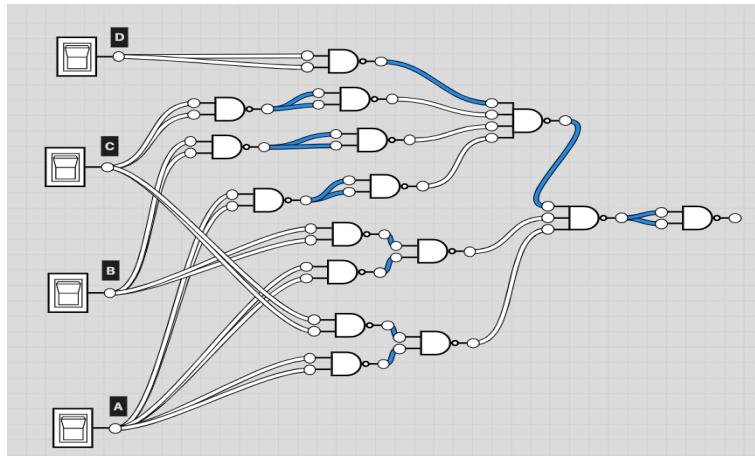
2.2.3. Implementación mediante compuertas AND, OR y NOT

Al igual que con la expresion anterior se implementó la función obtenida en 34 mediante compuertas AND, OR y NOT:



2.2.4. Implementación mediante compuertas NAND

Por último se realizó el circuito anterior solamente con compuertas nand:



3. Implementación de módulos en verilog

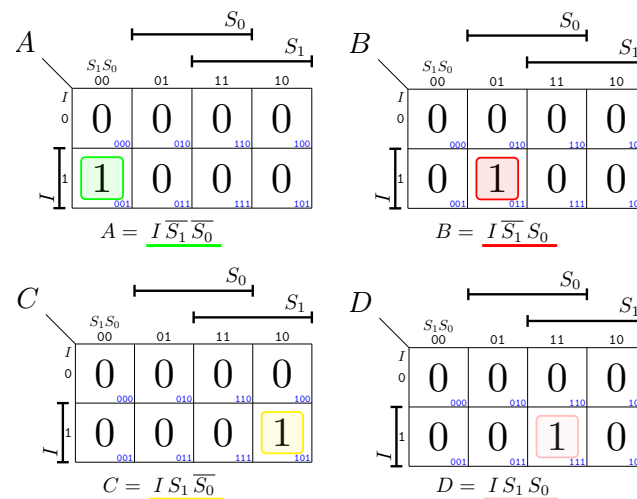
3.1. Demultiplexor de 4 salidas

A continuación, se analiza la tabla de verdad de un multiplexor de 4 salidas:

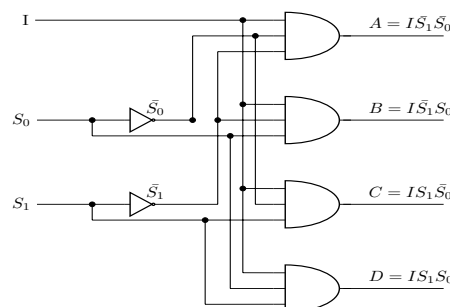
I	S_1	S_0	A	B	C	D
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	0	0	0	0
0	1	1	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1

Tabla 1: Tabla de verdad del Demultiplexor

Cada salida distinta nos permitirá diagramar un mapa de Karnaugh propio, los mismos se presentan a continuación:

**Figura 1:** Mapas de Karnaugh de las salidas del Demultiplexor

Se procede a implementar el circuito hallado mediante los mapas:

**Figura 2:** Circuito Demultiplexor de 4 salidas

Respecto del diseño en verilog,

3.2. Codificador de 4 entradas

A continuación, se analiza la tabla de verdad de un codificador de 4 entradas:

A	B	C	D	S_1	S_0	E
1	0	0	0	0	0	0
0	1	0	0	0	1	0
0	0	1	0	1	0	0
0	0	0	1	1	1	0
X	X	X	X	X	X	1

Tabla 2: Tabla de verdad del Codificador

Cada salida distinta nos permitirá diagramar un mapa de Karnaugh propio. Se contempla el caso de un error cuando las entradas no sean propias a las de un codificador. Los mapas se presentan a continuación:

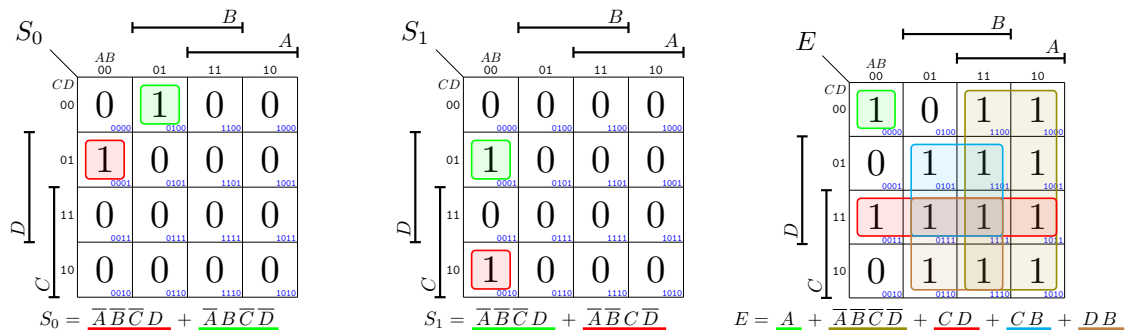


Figura 3: Mapas de Karnaugh de las salidas del Codificador

Por claridad, se coloca, por un lado, el circuito propio al codificador, y por otro, el utilizado para detectar un error. Sin embargo, los mismos podrían estar integrados. Los circuitos propuestos son los siguientes:

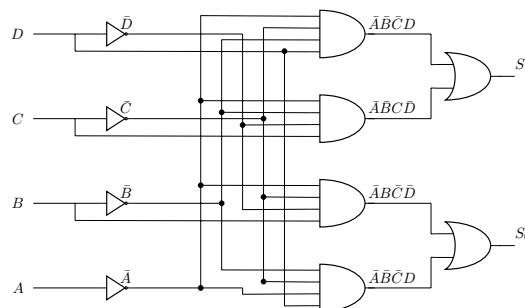


Figura 4: Circuito Codificador de 4 entradas

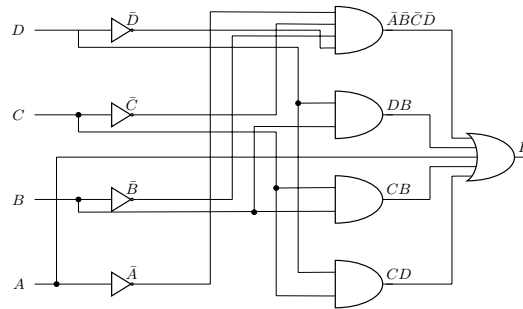


Figura 5: Circuito detector de error - Encoder

4. Ejercicio 4 - Conversor a código de Gray

Para este ejercicio, realizamos el desarrollo de un circuito lógico capaz de convertir un número binario de 4 bits a su equivalente de código de Gray, esto resulta en la siguiente tabla de verdad:

Entrada				Salida			
X_1	X_2	X_3	X_4	Y_1	Y_2	Y_3	Y_4
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

De la tabla de verdad obtenemos las siguientes ecuaciones en función de los minterminos:

$$\begin{aligned}
 Y_4 &= m_1 + m_2 + m_5 + m_6 + m_9 + m_{10} + m_{13} + m_{14} \\
 Y_3 &= m_2 + m_3 + m_4 + m_5 + m_{10} + m_{11} + m_{12} + m_{13} \\
 Y_2 &= m_4 + m_5 + m_6 + m_7 + m_8 + m_9 + m_{10} + m_{11} \\
 Y_1 &= m_8 + m_9 + m_{10} + m_{11} + m_{12} + m_{13} + m_{14} + m_{15}
 \end{aligned}$$

Que al reemplazar cada mintermino por su correspondiente expresión obtenemos:

$$\begin{aligned}
 Y_4 &= \overline{X_1} \cdot \overline{X_2} \cdot \overline{X_3} \cdot X_4 + \overline{X_1} \cdot \overline{X_2} \cdot X_3 \cdot \overline{X_4} + \overline{X_1} \cdot X_2 \cdot \overline{X_3} \cdot X_4 + \overline{X_1} \cdot X_2 \cdot X_3 \cdot \overline{X_4} + X_1 \cdot \overline{X_2} \cdot \overline{X_3} \cdot X_4 + X_1 \cdot \overline{X_2} \cdot X_3 \cdot \overline{X_4} + X_1 \cdot X_2 \cdot \overline{X_3} \cdot X_4 + X_1 \cdot X_2 \cdot X_3 \cdot \overline{X_4} \\
 Y_3 &= \overline{X_1} \cdot \overline{X_2} \cdot X_3 \cdot \overline{X_4} + \overline{X_1} \cdot \overline{X_2} \cdot X_3 \cdot X_4 + \overline{X_1} \cdot X_2 \cdot \overline{X_3} \cdot \overline{X_4} + \overline{X_1} \cdot X_2 \cdot \overline{X_3} \cdot X_4 + X_1 \cdot \overline{X_2} \cdot X_3 \cdot \overline{X_4} + X_1 \cdot \overline{X_2} \cdot X_3 \cdot X_4 + X_1 \cdot X_2 \cdot \overline{X_3} \cdot \overline{X_4} + X_1 \cdot X_2 \cdot \overline{X_3} \cdot X_4 \\
 Y_2 &= \overline{X_1} \cdot X_2 \cdot \overline{X_3} \cdot \overline{X_4} + \overline{X_1} \cdot X_2 \cdot \overline{X_3} \cdot X_4 + \overline{X_1} \cdot X_2 \cdot X_3 \cdot \overline{X_4} + \overline{X_1} \cdot X_2 \cdot X_3 \cdot X_4 + X_1 \cdot \overline{X_2} \cdot \overline{X_3} \cdot \overline{X_4} + X_1 \cdot \overline{X_2} \cdot \overline{X_3} \cdot X_4 + X_1 \cdot \overline{X_2} \cdot X_3 \cdot \overline{X_4} + X_1 \cdot \overline{X_2} \cdot X_3 \cdot X_4 \\
 Y_1 &= X_1 \cdot \overline{X_2} \cdot \overline{X_3} \cdot \overline{X_4} + X_1 \cdot \overline{X_2} \cdot \overline{X_3} \cdot X_4 + X_1 \cdot \overline{X_2} \cdot X_3 \cdot \overline{X_4} + X_1 \cdot \overline{X_2} \cdot X_3 \cdot X_4 + X_1 \cdot X_2 \cdot \overline{X_3} \cdot \overline{X_4} + X_1 \cdot X_2 \cdot \overline{X_3} \cdot X_4 + X_1 \cdot X_2 \cdot X_3 \cdot \overline{X_4} + X_1 \cdot X_2 \cdot X_3 \cdot X_4
 \end{aligned}$$

Tenemos unas funciones muy larga y como las tenemos expresadas en minterminos podemos simplificarlas por medio del mapa de Karnaugh. Ésto nos da a lugar a los siguientes mapas de Karnaugh y funciones de salida simplificadas:

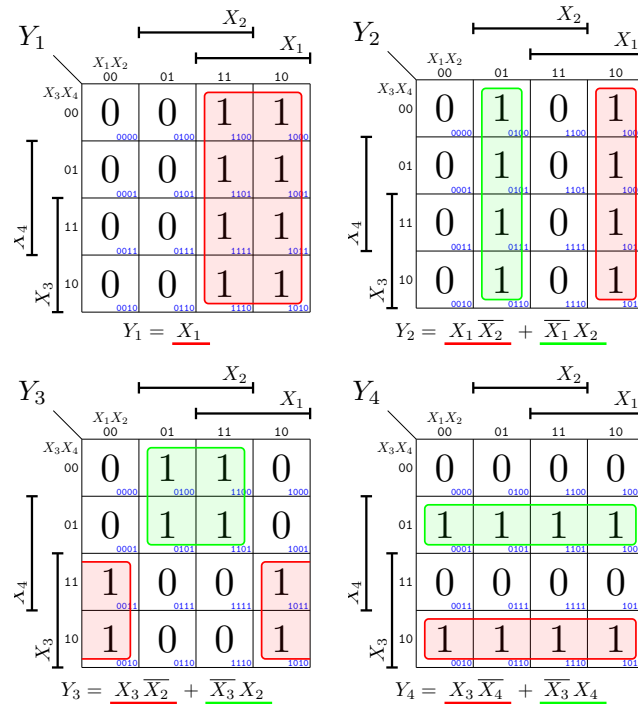


Figura 6: Mapas de Karnaugh de las salidas Y_1 , Y_2 , Y_3 e Y_4

De los valores obtenidos podemos realizar el siguiente circuito conformado por compuertas OR, AND y NOT:

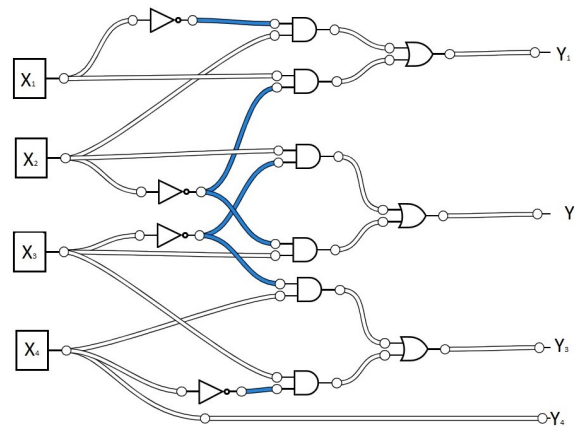


Figura 7: Implementación del conversor a código de Gray

5. Ejercicio 5

El objetivo de esta sección es implementar un programa en Verilog que multiplique dos números de un dígito en formato BCD y expresarlos a la salida como un número de dos dígitos en formato BCD. Como se trata de dos números de un dígito en BCD, ambos números serán de 4 bits con un rango de representación de 0 a 9. En cuanto a al producto, como son dos dígitos en BCD, serán dos "partes" de 4 bits cada una.

El primer problema a resolver es el de la multiplicación de dos números de 4 bits. Para estudiar el problema se desarrolla dicha operación. La misma se puede ver en la Figura 8.

			x_3	x_2	x_1	x_0	
	\times	y_3	y_2	y_1	y_0		
		$x_3 y_0$	$x_2 y_0$	$x_1 y_0$	$x_0 y_0$		
		$x_3 y_1$	$x_2 y_1$	$x_1 y_1$	$x_0 y_1$		
		$x_3 y_2$	$x_2 y_2$	$x_1 y_2$	$x_0 y_2$		
		$x_3 y_3$	$x_2 y_3$	$x_1 y_3$	$x_0 y_3$		
p_6	p_5	p_4	p_3	p_2	p_1	p_0	

Figura 8: Multiplicación de dos números de 4 bits

Como se puede observar se descompuso el producto en P_0 , P_1 , P_2 , P_3 , P_4 , P_5 , P_6 y P_7 . Cada uno de estos bits, que son parte del producto, están compuestos por sumas de operaciones ands y carrys. Por ejemplo: $P_2 = X_1 Y_1 + X_0 Y_2 + X_2 Y_0 + C_{in}$. Para poder resolver este tipo de operaciones se utilizan circuitos lógicos llamados Half Adder y Full Adder (ver Figura9). Ambos circuitos lógicos son de gran utilidad. El Half Adder permite hacer sumas de dos bits y devolver su carry. En cuanto al Full Adder, puede sumar (además de dos bits) un carry entrante y devolver su respectivo carry de salida. Además es posible combinar estos circuitos lógicos para obtener distintos resultados.

Volviendo al problema en cuestión, el circuito lógico que devuelve el producto de dos números con dos bits es el que se ve en la Figura 10.

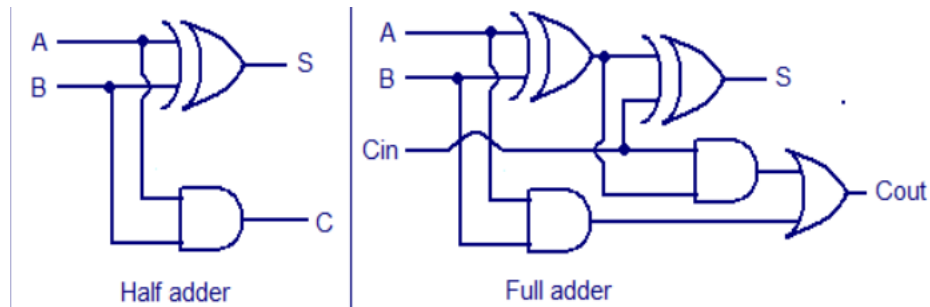


Figura 9: Half Adder y Full Adder

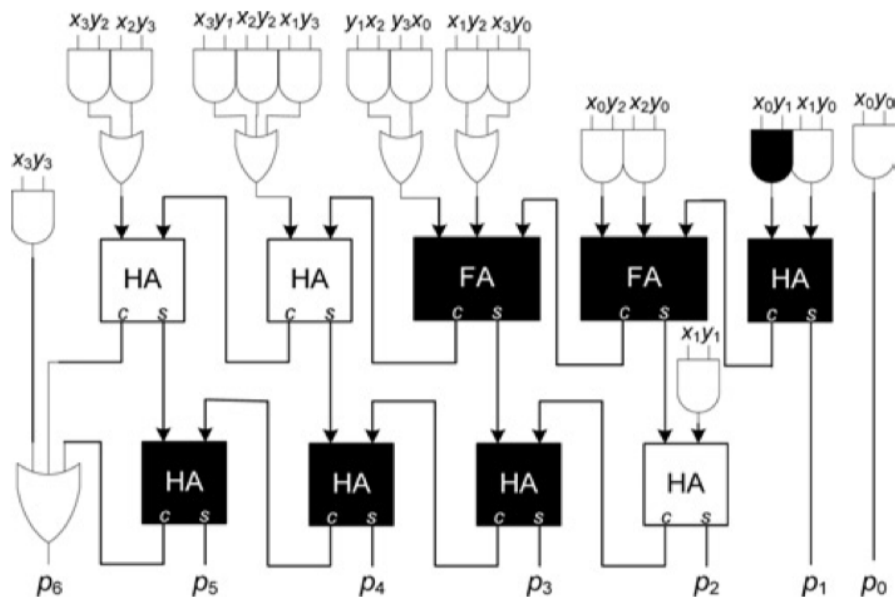


Figura 10: Circuito lógico para multiplicar dos números de 4 bits

Como se puede observar el circuito lógico está compuesto de Half y Full adders. Al tener el diagrama del circuito lógico completo es posible plasmarlo en un script de Verilog.

Habiendo sobrepasado el inconveniente de obtener el producto de dos números de cuatro bits, surge el problema de convertir dicho número a BCD. Esto se resuelve fácilmente mediante un algoritmo llamado Double Dabble. Mediante varias iteraciones del mismo es posible convertir un número binario en BCD. Para poder utilizar el programa se deben utilizar las siguientes instrucciones:

```
make
bash run
```

Estos comandos corren el programa de prueba *ej5_test.v* y el programa principal *ej5.v*. El programa de prueba somete al programa principal a todas las multiplicaciones posibles para que se devuelva el resultado en BCD.