INSTITUTO TECNOLÓGICO DE BUENOS AIRES

Trabajo Práctico Nº 1

Electrónica III - 2019

Grupo 1:

Farall, Facundo
Gaytan, Joaquín
Kammann, Lucas
Maselli, Carlos

4 de septiembre de 2019

ÍNDICE

1. EJERCICIO 1

Se realizó un programa bajo el nombre run.py, dentro de la carpeta EJ_1 del proyecto, el cual calcula el rango y resolución para una cierta convención de números en punto fijo. Recibe tres parámetros por línea de comando: un 1 o un 0 indicando si el sistema es signado, cantidad de bits de la parte entera, y cantidad de bits de la parte fraccionaria, en ese órden. Ante algún error en el formato en que se le pasan los parámetros, o ya sea porque los mismos exceden las capacidades del programa, se imprimirá en pantalla el mensaje ERROR. Los errores pueden estar ocasionados por alguna de las siguientes razones:

- No se pasa ningún argumento.
- Los argumentos no son números enteros.
- Se pasa una cantidad de argumentos distinta a 3.
- El primer argumento (que indica si el sistema es signado o no) es distinto de 0 o 1.

1.1. Limitaciones del programa

También puede devolverse ERROR a causa de que la cantidad de bits asignados a la parte entera o a la fraccionaria exceden los límites del lenguaje utilizado para el programa. El código se escribió en Python y se buscaron los casos límite para los cuales el programa deja de devolver valores con sentido:

- La cantidad de bits de la parte entera debe ser menor o igual a 1023.
- La cantidad de bits de la parte fraccionaria debe se menor o igual a 1074.

Además de las limitaciones mencionadas, existe un error en la representación de los números fraccionarios, inherente al sistema por el cual las computadoras guardan los números. La mayoría de los números que en sistema decimal tienen una representación con una cantidad finita de dígitos detrás de la coma, no poseen tal característica en un sistema binario. Tal es el caso del número 0,1 en base 10, que en Python, así como muchos otros lenguajes de programación que representan a los números fraccionarios bajo el formato normalizado de IEEE-754, es guardado como 0,1000000000000000055511151231257827021181583404541015625. En la mayoría de estos casos, al imprimir el número en pantalla, el lenguaje de Python decidirá aproximar el valor real guardado en memoria, a uno que considere más amigable para el usuario humano. Un caso en el cual estas limitaciones pueden verse reflejadas, es si se ingresan como parámetros 0, 5 y 740 (no signado, con 5 bits de parte entera y 740 bits de parte fraccionaria), la salida del programa será:

- Res: 1.7290327071306454e-223
- Ran: 32.0

A estos se les puede calcular el error absoluto de la siguiente manera:

$$E_{Res} = 2^{-740} - 1.7290327071306454 \cdot 10^{-223} \approx 1 \cdot 10^{-236} \tag{1.1}$$

$$E_{Ran} = 32 - \left(2^5 - 2^{-740}\right) \approx 1.7290327071286 \cdot 10^{-223} \tag{1.2}$$

Para la resolución, se observarán diferencias recién luego de 13 cifras significativas, mientras que para el rango, luego de 223. Ambos errores cometidos pueden considerarse despreciables para casi cualquier sistema real de medición.

2. Ejercicio 2

2.1. Expresión en mintérminos

La subsección presente tratará la siguiente expresión caracterizada como:

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

2.1.1. Simplificación con Álgebra booleana

En el siguiente desarrollo primero se escribe de forma completa la suma de productos que se describe en la expresión consignada, donde cada producto se compone de la combinación de entradas en cada estado indicado, de forma tal que el resultado de tal producto sea un estado lógico activo, esto es, un '1' binario. Con el objetivo de proveer la mayor claridad posible en el desarrollo, se subindican los términos, en donde el subíndice describe la identificación de un término.

$$f(e,d,c,b,a) = {}_0\,\overline{e}\cdot\overline{d}\cdot\overline{c}\cdot\overline{b}\cdot\overline{a} + {}_1\,\overline{e}\cdot\overline{d}\cdot\overline{c}\cdot b\cdot\overline{a} + {}_2\,\overline{e}\cdot\overline{d}\cdot c\cdot\overline{b}\cdot\overline{a} + {}_3\,\overline{e}\cdot d\cdot\overline{c}\cdot\overline{b}\cdot\overline{a} + {}_4\,\overline{e}\cdot\overline{d}\cdot c\cdot b\cdot a$$
$$+ {}_5\,\overline{e}\cdot d\cdot\overline{c}\cdot b\cdot\overline{a} + {}_6\,\overline{e}\cdot d\cdot c\cdot\overline{b}\cdot\overline{a} + {}_7\,e\cdot\overline{d}\cdot\overline{c}\cdot\overline{b}\cdot\overline{a} + {}_8\,e\cdot\overline{d}\cdot\overline{c}\cdot b\cdot\overline{a} + {}_9\,e\cdot\overline{d}\cdot c\cdot\overline{b}\cdot\overline{a}$$
$$+ {}_{10}\,e\cdot\overline{d}\cdot c\cdot b\cdot a + {}_{11}\,e\cdot d\cdot\overline{c}\cdot\overline{b}\cdot\overline{a} + {}_{12}\,e\cdot d\cdot\overline{c}\cdot\overline{b}\cdot a + {}_{13}e\cdot d\cdot\overline{c}\cdot b\cdot\overline{a} + {}_{14}\,e\cdot d\cdot\overline{c}\cdot b\cdot a$$
$$+ {}_{15}\,e\cdot d\cdot c\cdot\overline{b}\cdot\overline{a}$$

Luego se toman los términos de una manera adecuada de forma tal que se aplica sobre ellos la propiedad de abosrción y luego se reducen, obteniendo como resultado la siguiente expresión en la cual se subindican a izquierda y en rojo de dónde fueron simplificados aplicando la absorción.

$$f(e,d,c,b,a) = {}_{13,14}\,e\cdot d\cdot\overline{c}\cdot b + {}_{13,5}\,d\cdot\overline{c}\cdot b\cdot\overline{a} + {}_{13,8}\,e\cdot\overline{c}\cdot b\cdot\overline{a} + {}_{0,1}\,\overline{e}\cdot\overline{c}\cdot\overline{d}\cdot\overline{a} + {}_{0,2}\,\overline{e}\cdot\overline{d}\cdot\overline{b}\cdot\overline{a} + {}_{0,3}\,\overline{e}\cdot\overline{c}\cdot\overline{b}\cdot\overline{a}$$
$$+ {}_{3,6}\,\overline{e}\cdot d\cdot\overline{b}\cdot\overline{a} + {}_{0,7}\,\overline{d}\cdot\overline{c}\cdot\overline{b}\cdot\overline{a} + {}_{2,9}\,\overline{d}\cdot c\cdot\overline{b}\cdot\overline{a} + {}_{3,11}\,d\cdot\overline{c}\cdot\overline{b}\cdot\overline{a} + {}_{11,12}\,e\cdot d\cdot\overline{c}\cdot\overline{b} + {}_{11,15}\,e\cdot d\cdot\overline{b}\cdot\overline{a}$$
$$+ {}_{1,5}\,\overline{e}\cdot\overline{c}\cdot b\cdot\overline{a} + {}_{2,6}\,\overline{e}\cdot c\cdot\overline{b}\cdot\overline{a} + {}_{3,5}\,\overline{e}\cdot d\cdot\overline{c}\cdot\overline{a} + {}_{7,8}\,e\cdot\overline{d}\cdot\overline{c}\cdot\overline{a} + {}_{7,9}\,e\cdot\overline{d}\cdot\overline{b}\cdot\overline{a} + {}_{4,10}\,\overline{d}\cdot c\cdot b\cdot a$$

Con el fin de no recargar la notación y mantener el método empleado para vincular aquellos términos que son simplificados, se copian nuevamente la expresión previa con una numeración nueva que en este caso los identifica para su posterior simplificación.

$$f(e,d,c,b,a) = {}_0\,e\cdot d\cdot\overline{c}\cdot b + {}_1\,d\cdot\overline{c}\cdot b\cdot\overline{a} + {}_2\,e\cdot\overline{c}\cdot b\cdot\overline{a} + {}_3\,\overline{e}\cdot\overline{c}\cdot\overline{d}\cdot\overline{a} + {}_4\,\overline{e}\cdot\overline{d}\cdot\overline{b}\cdot\overline{a} + {}_5\,\overline{e}\cdot\overline{c}\cdot\overline{b}\cdot\overline{a}$$
$$+ {}_6\,\overline{e}\cdot d\cdot\overline{b}\cdot\overline{a} + {}_7\,\overline{d}\cdot\overline{c}\cdot\overline{b}\cdot\overline{a} + {}_8\,\overline{d}\cdot c\cdot\overline{b}\cdot\overline{a} + {}_9\,d\cdot\overline{c}\cdot\overline{b}\cdot\overline{a} + {}_{10}\,e\cdot d\cdot\overline{c}\cdot\overline{b} + {}_{11}\,e\cdot d\cdot\overline{b}\cdot\overline{a}$$
$$+ {}_{12}\,\overline{e}\cdot\overline{c}\cdot b\cdot\overline{a} + {}_{13}\,\overline{e}\cdot c\cdot\overline{b}\cdot\overline{a} + {}_{14}\,\overline{e}\cdot d\cdot\overline{c}\cdot\overline{a} + {}_{15}\,e\cdot\overline{d}\cdot\overline{c}\cdot\overline{a} + {}_{16}\,e\cdot\overline{d}\cdot\overline{b}\cdot\overline{a} + {}_{17}\,\overline{d}\cdot c\cdot b\cdot a$$

Repitiendo la modalidad, se simplifican algunos de los términos aplicando la regla de la absorción, se indican en los términos resultantes aquellos que fueron juntados.

$$f(e,d,c,b,a) = {}_{0,10}\,e\cdot d\cdot\overline{c} + {}_{1,9}\,d\cdot\overline{c}\cdot\overline{a} + {}_{8,9}\,\overline{d}\cdot\overline{b}\cdot\overline{a} + {}_{4,6}\,\overline{e}\cdot\overline{b}\cdot\overline{a} + {}_{2,12}\,\overline{c}\cdot b\cdot\overline{a} + {}_{3,14}\,\overline{e}\cdot\overline{c}\cdot\overline{a} + {}_{5,13}\,\overline{e}\cdot\overline{b}\cdot\overline{a}$$
$$+ {}_{11,16}\,e\cdot\overline{b}\cdot\overline{a} + {}_{17}\,\overline{d}\cdot c\cdot b\cdot a + {}_{3,15}\,\overline{d}\cdot\overline{c}\cdot\overline{a}$$

En la última expresión hay dos pares de términos de colores azul y marrón, sobre los cuales puede aplicarse propiedad de la absorción nuevamente y se reescribe adecuadamente como sigue:

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

Finalmente, se obtiene la expresión reducida:

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

2.1.2. Simplificación con mapas de Karnaugh

DC \ BA	00	01	11	10
00	1	0	0	1
01	1	0	1	0
11	1	0	0	0
10	1	0	0	1

Figura 2.1: E = 0

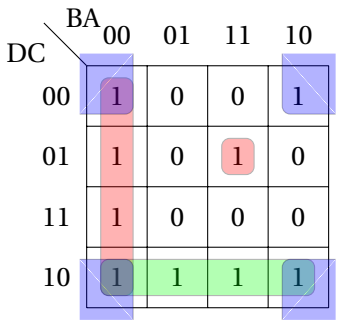

Figura 2.2: E = 1

Finalmente el resultado de tales selecciones de grupos resulta en la suma de productos:

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

2.1.3. Expresión desarrollada usando NOR

Partiendo de la primera expresión obtenida, se puede aplicar la propiedad de De Morgan para modificar las operaciones AND en operaciones OR aplicando un adecuado uso del algebra booleana y obteniendo finalmente todo escrito en compuertas NOR, como se hace a continuación.

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

2.1.4. Circuitos lógicos

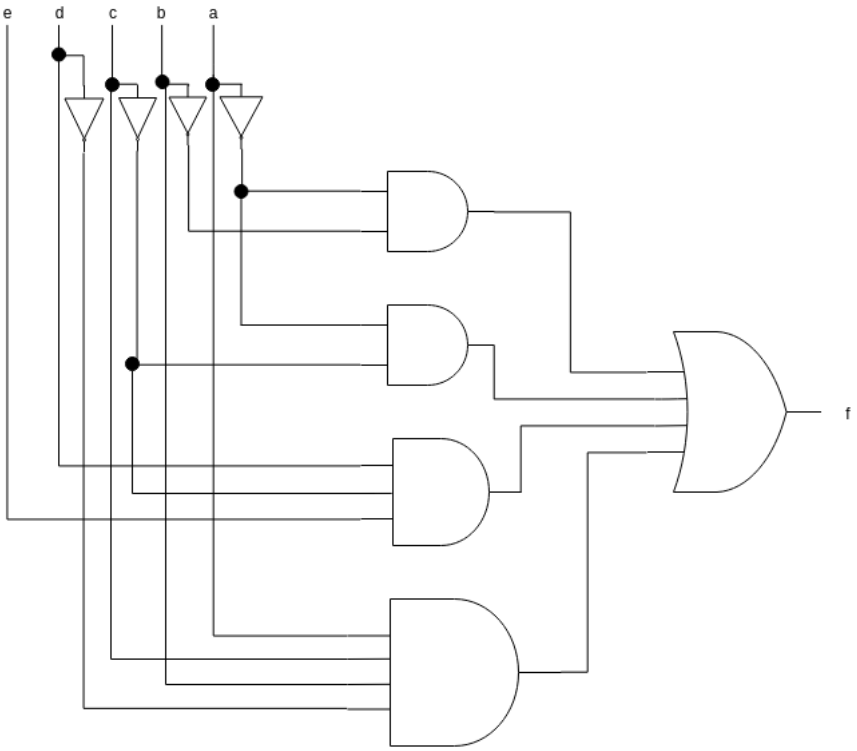

Figura 2.3: Circuito lógico según la expresión mínima hallada

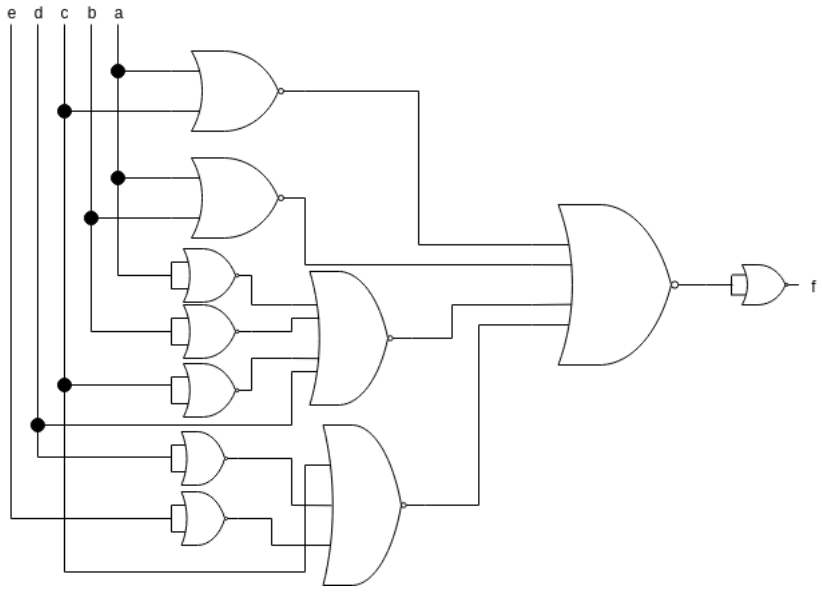

Figura 2.4: Circuito lógico implementado sólo con compuertas NOR

2.2. Expresión en maxtérminos

En forma análoga con la expresión de los mintérminos, se presenta la expresión de maxtérminos consignada para ser desarrollada y simplificada a través de diferentes métodos. Aquí también se emplea el uso de subíndices para describir e identificar los términos al momento de aplicar las propiedades.

$$f(d,c,b,a) = \prod M(0,2,4,7,8,10,12)$$

2.2.1. Simplificación con Álgebra booleana

$$f(d,c,b,a) = {}_{0}(d+c+b+a)\cdot {}_{1}(d+c+\overline{b}+a)\cdot {}_{2}(d+\overline{c}+b+a)\cdot {}_{3}(d+\overline{c}+\overline{b}+\overline{a})\cdot {}_{4}(\overline{d}+c+b+a)$$
$$\cdot {}_{5}(\overline{d}+c+\overline{b}+a)\cdot {}_{6}(\overline{d}+\overline{c}+b+a)$$

Se aplica la propiedad de absorción y, juntando los términos que son indicados en los subíndices de los resultantes, luego se llega a la siguiente expresión:

$$f(d,c,b,a) = {}_{0,1}(d+c+a)\cdot {}_{0,2}(d+b+a)\cdot {}_{0,4}(c+b+a)\cdot {}_{1,5}(c+\overline{b}+a)\cdot {}_{2,6}(\overline{c}+b+a)\cdot {}_{5}(d+\overline{c}+\overline{b}+\overline{a})$$
$$\cdot {}_{4,5}(\overline{d}+c+a)\cdot {}_{4,6}(\overline{d}+b+a)$$

Nuevamente aplicando absorción con los términos indicados en los colores azul y marrón, luego queda expresado de la siguiente forma para volver a aplicar la misma propiedad:

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

Finalmente absorbiendo estos miembros de las sumas:

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

2.2.2. Simplificación con mapas de Karnaugh

DC \ BA	00	01	11	10
00	0	1	1	0
01	0	1	0	1
11	0	1	1	1
10	0	1	1	0

De la selección de los grupos del mapa de Karnaugh se obtiene la siguiente expresión conformada por un producto de sumas de los maxtérminos reducidos por este método.

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

2.2.3. Expresión desarrollada usando NOR

En forma análoga a la otra parte del ejercicio la expresión obtenida anteriormente puede ser llevada a una forma donde se haga uso únicamente de operaciones NOR aplicando el teorema de De Morgan.

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

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

2.2.4. Circuitos lógicos

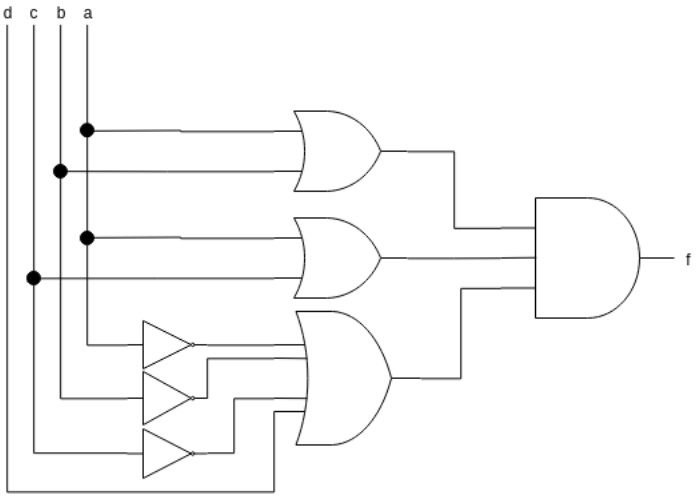

Figura 2.5: Circuito lógico según la expresión mínima hallada

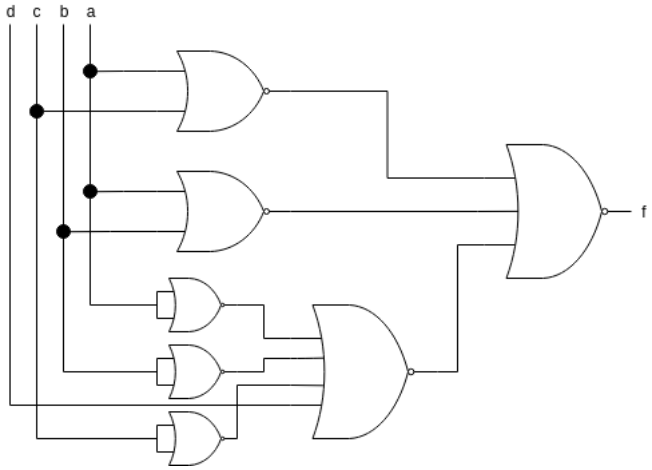

Figura 2.6: Circuito lógico implementado sólo con compuertas NOR

3. Ejercicio 3

De los dos módulos pedidos para implementar en Verilog se tomó la decisión de realizar uno mediante las compuertas lógicas que lo componen, y otro a través de la descripción de su comportamiento (behavioural). La razón detrás de esta decisión fue para hacer uso de las variantes provistas por Verilog, e interiorizarnos en su estilo de programación.

3.1. Decoder de 4 salidas

Esta implementación de un decoder de 4 salidas recibe una arreglo de 2 bits, que determinan cual de las cuatro salidas accionar. La misma se realizó a través de lógica de compuertas, donde únicamente la combinación correcta de los dos bits de entrada, ponen a la salida con ese número, en 1 lógico. Las relaciones lógicas son sencillas de comprender y quedan explicitadas en el mismo código. A continuación se presenta el código en Verilog:

```
module decoder4out(coded, y0, y1, y2, y3);
    input [1:0] coded;
    output y0, y1, y2, y3;

    assign y0 = ~coded[1] & ~coded[0];
    assign y1 = ~coded[1] & coded[0];
    assign y2 = coded[1] & ~coded[0];
    assign y3 = coded[1] & coded[0];

endmodule
```

3.2. MUX de 4 entradas

Esta implementación de un mux de 4 entradas recibe un arreglo de 4 bits con las 4 "fuentes", otro arreglo de 2 bits que hace las veces de selector, y cuenta con una salida de 1 bit. La salida copiará el valor de la entrada determinada por el selector. El módulo se logró mediante una descripción del comportamiento del mismo, en el cual se le especificó qué debía realizar ante cambios en alguna de sus entradas. A continuación se presenta el código en Verilog:

```
module mux4in (x, sel, y);
    input [3:0] x;
    input [1:0] sel;                    // sel selects the exit (x[3], x[2], x[1], x[0]).
    output reg y;

    always @(sel or x) begin
        if (sel == 0)
            assign y = x[0];
        else if (sel == 1)
            assign y = x[1];
        else if (sel == 2)
            assign y = x[2];
        else if (sel == 3)
            assign y = x[3];

    end

endmodule
```

3.3. TEST-BENCH

Se sometió a los dos módulos a un testeo de su respuesta a cada una de las posibles entradas, y los resultados fueron los esperados. Los mismos pueden ser replicados ejecutando los comandos:

```
user@computer: path/to/EJ_3/folder$ make
user@computer: path/to/EJ_3/folder$ ./run
```

4. EJERCICIO 4

4.1. TABLA DE VERDAD

En la siguiente tabla se muestra la equivalencia entre un número binario de 4 bits y el mismo en código de Gray. Según la convención adpotada, ."A"será el MSB y "D"será el LSB.

Tabla 4.1: Tabla de verdad

A	B	C	D	s_1	s_2	s_3	s_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

En base a esto, se puede expresar a cada bit de salida (s_1,s_2,s_3,s_4) como una combinación de entradas (A,B,C,D) en forma de minitérminos.
De esta forma, obtenemos las siguientes expresiones:

$$s_1 = A.\overline{B}.\overline{C}.\overline{D} + A.\overline{B}.\overline{C}.D + A.\overline{B}.C.\overline{D} + A.\overline{B}.C.D + A.B.\overline{C}.\overline{D} + A.B.\overline{C}.D + A.B.C.D \tag{4.1}$$

$$s_2 = A\overline{BCD} + A\overline{BC}D + A\overline{B}C\overline{D} + A\overline{B}CD + AB\overline{CD} + AB\overline{C}D + ABCD \tag{4.2}$$

$$s_3 = \overline{AB}C\overline{D} + \overline{AB}CD + \overline{A}B\overline{CD} + \overline{A}B\overline{C}D + A\overline{B}C\overline{D} + A\overline{B}CD + AB\overline{CD} + AB\overline{C}D \tag{4.3}$$

$$s_4 = \overline{ABC}D + \overline{AB}C\overline{D} + \overline{A}B\overline{C}D + \overline{A}BC\overline{D} + + A\overline{BCD} + A\overline{B}C\overline{D} + AB\overline{C}D + ABC\overline{D} \tag{4.4}$$

4.2. Simplificación. Mapas de Karnaugh

Las expresiones calculadas en el apartado anterior no son del todo eficientes a la hora de implementar el circuito lógico, debido a que no están apropiadamente simplificadas. Para encontrar la expresión óptima que satisfaga la tabla de verdad utilizaremos el método de mapas de Karnaugh, en este caso con minitérminos. Como tenemos cuatro salidas distintas debemos realizar cuatro de estos mapas, que resultarán en una expresión de suma de productos de las entradas. A continuación se reproducen estos diagramas y las expresiones derivadas de los mismos, donde en cada mapa cada color representa un grupo.

DC \ BA	00	01	11	10
00	0	0	1	1
01	0	0	1	1
11	0	0	1	1
10	0	0	1	1

Mapa de Karnaugh para s_1

DC \ BA	00	01	11	10
00	0	1	0	1
01	0	1	0	1
11	0	1	0	1
10	0	1	0	1

Mapa de Karnaugh para s_2

DC \ BA	00	01	11	10
00	0	1	1	0
01	0	1	1	0
11	1	0	0	1
10	1	0	0	1

Mapa de Karnaugh para s_3

DC \ BA	00	01	11	10
00	0	0	0	0
01	1	1	1	1
11	0	0	0	0
10	1	1	1	1

Mapa de Karnaugh para s_4

Realizando las correspondientes asociaciones obtenemos las siguientes expresiones que describen el comportamiento de cada bit de salida.

$$s_1 = A \tag{4.5}$$

$$s_2 = A.\overline{B} + \overline{A}.B \tag{4.6}$$

$$s_3 = B.\overline{C} + \overline{B}.C \tag{4.7}$$

$$s_4 = C.\overline{D} + \overline{C}.D \tag{4.8}$$

4.3. CIRCUITO LÓGICO

En base a las ecuaciones encontradas se construyó el circuito lógico mostrado abajo, utilizando compuertas OR, AND y NOT.

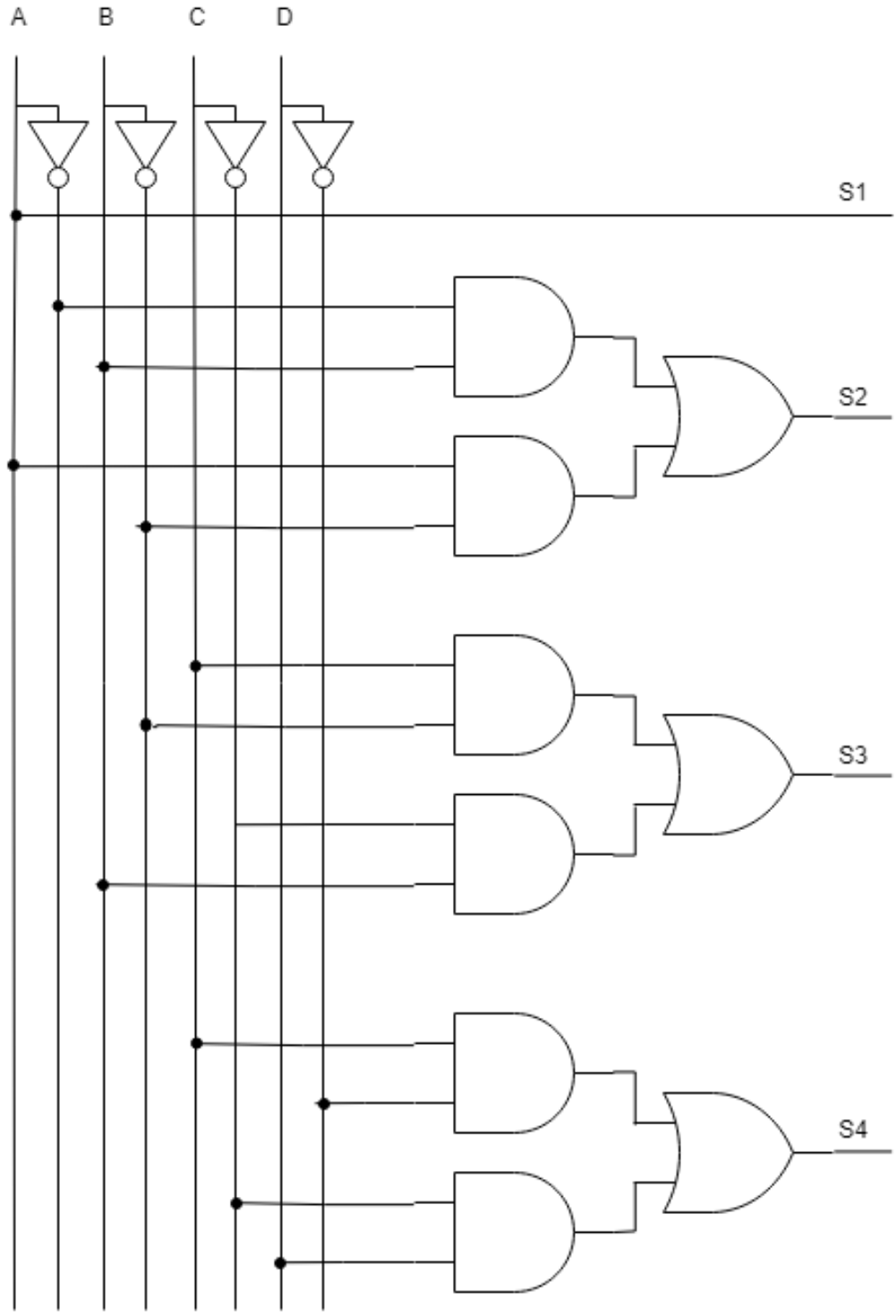

Figura 4.1: Circuito lógico

4.4. Programa en Verilog

Se adjunta en la entrega de este trabajo práctico el programa que representa al circuito mostrado en la sección previa.

5. Ejercicio 5

Se implementa en Verilog un multiplicador de 2 numeros de un dígito en codigo BCD que retorna el resultado como 2 digitos en formato BCD. Incluye además, un bit de error que será 1 en caso de que haya un error y 0 en otro caso.

5.1. Proceso de diseño

En un principio se decidió implementar este sistema a partir de realizar una tabla de verdad y un mapa de Karnaugh. Sin embargo, ese es un punto de vista que si bien parece ser simple, resulta muy engorroso de implementar. Esto se debe a la cantidad de entradas y salidas del sistema, 8 de cada una. Esto requiere resolver 8 mapas, con 8 variables de entrada. Se opta entonces, por implementarlo emulando la manera en que se resuelven multiplicaciones binarias a mano. Se puede observar en 5.1 un ejemplo de lo hablado.

$$\begin{array}{r} 1001 \\ \times \quad 0111 \\ \hline 1001 \\ 1001. \\ 1001.. \\ \underline{0000...} \\ 0111111 \end{array} \tag{5.1}$$

5.2. Implementacón

En la Figura 5.1 se puede observar la implementación de este proceso con Full-Adders y compuertas AND.

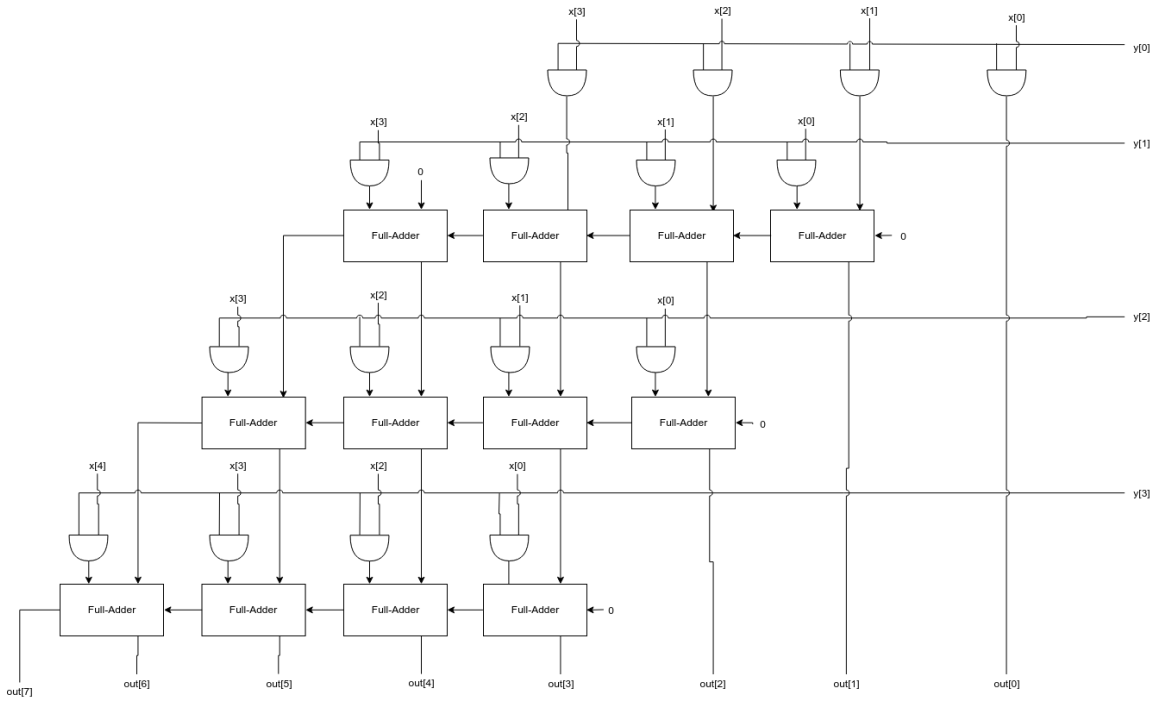

Figura 5.1: Circuito lógico que emula el caso descripto.

Este diagrama representa únicamente la seccion de la multiplicación binaria descripta en 5.1, el código esta escrito completamente con compuertas lógicas en Verilog. En cambio, tanto como para validar la información recibida como para codificar los resultados binarios en formato BCD se opta por usar bloques *behavioral*. Se adjunta junto con este informe, el codigo de Verilog con su correspondiente Makefile y un archivo *run*.

5.3. TEST-BENCH

Es posible ejecutar un *Test-bench* para verificar que el módulo funciona correctamente, con algunos ejemplos de las posibles entradas al sistema. El mismo puede se ejecutado por medio de los comandos:

```
user@computer: path/to/EJ_5/folder$ make
user@computer: path/to/EJ_5/folder$ ./run
```