

Organización del Computador 1

Práctica 2: Lógica Digital

1er cuatrimestre 2022

Índice

1. Ejercicio 1	2
2. Ejercicio 2	2
3. Ejercicio 3	3
4. Ejercicio 4	5
5. Ejercicio 5	5
6. Ejercicio 6	5
7. Ejercicio 7: demultiplexor	7
8. Ejercicio 8: codificador	8
9. Ejercicio 9: decodificador	10
10.Ejercicio 10: carry left shifter 3-4	11
11.Ejercicio 11: full adder de 1 bit	12
12.Ejercicio 12: full adder de 4 bits	13
13.Ejercicio 13	14

1. Ejercicio 1

Calculando las tablas de verdad podemos ver la equivalencia de las fórmulas booleanas.

1.a.

$$p = (p.q) + (p.\bar{q})$$

p	q	$(p.q)$	$+$	$(p.\bar{q})$
0	0	0	0	0
0	1	0	0	0
1	0	0	1	1
1	1	1	1	0

1.b.

$$x.z = (x + y).(x + \bar{y}).(\bar{x} + z)$$

x	y	z	$x.z$	$(x + y)$	$.$	$(x + \bar{y})$	$.$	$(\bar{x} + z)$
0	0	0	0	0	0	1	0	1
0	0	1	0	0	0	1	0	1
0	1	0	0	1	0	0	0	1
0	1	1	0	1	0	0	0	1
1	0	0	0	1	1	1	0	0
1	0	1	1	1	1	1	1	1
1	1	0	0	1	1	1	0	0
1	1	1	1	1	1	1	1	1

2. Ejercicio 2

Resolviendo mediante propiedades llegamos a 2 fórmulas que a priori no parecen ser equivalentes. Notar en la última línea que a la izquierda tenemos $\bar{y}.z$ mientras que a la derecha tenemos $\bar{y}.\bar{z}$.

$$\begin{aligned} x \oplus (y.z) &= (x \oplus y).(x \oplus z) \\ \bar{x}.y.z + x.\bar{y}.\bar{z} &= (\bar{x}.y + x.\bar{y}).(\bar{x}.z + x.\bar{z}) \\ \bar{x}.y.z + x.\bar{y}.\bar{z} &= (\bar{x}.y + x.\bar{y}).\bar{x}.z + (\bar{x}.y + x.\bar{y}).x.\bar{z} \\ \bar{x}.y.z + x.\bar{y}.\bar{z} &= \bar{x}.y.\bar{x}.z + x.\bar{y}.\bar{x}.z + \bar{x}.y.x.\bar{z} + x.\bar{y}.x.\bar{z} \\ \bar{x}.y.z + x.\bar{y}.\bar{z} &= \bar{x}.y.z + x.\bar{y}.\bar{z} \end{aligned}$$

Calculamos la tabla de verdad para verificar.

x	y	z	x	\oplus	$(y.z)$	$(x \oplus y)$	$.$	$(x \oplus z)$	
1	1	1	1	0	1	0	0	0	
1	1	0	1	1	0	0	0	1	✖
1	0	1	1	1	0	1	0	0	✖
1	0	0	1	1	0	1	1	1	
0	1	1	0	1	1	1	1	1	
0	1	0	0	0	0	1	0	0	
0	0	1	0	0	0	0	0	1	
0	0	0	0	0	0	0	0	0	

Conclusión: la propiedad planteada es falsa.

3. Ejercicio 3

3.a.

Verdadero, con el operador NAND ($p|q = \overline{p \cdot q}$) podemos representar todas las funciones booleanas: AND, OR, NOT.

Recordemos la tabla de verdad del NAND.

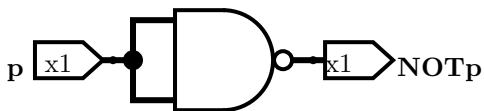
p	q	$p \cdot q$	$p q = \overline{p \cdot q}$
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

3.a.1. NOT

Utilizando la misma entrada 2 veces en un NAND podemos obtener un NOT.

$$p|p = \bar{p}$$

p	$p p$	\bar{p}
0	1	1
1	0	0

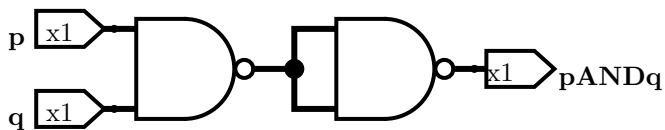


3.a.2. AND

Utilizando el NOT ya construido, podemos encadenarlo a la salida de un NAND para cancelar su negación y así obtener el resultado original del AND.

$$(p|q)|(p|q) = p \cdot q$$

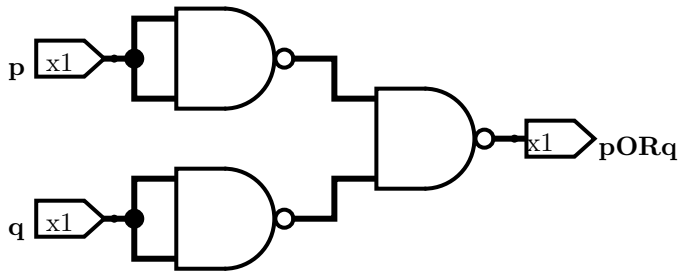
p	q	$p q$	$(p q) (p q)$	$p \cdot q$
0	0	1	0	0
0	1	1	0	0
1	0	1	0	0
1	1	0	1	1



3.a.3. OR

$$(p|p)|(q|q) = \bar{p}|\bar{q} = \overline{\bar{p} \cdot \bar{q}} = \bar{\bar{p} + \bar{q}} = p + q$$

p	q	$(p p)$	$(q q)$	$p + q$
0	0	1	1	0
0	1	1	0	1
1	0	0	1	1
1	1	0	0	1



3.b.

Verdadero, con el operador NOR ($p \downarrow q = \overline{p + q}$) podemos representar todas las funciones booleanas: AND, OR, NOT.

Recordemos la tabla de verdad del NOR.

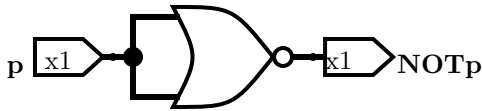
p	q	$p + q$	$p \downarrow q = \overline{p + q}$
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

Observemos que los circuitos para armar las funciones booleanas utilizando solo la compuerta NOR son análogos a los utilizando con la compuerta NAND.

3.b.1. NOT

$$p \downarrow p = \bar{p}$$

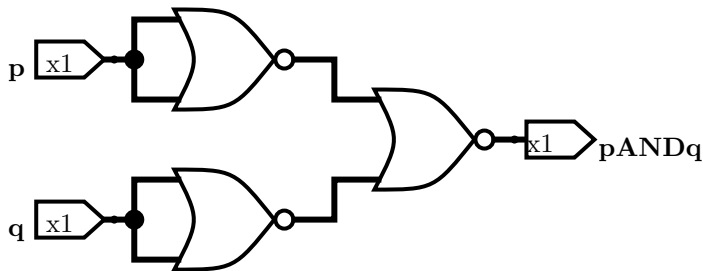
p	$p \downarrow p$	\bar{p}
0	1	1
1	0	0



3.b.2. AND

$$(p \downarrow p) \downarrow (q \downarrow q) = \bar{p} \downarrow \bar{q} = \overline{\bar{p} + \bar{q}} = \overline{\bar{p} \cdot \bar{q}} = p \cdot q$$

p	q	$(p \downarrow p)$	\downarrow	$(q \downarrow q)$	$p \cdot q$
0	0	1	0	1	0
0	1	1	0	0	0
1	0	0	0	1	0
1	1	0	1	0	1



3.b.3. OR

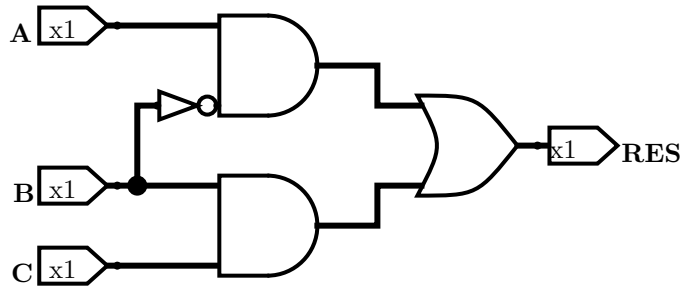
$$(p \downarrow q) \downarrow (p \downarrow q) = (\overline{p + q}) \downarrow (\overline{p + q}) = \overline{\overline{p + q} + \overline{p + q}} = \overline{\overline{p + q}} = p + q$$

6.b.

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

La implementación optimizada requiere un total de 4 compuertas: 1 OR, 2 AND y 1 NOT.

A	B	C	B.C	+	A. \overline{B}
0	0	0	0	0	0
0	0	1	0	0	0
0	1	0	0	0	0
0	1	1	1	1	0
1	0	0	0	1	1
1	0	1	0	1	1
1	1	0	0	0	0
1	1	1	1	1	0



7. Ejercicio 7: demultiplexor

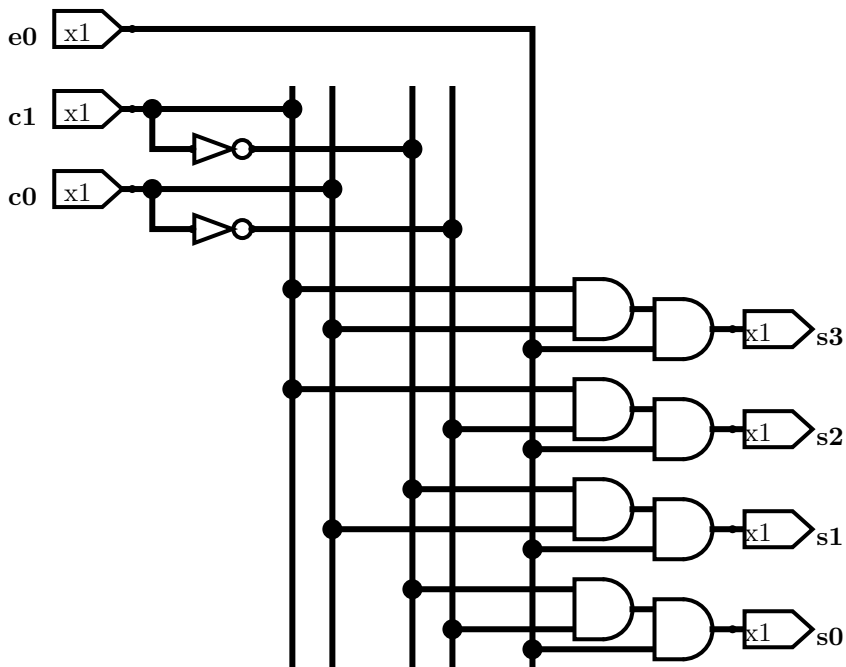
En la primer tabla de verdad planteamos una columna para los valores explícitos de e_0 . En la segunda tabla eliminamos esta columna y en cambio planteamos e_0 como el valor de salida cuando las líneas de control computan a 1 para esa fila. Es decir, dada la fórmula booleana para cada salida, primero vemos si el AND entre las 2 líneas de control daría 1, y solo en ese caso el resultado final va a ser determinado por e_0 . En todos los otros casos, como el AND entre las líneas de control da 0, es indistinto el valor de e_0 y la salida va a ser siempre 0.

c_1	c_0	e_0	s_3	s_2	s_1	s_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	0
0	0	1	0	0	0	1
0	1	1	0	0	1	0
1	0	1	0	1	0	0
1	1	1	1	0	0	0

c_1	c_0	s_3	s_2	s_1	s_0
0	0	0	0	0	e_0
0	1	0	0	e_0	0
1	0	0	e_0	0	0
1	1	e_0	0	0	0

Podemos plantear las siguientes fórmulas booleanas para cada una de las salidas.

$$s_3 = c_1 \cdot c_0 \cdot e_0 \quad s_2 = c_1 \cdot \overline{c_0} \cdot e_0 \quad s_1 = \overline{c_1} \cdot c_0 \cdot e_0 \quad s_0 = \overline{c_1} \cdot \overline{c_0} \cdot e_0$$



8. Ejercicio 8: codificador

8.a.

e_0	e_1	e_2	e_3	s_1	s_0
1	0	0	0	0	0
0	1	0	0	0	1
0	0	1	0	1	0
0	0	0	1	1	1

$$s_1 = \overline{e_0}.\overline{e_1}.e_2.\overline{e_3} + \overline{e_0}.\overline{e_1}.\overline{e_2}.e_3 = \overline{e_0}.\overline{e_1}.(e_2.\overline{e_3} + \overline{e_2}.e_3) = \overline{e_0 + e_1}.(e_2.\overline{e_3} + \overline{e_2}.e_3) = (e_0 \downarrow e_1).(e_2 \oplus e_3)$$

$$s_0 = \overline{e_0}.e_1.\overline{e_2}.\overline{e_3} + \overline{e_0}.\overline{e_1}.\overline{e_2}.e_3 = \overline{e_0}.\overline{e_2}.(e_1.\overline{e_3} + \overline{e_1}.e_3) = \overline{e_0 + e_2}.(e_1.\overline{e_3} + \overline{e_1}.e_3) = (e_0 \downarrow e_2).(e_1 \oplus e_3)$$

Nota: Consideramos s_1 como el bit más significativo de la representación binaria del número i .

8.b.

Para determinar si el estado de la entrada es válido necesitamos que $v = 1$ únicamente cuando hay una sola entrada en 1.

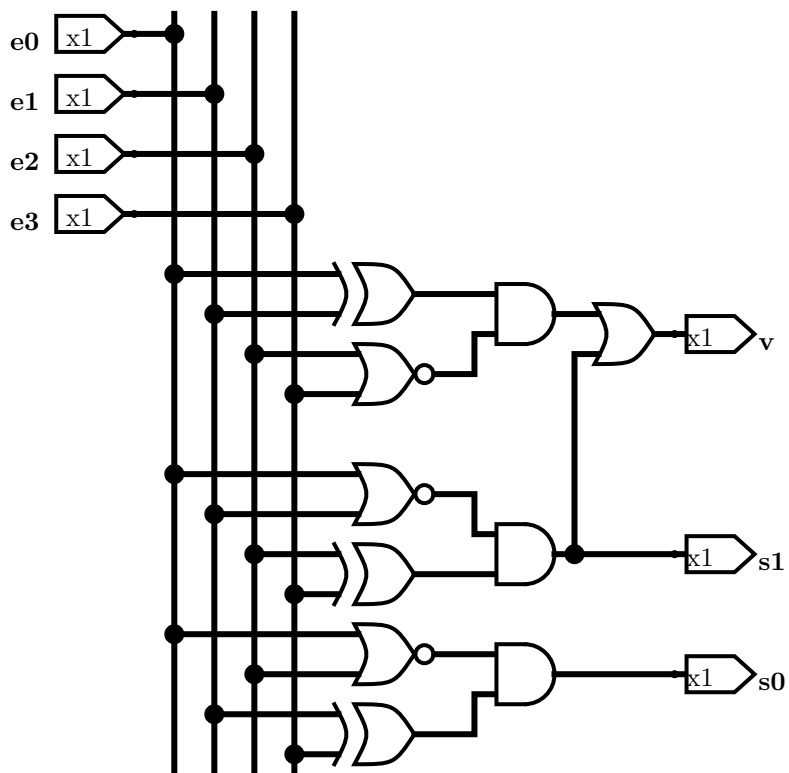
e_0	e_1	e_2	e_3	v
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

Traducimos la tabla de verdad a una fórmula booleana literal y simplificamos usando propiedades.

$$\begin{aligned}
 v &= e_0.\overline{e_1}.\overline{e_2}.\overline{e_3} + \overline{e_0}.e_1.\overline{e_2}.\overline{e_3} + \overline{e_0}.\overline{e_1}.e_2.\overline{e_3} + \overline{e_0}.\overline{e_1}.\overline{e_2}.e_3 \\
 &= (e_0.\overline{e_1} + \overline{e_0}.e_1).\overline{e_2}.\overline{e_3} + \overline{e_0}.\overline{e_1}.(e_2.\overline{e_3} + \overline{e_2}.e_3) \\
 &= (e_0 \oplus e_1).(e_2 \downarrow e_3) + (e_0 \downarrow e_1).(e_2 \oplus e_3)
 \end{aligned}$$

La fórmula se puede describir de la siguiente forma: la salida v será 1 si e_0 o e_1 son 1 (pero no ambos al mismo tiempo) y a su vez e_2 y e_3 son ambos 0. O, de forma análoga, si e_2 o e_3 son 1 (pero no ambos al mismo tiempo) y a su vez e_0 y e_1 son ambos 0.

Planteamos un único circuito para el codificador con la salida que indica si la entrada está en un estado válido.

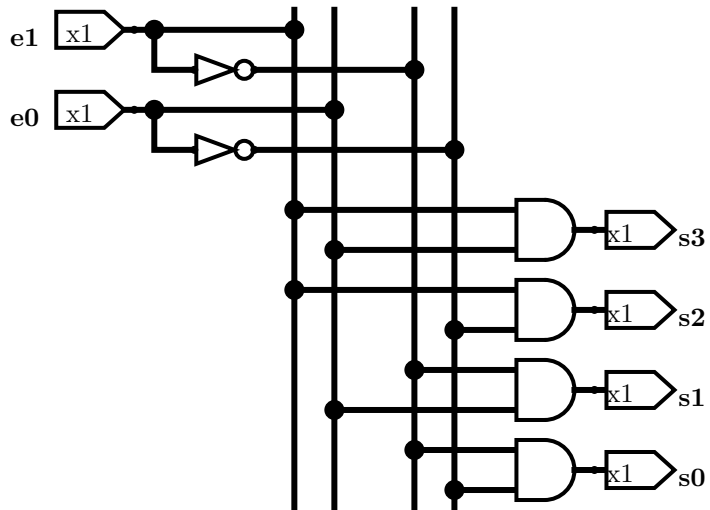


9. Ejercicio 9: decodificador

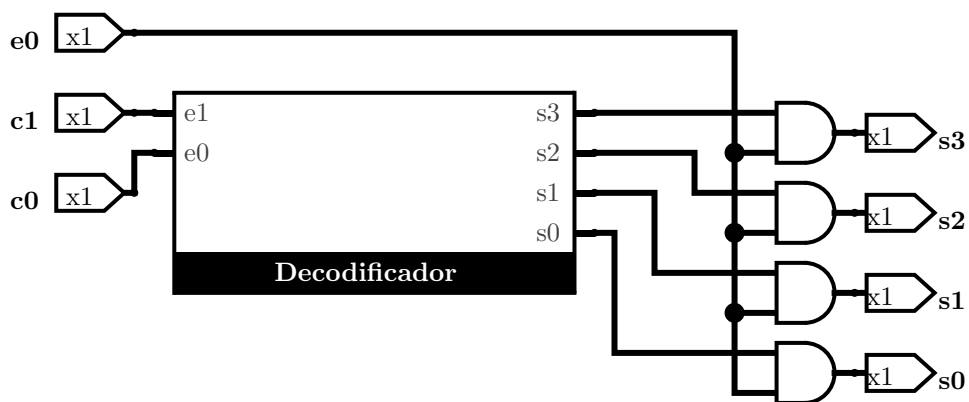
9.a.

e_1	e_0	s_3	s_2	s_1	s_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

$$s_3 = e_1 \cdot e_0 \quad s_2 = e_1 \cdot \overline{e_0} \quad s_1 = \overline{e_1} \cdot e_0 \quad s_0 = \overline{e_1} \cdot \overline{e_0}$$

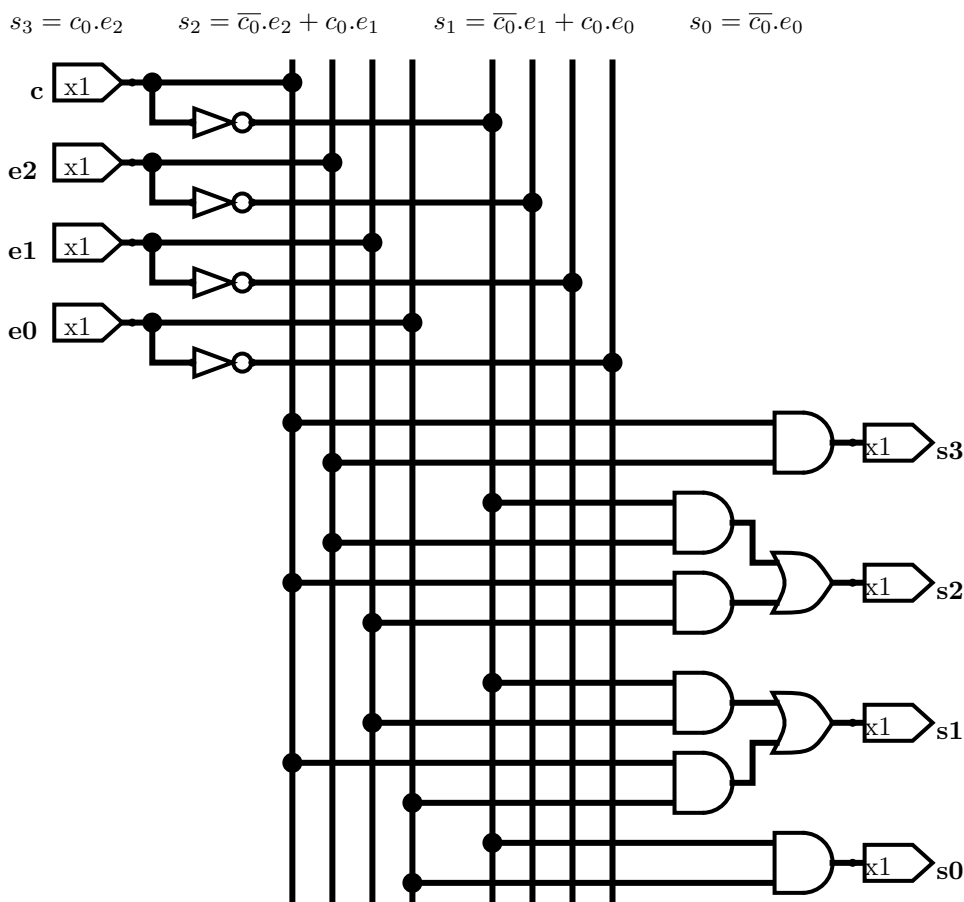


9.b.



10. Ejercicio 10: carry left shifter 3-4

10.a.



10.b.

El shift a la izquierda equivale a multiplicar por 2. El shift a la derecha equivale a la división entera por 2 (es decir, dividir por 2 redondeado hacia abajo).

$0110_2 = 6_{10}$
 $0110_2 \ll 1 = 1100_2 = 12_{10}$
 $0110_2 \gg 1 = 0011_2 = 3_{10}$

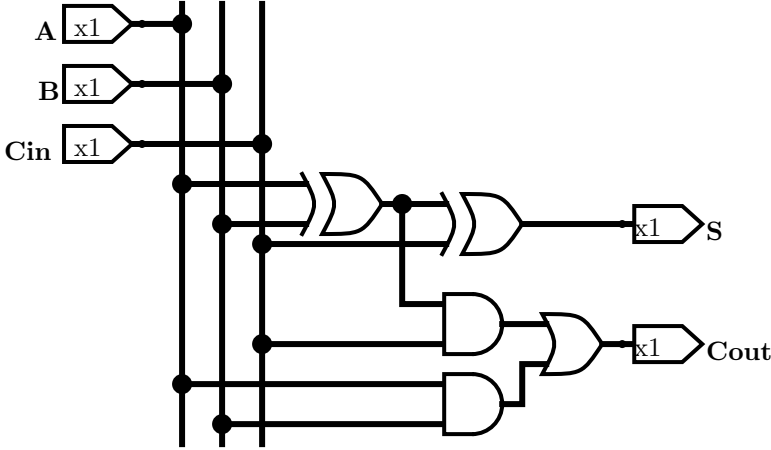
$0101_2 = 5_{10}$
 $0101_2 \ll 1 = 1010_2 = 10_{10}$
 $0101_2 \gg 1 = 0010_2 = 2_{10}$

11. Ejercicio 11: full adder de 1 bit

11.a.

$$\begin{aligned}
 S &= \overline{A}.\overline{B}.C_{in} + \overline{A}.B.\overline{C_{in}} + A.\overline{B}.\overline{C_{in}} + A.B.C_{in} \\
 &= (\overline{A}.\overline{B} + A.B).C_{in} + (\overline{A}.B + A.\overline{B}).\overline{C_{in}} \\
 &= (\overline{A \oplus B}).C_{in} + (A \oplus B).\overline{C_{in}} \\
 &= (A \oplus B) \oplus C_{in}
 \end{aligned}$$

$$\begin{aligned}
 C_{out} &= \overline{A}.B.C_{in} + A.\overline{B}.C_{in} + A.B.\overline{C_{in}} + A.B.C_{in} \\
 &= (\overline{A}.B + A.\overline{B}).C_{in} + (\overline{C_{in}} + C_{in}).A.B \\
 &= (A \oplus B).C_{in} + A.B
 \end{aligned}$$



11.b.

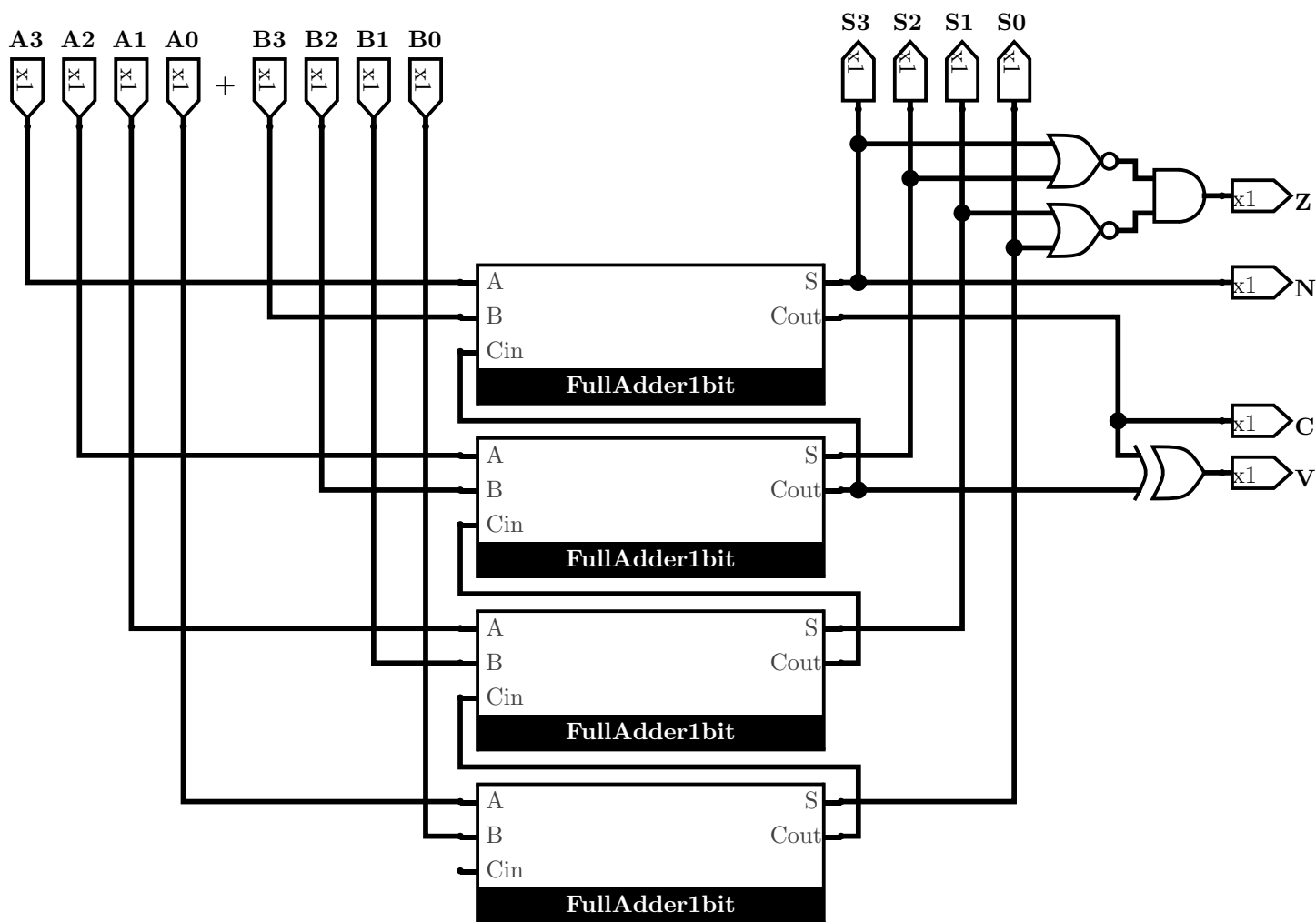
El retardo total del circuito será $\max\{\text{retardo}(S), \text{retardo}(C_{out})\}$. La salida S utiliza 2 compuertas mientras que C_{out} utiliza 4. Nos concentramos entonces en ver el retardo de C_{out} para cada instante.

1. Se activan las compuertas para las operaciones: $A \oplus B$ y $A.B$
2. Se activa la compuerta para la operación AND: $(A \oplus B).C_{in}$
3. Se activa la compuerta para la operación OR: $(A \oplus B).C_{in} + A.B$

Por lo tanto, el retardo total del circuito para producir todas su señales de salida es de $3t$.

12. Ejercicio 12: full adder de 4 bits

12.a.



Nota: Este circuito ya incluye los flags del item b.

12.b.

- Negative: $N = S_3$
- Overflow: $V = C_{in3} \oplus C_{out3}$
- Carry: $C = C_{out3}$
- Zero: $Z = (S_0 \downarrow S_1) \cdot (S_2 \downarrow S_3)$

12.c.

Sí, el circuito funciona para sumar números codificados en notación sin signo. Pero habría que ajustar los flags de la siguiente forma:

- Negative: no aplica ya que estamos sumando números sin signo
- Overflow: se ignora
- Carry: es el nuevo flag de overflow
- Zero: sin cambios

12.d.

Para realizar restas se podría modificar el circuito agregando una nueva señal de control O para indicar la operación: 0 representa suma, 1 representa resta.

Si los números están representados en complemento a 2, el procedimiento sería el siguiente:

- Invertir todos los bits de la entrada B .
- Suministrar un 1 en el C_{in} del full adder del bit menos significativo.

De esta forma encontramos el inverso aditivo de la entrada B y en efecto la operación realizada es: $A + (-B) = A - B$.

Para números representados en notación sin signo, si restringimos la entrada a 3 bits, el número se representa igual en complemento a 2 con 4 bits y podemos hacer la misma operación planteada anteriormente.

Tabla de verdad de la resta:

A	B	borrow_{in}	resta	borrow_{out}
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

12.e.

Pendiente

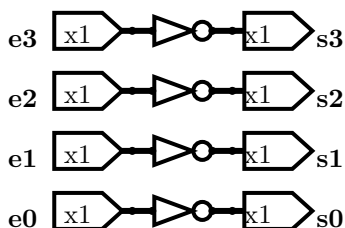
12.f.

Pendiente

13. Ejercicio 13

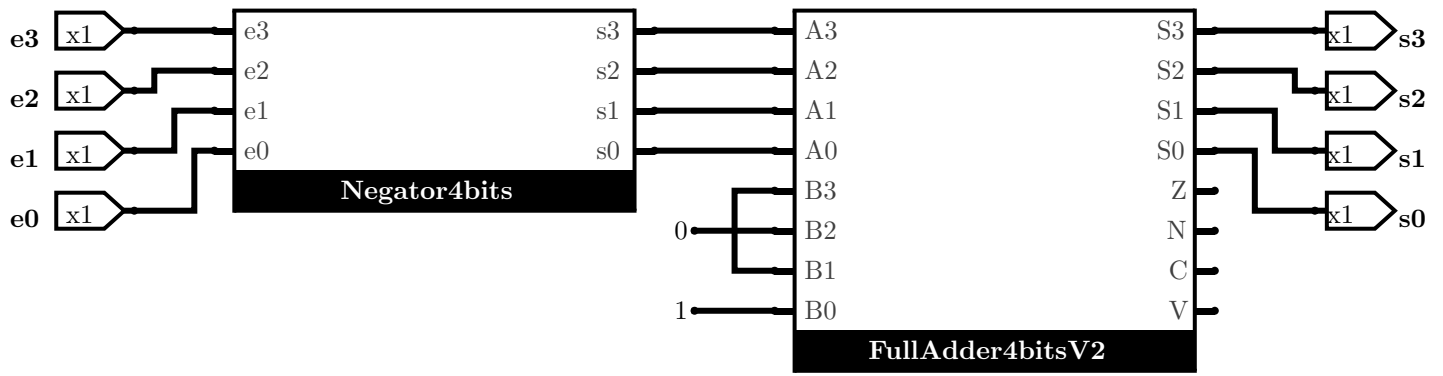
13.a. Negador

Simplemente negamos todas las entradas.



13.b. Inversor

Utilizando el negador y un full adder (para sumar 1) podemos aplicar el algoritmo para invertir un número en representación complemento a 2.



13.c. Inversor con flag

El único número que no tiene inverso aditivo en representación complemento a 2 es el -8_{10} , el cual se representa con el siguiente numeral: 1000_2 . Por lo tanto, la fórmula booleana para prender el flag es la siguiente: $F = e_3 \cdot \overline{e_2} \cdot \overline{e_1} \cdot \overline{e_0}$.

