



FIGURE 2.13: The basic logic gates of Boolean circuit design: (a) (left) The inverter, (b) (middle) The AND gate, and (c) (right) The OR gate.

Because of associativity and commutativity, the AND and OR gates may take any number of inputs (in any order) to produce (unambiguous) outputs. From these basic logic gates, more intricate circuits can be designed that will correspond to any given Boolean function. For a given Boolean expression representing a Boolean function, there is some flexibility in how the corresponding circuit can be drawn (by putting together the basic logic gates), we give an example to illustrate two of the more commonly used schemes.

**EXAMPLE 2.16:** Draw a circuit design for the Boolean function represented by the expression  $xyz + x(\bar{y} + \bar{z})$ .

**SOLUTION:** The gates needed to put together a circuit for this Boolean expression share some common inputs. Such common inputs can either be branched off from single input lines for each variable, or input lines can be drawn separately for each initial gate. Circuit diagrams corresponding to these two possible approaches are illustrated in Figure 2.14.

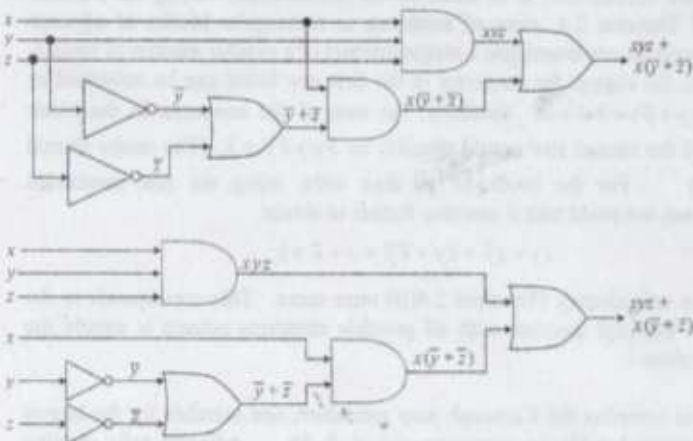


FIGURE 2.14: Two circuit drawings for the Boolean expression of Example 2.16: (a) (top) Using single input lines for each input variable. (b) (bottom) Using separate input lines for each occurrence of each input variable.

**EXERCISE FOR THE READER 2.17:** Draw a circuit design for the Boolean function represented by the expression  $(xy\bar{z})(x + \bar{y})$ .

**EXAMPLE 2.17:** Design a circuit that will control a light using three (standard light) switches in such a way that whenever one of the switches is flipped the status of the light will change (either from “on” to “off,” or from “off” to “on”).

**SOLUTION:** Each switch can be thought of as an inputted Boolean variable. We let the value 0 correspond to a switch being in the down position, and 1 correspond to the switch being in the up position. We let the variables  $x$ ,  $y$ , and  $z$  correspond to the three switch inputs, and  $f(x, y, z)$  denote the corresponding Boolean function, where the output will be 1 if the light is “on,” and 0 if the light is “off.” Any particular state of the switches can be assigned to produce either a 0 or a 1 output, and from this all other states (and thus the Boolean function) will be completely determined. We arbitrarily assign the value  $f(0, 0, 0) = 0$  (i.e., when all switches are down, we stipulate that the light is “off”). From this state, if exactly one of the switches is flipped up, the light turns “on,” so  $f(1, 0, 0) = f(0, 1, 0) = f(0, 0, 1) = 1$ . Next, if exactly two switches are up, the light will be off, i.e.,  $f(1, 1, 0) = f(1, 0, 1) = f(0, 1, 1) = 0$ . Finally, with all three switches up, the light will be on again:  $f(1, 1, 1) = 1$ . The resulting sum-of-products expansion for this Boolean (switching) function is thus:

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

A corresponding circuit diagram for this function is shown in Figure 2.15.

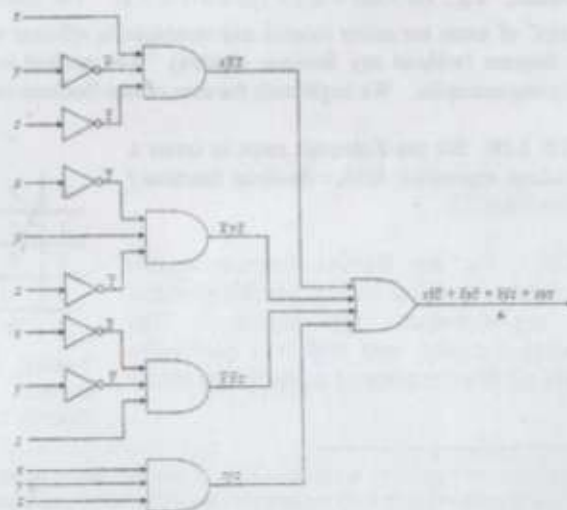


FIGURE 2.15: Diagram for the switching circuit of Example 2.17.

$x$	$y$	$x'$ NOT	$x \vee y$ OR	$(x \vee y)'$ NOR	$xy$ AND	$(xy)'$ NAND	$x \oplus y$ XOR
0	0	1	0	1	0	1	0
0	1	1	1	0	0	1	1
1	0	0	1	0	0	1	1
1	1	0	1	0	1	0	0

**EJEMPLO 1**

- a) La compuerta que se muestra en la figura 2a corresponde a la función booleana  $(x \vee y)'$ , o equivalentemente  $x'y$ .
- b) La compuerta AND de 3 entradas de la figura 2b va con la función  $x'yz$ .
- c) La compuerta de la figura 2c da  $(x'y')'$  o  $x \vee y$ , por lo que actúa como la puerta OR. ■

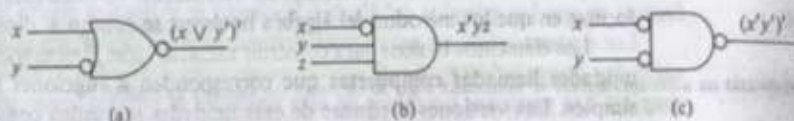


Figura 2

Consideramos el problema para diseñar redes de puertas que produzcan una complicada función booleana de varias variables. Una consideración importante es mantener un número pequeño de compuertas. Otra, es mantener una longitud pequeña de la cadena más larga de puertas. Otros criterios aparecen en aplicaciones prácticas concretas.

**EJEMPLO 2**

Consideremos la red diseñada con simpleza que se muestra en la figura 3a. [Los puntos indican lugares donde las líneas de entrada se dividen.] Hay cuatro compuertas en la red. Leyendo de izquierda a derecha hay dos cadenas con tres compuertas a lo largo. Calculamos las funciones booleanas en  $A$ ,  $B$ ,  $C$  y  $D$ :

$$A = (x \vee y)'; \quad B = x \vee z;$$

$$C = (A \vee B)' = ((x \vee y)' \vee (x \vee z))';$$

$$D = C \vee y = ((x \vee y)' \vee (x \vee z))' \vee y.$$

Las leyes del álgebra booleana nos dan

$$\begin{aligned} D &= ((x \vee y)(x \vee z))' \vee y = ((x \vee y)x'z') \vee y \\ &= (xx'z' \vee yx'z') \vee y = yx'z' \vee y = (y' \vee y)(x'z' \vee y) \\ &= x'z' \vee y. \end{aligned}$$

La red que se muestra en la figura 3b produce la misma salida, ya que

$$E = (x \vee z)' = x'z' \quad y \quad F = E \vee y = x'z' \vee y. \quad \blacksquare$$

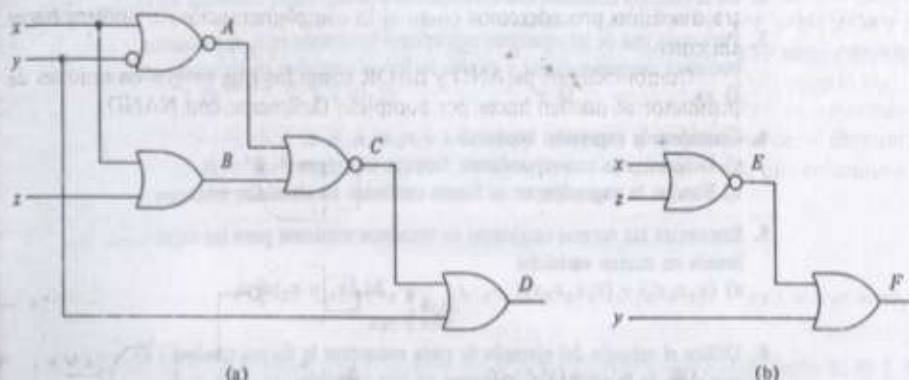


Figura 3

Este sencillo ejemplo muestra cómo a veces es posible rediseñar una complicada red en otra con menos puertas. Una de las razones para tratar de reducir la longitud de las cadenas de puertas es que en muchas situaciones, incluyendo simulaciones programadas de circuitos de alambres, la operación de cada compuerta se lleva una unidad fija de tiempo, y las puertas en la cadena se deben operar una después de la otra. Las cadenas largas significan operaciones más lentas.

Resulta que la expresión  $x'z' \vee y$  que obtuvimos de la complicada expresión  $D$  en el último ejemplo es una expresión óptima, en el sentido de § 10.4. Las expresiones óptimas no dan siempre las redes más sencillas. Por ejemplo, podemos mostrar [ejercicio 7a de § 10.6] que  $xz \vee yz$  es una expresión óptima en  $x$ ,  $y$ ,  $z$ . Ahora,  $xz \vee yz = (x \vee y)z$ , que se puede implementar con una puerta OR y una puerta AND, mientras que para implementar directamente  $xz \vee yz$  se requieren dos puertas AND para formar  $xz$  y  $yz$  y una puerta OR para terminar el trabajo. Lo que significa esto es que en situaciones prácticas nuestra definición de "óptimo" debe cambiar para hacer útil el hardware.

En algunos contextos es deseable tener todas las puertas del mismo tipo o a lo más de dos tipos. Resulta que podemos hacer todo exclusivamente con NAND o sólo con NOR. Determinar cuál de estos dos tipos de puertas es el que conviene usar dependerá de la tecnología particular de transistores que se vayan a utilizar. La tabla de la figura 4a muestra cómo escribir NOT, OR y AND en términos de NAND. La figura 4b muestra sus redes correspondientes. Esta tabla también responde la pregunta 12 de § 2.3 ya que



located, an alarm bell rings until the master switch is turned off. If the master switch is not turned on then the bell does not ring whatever the state of the sensors.




Design a switching circuit incorporating switches activated by the movement sensors and the master switch which will achieve this effect.

Do you think that the circuit you have designed is the simplest one? If not, try and design a simpler one.

### 9.5 Logic Networks

In this section we deal with 'logic gates'. These are electronic devices which may be viewed as the basic functional components of a digital computer. A **logic gate** is an electronic component, incorporated within a circuit, which operates on one or more inputs to produce one output. Each input and each output can take one of two values (normally low and high voltage) which are denoted by 0 and 1. Because of the 'two-value' nature of the input and output variables, a logic gate is an example of a **binary device**. It also falls in the category of **combinational devices** because the output value depends only on the input values. (This is in contrast to a **sequential device** where the output value is also determined by such factors as the time or the past history of the circuit.)

The three most important types of logic gates are the **AND-gate**, the **OR-gate** and the **NOT-gate** (or **inverter**). These gates are so named because of their association (which will become obvious) with the corresponding logical connective. We use  $x_i$  to represent the value of the input(s) to a gate and the variable  $z$  to represent its output. The following table summarizes the operation of each of the three gates.

	AND-gate	OR-gate	NOT-gate																																				
Circuit symbol																																							
Input/output table	<table> <tr> <th><math>x_1</math></th> <th><math>x_2</math></th> <th><math>z</math></th> </tr> <tr> <td>0</td> <td>0</td> <td>0</td> </tr> <tr> <td>0</td> <td>1</td> <td>0</td> </tr> <tr> <td>1</td> <td>0</td> <td>0</td> </tr> <tr> <td>1</td> <td>1</td> <td>1</td> </tr> </table>	$x_1$	$x_2$	$z$	0	0	0	0	1	0	1	0	0	1	1	1	<table> <tr> <th><math>x_1</math></th> <th><math>x_2</math></th> <th><math>z</math></th> </tr> <tr> <td>0</td> <td>0</td> <td>0</td> </tr> <tr> <td>0</td> <td>1</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>1</td> <td>1</td> </tr> </table>	$x_1$	$x_2$	$z$	0	0	0	0	1	1	1	0	1	1	1	1	<table> <tr> <th><math>x</math></th> <th><math>z</math></th> </tr> <tr> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> </tr> </table>	$x$	$z$	0	1	1	0
$x_1$	$x_2$	$z$																																					
0	0	0																																					
0	1	0																																					
1	0	0																																					
1	1	1																																					
$x_1$	$x_2$	$z$																																					
0	0	0																																					
0	1	1																																					
1	0	1																																					
1	1	1																																					
$x$	$z$																																						
0	1																																						
1	0																																						
Boolean expression	$z = x_1 x_2$	$z = x_1 \oplus x_2$	$z = \bar{x}$																																				

The AND-gate and the OR-gate each have two inputs and one output. For the AND-gate the output  $z$  has the value 1 only when the two input values are 1. The value of  $z$  is 0 otherwise. Viewing  $x_1$ ,  $x_2$  and  $z$  as variables whose domain is the underlying set  $\{0, 1\}$  of the Boolean algebra  $(\{0, 1\}, *, \oplus, \neg, 0, 1)$  we have  $z = x_1 x_2$ .

The OR-gate has output value 1 only if either or both of the input values are 1 and therefore the Boolean expression for  $z$  is given by  $z = x_1 \oplus x_2$ .

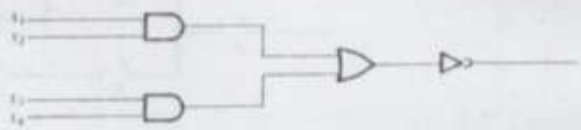
In contrast to the other two gates the NOT-gate has only one input. The gate has the effect of reversing the value of the input variable so that the Boolean expression for  $z$  is given by  $z = \bar{x}$ .

Comparison of the input/output tables with the truth tables for conjunction, disjunction and negation should make it clear why these gates are named as they are.

Within a circuit, a number of these gates may be linked together, the output from one gate acting as the input to one or more others. Such a circuit is termed a **logic network**. We can describe the output of the system of gates by a Boolean expression in terms of the various input variables.

#### Examples 9.10

1. Give the Boolean expression for the output of the following system of gates.

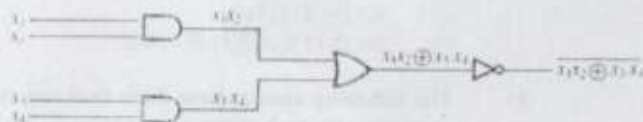


#### Solution

The output from the AND-gate having  $x_1$  and  $x_2$  as inputs is  $x_1 x_2$  and that from the AND-gate having  $x_3$  and  $x_4$  as inputs is  $x_3 x_4$ . The output from the OR-gate is therefore  $x_1 x_2 \oplus x_3 x_4$  and this is the input to the NOT-gate. The final output is therefore

$$\overline{x_1 x_2 \oplus x_3 x_4}.$$

We show these stages on the following diagram.



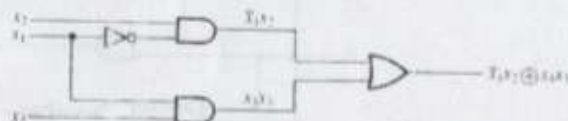
From the Boolean expression for the output, we can determine its value for each of the 16 possible sets of values of the input variables.

2. Design a system of logic gates with input variables  $x_1$ ,  $x_2$  and  $x_3$  which will produce an output defined by the Boolean expression  $\bar{x}_1x_2 \oplus x_1x_3$ .

*Solution*

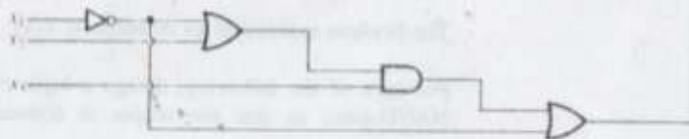
The final output can be achieved by an OR-gate whose input values are  $\bar{x}_1x_2$  and  $x_1x_3$ . The first of these expressions is the output of an AND-gate with inputs  $\bar{x}_1$  and  $x_2$ . The second is the output of an AND-gate with inputs  $x_1$  and  $x_3$ .

Thus we have the following diagram.



The input variable  $x_1$  branches, one branch passing through the NOT-gate and the other through the AND-gate having the variable  $x_3$  as its other input. The point at which the circuit branches is shown as a filled-in circle on the circuit diagram.

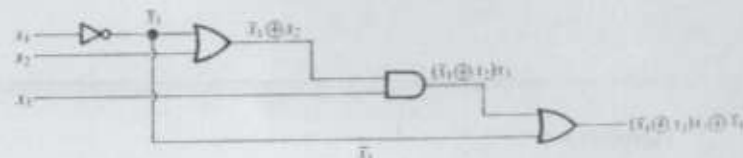
3. Determine the Boolean expression for the output of the following system of gates.



*Solution*

In this example the output from the inverter branches and is used as one input to the final OR-gate. Proceeding as in the first example we have the following

diagram.

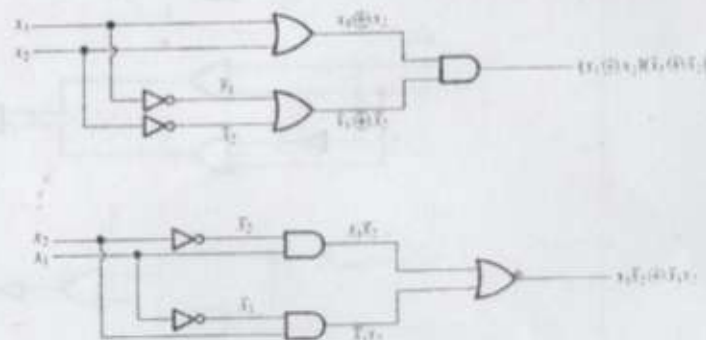


In all these examples the output is represented by a Boolean expression in terms of the input variables. We can, as usual, regard this Boolean expression as defining a function of the variables representing the states of the inputs. As we have seen, a particular function may be defined by a number of equivalent Boolean expressions. It follows therefore that, given any network of logic gates, there may be a number of equivalent networks. By 'equivalent' networks we mean that, given any set of values of the input variables, the output of any of these networks is the same.

For example, it is a simple matter to establish the equivalence of the two Boolean expressions.

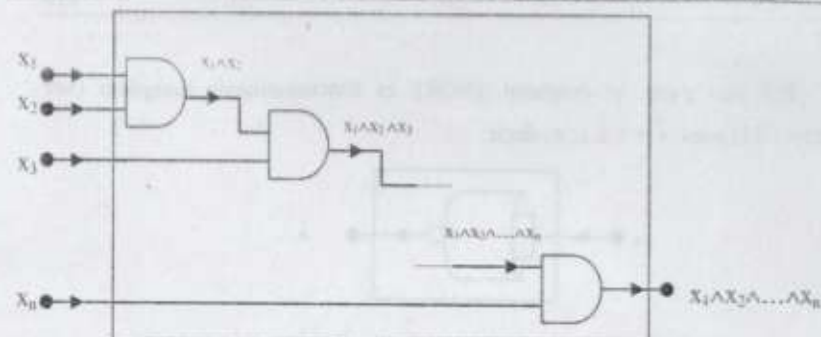
$$(x_1 \oplus x_2)(\bar{x}_1 \oplus \bar{x}_2) \quad \text{and} \quad x_1\bar{x}_2 \oplus \bar{x}_1x_2.$$

(Note that these are the conjunctive and disjunctive normal forms of the expression.) Thus the two logic networks whose outputs can be described by these expressions are equivalent. These are as follows.



Since the two Boolean expressions are equivalent, the output is the same for any given set of values of the input variables  $x_1$  and  $x_2$ .

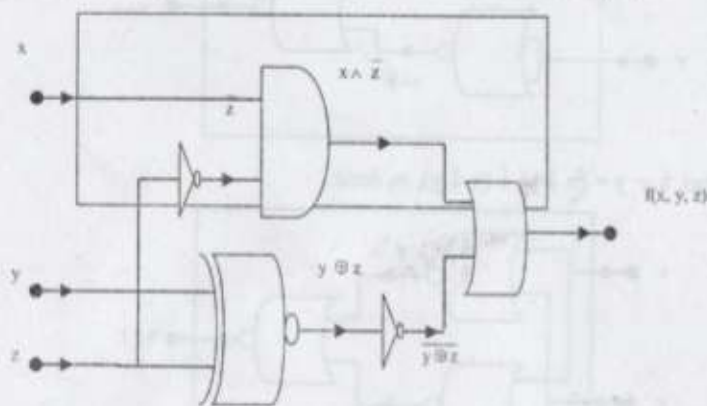
The fact that a number of different logic networks may be equivalent raises again the question of how we may determine, given a particular Boolean expression



**Ejercicio 3.49.** Representar gráficamente la función booleana  $f: (B_2)^3 \rightarrow B_2$  dada por la expresión  $f(x, y, z) = (x \wedge \bar{z}) \vee (y \oplus z)$ .

**Solución:**

La puerta lógica que representa la función  $f$  vendrá dada por:



**Ejercicio 3.50.** Demostrar la Proposición 3.14.

**Solución:**

Teniendo en cuenta la Proposición 3.12, como toda función booleana

elemental, distinta de la constante cero, se escribe como supremo de minterminos, es inmediato que toda puerta lógica, distinta de la constante cero, se puede sintetizar a partir de las puertas lógicas del conjunto {AND, OR, NOT}.

Además la constante cero de  $n$ -variables y  $n > 0$ , claramente también se puede sintetizar a partir del conjunto anterior de puertas lógicas, en efecto:

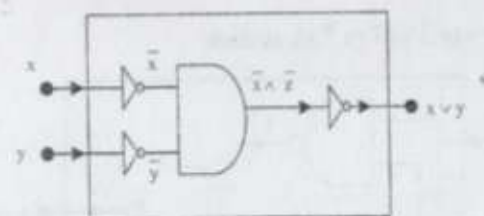
$$0 = x_1 \cdot \bar{x}_1 \cdot x_2 \cdot \dots \cdot x_n$$

**Ejercicio 3.51.** Demostrar la Proposición 3.15.

**Solución:**

Utilizando la Proposición 3.14, como para cualquier función booleana podemos obtener una síntesis en la que sólo aparecen los funcionales AND, OR y NOT, probaremos que cada uno de los conjuntos dados en la Proposición 3.15 es funcionalmente completo, viendo que cada uno de los funcionales del conjunto {AND, OR, NOT} se pueden obtener utilizando sólo los funcionales de dichos conjuntos.

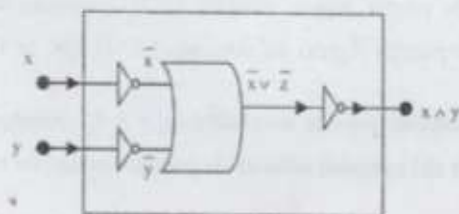
En primer lugar, el conjunto {AND, NOT} es funcionalmente completo pues por las leyes de De Morgan, podemos obtener una síntesis para el funcional OR mediante:  $x \vee y = \overline{\bar{x} \wedge \bar{y}}$ , es decir:



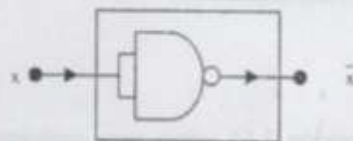
De forma análoga, el conjunto {OR, NOT} también es funcionalmente completo pues de nuevo por las leyes de De Morgan sabemos que  $x \wedge y = \overline{\bar{x} \vee \bar{y}}$ .



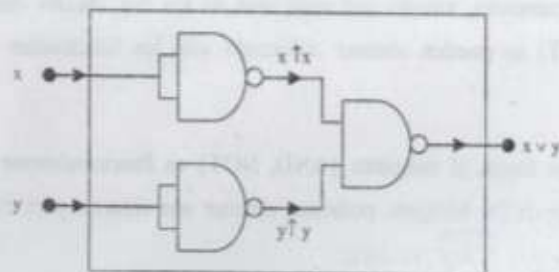
es decir, tenemos la siguiente síntesis para el funcional AND en función de los funcionales OR y NOT:



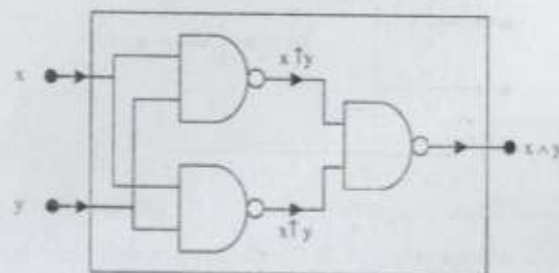
Por otra parte, el conjunto {NAND} es funcionalmente completo (ver ejercicio 1.34), ya que  $\bar{x} = x \uparrow x$ , es decir:



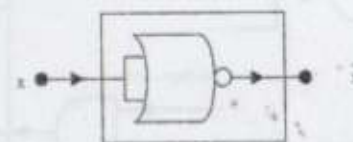
Además el operador OR se puede sintetizar mediante la expresión booleana  $x \vee y = (x \uparrow x) \uparrow (y \uparrow y)$ , esto es:



Y por último,  $x \wedge y = (x \uparrow y) \uparrow (x \uparrow y)$ , es decir:



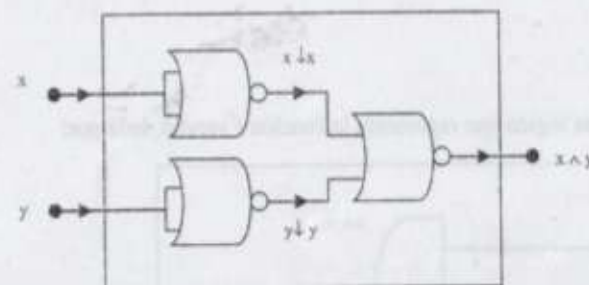
Por otra parte, el conjunto {NOR} es funcionalmente completo (ver ejercicio 1.33) pues  $\bar{x} = x \downarrow x$ , es decir:



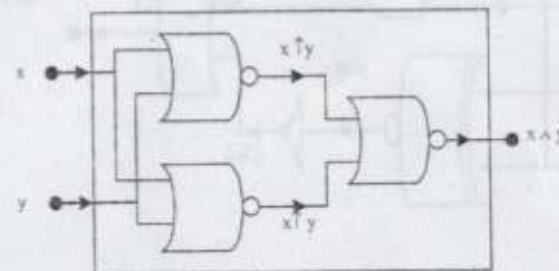
Además el operador OR se puede sintetizar mediante la expresión booleana:

$$x \wedge y = (x \downarrow x) \downarrow (y \downarrow y),$$

esto es:



Y por último,  $x \vee y = (x \downarrow y) \downarrow (x \downarrow y)$ , es decir:



**Ejercicio 3.52.** Sea  $f: (\mathbb{B}_2)^3 \rightarrow \mathbb{B}_2$  la función booleana definida por la expresión  $f(x, y, z) = (x \wedge \bar{z}) \vee (y \oplus z)$ . Encontrar expresiones equivalentes en las que sólo se usen los siguientes conjuntos de operadores:

a)  $\{\wedge, \vee, -\}$ .

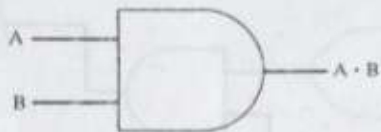
para obtener el área total. Esta área es llamada área de *unión*. La operación lógica *y* proporciona en realce la *intersección* de dos conjuntos. En la Fig. 6-1 está representada por el área del emparrillado. La Fig. 6-3 contiene varios diagramas de Venn y funciones booleanas asociadas.

Repaso

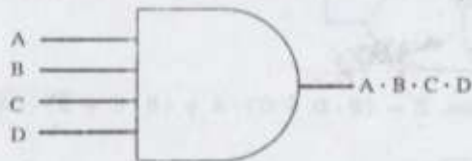
- 17. Dibuje un diagrama de Venn para B.
- 18. Dibuje un diagrama de Venn para  $\overline{A + B}$ .
- 19. Dibuje un diagrama de Venn para  $A + A \cdot B$ .
- 20. Dibuje un diagrama de Venn para  $A \cdot B + B$ .

6.6 Compuertas

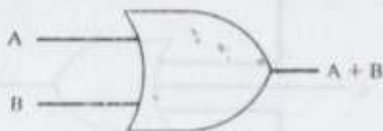
Muchos circuitos de computadoras electrónicas pueden ser construidos a partir de tres elementos de circuitos llamados, respectivamente, *compuerta y* (*and*), *compuerta o* (*or*) y *compuerta invertida*. La compuerta normal *y* tiene dos entradas A y B y una salida  $A \cdot B$ . La compuerta *y* para dos entradas es ilustrada a continuación.



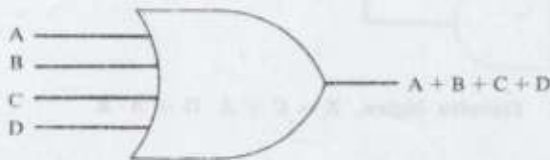
Una compuerta *y* con cuatro entradas podría ser representada como sigue:



La compuerta normal *o* tiene dos entradas A, B, y una salida  $A + B$ . La compuerta *o* para dos entradas es ilustrada abajo:



Una compuerta *o* con más de dos entradas puede aparecer como sigue:



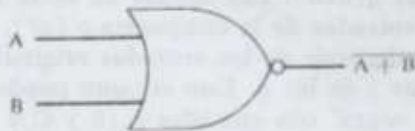
Una *invertida* tiene una entrada A y una salida  $\overline{A}$ . La invertida simplemente cambia una entrada a su estado opuesto —por ejemplo, si la entrada es un pulso, la salida será un NO pulso—. La invertida es ilustrada abajo.



Otros dos circuitos básicos son construidos conectando compuertas *y* y *o* con las invertidas. La compuerta *noy* (*nand*) está formada por una compuerta *y* (*and*) seguida de una *invertida*. Esto puede ser ilustrado de la siguiente manera.



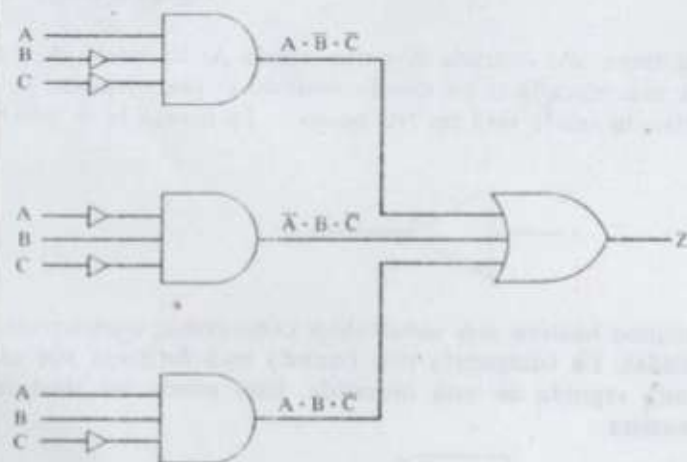
La compuerta *noo* (*nor*) está formada por una compuerta *o* (*or*) seguida de una *invertida*; lo cual puede ser ilustrado como sigue:



DEPTO. DE INGENIERIA

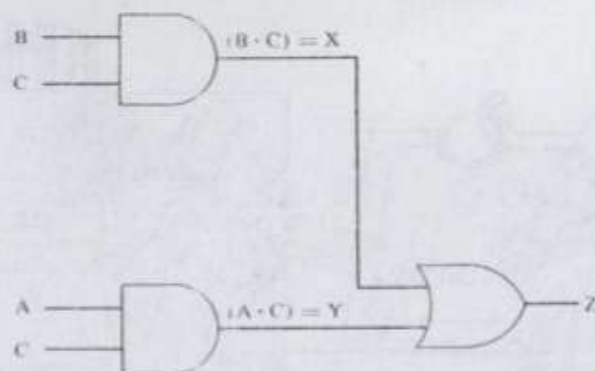
“¿No debería ser un o (or) exclusivo?”



FIG. 6-4. Circuito lógico,  $Z = A \cdot \bar{B} \cdot \bar{C} + \bar{A} \cdot \bar{B} \cdot \bar{C} + \bar{A} \cdot B \cdot \bar{C}$ 

Complicados circuitos de computadoras pueden ser construidos a través de las conexiones de las salidas de tales compuertas a las terminales de entrada de otras compuertas. Por ejemplo, considere el diagrama mostrado en la Fig. 6-4. Las salidas de cinco *invertidas* son conectadas a las entradas de tres compuertas *y* (*and*). Las salidas de estas tres compuertas *y* son conectadas a las entradas de la compuerta *o* (*or*). Por tanto, la salida  $Z$  de la compuerta *o* depende de las entradas originales  $A$ ,  $B$  y  $C$ , de las compuertas *invertidas* y de las *y*. Este circuito puede ser considerado como el de una "caja negra" con entradas  $A$ ,  $B$  y  $C$  y una salida  $Z$  tal que:  $Z = A \cdot \bar{B} \cdot \bar{C} + \bar{A} \cdot \bar{B} \cdot \bar{C} + \bar{A} \cdot B \cdot \bar{C}$ .

Dado un diagrama de circuito, uno puede fácilmente derivar la expresión booleana asociada. Por ejemplo, considere el circuito de la Fig.

FIG. 6-5. Circuito lógico,  $Z = B \cdot C + A \cdot C$ 

6-5. El problema es escribir  $Z$  en términos de  $A$ ,  $B$  y  $C$ , como está especificado en el diagrama del circuito. Esto puede ser realizado del modo que sigue:

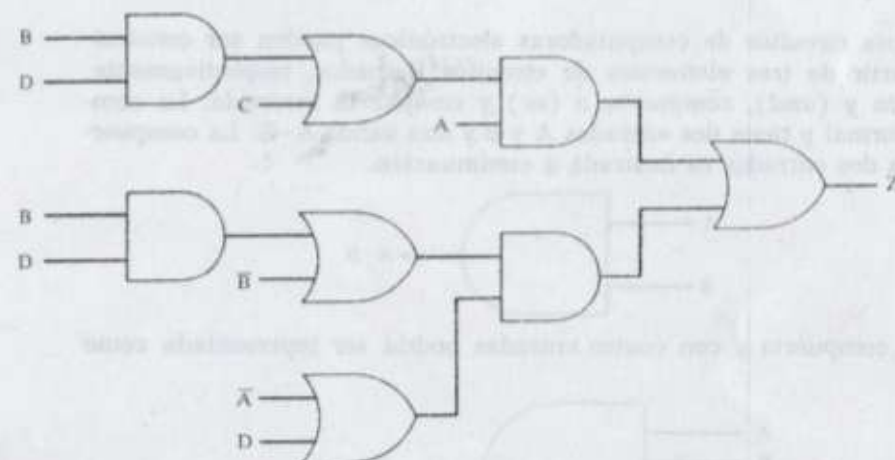
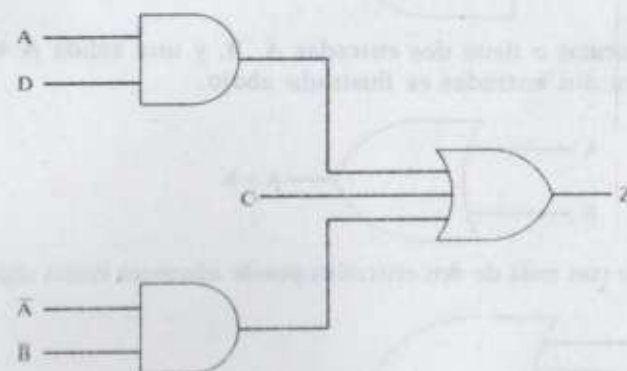
$$Z = X + Y, X = B \cdot C, Y = A \cdot C; \text{ Por lo tanto } Z = B \cdot C + A \cdot C.$$

Simplifiquemos ahora el circuito mostrado en la Fig. 6-6. La expresión para este circuito es  $Z = (B \cdot D + C) \cdot A + (B \cdot C + \bar{B}) \cdot (\bar{A} + D)$ . Rearreglando y simplificando obtenemos

$$\begin{aligned} Z &= A \cdot B \cdot D + A \cdot C + \bar{A} \cdot B \cdot C + \bar{B} \cdot C \cdot D + \bar{A} \cdot \bar{B} + \bar{B} \cdot D \\ &= A \cdot B \cdot D + C \cdot (A + \bar{A}) + C \cdot D + \bar{A} \cdot \bar{B} + \bar{B} \cdot D \\ &= C + D \cdot (\bar{B} + B \cdot A) + \bar{A} \cdot \bar{B} \end{aligned}$$

y finalmente

$$Z = C + A \cdot D + \bar{A} \cdot \bar{B}$$

FIG. 6-6. Circuito lógico,  $Z = (B \cdot D + C) \cdot A + (B \cdot C + \bar{B}) \cdot (\bar{A} + D)$ FIG. 6-7. Circuito lógico,  $Z = C + A \cdot D + \bar{A} \cdot \bar{B}$

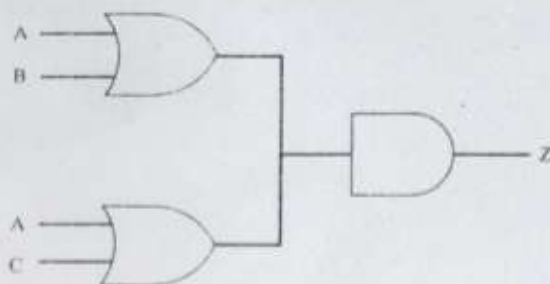


La forma simplificada de esta nueva malla es mostrada en la Fig. 6-7. A través de técnicas simplificadas de simplificación booleana, el circuito original puede ser reducido de 8 a 3 compuertas y de 16 a 7 elementos de entrada.

### Repaso

21. Dibuje una compuerta *y* de cinco entradas.
22. Dibuje una compuerta *o* de tres entradas.
23. Dibuje el diagrama de circuito de  

$$Z = A \cdot \bar{B} + \bar{A} \cdot B,$$
 usando compuertas *y*, compuertas *o* y compuertas *invertidas*.
24. Determine la función booleana del diagrama mostrado a continuación.



25. Dibuje el diagrama de circuito, usando compuertas *y*, *o* e *invertidas*, para:  

$$Z = \overline{A \cdot B + C}.$$
26. Dibuje el diagrama de circuito para el circuito de computadora tal que  

$$Z = (A + \bar{B}) \cdot (C + \bar{D}).$$

### □ Términos Clave

álgebra booleana

George S. Boole

Claude E. Shannon

elemento lógico

proposición

operación proposicional

producto lógico

suma lógica

operadores booleanos

hipótesis

tautología

tabla de verdad

circuito de computadora

circuitos en serie

circuitos en paralelo

diagrama de Venn

intersección

unión

compuertas invertidas

compuerta *y* (*and*)compuerta *o* (*or*)compuerta NOY (*nand*)