2. Álgebra booleana y puertas lógicas

A lo largo de los años muchas personas han intentado conocer los mecanismos del razonamiento humano desde distintas perspectivas. Una de ellas ha sido la matemática, en la que han ido apareciendo diferentes sistemas de representación numérica y álgebras que intentan emular ciertas operaciones de razonamiento. Seguramente la más conocida es el álgebra booleana puesto que su simplicidad permitió que rápidamente pudiera ser implementada por máquinas.

El álgebra booleana debe su nombre al filósofo y matemático inglés **George Boole** que en 1854 publica un trabajo de investigación titulado “An investigation of the laws of thought”. En dicho trabajo Boole propone un marco algebraico con un conjunto de postulados que opera con variables que tan sólo pueden tomar dos posibles valores, “cierto” o “falso”. De esta manera presenta unas operaciones matemáticas que emulan la manera en la que los humanos llegamos a ciertas decisiones a partir de proposiciones básicas, es decir, partiendo de ciertas premisas que son ciertas o falsas y proposiciones que relacionan dichas premisas con conclusiones, logra definir cómo se llega a afirmar si dichas conclusiones son ciertas o bien falsas.

El trabajo de George Boole ha llegado a ser de los más relevantes para el progreso tecnológico. El momento clave lo encontramos en la primera mitad del siglo XX y cerca de la Segunda Guerra Mundial, cuando en los Estados Unidos se estaba investigando sobre el desarrollo de máquinas que fueran capaces de realizar ciertas operaciones de cálculo para la toma de decisiones. Para conocer los detalles nos tendríamos que trasladar en el tiempo hasta los laboratorios de la Bell donde se estaba trabajando en máquinas de conmutación que tenían que operar con múltiples variables las cuales sólo podían tener dos posibles valores, o bien estaban conectadas o bien estaban desconectadas, y allí encontrar cómo uno de sus principales ingenieros, **Claude E. Shannon**, propone que las operaciones que se utilicen para realizar las distintas conmutaciones se basen en el álgebra de Boole. El trabajo de Shannon se publicó en 1938 bajo el título “A symbolic analysis of relay and switching circuits”.

Fueron pues estos desarrollos los que llevaron a la creación de sistemas electrónicos de conmutación y que terminaron dando lugar a los circuitos integrados o chips por medio de los cuales se inició el desarrollo de las primeras máquinas de cálculo y posteriormente a los primeros ordenadores. Así pues y por sorprendente que parezca, en el álgebra de Boole encontramos los principios de funcionamiento de los ordenadores y en general de cualquier sistema electrónico de proceso.

2.1. Álgebra booleana

El álgebra boolena o álgebra de Boole es un álgebra de variables binarias o bivaluadas, es decir variables que sólo pueden tomar dos valores, sobre las que mediante postulados se definen dos posibles operaciones: la suma booleana y el producto booleano. En posteriores temas veremos cómo se pueden realizar operaciones y sistemas cada vez más complejos a partir de estas dos operaciones elementales.

Consideraremos entonces que las variables booleanas sólo tomaran los valores 0 ó 1, si bien la definición del álgebra permitiría utilizar cualquier otro par de valores como por

ejemplo “falso” o “cierto”, “no” o “sí”, etc. Por consiguiente si a es una variable booleana entonces a puede valer 0 o bien 1, es decir, $a \in \{0,1\}$.

Asimismo la simbología que utilizaremos para representar las operaciones de suma booleana y producto booleano son los símbolos $+$ y $\times$, respectivamente, si bien dichos símbolos pueden variar según personas y costumbres, siendo los más habituales los siguientes:

- Suma: $+$ $\cup$ $\vee$ | OR
- Producto: $\times$ $\cap$ $\wedge$ & AND

Para simplificar la nomenclatura y teniendo en cuenta que sólo operaremos mediante el álgebra de Boole, a partir de este punto consideraremos equivalentes los nombres de variable booleana por los de variable binaria o simplemente variable. De la misma manera consideraremos equivalentes los nombres de suma booleana, suma binaria o simplemente suma; así como producto booleano, producto binario o simplemente producto.

Estas operaciones quedarán definidas a partir de los siguientes postulados o dicho de otra manera, ambas operaciones veremos que satisfacen los siguientes postulados.

Postulado 1. Tanto la suma como el producto son operaciones conmutativas, es decir, si a y b son dos variables booleanas entonces:

$$a + b = b + a$$
$$a \times b = b \times a$$

Postulado 2. Existen dos elementos neutros, el 0 para la suma y el 1 para el producto, de tal manera que si a es una variable booleana entonces:

$$a + 0 = a$$
$$a \times 1 = a$$

Postulado 3. Tanto la suma como el producto son operaciones distributivas respecto a la otra operación por lo que si tenemos tres variables booleanas a, b y c entonces:

$$a \times (b + c) = (a \times b) + (a \times c)$$
$$a + (b \times c) = (a + b) \times (a + c)$$

Observad que en este postulado aparece una de las principales diferencias entre las operaciones de suma y producto del álgebra de Boole respecto las operaciones de suma aritmética y producto aritmético con las que estamos más habituados a realizar operaciones de cálculo, siendo uno de los más difíciles de utilizar por su carácter novedoso.

Postulado 4. Cada variable booleana tiene su variable complementada de manera que si a es una variable booleana entonces su variable complementada se designa por $\bar{a}$ y satisface que:

$$a + \bar{a} = 1$$
$$a \times \bar{a} = 0$$

Si bien el complemento se define a partir de un postulado, a menudo se le considera como la tercera operación del álgebra de Boole puesto que permite invertir el valor de

una variable con la finalidad que ésta adopte su valor opuesto. En ciertos ámbitos al complemento también se le denomina inversión.

2.2. Funciones booleanas

Una función booleana es una función en la que una variable booleana depende del valor que adoptan otras variables booleanas relacionadas entre sí mediante operaciones booleanas (suma, producto, complemento).

Por ejemplo podemos tener una función f en la que una variable q dependa de otras variables $a, b, c, ...$, siendo todas ellas variables booleanas, es decir $q = f(a, b, c, ...) \mid q, a, b, c, ... \in \{0,1\}$. Es habitual que a la variable q se la denomine variable de salida mientras que al resto de variables $a, b, c, ...$, se las denomine variables de entrada.

Existen distintas formas para expresar o representar una función booleana, siendo las más habituales las siguientes:

- **Expresión algebraica** en la que describe la función mediante una expresión que relaciona las distintas variables con operaciones booleanas. Si conocemos el valor que adoptan las variables de entrada entonces podemos deducir el valor de la variable de salida realizando las distintas operaciones.

  Un ejemplo concreto de función booleana podría ser $q = a + b \times \bar{c}$ siendo q la variable de salida y a, b, c las variables de entrada. Supongamos por ejemplo que en un momento dado las variables de entrada presentan los siguientes valores $\{a, b, c\} = \{0,1,0\}$, entonces $q = 0 + 1 \times \bar{0} = 0 + 1 \times 1$ puesto que $\bar{0} = 1$ según el postulado 4; entonces $q = 0 + 1 \times 1 = 0 + 1$ puesto que $1 \times 1 = 1$ según el postulado 2; entonces $q = 0 + 1 = 1$ según el postulado 2.

  Un caso concreto de expresión algebraica es cuando ésta se considera como **expresión algebraica canónica**. En este caso la expresión ha de estar formada por 1) un producto de conjuntos de términos en los que en cada término aparecen todas las variables de entrada operadas por sumas o bien por 2) una suma de conjuntos de términos en los que en cada término aparecen todas las variables de entrada operadas por producto. Más adelante dedicaremos un apartado a estudiar cómo se obtienen, aunque tan sólo a modo de ejemplo podemos facilitar las dos posibles formas canónicas del ejemplo anterior, concretamente $q = a + b \times \bar{c} = (a + b + c) \times (a + b + \bar{c}) \times (a + \bar{b} + \bar{c})$ para el primer caso o bien tendremos que $q = a + b \times \bar{c} = (\bar{a} \times b \times \bar{c}) + (a \times \bar{b} \times \bar{c}) + (a \times \bar{b} \times c) + (a \times b \times \bar{c}) + (a \times b \times c)$ para el segundo. Si analizáramos el valor que adopta q para cada posible combinación de valores de entrada veríamos que todas las expresiones dan el mismo resultado. Son por consiguiente expresiones equivalentes.

  Otro caso concreto de expresión algebraica opuesto al anterior es cuando ésta se considera como **expresión algebraica simplificada**, habiendo obtenido ésta a partir del uso de distintos postulados y teoremas con la finalidad de obtener una expresión que opere de la misma manera pero con el menor número posible de operadores. De hecho la expresión $q = a + b \times \bar{c}$ ya es una expresión simplificada pues no se puede obtener otra expresión equivalente con menos

operadores. Más adelante veremos todo un conjunto de teoremas que pueden utilizarse para realizar las simplificaciones.

- **Tabla de verdad** en las que se expresa la función mediante una tabla que incluye todas las combinaciones posibles que pueden adoptar las variables de entrada así como el valor que adopta la variable de salida para cada una de dichas combinaciones. Estas tablas tienen pues dos partes, la de la izquierda donde se indican las combinaciones que pueden adoptar las variables de entrada y la de la derecha en la que se representa el valor que adopta la variable de salida para cada combinación.

  Si consideramos el ejemplo anterior en el que $q = a + b \times \bar{c}$, podemos expresar la misma función mediante la siguiente tabla de verdad en la que hemos calculado el valor que adopta q para cada una de las ocho combinaciones posibles de entrada. Observad que habitualmente las combinaciones se ordenan según el código binario natural aunque no sería necesario:

a	b	c	q
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Tabla 2.1. Ejemplo de tabla de verdad.

- **Tabla de Karnaugh**, siendo ésta otra representación en forma de tabla que facilita la búsqueda de expresiones algebraicas simplificadas. Las estudiaremos más adelante.

- **Diagrama de puertas lógicas** en el que se representan gráficamente las distintas operaciones mediante unos símbolos que denotan dichas operaciones. En el siguiente apartado veremos los diagramas de las operaciones más habituales, pero a modo de ejemplo podemos ver cómo quedaría la representación de la expresión anterior $q = a + b \times \bar{c}$:

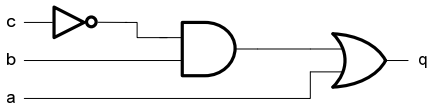

Figura 2.1. Ejemplo de diagrama de puertas lógicas. En él se observa como la variable c se aplica a una puerta que realiza la operación de complemento y su resultado, junto con la variable b, se aplica a una puerta que realiza la operación de producto. Finalmente el resultado de esta operación se aplica junto con la variable a a una puerta que realiza la operación de suma y obtiene el valor de la variable de salida q.

De hecho, estos diagramas representan una posible implementación real mediante dispositivos electrónicos, puesto que estas puertas se comercializan en forma de circuitos integrados, los cuales se conectan entre sí según el cableado que indica el mismo diagrama de puertas lógicas. De este modo

podemos aplicar una cierta tensión a los nodos de entrada, por ejemplo 5 voltios para aquellas variables que han de adoptar el valor booleano 1 o bien 0 voltios para aquellas variables que han de adoptar el valor booleano 0, con el fin de medir la tensión en el nodo de salida y deducir su valor booleano.

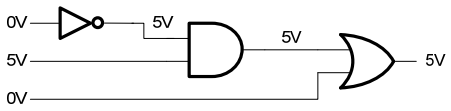

Figura 2.2. Ejemplo de diagrama de puertas lógicas en el que hemos aplicado unas ciertas tensiones en los nodos de entrada y podemos observar la tensión que mediríamos en los restantes nodos del circuito, incluido el de salida en el que al medir 5 voltios deduciríamos que la variable de salida toma el valor booleano 1 para esta combinación de entrada.

- **Lenguajes descriptores de hardware de alto nivel**, como el VHDL o el Verilog, que utilizan una sintaxis y construcciones similares a las de lenguajes de programación de alto nivel (como C o Java) para describir el comportamiento de sistemas digitales descritos mediante funciones booleanas. En realidad, estos lenguajes están pensados para describir sistemas digitales mucho más complejos que meras funciones booleanas, pero también permiten representar sistemas sencillos que implementan pocas o una única ecuación booleana. En el apartado 2.8 podemos observar, a modo de ejemplo, la descripción completa de un sistema que implementaría en lenguaje VHDL la función $q = a + b \times \bar{c}$.

  La utilidad de describir un sistema (sobre todo si es complejo) mediante un lenguaje de este tipo es que existen herramientas de ayuda al diseño electrónico que permiten, a partir de una descripción del sistema en estos lenguajes, sintetizar dicho sistema de forma automática (obtener la implementación con puertas lógicas) de todas las funciones que lo forman, simplificando mucho el trabajo del diseñador.

2.3. Operaciones booleanas

En el apartado anterior hemos visto distintas operaciones booleanas como son la suma, el producto y, a partir de los postulados del álgebra de Boole, también el complemento. Vamos a repasar el funcionamiento de todas ellas.

2.3.1. Operación AND

La operación AND es una operación de dos variables de entrada y una variable de salida que realiza la operación de producto booleano definida según el álgebra de Boole. La operación AND hace que la variable de salida siempre valga 0 excepto cuando las dos variables de entrada valen 1. Su representación más habitual en forma de expresión algebraica es $q = a \times b$ siendo a y b las variables de entrada y q la variable de salida.

Es una función que emula el conector lingüístico “y” mediante el cual una proposición que contiene dos variables unidas por este operador tan sólo es cierta cuando ambas variables son ciertas.

Su representación en forma de tabla de verdad muestra claramente su funcionamiento para cada combinación de valores que puedan adoptar las variables de entrada.

a	b	q
0	0	0
0	1	0
1	0	0
1	1	1

Tabla 2.2. Tabla de verdad de la función AND.

Finalmente su representación en un diagrama de puertas lógicas queda de la siguiente manera:

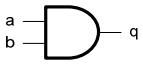

Figura 2.3. Representación según diagrama de puertas de la función AND.

De hecho esta operación se puede realizar de manera sucesiva a más de dos variables de entrada y su funcionamiento es siempre idéntico, es decir, tan sólo adopta el valor 1 cuando todas las variables de entrada valen 1. Así, si por ejemplo tenemos 3 variables de entrada $\{a, b, c\}$ sobre las que efectuamos la operación AND tendremos que $q = a \times b \times c = a \times (b \times c)$ en dónde q sólo adopta el valor 1 para la combinación $\{a, b, c\} = \{1,1,1\}$, siendo 0 para el resto de combinaciones.

Incluso existen circuitos comerciales que implementan el cálculo de la AND de más de 2 entradas, por lo que existen diagramas de puertas lógicas específicos para estos casos.

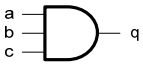

Figura 2.4. Representación según diagrama de puertas de la función AND de 3 entradas. A nivel comercial también se pueden encontrar circuitos integrados que implementan la función AND de 4 entradas.

2.3.2. Operación OR

La operación OR es una operación de dos variables de entrada y una variable de salida que realiza la operación de suma booleana definida según el álgebra de Boole. La operación OR hace que la variable de salida siempre valga 1 excepto cuando las dos variables de entrada valen 0. Su representación más habitual en forma de expresión algebraica es $q = a + b$ siendo a y b las variables de entrada y q la variable de salida.

Es una función que emula el conector lingüístico “o” mediante el cual una proposición que contiene dos variables unidas por este operador siempre es cierta excepto cuando ambas variables son falsas.

Su representación en forma de tabla de verdad muestra claramente su funcionamiento para cada combinación de valores que puedan adoptar las variables de entrada.

a	b	q
0	0	0
0	1	1
1	0	1
1	1	1

Tabla 2.3. Tabla de verdad de la función OR.

Finalmente su representación en un diagrama de puertas lógicas queda de la siguiente manera:

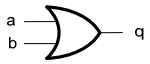

Figura 2.5. Representación según diagrama de puertas de la función OR.

De manera análoga a la función AND, la operación OR se puede realizar de manera sucesiva a más de dos variables de entrada y su funcionamiento es siempre idéntico, es decir, tan sólo adopta el valor 0 cuando todas las variables de entrada valen 0. Así, si por ejemplo tenemos 3 variables de entrada $\{a, b, c\}$ sobre las que efectuamos la operación OR tendremos que $q = a + b + c = a + (b + c)$ en donde q sólo adopta el valor 0 para la combinación $\{a, b, c\} = \{0,0,0\}$, siendo 1 para el resto de combinaciones.

Existen circuitos comerciales que implementan el cálculo de la OR de más de 2 entradas, por lo que existen diagramas de puertas lógicas específicos para estos casos.

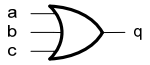

Figura 2.6. Representación según diagrama de puertas de la función OR de 3 entradas. También existen circuitos integrados que implementan la OR de 4 entradas.

2.3.3. Operación NOT

La operación NOT es una operación de una variable de entrada y una variable de salida que realiza la operación de complemento booleano definida según el postulado 4 del álgebra de Boole. La operación NOT asigna a la variable de salida el valor inverso al que presente la variable de entrada.

Su representación más habitual en forma de expresión algebraica es $q = \bar{a}$ siendo a la variable de entrada y q la variable de salida, si bien a veces se utiliza también el símbolo ! es decir $q = !\,a$.

Es una función que emula la palabra lingüística “no” mediante la cual cambia el valor de una proposición pasando de cierta a falsa y viceversa.

Su representación en forma de tabla de verdad muestra su funcionamiento:

a	q
0	1
1	0

Tabla 2.4. Tabla de verdad de la función NOT.

Finalmente su representación en un diagrama de puertas lógicas queda de la siguiente manera:

Figura 2.7. Representación según diagrama de puertas de la función NOT.

2.3.4. Operación NAND

La operación NAND es una combinación de dos operaciones anteriores. Concretamente surge de aplicar la operación NOT a la variable de salida de una operación AND. De esta manera la operación NAND funciona de manera complementada a la operación AND, por lo que la variable de salida siempre vale 1 excepto cuando las dos variables de entrada valen 0. Su representación más habitual en forma de expresión algebraica es $q = \overline{a \times b}$ siendo a y b las variables de entrada y q la variable de salida.

Su representación en forma de tabla de verdad muestra claramente su funcionamiento para cada combinación de valores que puedan adoptar las variables de entrada.

a	b	q
0	0	1
0	1	1
1	0	1
1	1	0

Tabla 2.5. Tabla de verdad de la función NAND.

Finalmente su representación en un diagrama de puertas lógicas queda de la siguiente manera:

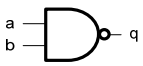

Figura 2.8. Representación según diagrama de puertas de la función NAND. Observamos como aparece un círculo a la salida de la puerta que es la manera habitual para indicar una operación de complemento sobre una variable, al igual que sucedía para el símbolo de la operación NOT.

De manera análoga a la función AND también existen circuitos integrados que implementan la función NAND para más de 2 variables de entrada, siendo el símbolo idéntico tan sólo que en el lado izquierdo aparecen tantas entradas como requiera la función.

2.3.5. Operación NOR

La operación NOR nuevamente es una combinación de dos operaciones anteriores y es fácil de deducir a partir de los casos vistos. Concretamente surge de aplicar la operación NOT a la variable de salida de una operación OR. De esta manera la operación NOR funciona de manera complementada a la operación OR, por lo que la variable de salida siempre vale 0 excepto cuando las dos variables de entrada valen 0. Su representación más habitual en forma de expresión algebraica es $q = \overline{a + b}$ siendo a y b las variables de entrada y q la variable de salida.

Su representación en forma de tabla de verdad muestra claramente su funcionamiento para cada combinación de valores que puedan adoptar las variables de entrada.

a	b	q
0	0	1
0	1	0
1	0	0
1	1	0

Tabla 2.6. Tabla de verdad de la función NOR.

Finalmente su representación en un diagrama de puertas lógicas sería el siguiente:

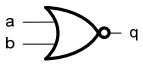

Figura 2.9. Representación según diagrama de puertas de la función NOR.

Al igual que en los casos anteriores podemos encontrar circuitos integrados que implementen una función NOR a más de 2 variables de entrada, indicándose entonces con un símbolo como el anterior pero añadiendo en el lado izquierdo tantas entradas como sea necesario.

2.3.6. Operación XOR

La operación XOR o también denominada OR-exclusiva es una función de dos variables de entrada y una variable de salida, la cual toma el valor 1 cuando las dos variables de entrada toman valores distintos entre sí, mientras que toma el valor 0 cuando las dos variables de entrada coinciden en su valor.

Al no ser una operación elemental definida directamente en el álgebra de Boole su representación ha de poder hacerse mediante los operadores habituales de suma y producto, quedando de la siguiente manera: $q = a \times \overline{b} + \overline{a} \times b$ siendo a y b las variables de entrada y q la variable de salida. En esta expresión podemos observar que cuando las variables de entrada son $\{a, b\} = \{0,0\}$ entonces $q = 0 \times \overline{0} + \overline{0} \times 0 = 0 \times 1 + 1 \times 0 = 0 + 0 = 0$. Lo mismo ocurre cuando $\{a, b\} = \{1,1\}$ pues entonces $q = 1 \times \overline{1} + \overline{1} \times 1 = 1 \times 0 + 0 \times 1 = 0 + 0 = 0$. En cambio cuando $\{a, b\} = \{1,0\}$ o bien $\{a, b\} = \{0,1\}$ observaremos que uno de los dos términos de la suma adopta el valor unitario y fuerza que la variable de salida también valga 1.

Su representación en forma de tabla de verdad muestra de manera mucho más clara su funcionamiento:

a	b	q
0	0	0
0	1	1
1	0	1
1	1	0

Tabla 2.7. Tabla de verdad de la función XOR.

Finalmente su representación en un diagrama de puertas lógicas sería el siguiente:

Figura 2.10. Representación según diagrama de puertas de la función XOR.

De hecho también podríamos haber representado la función XOR mediante unas puertas lógicas que implementen la expresión original $q = a \times \bar{b} + \bar{a} \times b$, quedando el diagrama que se muestra a continuación, si bien la manera habitual de representar la expresión es mediante el símbolo propio de la XOR. Veamos pues esta representación tan sólo a modo de ejemplo:

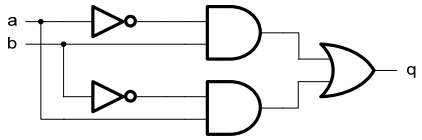

Figura 2.11. Representación según diagrama de puertas de la función XOR mediante puertas elementales de tipo AND, OR y NOT. Si bien esta implementación responde a la expresión que define la función XOR, su uso no es habitual pues ya existen de manera comercial circuitos integrados que implementan la función XOR y que se representan por su propio símbolo.

Si aplicáramos una puerta XOR de manera sucesiva a varias variables de entrada veríamos que la variable de salida adopta el valor 1 cuando tenemos un número impar de variables de entrada con valor 1 mientras que adopta el valor 0 cuando tenemos un número par de variables de entrada con valor 1. Por ejemplo, supongamos que tenemos 6 variables de entrada $\{a, b, c, d, e, f\}$ sobre las que aplicamos de manera sucesiva la función XOR, es decir $q = a \oplus (b \oplus (c \oplus (d \oplus (e \oplus f))))$. Supongamos que por ejemplo las variables de entrada adoptan la siguiente combinación de valores $\{a, b, c, d, e, f\} = \{0,1,1,0,1,0\}$, entonces nos encontraremos que la variable de salida sería 1 pues $q = 0 \oplus (1 \oplus (1 \oplus (0 \oplus (1 \oplus 0)))) = 0 \oplus (1 \oplus (1 \oplus (0 \oplus 1))) = 0 \oplus (1 \oplus (1 \oplus 1)) = 0 \oplus (1 \oplus 0) = 0 \oplus 1 = 1$, coincidiendo con el hecho que tenemos un número impar de variables de entrada que adoptan el valor unitario, concretamente tenemos 3 variables con dicho valor. Esta propiedad hace que se utilice como elemento para detección de paridad de un sistema al detectar el número par o impar de unos recibidos.

Comentar finalmente que para simplificar su expresión de manera algebraica se suele usar el símbolo $\oplus$ por lo que $q = a \times \bar{b} + \bar{a} \times b = a \oplus b$.

2.3.7. Operación NXOR

La operación NXOR es una función de dos variables de entrada y una variable de salida que funciona de manera complementada a la función XOR, por lo que la variable de salida toma el valor 0 cuando las dos variables de entrada toman valores distintos entre sí, mientras que toma el valor 1 cuando las dos variables de entrada coinciden en su valor. Su representación en forma de expresión algebraica quedaría $q = \overline{a \times \bar{b} + \bar{a} \times b} = \overline{a \oplus b}$ siendo a y b las variables de entrada y q la variable de salida.

Su representación en forma de tabla de verdad muestra de manera mucho más clara su funcionamiento:

a	b	q
0	0	1
0	1	0
1	0	0
1	1	1

Tabla 2.8. Tabla de verdad de la función NXOR.

Finalmente su representación en un diagrama de puertas lógicas sería el siguiente:

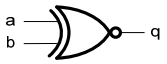

Figura 2.12. Representación según diagrama de puertas de la función NXOR.

2.3.8. Otras operaciones

Cualquier operación booleana puede expresarse mediante las operaciones vistas en los apartados anteriores. De hecho tan sólo serían necesarias las operaciones AND, OR y NOT, si bien el resto de operaciones también se consideran elementales por el hecho de simplificar ciertas expresiones y las correspondientes implementaciones.

En esta sección veremos algunos ejemplos de posibles funciones booleanas con el fin que seáis capaces vosotros mismos de deducir, a partir de estos ejemplos, cualquier otro caso.

Ejemplo 2.1

Partimos de la función $q = \overline{(a \times b + c)} \times \bar{d}$ y expresamos dicha función mediante una tabla de verdad y mediante diagrama de puertas lógicas.

a	b	c	d	q
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
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

Tabla 2.9. Tabla de verdad de la función $q = \overline{(a \times b + c)} \times \bar{d}$. Podemos observar que cuando $d = 1$ entonces su valor complementado hará que aplicado sobre el producto fuerce que la salida tome el valor 0. Para los otros casos tenemos que analizar el valor que adopta la expresión $a \times b + c$ y complementar su valor para conocer el valor de salida.

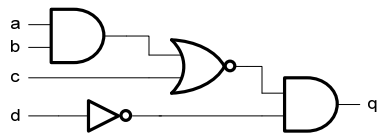

Figura 2.13. Representación según diagrama de puertas de la función $q = \overline{(a \times b + c)} \times \bar{d}$.

Ejemplo 2.2

Consideremos ahora la función $q = \overline{(a + b)} \oplus c \oplus d$ y aprovecharemos este ejemplo para ver cómo a menudo la tabla de verdad se expande con la finalidad de indicar ciertas operaciones que faciliten el cálculo de la función. Así pues:

a	b	c	d	$\overline{(a+b)}$	c	d	q
0	0	0	0	1	0	0	1
0	0	0	1	1	0	1	0
0	0	1	0	1	1	0	0
0	0	1	1	1	1	1	1
0	1	0	0	0	0	0	0
0	1	0	1	0	0	1	1
0	1	1	0	0	1	0	1
0	1	1	1	0	1	1	0
1	0	0	0	0	0	0	0
1	0	0	1	0	0	1	1
1	0	1	0	0	1	0	1
1	0	1	1	0	1	1	0
1	1	0	0	0	0	0	0
1	1	0	1	0	0	1	1
1	1	1	0	0	1	0	1
1	1	1	1	0	1	1	0

Tabla 2.10. Tabla de verdad de la función $q = \overline{(a+b)} \oplus c \oplus d$. Observamos como tan sólo tenemos que efectuar la XOR, es decir mirar la paridad impar, de 3 valores: la salida de una NOR junto con dos de las variables de entrada.

Su representación mediante diagrama de puertas lógicas sería el siguiente:

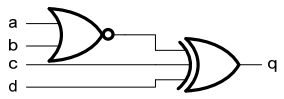

Figura 2.14. Representación según diagrama de puertas de la función $q = \overline{(a+b)} \oplus c \oplus d$ en la que podemos ver el símbolo de una puerta XOR con más de 2 entradas, análogo al símbolo que habíamos visto para el caso de puertas AND de más de 2 entradas y también para el caso de puertas OR de más de 2 entradas.

Ejemplo 2.3

Os propongo que representéis la tabla de verdad y el diagrama de puertas lógicas de la función $q = \overline{(a \times b) \oplus \overline{\overline{c + \bar{d}}}}$. Para la tabla de verdad os aconsejo que busquéis ciertos resultados intermedios, de manera similar a como lo hemos hecho en el ejemplo anterior.

2.4. Formas canónicas

Las formas canónicas son expresiones booleanas formadas a partir de términos que contienen todas las variables de entrada de la función booleana. Estos términos pueden ser de dos tipos diferentes y dar lugar a dos tipos de formas canónicas.

- Se denomina **minterm** a un término formado por el producto de todas las variables de entrada, pudiendo estar éstas en su forma directa o bien complementada. De esta manera se puede construir una forma canónica por medio de una suma de minterms.

  Supongamos como ejemplo que tenemos una función de 3 variables de entrada $\{a, b, c\}$. En este caso los minterms son términos del tipo $(a \times \bar{b} \times c)$ o bien $(\bar{a} \times \bar{b} \times c)$ o bien $(\bar{a} \times b \times \bar{c})$ en tanto que están formados por un producto de todas las variables de entrada. En cambio expresiones del tipo $(a \times c)$ o bien $\left(a \times \overline{b \times c}\right)$ o bien $\overline{(\bar{a} \times \bar{b} \times c)}$ no son minterms. En general, si tenemos N variables de entrada tenemos 2^Nposibles minterms, por lo que en este ejemplo tenemos hasta 8 minterms distintos.

  Los minterms son términos a partir de los cuales se puede expresar cualquier función booleana, mediante la suma de algunos de ellos. Así por ejemplo la función $q(a, b, c) = a + b \times \bar{c}$ es equivalente a la expresión $q = a + b \times \bar{c} = (\bar{a} \times b \times \bar{c}) + \left(a \times \bar{b} \times \bar{c}\right) + \left(a \times \bar{b} \times c\right) + (a \times b \times \bar{c}) + (a \times b \times c)$ siendo esta última la forma canónica de minterms de la expresión original. Más adelante veremos cómo se puede desarrollar una expresión booleana con la finalidad de obtener su forma canónica de minterms.

- Se denomina **maxterm** a un término formado por la suma de todas las variables de entrada, pudiendo estar éstas en su forma directa o bien complementada. De esta manera se puede construir una forma canónica por medio de un producto de maxterms.

  Supongamos como ejemplo que tenemos una función de 3 variables de entrada $\{a, b, c\}$. En este caso los maxterms son términos del tipo $(a + \bar{b} + c)$ o bien $(\bar{a} + \bar{b} + c)$ o bien $(\bar{a} + b + \bar{c})$ en tanto que están formados por una suma de todas las variables de entrada, en forma directa o en forma complementada. En cambio expresiones del tipo $(a + c)$ o bien $\left(a + \overline{b + c}\right)$ o bien $\overline{(\bar{a} + \bar{b} + c)}$ no son maxterms. Como en el caso de los minterms, si tenemos N variables de entrada tenemos 2^Nposibles maxterms, por lo que en este ejemplo tenemos hasta 8 maxterms distintos.

  Los maxterms son términos a partir de los cuales se puede expresar cualquier función booleana, mediante el producto de algunos de ellos. Así por ejemplo la función $q(a, b, c) = a + b \times \bar{c}$ es equivalente a la expresión $q = a + b \times \bar{c} = (a + b + c) \times (a + b + \bar{c}) \times \left(a + \bar{b} + \bar{c}\right)$, siendo esta última la forma canónica de maxterms de la expresión original. Más adelante veremos el desarrollo de una expresión booleana con la finalidad de obtener su forma canónica de maxterms.

2.4.1. Búsqueda de formas canónicas por medio de la evaluación de la función

Teorema. Toda expresión booleana se puede expresar en forma canónica, ya sea mediante producto de maxterms o bien mediante suma de minterms.

Veamos la demostración para el caso de minterms.

Dada una función $f(a,b,c,\dots)$ ésta se puede descomponer en $a \times f(1,b,c,\dots) + \bar{a} \times f(0,b,c,\dots)$. Esta descomposición la podemos realizar de manera iterativa para el resto de variables, es decir, la función anterior podría expresarse como $a \times [b \times f(1,1,c,\dots) + \bar{b} \times f(1,0,c,\dots)] + \bar{a} \times [b \times f(0,1,c,\dots) + \bar{b} \times f(0,0,c,\dots)]$ y también $a \times b \times [c \times f(1,1,1,\dots) + \bar{c} \times f(1,1,0,\dots)] + a \times \bar{b} \times [c \times f(1,0,1,\dots) + \bar{c} \times f(1,0,0,\dots)] + \bar{a} \times b \times [c \times f(0,1,1,\dots) + \bar{c} \times f(0,1,0,\dots)] + \bar{a} \times \bar{b} \times [c \times f(0,0,1,\dots) + \bar{c} \times f(0,0,0,\dots)]$ y así sucesivamente. Observad que al final nos quedarían todos los posibles minterms multiplicados cada uno de ellos por el valor que adopta la función ante una combinación concreta de valores de entrada, es decir, $a \times b \times c \times f(1,1,1,\dots) + a \times b \times \bar{c} \times f(1,1,0,\dots) + a \times \bar{b} \times c \times f(1,0,1,\dots) + a \times \bar{b} \times \bar{c} \times f(1,0,0,\dots) + \bar{a} \times b \times c \times f(0,1,1,\dots) + \bar{a} \times b \times \bar{c} \times f(0,1,0,\dots) + \bar{a} \times \bar{b} \times c \times f(0,0,1,\dots) + \bar{a} \times \bar{b} \times \bar{c} \times f(0,0,0,\dots) +$ ..., quedando pues al final una suma de aquellos minterms para los que la función toma el valor 1.

Dicho de otra manera, cada combinación de valores de entrada tiene un minterm asociado y la forma canónica de la función es la suma de aquellos minterms por los que la combinación de valores de entrada vale 1.

Ejemplo 2.4

Anteriormente habíamos comentado que la función $q(a,b,c) = a + b \times \bar{c}$ es equivalente a la forma canónica $q = (\bar{a} \times b \times \bar{c}) + (a \times \bar{b} \times \bar{c}) + (a \times \bar{b} \times c) + (a \times b \times \bar{c}) + (a \times b \times c)$.

Veamos cómo habríamos podido llegar a esta expresión a partir de evaluar la función original para cada posible combinación de valores de entrada y asociando a cada combinación un minterm.

Combinaciones			Minterm asociado a la combinación	Valor de salida	Conclusión del minterm asociado y la forma canónica
a	b	c		q	
0	0	0	$(\bar{a} \times \bar{b} \times \bar{c})$	0	No pertenece a la forma
0	0	1	$(\bar{a} \times \bar{b} \times c)$	0	No pertenece a la forma
0	1	0	$(\bar{a} \times b \times \bar{c})$	1	Pertenece a la forma
0	1	1	$(\bar{a} \times b \times c)$	0	No pertenece a la forma
1	0	0	$(a \times \bar{b} \times \bar{c})$	1	Pertenece a la forma
1	0	1	$(a \times \bar{b} \times c)$	1	Pertenece a la forma
1	1	0	$(a \times b \times \bar{c})$	1	Pertenece a la forma
1	1	1	$(a \times b \times c)$	1	Pertenece a la forma

Tabla 2.11. Tabla de verdad de la función booleana que permite analizar los minterms que constituyen la forma canónica de dicha función. Los minterms asociados a cada combinación se construyen escribiendo la variable en forma directa si para esa combinación la variable en cuestión toma el valor 1 o bien en forma complementada si para esa combinación toma el valor 0.

La demostración de este teorema para el caso de maxterms podría hacerse de manera similar. No obstante existe un teorema que presentaremos en próximas sesiones, llamado teorema de dualidad, mediante el cual la demostración es inmediata, por lo que pasaremos directamente a estudiar cómo se puede hallar la forma canónica de maxterms a partir de una expresión booleana y asumiremos por cierta la demostración del teorema.

La búsqueda de una forma canónica con maxterms de una cierta expresión booleana es similar al método expuesto para los minterms. Es este caso nuevamente tendremos un maxterm asociado a cada posible combinación de valores de entrada, de tal manera que en la forma canónica sólo aparecerán aquellos maxterms que se correspondan a aquellas combinaciones de entrada para las que la expresión booleana vale 0.

Ejemplo 2.5

Hemos comentado que la expresión en forma canónica de maxterms de la función $q(a,b,c) = a + b \times \bar{c}$ es $q = (a+b+c) \times (a+b+\bar{c}) \times (a+\bar{b}+\bar{c})$. De manera dual al ejemplo anterior, veamos cómo habríamos podido llegar a esta expresión a partir de evaluar la función original para cada posible combinación de valores de entrada y asociando a cada combinación un maxterm.

Combinaciones			Maxterm asociado a la combinación	Valor de salida	Conclusión del maxterm asociado y la forma canónica
a	b	c		q	
0	0	0	$(a+b+c)$	0	Pertenece a la forma
0	0	1	$(a+b+\bar{c})$	0	Pertenece a la forma
0	1	0	$(a+\bar{b}+c)$	1	No pertenece a la forma
0	1	1	$(a+\bar{b}+\bar{c})$	0	Pertenece a la forma
1	0	0	$(\bar{a}+b+c)$	1	No pertenece a la forma
1	0	1	$(\bar{a}+b+\bar{c})$	1	No pertenece a la forma
1	1	0	$(\bar{a}+\bar{b}+c)$	1	No pertenece a la forma
1	1	1	$(\bar{a}+\bar{b}+\bar{c})$	1	No pertenece a la forma

Tabla 2.12. Tabla de verdad de la función booleana que permite analizar los maxterms que constituyen la forma canónica de dicha función. Los maxterms asociados a cada combinación se construyen escribiendo la variable en forma directa si para esa combinación la variable en cuestión toma el valor 0 o bien en forma complementada si para esa combinación toma el valor 1.

A partir de los dos últimos ejemplos habéis visto una forma sistemática de hallar las formas canónicas de una cierta función. Observad que sólo tenemos que evaluar la función correspondiente para encontrar aquellas combinaciones para las que la función toma el valor 1 a fin de utilizar los minterms asociados para construir la forma canónica de minterms, o bien encontrar aquellas combinaciones para las que la función toma el valor 0 a fin de utilizar los maxterms asociados para construir la forma canónica de maxterms.

Esto responde al hecho que los minterms son términos que sólo valen 1 para una cierta combinación de entrada, de tal manera que sumados a otros minterms forzaran que para esa combinación concreta la función resultante valga 1. En cambio los maxterms son términos que sólo valen 0 para otras combinaciones de entrada, de tal manera que haciendo el producto con otros maxterms forzaran que para esa combinación concreta la función resultante valga 0.

Ejemplo 2.6

Veamos la obtención de las formas canónicas de minterms y de maxterms para la función $q(a,b,c) = \overline{(a \times \bar{b} + c)}$.

Por lo que hemos comentado básicamente tenemos que evaluar la función para determinar las combinaciones para las que la función vale 1 o bien 0, a fin de determinar los minterms o maxterms correspondientes.

Combinaciones			Salida	Conclusión y término asociado a la forma canónica de minterms o maxterms
a	b	c	q	
0	0	0	1	Pertenece a la forma de minterms con $(\bar{a} \times \bar{b} \times \bar{c})$
0	0	1	0	Pertenece a la forma de maxterms con $(a + b + \bar{c})$
0	1	0	1	Pertenece a la forma de minterms con $(\bar{a} \times b \times \bar{c})$
0	1	1	0	Pertenece a la forma de maxterms con $(a + \bar{b} + \bar{c})$
1	0	0	0	Pertenece a la forma de maxterms con $(\bar{a} + b + c)$
1	0	1	0	Pertenece a la forma de maxterms con $(\bar{a} + b + \bar{c})$
1	1	0	1	Pertenece a la forma de minterms con $(a \times b \times \bar{c})$
1	1	1	0	Pertenece a la forma de maxterms con $(\bar{a} + \bar{b} + \bar{c})$

Tabla 2.13. Tabla de verdad de la función booleana que permite buscar los minterms o maxterms que constituyen las posibles formas canónicas de dicha función.

Las formas canónicas son pues las siguientes. Para el caso de minterms tendremos que $q = (\bar{a} \times \bar{b} \times \bar{c}) + (\bar{a} \times b \times \bar{c}) + (a \times b \times \bar{c})$ mientras que para el caso de maxterms tendremos que $q = (a + b + \bar{c}) \times (a + \bar{b} + \bar{c}) \times (\bar{a} + b + c) \times (\bar{a} + b + \bar{c}) \times (\bar{a} + \bar{b} + \bar{c})$.

2.4.2. Búsqueda de formas canónicas por medio de la manipulación algebraica de la función

Llegado a este punto ya hemos aprendido cómo encontrar las formas canónicas por medio de evaluar la expresión original para cada posible combinación de entrada, pero también es posible hallar estas formas desarrollando las funciones originales hasta obtener la forma deseada. Se trata pues de un proceso de manipulado algebraico de la función original.

De manera sintética los pasos a realizar son los siguientes:

- Para una forma canónica de minterms:

  1. Desarrollamos la función hasta obtener una suma de términos que incluyan productos de variables. No obstante en este punto puede ser que los términos hallados no sean propiamente minterms puesto que no incluyan todas las variables de entrada.
  2. Se convierten los términos en minterms incluyendo las variables que falten hasta completar la aparición de todas las variables de entrada. Para mantener inalterado el valor de la función tendremos que añadir la suma de cada variable y la variable complementada puesto que de esta manera su valor siempre será 1 y no altera el producto.
  3. Se expande la expresión y se eliminan los minterms repetidos.

- Para una forma canónica de maxterms:

  1. Desarrollamos la función hasta obtener un producto de términos que incluyan sumas de variables. No obstante en este punto puede ser que los términos hallados no sean propiamente maxterms puesto que no incluyan todas las variables de entrada.

2. Se convierten los términos en maxterms incluyendo las variables que falten hasta completar la aparición de todas las variables de entrada. Para mantener inalterado el valor de la función tendremos que añadir el producto de cada variable y la variable complementada puesto que de esta manera su valor siempre será 0 y no altera la suma.
3. Se expande la expresión y se eliminan los maxterms repetidos.

Comentar que el primer punto de este procedimiento no siempre es trivial y a menudo tendríamos que recurrir a teoremas que aún no hemos estudiado para su obtención. No obstante si las funciones son relativamente simples entonces el desarrollo algebraico de las mismas para convertirlas en una forma canónica es bastante rápido, a menudo utilizando directamente los postulados del álgebra de Boole.

Ejemplo 2.7

Veamos la búsqueda de las formas canónicas de la función $q(a,b,c) = a + b \times \bar{c}$ por medio de manipulaciones algebraicas.

Para el caso de la forma canónica de minterms primero tenemos que obtener una expresión que sea una suma de términos. En este caso podemos diferenciar dos términos sumados en la expresión original que indicamos entre paréntesis $q = (a) + (b \times \bar{c})$. Para cada término tenemos que incluir variables hasta que en cada término aparezcan todas las variables de entrada. Para ello, en el caso que falte alguna variable tendremos que añadir la suma de dichas variables y las variables complementadas para no alterar la función original, es decir $q = \left(a \times (b + \bar{b}) \times (c + \bar{c})\right) + ((a + \bar{a}) \times b \times \bar{c})$. Finalmente expandimos esta expresión y eliminamos los minterms repetidos, es decir $q = (a \times b \times c) + \left(a \times \bar{b} \times c\right) + (a \times b \times \bar{c}) + \left(a \times \bar{b} \times \bar{c}\right) + (a \times b \times \bar{c}) + (\bar{a} \times b \times \bar{c}) = (a \times b \times c) + \left(a \times \bar{b} \times c\right) + (a \times b \times \bar{c}) + \left(a \times \bar{b} \times \bar{c}\right) + +(\bar{a} \times b \times \bar{c})$.

De manera análoga para el caso de la forma canónica de maxterms primero obtenemos una expresión que sea un producto de términos. En este caso podemos diferenciar dos términos en la expresión original sobre los que se aplica el producto, a partir del postulado de propiedad distributiva del álgebra de Boole, y que indicamos entre paréntesis $q = (a + b) \times (a + \bar{c})$. Para cada término tenemos que incluir variables hasta que en cada término aparezcan todas las variables de entrada. Para ello, en el caso que falte alguna variable tendremos que añadir el producto de dichas variables y las variables complementadas para no alterar la función original, es decir $q = (a + b + c \times \bar{c}) \times (a + b \times \bar{b} + \bar{c})$. Finalmente expandimos esta expresión y eliminamos los maxterms repetidos, es decir $q = (a + b + c) \times (a + b + \bar{c}) \times (a + b + \bar{c}) \times \left(a + \bar{b} + \bar{c}\right) = (a + b + c) \times (a + b + \bar{c}) \times \left(a + \bar{b} + \bar{c}\right)$.

Ejemplo 2.8

Veamos la búsqueda de las formas canónicas de la función $q(a,b,c) = b \times (a + \bar{c}) + b \times c$ por medio de manipulaciones algebraicas.

Para la forma canónica de minterms haríamos los siguientes desarrollos: $q = a \times b + b \times \bar{c} + b \times c = a \times b \times (c + \bar{c}) + (a + \bar{a}) \times b \times \bar{c} + (a + \bar{a}) \times b \times c = a \times b \times c + a \times b \times \bar{c} + a \times b \times \bar{c} + \bar{a} \times b \times \bar{c} + a \times b \times c + \bar{a} \times b \times c = a \times b \times c + a \times b \times \bar{c} + \bar{a} \times b \times \bar{c} + \bar{a} \times b \times c$.

Mientras que para la forma canónica de maxterms haríamos los siguientes pasos: $q = (b) \times (a + \bar{c} + c) = (b) \times (a + 1) = (b) = (a \times \bar{a} + b + c \times \bar{c}) = (a + b + c) \times (a + b + \bar{c}) \times (\bar{a} + b + c) \times (\bar{a} + b + \bar{c})$.

Una comprobación que podemos hacer para verificar la coherencia del resultado que hemos hallado es buscar las combinaciones de entrada asociadas a cada minterm y maxterm para ver si cubrimos todas las combinaciones con unos y otros términos. En el caso de este ejemplo vemos que los minterms están asociados a las combinaciones 7, 6, 2 y 3, mientras que los maxterms están asociados a las combinaciones 0,1, 4 y 5.

Ejemplo 2.9

Dada la función $q(a, b, c) = (a + \bar{c}) \times b + \bar{a} \times c$, os propongo que encontréis sus formas canónicas:

- Por medio de evaluar la función original para cada posible combinación y hallando los minterms y maxterms asociados a cada combinación.
- Por medio de la manipulación algebraica de la función e intentando hallar algebraicamente tanto la forma canónica de minterms como la de maxterms.

2.4.3. Formas canónicas abreviadas

Hemos visto cómo hallar las formas canónicas por medio de la evaluación de la función original o bien por medio de la manipulación algebraica de la misma. En este apartado no pretendemos presentar un tercer método sino simplemente explicar una forma abreviada de expresar las formas canónicas a fin que éstas se puedan interpretar de manera más rápida.

La idea es aprovechar el hecho que cada posible minterm o maxterm está asociado a una combinación concreta de valores de entrada para indicar esta combinación en vez del minterm o maxterm. La manera de indicar la combinación será usar el valor decimal correspondiente a dicha combinación binaria, asumiendo como si esta estuviera expresada en binario natural.

Por ejemplo si tenemos la siguiente forma canónica de minterms $q = (\bar{a} \times b \times \bar{c}) + (a \times \bar{b} \times \bar{c}) + (a \times \bar{b} \times c) + (a \times b \times \bar{c}) + (a \times b \times c)$ y vemos que los minterms asociados corresponden a las combinaciones 2, 4, 5, 6 y 7 entonces la forma canónica abreviada la denotaremos por $q(a, b, c) = \sum_3(2,4,5,6,7)$. El hecho de denotar el nombre de las variables en la expresión responde a la necesidad de indicar en qué orden tenemos que considerarlas en el momento de interpretar los valores decimales. Así mismo el subíndice que aparece en el sumatorio expresa el número de variables de entrada.

Por ejemplo la misma función en forma canónica de maxterms es $q = (a + b + c) \times (a + b + \bar{c}) \times (a + \bar{b} + \bar{c})$ y en este caso su expresión abreviada sería la siguiente al ver que los maxterms que aparecen están asociados a las combinaciones 0, 1 y 3, $q(a, b, c) = \prod_3(0,1,3)$. En este caso en vez de utilizar el símbolo de sumatorio se utiliza el de productorio.

Podemos comprobar que las combinaciones que aparecen para la forma canónica de minterms son distintas de las que aparecen para la forma canónica de maxterms y que todas ellas cubren el total de las combinaciones de entrada

Ejemplo 2.10

Trataremos de hallar las formas canónicas abreviadas de la función $q(a, b, c) = a \times (b + \bar{a} \times \bar{c})$.

Primero tenemos que hallar cualquiera de las dos posibles formas canónicas, o bien la de minterms o bien la de maxterms. Para hallarla además podemos elegir si utilizar el método de evaluación de la función original para todas las combinaciones o bien si utilizamos el método de manipulación algebraica de dicha función.

Si optamos por este último método vemos que $q = a \times b + a \times \bar{a} \times \bar{c} = a \times b = a \times b \times (c + \bar{c}) = a \times b \times c + a \times b \times \bar{c} = \sum_3(7,6) = \prod_3(0,1,2,3,4,5)$.

Ejemplo 2.11

Dadas las formas canónicas halladas en el ejemplo 2.9 para la función $q = (a + \bar{c}) \times b + \bar{a} \times c$, reescribid dichas formas canónicas en su forma abreviada.

2.5. Teoremas booleanos

Partiendo de los postulados del álgebra de Boole podemos demostrar un conjunto de teoremas que pueden ser útiles para reescribir funciones booleanas. Recordemos brevemente los postulados que utilizaremos para demostrar los teoremas.

Postulado 1.
$a + b = b + a$
$a \times b = b \times a$

Postulado 2.
$a + 0 = a$
$a \times 1 = a$

Postulado 3.
$a \times (b + c) = (a \times b) + (a \times c)$
$a + (b \times c) = (a + b) \times (a + c)$

Postulado 4.
$a + \bar{a} = 1$
$a \times \bar{a} = 0$

A continuación vamos a ver cuáles son estos teoremas y también sus demostraciones. Cuando hagamos referencia al uso de un postulado para dichas demostraciones

indicaremos mediante P1 si usamos el primer postulado, P2 si usamos el segundo, P3 para el tercer postulado y P4 para el último.

Teorema 1. Elemento nulo:

$$a + 1 = 1 + a = 1$$
$$a \times 0 = 0 \times a = 0$$

Veamos la demostración usando en este orden P4, P2, P3, P4, P2 y P1:

$$1 = a + \bar{a} = a + \bar{a} \times 1 = (a + \bar{a}) \times (a + 1) = 1 \times (a + 1) = a + 1 = 1 + a$$
$$0 = a \times \bar{a} = a \times (\bar{a} + 0) = (a \times \bar{a}) + (a \times 0) = 0 + (a \times 0) = a \times 0 = 0 \times a$$

Teorema 2. Propiedad de idempotencia:

$$a + a = a$$
$$a \times a = a$$

Veamos la demostración usando en este orden P2, P4, P3, P4 y P2:

$$a = a + 0 = a + a \times \bar{a} = (a + a) \times (a + \bar{a}) = (a + a) \times 1 = a + a$$
$$a = a \times 1 = a \times (a + \bar{a}) = (a \times a) + (a \times \bar{a}) = (a \times a) + 0 = a \times a$$

Teorema 3. Ley de absorción:

$$a + (a \times b) = a$$
$$a \times (a + b) = a$$

Veamos la demostración usando en este orden P2, Teorema 1, P3, y P2:

$$a = 1 \times a = (1 + b) \times a = (1 \times a) + (b \times a) = a + (b \times a)$$
$$a = 0 + a = (0 \times b) + a = (0 + a) \times (b + a) = a \times (b + a)$$

La ley de absorción es extensible a funciones booleanas y no tan solo a variables. Es decir, si consideramos dos funciones booleanas f y g entonces se satisface que $f + (f \times g) = f$ y también que $f \times (f + g) = f$. Esta propiedad a menudo se utiliza en la simplificación algebraica de expresiones booleanas.

Teorema 4. Principio de dualidad según el cual una identidad del álgebra de Boole se mantiene si se intercambian entre sí las operaciones $+$ y $\times$, así como los valores lógicos 0 y 1.

$$a + 1 = 1 \Leftrightarrow a \times 0 = 0$$

En las demostraciones de los teoremas anteriores podemos ver cómo se cumple el principio de dualidad en tanto que para cada teorema existen dos versiones distintas. El principio de dualidad permite pues realizar demostraciones o cálculos booleanos mediante un formato de expresión concreto y posteriormente extender dicha demostración o cálculo a otro formato equivalente. Desde el punto de vista práctico veremos cómo se puede utilizar para hallar diversas implementaciones equivalentes.

Teorema 5. Propiedad asociativa:

$$a + (b + c) = (a + b) + c = a + b + c$$
$$a \times (b \times c) = (a \times b) \times c = a \times b \times c$$

Se demostraría a partir de la tabla de verdad de las funciones, observando que por todas y cada una de las combinaciones de las tres variables implicadas, se satisface que las expresiones dan el mismo valor.

Teorema 6. Ley de convolución por la que para todo elemento a se cumplirá que el complemento de su variable complementada da nuevamente a:

$$a = \bar{\bar{a}}$$

Se satisface por tabla de verdad, es decir cuando $a = 0$ entonces $\bar{a} = 1$ y por consiguiente $\bar{\bar{a}} = 0 = a$. Del mismo modo cuando $a = 1$ entonces $\bar{a} = 0$ y por consiguiente $\bar{\bar{a}} = 1 = a$.

Teorema 7. Leyes de De Morgan:

$$\overline{a + b + c + d + \cdots} = \bar{a} \times \bar{b} \times \bar{c} \times \bar{d} \times \ldots$$
$$\overline{a \times b \times c \times d \times \ldots} = \bar{a} + \bar{b} + \bar{c} + \bar{d} + \cdots$$

Veamos la demostración para la primera de las dos expresiones puesto que la segunda queda demostrada por el principio de dualidad que hemos indicado en el cuarto teorema,

Partiremos del caso de tener tan sólo dos variables en las que según la ley de De Morgan han de satisfacer que $\overline{a + b} = \bar{a} \times \bar{b}$ y renombraremos ambos lados de la igualdad por $f = \overline{a + b}$ y $g = \bar{a} \times \bar{b}$. Si la igualdad es cierta se tienen que satisfacer las siguientes dos propiedades, $\bar{f} \times g = 0$ y también que $\bar{f} + g = 1$. Hagamos pues la verificación de la primera propiedad, es decir $\overline{\overline{a + b}} \times \bar{a} \times \bar{b} = (a + b) \times \bar{a} \times \bar{b} = a \times \bar{a} \times \bar{b} + b \times \bar{a} \times \bar{b} = 0 + 0 = 0$. Para la segunda propiedad tenemos que $\overline{\overline{a + b}} + \bar{a} \times \bar{b} = (a + b) + \bar{a} \times \bar{b} = (a + b + \bar{a}) \times \left(a + b + \bar{b}\right) = (1 + b) \times (1 + a) = 1 \times 1 = 1$.

En el caso de más variables seguiremos un proceso iterativo que empezaremos por definir $p = b + c + d + \cdots$ con lo que según la ley de De Morgan se cumplirá que $\overline{a + p} = \bar{a} \times \bar{p} \Longrightarrow \overline{a + b + c + d + \cdots} = \bar{a} \times \overline{b + c + d + \cdots}$. Ahora definamos $q = c + d + \cdots$ por lo que $\overline{a + b + c + d + \cdots} = \bar{a} \times \overline{b + q} = .\bar{a} \times \bar{b} \times \bar{q} = \bar{a} \times \bar{b} \times \overline{c + d + \cdots}$. Siguiendo el mismo proceso podemos nombrar $r = d + \cdots$ por lo que $\overline{a + b + c + d + \cdots} = \bar{a} \times \bar{b} \times \overline{c + r} = \bar{a} \times \bar{b} \times \bar{c} \times \bar{r} = \bar{a} \times \bar{b} \times \bar{c} \times \overline{d + \cdots}$. Y el proceso se podría repetir indefinidamente hasta la igualdad inicial.

De Morgan fue contemporáneo a George Boole y desarrolló diversos estudios y libros como “Formal Logic” al mismo tiempo que el precursor del álgebra de Boole.

Más adelante veremos la aplicación de las leyes de De Morgan en la búsqueda de expresiones que permitan implementar cualquier función booleana sólo mediante el uso de puertas lógicas de tipo NAND o bien de tipo NOR. No obstante primero veremos algunos ejemplos en el uso de los anteriores teoremas puesto que su uso no resulta trivial, más todo lo contrario..

Ejemplo 2.12

Intentemos simplificar mediante el uso de los postulados y teoremas del álgebra de Boole la función $f(a,b,c,d) = \overline{(a+c)\times(a+\bar{c})\times\overline{b\times d+\bar{b}\times d}}$.

Primero podemos mirar de simplificar el término complementado que incluye las variables b y d. Para ello vemos que $\overline{b\times d+\bar{b}\times d} = \overline{d\times(b+\bar{b})} = \overline{d\times 1} = \bar{d}$. De tal manera que la función inicial equivale a $\overline{(a+c)\times(a+\bar{c})\times\bar{d}}$.

Igualmente podemos expandir los dos términos que incluyen las variables a y c con lo que veremos lo siguiente $(a+c)\times(a+\bar{c}) = a\times a + a\times c + a\times\bar{c} + c\times\bar{c} = a + a\times c + a\times\bar{c} + 0 = a\times(1+c+\bar{c}) = a\times(1+1) = a\times 1 = a$.

De este modo tenemos que $f(a,b,c,d) = \overline{a\times\bar{d}} = \bar{a} + d$.

Ejemplo 2.13

Imaginemos que nos piden que demostremos que la expresión $f(a,b,c,d,e) = \overline{\overline{\bar{b}\times c}+a} + \overline{a+b+\bar{c}+d\times\bar{e}}$ es equivalente a $f(a,b,c,d,e) = \bar{a}\times\bar{b}\times c$..

Al fin y al cabo nos están pidiendo simplificar la función original hasta obtener la función equivalente simplificada. Primero podemos utilizar De Morgan para agrupar los dos términos que suma la función original, es decir $\overline{\overline{\bar{b}\times c}+a} + \overline{a+b+\bar{c}+d\times\bar{e}} = \overline{\left(\overline{\bar{b}\times c}+a\right)\times(a+b+\bar{c}+d\times\bar{e})}$.

Ahora podemos aplicar De Morgan sobre las variables b y $\bar{c}$ del segundo término y veremos que obtenemos una expresión que también es común al primer término, es decir $\overline{\left(\overline{\bar{b}\times c}+a\right)\times(a+b+\bar{c}+d\times\bar{e})} = \overline{\left(\overline{\bar{b}\times c}+a\right)\times\left(a+\overline{\bar{b}\times c}+d\times\bar{e}\right)}$ de tal manera que entonces $\overline{\left(\overline{\bar{b}\times c}+a\right)\times\left(a+\overline{\bar{b}\times c}+d\times\bar{e}\right)} = \overline{(a+\overline{\bar{b}\times c})\times(1+d\times\bar{e})}$ por lo que finalmente $f(a,b,c,d,e) = \overline{a+\overline{\bar{b}\times c}}$ y por De Morgan $f(a,b,c,d,e) = \bar{a}\times\bar{b}\times c$.

2.6. Implementación de funciones con puertas lógicas de tipo NAND o NOR

En esta sección veremos que toda función booleana se puede implementar con puertas de tipo NAND o NOR. Para lograrlo tenemos que mirar de reescribir la función booleana hasta obtener un formato en el que aparezca solamente una suma de variables, para hacer una implementación con puertas NOR, o bien solamente un producto de variables, para hacer una implementación en este caso con puertas NAND. Dichos cambios se suele hacer mediante un uso intensivo de las leyes de De Morgan.

De manera sintetizada las pautas que en general se siguen son las siguientes para una implementación con puertas NAND:

- Obtener una función simplificada de la función original.
- Si la operación más externa es una suma entonces se complementa dos veces la función simplificada para no alterar su valor y se aprovecha uno de los dos complementos para aplicar la ley de De Morgan que permita convertir la suma en un producto.
- Si en la anterior expresión continúan habiendo sumas lógicas entonces se tiene que aplicar el complemento dos veces a los correspondientes términos y operar como en el paso anterior para convertir las sumas en productos.
- El proceso se va repitiendo hasta que sólo queden productos.

Las pautas anteriores básicamente lo que aconsejan es trabajar con funciones lo más simplificadas posibles e ir utilizando paulatinamente De Morgan para convertir cualquier operador de suma que tengamos en un operador de producto, empezando en general por los operadores más externos de la función. En cualquier caso sólo son pautas y a menudo se pueden hallar otras vías o alternativas para hallar la función final que pueda implementarse con puertas NAND.

Ejemplo 2.14

Supongamos que queremos implementar la función $f(a,b,c) = \bar{a} \times b + c$ sólo con puertas NAND.

Empezamos complementado dos veces el operador suma y los términos que abarca quedando $\overline{\overline{\bar{a} \times b + c}}$. De esta manera podemos utilizar De Morgan para cambiar el operador suma por un producto, recordando que entonces también se han de complementar los términos que quedan a ambos lados de la suma, es decir $\overline{\overline{\bar{a} \times b} \times \bar{c}}$. Esta expresión se puede implementar directamente sólo con puertas NAND como se puede ver en la siguiente figura.

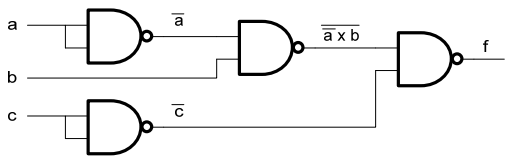

Figura 2.15. Implementación de la función $f(a,b,c) = \bar{a} \times b + c$ sólo con puertas NAND. Es interesante de observar cómo se implementa un inversor con una puerta NAND sobre la que se aplica la variable a complementar a ambas entradas. Observad que si la variable vale 1 entonces la NAND dará $\overline{1 \times 1} = \bar{1} = 0$ mientras que si la variable vale 0 entonces la NAND dará $\overline{0 \times 0} = \bar{0} = 1$.

Ejemplo 2.15

Miremos de implementar la función $f(a,b,c,d) = a + \overline{\overline{b \times c} + d}$ mediante puertas NAND.

Como que esta expresión ya parte de un formato bastante simplificado, empezaremos mirando de convertir la suma más externa en un producto. Para ello vamos a complementar dos veces dicha suma y los términos que abarca, es decir, $\overline{\overline{a + \overline{b \times c} + d}}$ y aplicamos De Morgan quedando entonces $\overline{\bar{a} \times \overline{\overline{b \times c} + d}}$. Ahora podemos aplicar De Morgan sobre el operador suma que aún queda, observando que en este caso no es necesario poner un doble complemento puesto que el operador ya se halla complementado. Así pues nos queda $\overline{\bar{a} \times \overline{\overline{b \times c} \times \bar{d}}} = \overline{\bar{a} \times b \times c \times \bar{d}}$ la cual podemos implementar con puertas NAND.

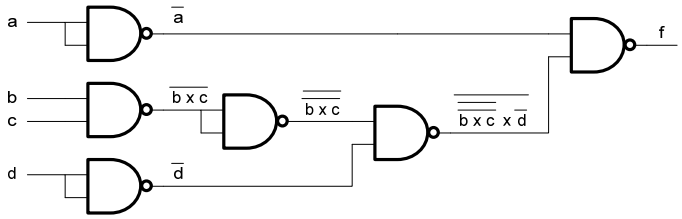

Figura 2.16. Implementación de $f(a,b,c,d) = a + \overline{\overline{b \times c} + d}$ sólo con puertas NAND.

Ejemplo 2.16

Os propongo que implementéis la función $f(a,b,c,d) = \overline{\bar{a} \times \bar{b} + \bar{c} \times (\bar{d} \times (a+b))}$ mediante puertas NAND. En un principio necesitaréis 10 puertas de este tipo.

Las pautas anteriores para la implementación con puertas NAND son las mismas que para una implementación con puertas NOR, simplemente cambiando la palabra suma por la palabra producto y viceversa. Es decir:

- Obtener una función simplificada de la función original.
- Si la operación más externa es un producto entonces se complementa dos veces la función simplificada para no alterar su valor y se aprovecha uno de los dos complementos para aplicar la ley de De Morgan que permita convertir el producto en una suma.
- Si en la anterior expresión continúan habiendo productos lógicos entonces se tiene que aplicar el complemento dos veces a los correspondientes términos y operar como en el paso anterior para convertir los productos en sumas.
- El proceso se va repitiendo hasta que sólo queden sumas.

Ejemplo 2.17

Implementemos la función $f(a,b,c) = a \times b + b \times \bar{c}$ sólo mediante puertas NOR.

En este caso complementamos dos veces directamente los términos que aparecen a ambos lados del operador suma en tanto que sólo tenemos que mirar de cambiar los operadores producto que aparecen en estos términos por operadores suma. Así pues quedaría $\overline{\overline{a \times b}} + \overline{\overline{b \times \bar{c}}}$ y aplicando De Morgan a estos términos de los extremos resulta

la expresión $\overline{\bar{a}+\bar{b}}+\overline{\bar{b}+\bar{c}} = \overline{\bar{a}+\bar{b}}+\overline{\bar{b}+c}$ que se puede implementar sólo con puertas NOR.

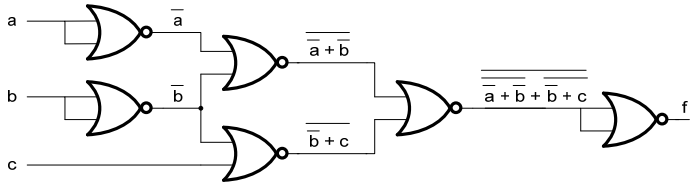

Figura 2.17. Implementación de $f(a,b,c) = a \times b + b \times \bar{c}$ mediante puertas NOR.

Ejemplo 2.18

Miremos de implementar ahora la función $f(a,b,c,d) = \overline{\bar{a} \times \bar{b} + \bar{c} \times (\bar{d} \times (a+b))}$ sólo mediante puertas NOR.

Empecemos complementando dos veces el operador producto del primer término y aplicando De Morgan, es decir $\overline{\overline{\bar{a} \times \bar{b}} + \bar{c} \times (\bar{d} \times (a+b))} = \overline{\overline{(a+b)} + \bar{c} \times (\bar{d} \times (a+b))}$. Ahora podemos hacer lo mismo para el operador suma del siguiente término empezando por el operador más externo, es decir $\overline{\overline{(a+b)} + \overline{\bar{c} \times (\bar{d} \times (a+b))}} = \overline{\overline{(a+b)} + \overline{\overline{c + (\bar{d} \times (a+b))}}}$. Finalmente podemos aplicar De Morgan al último operador producto que nos queda. En este caso no hace falta complementar dos veces pues este término ya está complementado, por lo que resulta $\overline{\overline{(a+b)} + \overline{c+d+\overline{a+b}}}$.

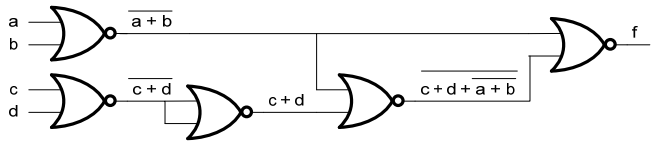

Figura 2.18. Implementación de $f(a,b,c,d) = \overline{\bar{a} \times \bar{b} + \bar{c} \times (\bar{d} \times (a+b))}$ sólo mediante puertas NOR.

Ejemplo 2.19

Os propongo que implementéis la función $f(a,b,c) = a + \overline{\overline{b \times (\bar{a}+c)} + \bar{b}}$ mediante puertas NOR.

Seguramente no deja de ser curioso el hecho que cualquier expresión booleana pueda implementarse sólo con el uso de estas puertas lógicas, sobre todo si pensamos en la cantidad de circuitos digitales que nos rodean. ¿Podéis llegar a pensar por ejemplo que un procesador de cálculo esté implementado de esta manera? El hecho es que si bien en teoría cualquier circuito digital basado en álgebra de Boole ha de poder implementarse mediante estas puertas, esto no asegura que la correspondiente implementación sea la más óptima posible, más todo lo contrario para circuitos

complejos. En este caso se hace uso de otros elementos que facilitan un diseño más rápido, modular, etc. tal y como veremos en próximas sesiones.

2.7. Diseño e implementación de sistemas mediante puertas lógicas

Las puertas lógicas no son sólo una manera más de representar funciones booleanas sino que además son la manera real de implementarlas. Así existen en el mercado circuitos integrados que implementan puertas de tipo NAND, NOR, AND, OR, XOR, etc. de tal manera que su interconexión da lugar a funciones de mayor complejidad.

A modo de ejemplo podemos ver en la siguiente figura el esquema interno de un par de circuitos integrados. Estos circuitos integrados habitualmente tienen dos terminales de entrada o salida (se suelen identificar por su nombre en inglés, “pin”) donde se aplica la alimentación del circuito, en este caso +5V y 0V, mediante la cual el circuito integrado hará que las distintas puertas lógicas que incorpora funcionen correctamente. La implementación física del resto de entradas y salidas también se realiza mediante otros terminales (“pins”) de tal modo que se denomina “pin-out” a la descripción del conjunto de terminales del circuito integrado.

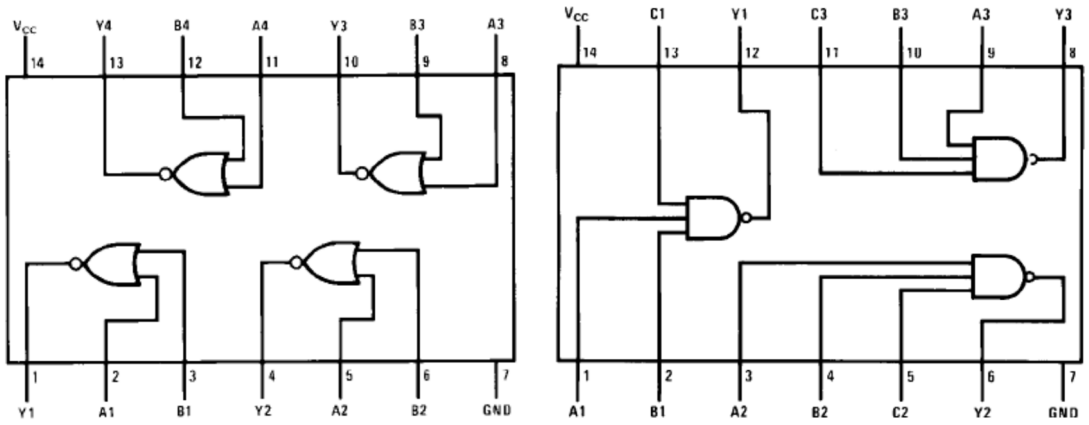

Figura 2.19. “Pin-out” de los circuitos integrados 7402 y 7410 que implementan respectivamente, 4 puertas de tipo NOR y 3 puertas de tipo NAND. Los circuitos se alimentan aplicando +5V entre los terminales 14 y 7. De esta manera entre los terminales 1,2 y 3 del 7402 tenemos las entradas y la salida de una NOR; entre los terminales 4,5 y 6 las entradas y la salida de otra; y así sucesivamente; o bien entre los terminales 3, 4, 5, y 6 del 7410 tenemos las entradas y la salida de una NAND; y así sucesivamente.

El modo de operar es muy simple. Si queremos aplicar un 1 lógico a la entrada de una puerta aplicaremos +5V al terminal correspondiente, mientras que si queremos aplicar un 0 lógico a la entrada entonces aplicaremos 0V. De la misma manera si la puerta tiene que dar un 1 lógico en su salida entonces sacará +5V por el terminal de salida de esa puerta o bien sacará 0V en caso de tener que dar un 0 lógico.

A modo de ejemplo podemos ver en la siguiente figura cómo se han conectado estos circuitos integrados para implementar una función booleana concreta.

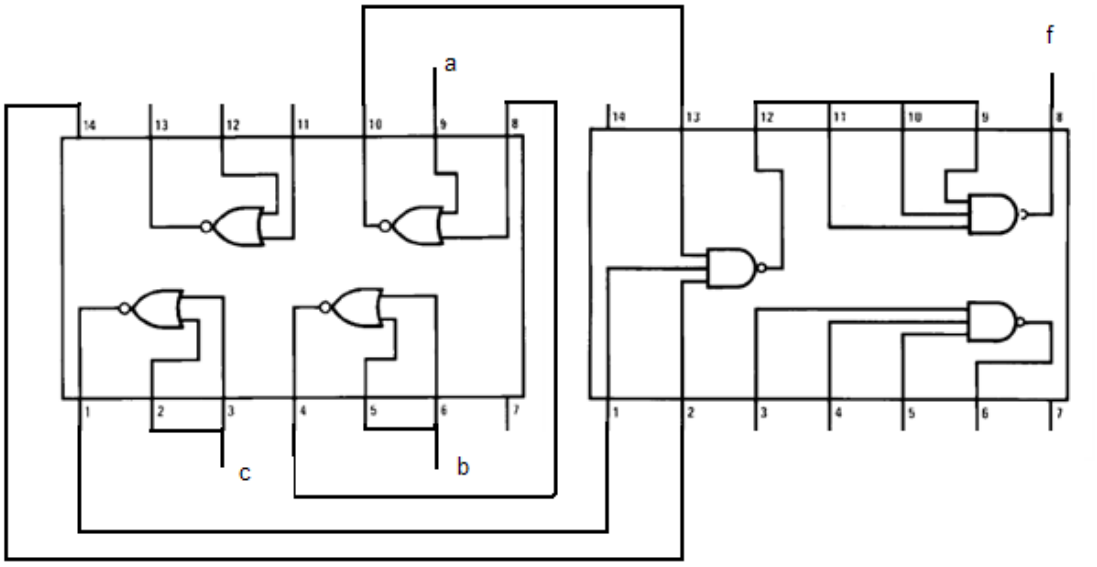

Figura 2.20. Interconexiones entre un circuito integrado 7402 y un 7410 para implementar la función $f(a,b,c) = \overline{a + \overline{b}} \times \overline{c}$. Podemos observar cómo se han utilizado un par de puertas NOR del 7402 para implementar los términos $\overline{b}$ y $\overline{c}$. También es interesante observar cómo una de las entradas de una NAND se ha conectado directamente a alimentación mediante la interconexión del pin 2 del 7410 al pin 14 del 7402, pues tan sólo necesitamos dos de las tres entradas para la operación producto. Finalmente observad también el uso de una NAND para implementar un último complemento. A modo de ejemplo, si aplicásemos +5V en el pin 9 del 7402, +5V en el pin 6 del 7402 y 0V en el pin 3 del 7402 entonces tendremos +5V en el pin 1 del 7402 que se aplica al pin 1 del 7410; tendremos +5V en el pin 4 del 7402 que se aplican al pin 8 del mismo integrado y también tendremos 0V en el pin 10 del 7402 que se aplicarán al pin 13 del 7410. Finalmente esta combinación dará lugar a 0V en el pin 8 del 7410 que indicará un 0 lógico como resultado de combinación.

Existen múltiples circuitos integrados que implementan puertas lógicas, así como muchas otras funciones booleanas que veremos en próximas sesiones. No obstante en la siguiente tabla podéis ver algunos de los circuitos integrados más utilizados cuando se quieren implementar funciones con las puertas lógicas que hasta el momento hemos estudiado.

Dispositivo	Descripción
7400	4 puertas NAND de 2 entradas cada una
7402	4 puertas NOR de 2 entradas cada una
7404	6 puertas NOT de 1 entrada cada una
7408	4 puertas AND de 2 entradas cada una
7432	4 puertas OR de 2 entradas cada una
7486	4 puertas XOR de 2 entradas cada una
7410	3 puertas NAND de 3 entradas cada una
7427	3 puertas NOR de 3 entradas cada una
7411	3 puertas AND de 3 entradas cada una

Tabla 2.14. Circuitos integrados de la serie 74 que implementan las principales puertas lógicas. La serie 74 es una familia muy extensa de circuitos integrados que implementan distintas funciones lógicas. Dicha serie fue iniciada por la empresa Texas Instruments para integrar puertas lógicas mediante una tecnología llamada TTL con finalidades comerciales, a partir del éxito que había tenido la serie 54 que inició en 1964 para usos militares. Su uso fue tan popular que se convirtió en un estándar *de facto* en la nomenclatura de circuitos integrados y hoy en día la usan múltiples fabricantes.

Ejemplo 2.20

Veamos cómo interconectar los terminales del circuito integrado 7400 que se muestra a continuación para implementar la función $f(a,b) = a \times \overline{b}$.

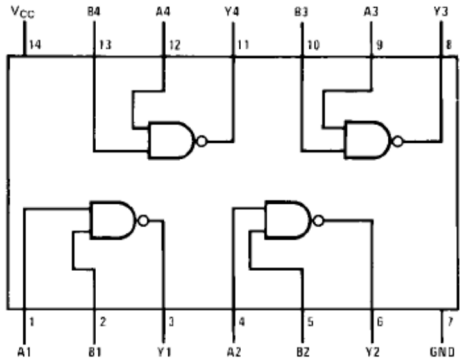

Figura 2.21 Circuito integrado 7400.

En este caso tendremos que utilizar una NAND para implementar el complemento de b, es decir $\overline{b}$. Luego utilizaremos otra NAND para implementar propiamente la operación producto, es decir $\overline{a \times \overline{b}}$. Finalmente necesitaremos otra NAND para implementar el complemento del producto anterior y así obtener la función solicitada, es decir $\overline{\overline{a \times \overline{b}}} = a \times \overline{b}$.

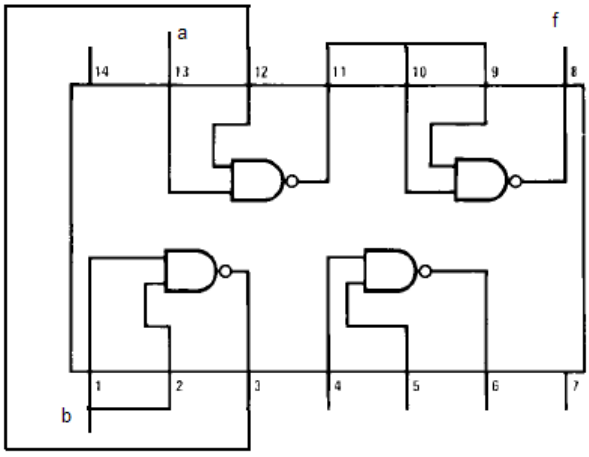

Figura 2.22 Interconexión de un 7400 para implementar la función $f(a,b) = a \times \overline{b}$.

Ejemplo 2.21

Tratemos ahora de descubrir qué función implementa la siguiente interconexión real entre dos circuitos integrados, un 7404 que contiene 6 puertas inversoras y un 7408 que contiene 4 puertas AND.

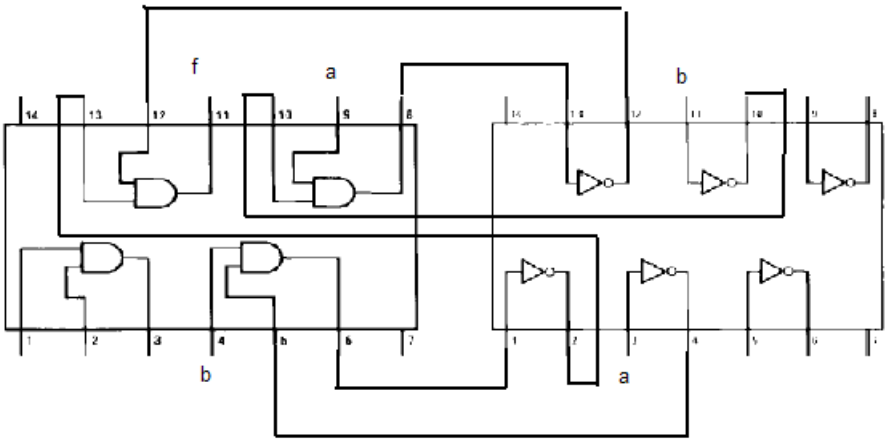

Figura 2.23 Implementación de una función lógica mediante circuitos integrados.

Primero observamos que tenemos dos puntos de entrada que son las variables a y b. Éstas se aplican a puertas inversoras para obtener $\bar{a}$ y $\bar{b}$ (terminales 4 y 10 del 7404). Posteriormente vemos que tenemos un par de puertas AND que implementan las funciones $\bar{a} \times b$ y $\bar{b} \times a$ (terminales 6 y 8 del 7408). Estos dos últimos términos se vuelven a aplicar a puertas inversoras que darán lugar a los términos $\overline{\bar{a} \times b}$ y $\overline{\bar{b} \times a}$ (terminales 2 y 12 del 7404). Finalmente existe otra puerta AND que implementa la operación producto de estos dos términos (terminal 11 del 7408), por lo que la función resultante será $f(a,b) = \overline{\bar{a} \times b} \times \overline{\bar{b} \times a}$ que de hecho es equivalente a una XOR complementada.

2.7.1. Problemas de diseño

Ahora que hemos visto la existencia de circuitos integrados que permiten implementar los esquemas de puertas lógicas de funciones booleanas, tenemos que ser capaces de ver cómo es posible implementar simples aplicaciones que basan su funcionamiento en una función booleana.

Las pautas básicas de estos problemas de diseño suelen ser:

- Identificación de la función booleana que resuelve el problema planteado. En general esta función se expresa mediante tabla de verdad, en tanto que para cada combinación de las variables de entrada tenemos que ir "razonando" el valor que tiene que tomar la salida de la función según los requisitos indicados en el enunciado.
- Expresión de la función booleana de forma algebraica. A menudo se busca una simplificación de la misma.
- Implementación de la función con puertas lógicas. Ésta puede ser usando la técnica de implementación basada en puertas NAND o NOR, o bien directamente usando cualquier tipo de puerta lógica.

A continuación veremos algunos ejemplos de problemas de diseño. Comentar que no existe un método universal para resolverlos y en este punto tan solo pretendemos que estos problemas os permitan empezar a introduciros en el diseño e implementación de pequeñas aplicaciones.

Ejemplo 2.22

Queremos diseñar un sistema que compare dos números digitales.

El sistema en cuestión tendrá dos entradas a y b, cada una de las cuales tendrá 2 bits, por lo que realmente tendremos 4 variables de entrada a_1, a_0, b_1, b_0. El sistema tendrá 2 salidas denominadas s y c.

Cuando las dos entradas presenten el mismo valor, es decir que $a_1 = b_1$ y $a_0 = b_0$ entonces activaremos $s = 1$ y $c = 0$. En cambio si son distintas entonces activaremos $s = 0$ y $c = 1$ cuando el valor binario de a sea mayor que el de b, o bien las salidas valdrán $s = 0$ y $c = 0$ cuando el valor binario de a sea menor que el de b.

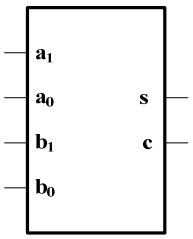

Figura 2.24. Sistema comparador que queremos diseñar.

Primero miramos de construir la tabla de verdad del sistema. De hecho son dos tablas de verdad, una para implementar la función que dará lugar a la variable s y otra para implementar la función de la variable c. Para simplificar su representación vamos a indicar en una misma tabla ambas funciones, tal y como se muestra a continuación.

a_1	a_0	b_1	b_0	s	c
0	0	0	0	1	0
0	0	0	1	0	0
0	0	1	0	0	0
0	0	1	1	0	0
0	1	0	0	0	1
0	1	0	1	1	0
0	1	1	0	0	0
0	1	1	1	0	0
1	0	0	0	0	1
1	0	0	1	0	1
1	0	1	0	1	0
1	0	1	1	0	0
1	1	0	0	0	1
1	1	0	1	0	1
1	1	1	0	0	1
1	1	1	1	1	0

Tabla 2.15. Tabla de verdad del sistema comparador.

A partir de la tabla de verdad es relativamente simple identificar una función algebraica para las dos variables de salida. Si por ejemplo las expresamos por medio de minterms tendremos que $s = \overline{a_1} \times \overline{a_0} \times \overline{b_1} \times \overline{b_0} + \overline{a_1} \times a_0 \times \overline{b_1} \times b_0 + a_1 \times \overline{a_0} \times b_1 \times \overline{b_0} + a_1 \times a_0 \times b_1 \times b_0$ y que $c = \overline{a_1} \times a_0 \times \overline{b_1} \times \overline{b_0} + a_1 \times \overline{a_0} \times \overline{b_1} \times \overline{b_0} + a_1 \times \overline{a_0} \times \overline{b_1} \times b_0 + a_1 \times a_0 \times \overline{b_1} \times \overline{b_0} + a_1 \times a_0 \times \overline{b_1} \times b_0 + a_1 \times a_0 \times b_1 \times \overline{b_0}$.

Estas funciones las podemos implementar por medio de puertas lógicas. Por ejemplo una posible solución para la función s podría ser la que mostraremos a continuación y dejamos para vosotros que encontréis una implementación para la función c. Una vez hubiésemos obtenido los esquemas de puertas lógicas tan solo restaría comprar los circuitos integrados necesarios para su implementación real e interconectarlos según el esquema propuesto.

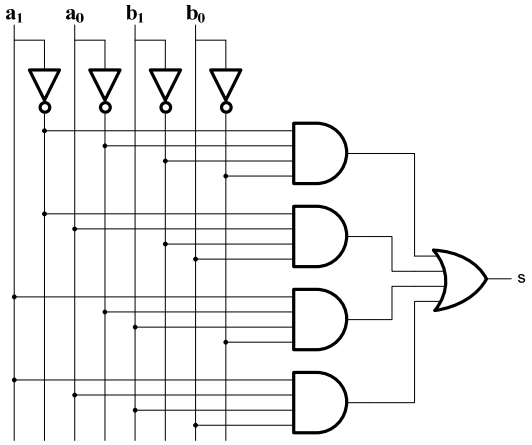

Figura 2.25. Diagrama de puertas lógicas de la función s del sistema comparador.

Ejemplo 2.23

Supongamos que tenemos que diseñar un circuito lógico que implemente el control de luces intermitentes de un vehículo según las siguientes especificaciones. También aprovecharemos este problema para repasar algunos de los conceptos vistos en sesiones anteriores.

El sistema dispondrá de dos líneas de entrada:

- CLK será una entrada por la que en el sistema se introducirá una secuencia alternada 1,0,1,0,1,0,1,0, etc.
- M será un bus de entrada de 2 bits que indicará el modo de funcionamiento.

Y tendrá las siguientes salidas:

- L será la línea que activará el intermitente de la izquierda por lógica positiva, esto quiere decir que cuando valga 1 el intermitente estará encendido mientras que estará apagado cuando valga 0.
- R funcionará como la salida L pero para el intermitente de la derecha.

Figura 2.26. Sistema de luces del vehículo que queremos implementar. El número que se indica sobre cada línea de entrada expresa la longitud en bits de dicha línea, es decir, M realmente es un bus de 2 bits por lo que en verdad tendrá dos líneas de entrada M_1 y M_0, mientras que el resto de líneas son directamente de 1 bit.

El modo de funcionamiento nos dicen que será el siguiente según M:

- Cuando $M_1 = 0$ y $M_0 = 0$ entonces las luces estarán siempre apagadas.
- Cuando $M_1 = 0$ y $M_0 = 1$ entonces las luces de los intermitentes se activaran de manera alternada, es decir, cuando $CLK = 1$ se activará una de ellas y cuando $CLK = 0$ se activará la otra.
- Cuando $M_1 = 1$ y $M_0 = 0$ entonces las luces de los intermitentes se activaran de manera simultánea, es decir, cuando $CLK = 1$ se activarán ambas luces y cuando $CLK = 0$ se apagarán las dos.
- Cuando $M_1 = 1$ y $M_0 = 1$ entonces las luces estarán siempre encendidas.

Empecemos a diseñar el sistema. Para ello vamos a crear la tabla de verdad del sistema según las especificaciones que nos han indicado.

M_1	M_0	CLK	L	R
0	0	0	0	0
0	0	1	0	0
0	1	0	0	1
0	1	1	1	0
1	0	0	0	0
1	0	1	1	1
1	1	0	1	1
1	1	1	1	1

Tabla 2.16. Tabla de verdad del sistema de luces del vehículo.

A partir de la tabla de verdad buscamos una función algebraica para ambas salidas en cualquier formato. Por ejemplo podríamos expresar L por minterms, es decir $L = \overline{M_1} \times M_0 \times CLK + M_1 \times \overline{M_0} \times CLK + M_1 \times M_0 \times \overline{CLK} + M_1 \times M_0 \times CLK$ y podríamos expresar R por maxterms, es decir $R = (M_1 + M_0 + CLK) \times (M_1 + M_0 + \overline{CLK}) \times (M_1 + \overline{M_0} + \overline{CLK}) \times (\overline{M_1} + M_0 + CLK)$.

Finalmente sólo nos queda implementar ambas funciones mediante puertas lógicas. Supongamos que nos piden hacer la implementación sólo con puertas NAND, por lo que primero tendremos que reescribir ambas funciones para que éstas sólo incluyan operadores producto. De esta manera escribimos la primera de las salidas como $L = \overline{\overline{M_1} \times M_0 \times CLK + M_1 \times \overline{M_0} \times CLK + M_1 \times M_0 \times \overline{CLK} + M_1 \times M_0 \times CLK} = \overline{\overline{M_1} \times M_0 \times CLK} \times \overline{M_1 \times \overline{M_0} \times CLK} \times \overline{M_1 \times M_0 \times \overline{CLK}} \times \overline{M_1 \times M_0 \times CLK}$, que podríamos implementar de la siguiente manera mediante puertas NAND.

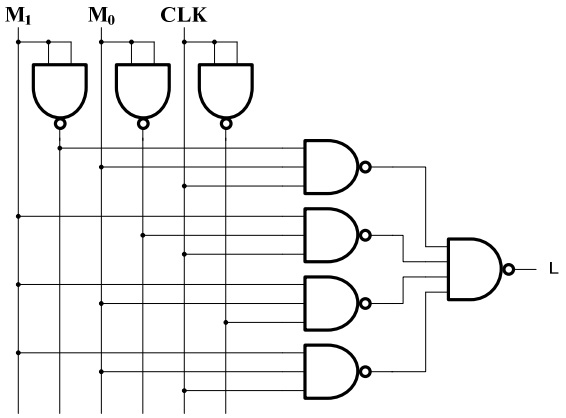

Figura 2.27. Implementación con puertas lógicas de tipo NAND de la salida L del sistema de luces del vehículo.

Os propongo que terminéis el problema implementando la otra salida R de manera similar a L.

Ejemplo 2.24

Os propongo que miréis de diseñar un sistema que sume dos números A y B que se introducen al sistema en binario natural. Estos números tan sólo pueden ser 0, 1, 2 ó 3, por lo que cada una de estas entradas será de 2 bits. El sistema tendrá que devolver el resultado R expresado en código Gray mediante un bus de 3 bits.

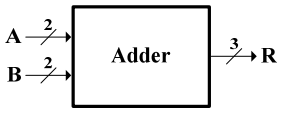

Figura 2.28. Sistema sumador.

Para hacer el desarrollo os propongo que:

- Primero hagáis la tabla de verdad del sistema, en el que para cada posible combinación de valores de entrada se indique el valor que tomará el bus de salida.
- Luego expresáis las 3 salidas mediante una forma canónica.
- Finalmente implementáis las 3 salidas mediante el uso de puertas NAND o bien NOR.