A Mamen, gracias por estar siempre a mi lado.

Marcos García

A mi mujer Laura, por su apoyo y ánimo constante.

Pablo Huerta

A mi familia.

Carlos Sánchez

A Beatriz, mis padres y mis hermanos, por todo el apoyo que me dan siempre.

Pablo Toharia

Introducción

Este libro surge con el objetivo de acercar al lector a los aspectos más importantes que encierra la electrónica ante la creciente demanda de personal cualificado para su administración. Con tal propósito, puede servir de apoyo también para estudiantes del Ciclo Formativo de Grado Medio de Instalaciones Eléctricas y Automáticas y para profesionales de distinto rango.

Para todo aquel que use este libro en el entorno de la enseñanza (Ciclos Formativos, Profesionales o Universidad) se ofrecen varias posibilidades: utilizar los conocimientos aquí expuestos para inculcar aspectos genéricos de la electrónica o simplemente centrarse en preparar a fondo alguno de ellos. La extensión de los contenidos aquí incluidos hace imposible su desarrollo completo en la mayoría de los casos.

Ra-Ma pone a disposición de los profesores una guía didáctica para el desarrollo del tema que incluye las soluciones a los ejercicios expuestos en el texto. Puede solicitarlo a *editorial@ra-ma.com*, acreditándose como docente y siempre que el libro sea utilizado como texto base para impartir las clases.

1 Sistemas digitales

OBJETIVOS DEL CAPÍTULO

- ✓ Entender las diferencias principales que existen entre los sistemas analógicos y digitales.
- ✓ Comprender los fundamentos del álgebra de Boole y su aplicación en el diseño y análisis de los sistemas digitales.
- ✓ Estudiar los fundamentos del diseño de circuitos combinacionales.
- ✓ Introducir al alumno en el diseño de circuitos utilizando bloques combinacionales básicos.

1.1 INTRODUCCIÓN A LOS SISTEMAS DIGITALES

Podríamos definir un **computador** como una máquina capaz de procesar información. Dicho procesado implica manipular la información de partida con el objetivo de resolver un determinado problema. Sin embargo, en esta definición no se hace referencia a la tecnología con la que se fabrican dichos computadores.

A lo largo de la historia de la computación se han utilizado distintas tecnologías en su construcción: desde las primeras máquinas mecánicas de *Pascal*, *Babbage* y *Leibniz*, pasando por las máquinas compuestas de piezas electromecánicas como relés, entre los que destaca el *Harvard Mark I*, y por las máquinas electrónicas actuales hasta los novedosos sistemas ópticos o los aún experimentales sistemas cuánticos y de procesado basado en ADN. En este libro nos centraremos en los más extendidos actualmente: los sistemas electrónicos.

1.1.1 MAGNITUDES ANALÓGICAS Y MAGNITUDES DIGITALES

La información que se suministrará al sistema, independientemente de la tecnología utilizada, viene caracterizada por una o varias magnitudes. Una **magnitud** es una propiedad física que puede medirse cuantitativamente. Dependiendo de la naturaleza de las mismas podemos dividirlas en analógicas y digitales. Se denomina **señal** a la evolución en el tiempo de dichas magnitudes.

Las **magnitudes analógicas** son aquellas que toman valor en un rango continuo. Matemáticamente se asocian con números reales o conjuntos de los mismos. Se pueden poner muchos ejemplos de magnitudes analógicas: temperatura, voltaje, fuerza, etc. La mayoría de los fenómenos naturales se miden utilizando magnitudes analógicas.

Por el contrario las **magnitudes digitales** toman valor en un rango discreto. Al igual que las magnitudes analógicas podemos asociarlas con un subconjunto matemático: los números enteros. Como ejemplos de magnitudes digitales podemos destacar el número de habitantes de una población o el número de coches de un determinado aparcamiento.

Recapitulando, las magnitudes digitales toman valor en el rango de los números reales y las magnitudes digitales toman valor en el rango de los números enteros. De este modo, tiene sentido hablar de una intensidad de corriente de 1,25 Amperios y no tiene sentido decir que en una determinada librería hay 30,6 libros. Una determinada magnitud puede variar en el tiempo. Esta variación de la magnitud en el tiempo recibe el nombre de señal y al igual que las magnitudes pueden ser analógicas o digitales.

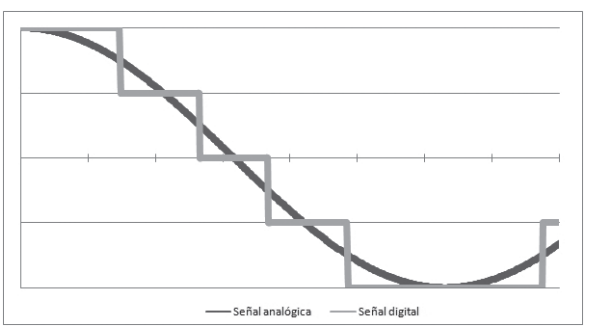


Figura 1.1. *Comparativa entre una señal analógica y una señal digital*

Es probable que el lector ya sepa que los sistemas electrónicos se dividen en sistemas analógicos y sistemas digitales. Los analógicos son aquellos que procesan información analógica, mientras que por el contrarío los sistemas digitales procesan información digital.

Atendiendo a esta definición y sabiendo que la mayoría de los fenómenos naturales se miden mediante magnitudes analógicas, se podría llegar a pensar que los sistemas digitales tienen un ámbito de aplicación más reducido que los sistemas analógicos. Esto no es cierto en absoluto, si bien es cierto que los sistemas digitales solo pueden manipular información digital, la información analógica puede transformarse en información digital mediante **conversores analógico/digitales** y la información digital puede transformarse en información analógica mediante **conversores digital/analógicos** (a partir de ahora D/A).

A los conversores analógico/digitales a partir de ahora se les denominará A/D.

A los conversores digital/analógicos a partir de ahora se les denominará D/A.

EJEMPLO 1.1

Se pretende diseñar un sistema que regule la temperatura de una habitación. El sistema recibe como entrada la temperatura de la habitación y como salida debe devolver el voltaje que hay que suministrar al ventilador encargado de enfriarla. Podríamos construir dicho sistema de control utilizando un computador digital. Para ello deberíamos utilizar un conversor A/D que transforme la señal de entrada para que pueda ser procesada por nuestro computador y un conversor D/A que transforme la salida de nuestro computador en la señal que necesita el ventilador.

ACTIVIDADES 1.1

- Buscar información sobre conversores A/D y D/A.
- Entender la diferencia entre conversores causales y no causales.

1.1.2 VENTAJAS E INCONVENIENTES DE LOS SISTEMAS DIGITALES

Actualmente la electrónica digital está experimentado un gran empuje y se está imponiendo en mercados en los que tradicionalmente se imponía la electrónica analógica. Esto se debe sus numerosas ventajas:

- ✓ **Sencillez**. El diseño de circuitos digitales es relativamente simple reduciendo el tiempo de diseño y abaratando el coste del producto. Existen numerosas herramientas tanto de alto nivel (lenguajes de descripción como el VHDL o el VERILOG) como de bajo nivel hardware (herramientas CAD como Virtuoso Layout Suite) que nos permiten definir de forma sencilla circuitos digitales.
- ✓ **Menor sensibilidad a ruidos**. Al trasmitir la señal por un determinado medio ésta puede atenuarse y degradarse. La señal digital en muchos casos puede amplificarse y regenerarse.
- ✓ **Tolerancia a fallos**. Existen numerosos sistemas de codificación digital capaces de detectar información corrupta y en algunos casos regenerarla (bits de paridad, códigos de *Hamming*…).
- ✓ **Facilidad de almacenamiento**. Existen numerosos sustratos capaces de almacenar de forma barata gran cantidad de información digital (memorias flash, discos duros magnéticos…). Además, se pueden encontrar algoritmos de compresión mucho más eficientes que los existentes para información analógica (codificación incremental, códigos de Huffman…).

ACTIVIDADES 1.2

- Buscar información sobre el problema del *aliasing* en las señales.

1.1.3 SEÑALES DIGITALES

Este capítulo se centra en describir los fundamentos del diseño de sistemas digitales. En la actualidad la información manejada por dichos sistemas es **binaria**, es decir, la información se compone por secuencias de dos dígitos: el "0" y el "1". Se suele denominar al "1" como valor cierto y al "0" como valor falso. Esta información se codifica mediante una magnitud física: el voltaje. Cada tecnología define dos valores de voltaje: valor bajo (VL) y valor alto (VH). Generalmente se hace coincidir a VL con el valor "0" y VH con el valor "1".

ACTIVIDADES 1.3

- Ampliar la información de este apartado con la que se adjunta en el apéndice del libro.

En el análisis y diseño de sistemas digitales, con el objetivo de facilitar la comprensión del sistema, se suele utiliza una señal idealizada en lugar del voltaje. El voltaje como señal analógica puede tener ruido y, como se ha comentado anteriormente, VL y VH no tienen valores precisos. Por este motivo, las señales del sistema se suelen mostrar como una onda cuadrada ideal que puede tomar los valores "0" y "1".

En estas señales denominaremos **flanco** a una transición entre dos niveles. Si esta transición es de "0" a "1", recibirá el nombre de flanco de subida, mientras que si es de "1" a "0" recibirá el nombre de flanco de bajada. La señal entre dos flancos recibirá el nombre de pulso. Si se encuentra entre un flanco de subida y uno de bajada se estará hablando de un **pulso** positivo, mientras que si se encuentra entre un flanco de bajada y uno de subida estaremos hablando de un pulso negativo.

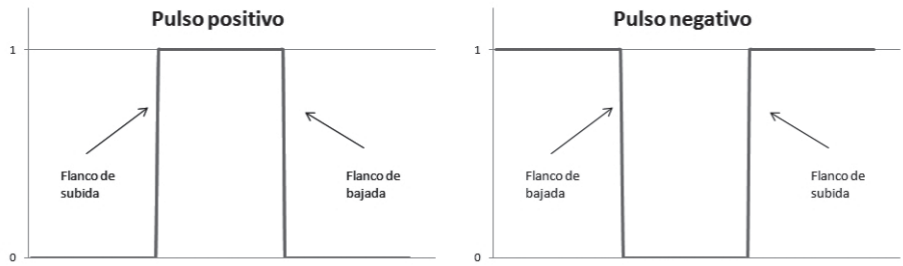


Figura 1.2. *Pulsos digitales*

Muchos sistemas digitales utilizan señales que se repiten de forma periódica para su sincronización. Dichas señales reciben el nombre de *reloj*. Es importante destacar que la duración del pulso positivo no tiene porque ser igual a la duración del pulso negativo, siempre y cuando todos los pulsos positivos y todos los pulsos negativos tengan la misma duración.

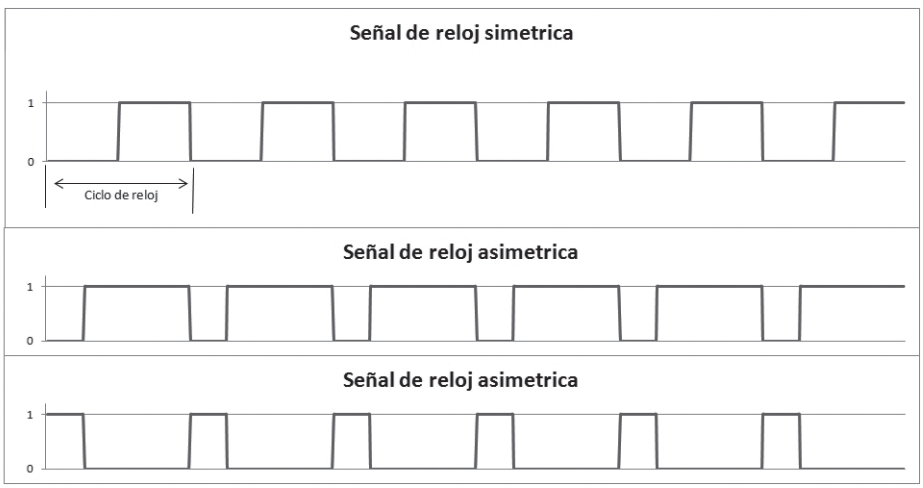


Figura 1.3. *Distintos tipos de señales de reloj*

El conjunto de señales de un sistema recibe el nombre de **cronograma**. Los cronogramas son herramientas imprescindibles en el diseño, el análisis y la depuración de circuitos digitales.

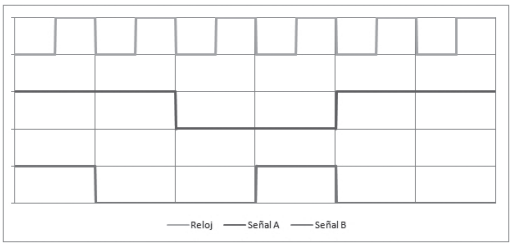


Figura 1.4. *Cronograma de un sistema con 2 señales (A y B)*

1.2 EL ÁLGEBRA DE BOOLE Y LAS PUERTAS LÓGICAS: LAS PIEDRAS ANGULARES DEL DISEÑO DE CIRCUITOS DIGITALES

En el capítulo 3 se describirán distintas codificaciones que nos permiten representar información de distinta naturaleza. El complemento a 2 permite representar números enteros y el sistema ANSI permite representar caracteres alfanuméricos. El objetivo de este libro no es que el lector sea capaz de operar con distintas codificaciones, sino que sea capaz de diseñar circuitos que realicen estas operaciones. Una de las herramientas básicas en la síntesis de circuitos digitales es el **álgebra de Boole** .El álgebra de *Boole* fue descrita por el matemático británico George Boole en libro *An Investigation of the Laws of Thought* en 1854. Lo que no sabía George Boole es que casi un siglo después, en 1939, Claude E. Shannon propondría el uso de su formalismo matemático para el análisis de circuitos digitales en su tesis de máster *A Symbolic Analysis of Relay and Switching Circuits*, convirtiéndolo en la piedra angular del diseño de estos sistemas. Esta herramienta simplifica su descripción, siendo la gran responsable del auge de esta tecnología, omnipresente en nuestros días y que actualmente se está permitiendo ocupar nichos hasta hace poco reservados a los sistemas analógicos como el control industrial.

¿SABÍAS QUE...?

Toda operación realizada en un sistema digital, ya sea un computador, un teléfono, un reloj o una calculadora utiliza las operaciones definidas por el algebra de Boole. Unas veces estas funciones vendrán implementadas en hardware y otras en software.

Se puede definir el álgebra de Boole bivaluada (desde este punto en adelante la denominaremos solo álgebra de Boole) a partir de sus operadores, siendo el conjunto B que cumple las siguientes propiedades:

- ✓ Todo elemento a del conjunto toma o bien el valor 0 o bien el valor 1:

  $\forall\, a \in B \mid a = 0$ ó $a = 1$

- ✓ Para todos los elementos del conjunto se define la operación unitaria (un único operando) **complemento** o negación (') de la siguiente forma:

  Si a es igual a 0, a' es igual a 1: $a' = 1 \rightarrow a = 0$
  Si a es igual a 1, a' es igual a 0: $a' = 0 \rightarrow a = 1$

- ✓ Para todos los elementos del conjunto se define la operación binaria (dos operandos) **producto lógico** (*) como:

  Si a es igual a 0 y b es igual a 0, $a*b$ es igual a 0: $a*b = 0 \rightarrow a = 0\ y\ b = 0$.
  Si a es igual a 0 y b es igual a 1, $a*b$ es igual a 0: $a*b = 0 \rightarrow a = 0\ y\ b = 1$.
  Si a es igual a 1 y b es igual a *0*, $a*b$ es igual a 0: $a*b = 0 \rightarrow a = 1\ y\ b = 0$.
  Si a es igual a 1 y b es igual a 1, $a*b$ es igual a 1: $a*b = 1 \rightarrow a = 1\ y\ b = 1$.

- ✓ Para todos los elementos del conjunto se define la operación binaria (dos operandos) **suma lógica** (+) como:

  Si a es igual a 0 y b es igual a 0, $a*b$ es igual a 0: $a*b = 0 \rightarrow a = 0\ y\ b = 0$.
  Si a es igual a 0 y b es igual a 1, $a*b$ es igual a 1: $a*b = 1 \rightarrow a = 0\ y\ b = 1$.
  Si a es igual a 1 y b es igual a 0, $a*b$ es igual a 1: $a*b = 1 \rightarrow a = 1\ y\ b = 0$.
  Si a es igual a 1 y b es igual a 1, $a*b$ es igual a 1: $a*b = 1 \rightarrow a = 1\ y\ b = 1$.

Estos operadores (+, *, ‘) combinan variables y constantes formando *expresiones lógicas*.

EJEMPLO 1.2

Podríamos definir la función *F* que depende de las variables *a*, *b* y *c* como *F* (*a*, *b*, *c*) = *a*b′ + c + 0*. De esta forma *F* queda definida para cualquier combinación de valores de *a, b y c*. Si *a* es igual a 1, *b* es igual a 0 y *c* es igual a 0, *F* tomará el valor de *1*:

F(1, 0, 0) = 1 * 0′ + 0 + 0 = 1 * 1 + 0 + 0= 1 + 0 + 0 = 1 + 0 = 0

Es importante que los operadores se apliquen en el orden correcto. Si no es así, la expresión podría tomar un valor erróneo. Para poder entender cualquier función lógica es necesario conocer cuál es la prioridad de los operadores. Se dice que un operador *x* tiene prioridad sobre otro operador *y* cuando ante la posibilidad de aplicar los dos operadores, el operador *x* se aplicará siempre primero. Decimos que el operador complemento (‘) tiene prioridad sobre el operador producto lógico (*), puesto que debe aplicarse primero.

EJEMPLO 1.3

Si se define *F*(*a*,*b*)=*a*b′* diremos que *F* es igual al producto de *a* por el complemento de *b* y **NO** que *F* es el complemento del producto de *a* y *b*. De esta forma:

F (0, 1) = 0 * 1′ = 0 * 0 = 0

y **NO**: *F* (0, 1) = 0 * 1′ = 0′ = 1

Las prioridades de los operadores están preestablecidas de forma arbitraria por convenio, siendo el operador de mayor prioridad la negación o complemento y el de menor prioridad la suma lógica. El complemento se aplicará antes que el producto y que la suma; y el producto se aplicará antes que la suma. Para poder modificar la prioridad de un operador se pueden utilizar paréntesis, priorizando de esta manera la expresión comprendida entre dos paréntesis, al igual que se hace en cualquier expresión algebraica.

EJEMPLO 1.4

Si se desea que la función aritmética *F* represente el complemento del producto de dos variables *a* y *b*, podríamos utilizar la siguiente expresión lógica *F*(*a*, *b*) = (*a* * *b*)'. De esta forma:

F (0, 1) = (0 * 1)' = (0)' = 1

¿SABÍAS QUE...?

En inglés:

- Puerta lógica: *logic gate*.
- Salida y entrada digital: *digital output and input.*
- Símbolo: *symbol.*
- Sonda lógica: *logic probe.*
- Tabla de verdad: *truth table.*
- Circuito integrado (C.I.): *integrated circuit* (I.C.).
- Placa de inserción: *protoboard*.
- Hoja de características: *datasheet*.

Antes de finalizar este apartado en el que se han introducido los conceptos de álgebra de *Boole* y expresiones lógicas, se quiere destacar que los operadores producto lógico, suma lógica y conjunción pueden representarse de forma distinta a la que se ha indicado aquí. Es también común en la bibliografía que al operador producto lógico se le denomine *conjunción* o *and lógico* y se le represente como ∧ o *AND,* de la misma forma al operador suma lógica se le puede denominar *disyunción* o *or lógico* y se le representa como ∨ o *OR* y por último al operador conjunción se le puede llamar *negación* o *not lógico* y se puede representar con un subrayado alto ¯, con ~, ¬ o con un NOT. Los elementos del conjunto *B* también pueden recibir otros nombres: en algunas ocasiones el valor *1* puede sustituirse por *True*, *Verdadero*, *V* o *T* y el valor *0* por *False*, *Falso o F*. De esta forma, las siguientes expresiones lógicas representan la misma función:

$$F\,(a, b, c) = (a * b)' + c$$

$$F\,(a, b, c) = \neg\,(a \wedge b) \vee c$$

$$F(a, b, c) = \overline{(a \wedge b)} \vee c$$

$$F\,(a, b, c) = NOT\,(a\ AND\ b)\ OR\ c$$

1.2.1 PROPIEDADES Y TEOREMAS DEL ÁLGEBRA DEL BOOLE

El álgebra de Boole verifica ciertos teoremas, propiedades y principios que nos permitirán operar con las expresiones lógicas dándonos la posibilidad de modificarlas con distintos propósitos, por ejemplo simplificar una expresión. De esta forma, el lector debe tener claro que una función lógica puede representarse con distintas expresiones.

Uno de los principios más importantes del álgebra de *Boole*, es el **principio de dualidad**. Este principio nos permite inferir nuevos teoremas a partir de teoremas ya existentes. El nuevo teorema recibirá el nombre de **teorema dual**. El principio de dualidad indica que *dado un teorema*, *existe un teorema dual que se obtiene sustituyendo los 0 por 1*, *los 1 por 0*, *los productos lógicos por sumas lógicas y las sumas lógicas por productos lógicos*.

EJEMPLO 1.5

Una vez comprobada la propiedad asociativa de la suma lógica quedará demostrada la propiedad asociativa del producto lógico:

$(a * b) * c = a * (b * c) \rightarrow (a + b) + c = a + (b + c)$

A continuación se detallarán las principales propiedades del álgebra de Boole:

- **Propiedad conmutativa** de la suma lógica:

  $a + b = b + a \qquad (\forall\, a, b \in B)$

- **Propiedad asociativa** de la suma lógica:

  $(a + b) + c = a + (b + c) \qquad (\forall\, a, b, c \in B)$

- **piedad distributiva** de la suma lógica:

  $a + (b * c) = (a + b) * (a + c) \qquad (\forall\, a, b, c \in B)$

- **Elemento neutro** de la suma lógica:

  $a + 0 = a \qquad (\forall\, a \in B)$

Utilizando el principio de dualidad se pueden enunciar las mismas propiedades del producto lógico:

- **Propiedad conmutativa** del producto lógico:

  $a * b = b * a \qquad (\forall\, a, b \in B)$

- **Propiedad asociativa** del producto lógico:

  $(a * b) * c = a * (b * c) \qquad (\forall\, a, b, c \in B)$

- **Propiedad distributiva** del producto lógico:

  $a * (b + c) = (a * b) + (a * c) \qquad (\forall\, a, b, c \in B)$

- **Elemento neutro** del producto lógico:

  $a * 1 = a \qquad (\forall\, a \in B)$

Además de las propiedades anteriores, todos los elementos de B deben cumplir la propiedad de ortocomplementariedad o **involución**:

$(a')' = a \qquad (\forall\, a \in B)$

También son de vital importancia los siguientes teoremas (a partir de ahora enunciaremos el teorema junto con su teorema dual):

- ✓ **Teorema de identidad**:

  $a + a' = 1 \qquad (\forall\, a \in B)$

  $a * a' = 0 \qquad (\forall\, a \in B)$

- ✓ **Teorema de idempotencia**:

  $a + a = a \qquad (\forall\, a \in B)$

  $a * a = a \qquad (\forall\, a \in B)$

- ✓ **Teorema de la identidad del 1 y del 0**:

  $a + 1 = 1 \qquad (\forall\, a \in B)$

  $a * 0 = 0 \qquad (\forall\, a \in B)$

- ✓ **Elemento neutro**:

  $a + 0 = a \qquad (\forall\, a \in B)$

  $a * 1 = a \qquad (\forall\, a \in B)$

- ✓ **Teoremas de absorción**:

  $a + a * b = a \qquad (\forall\, a, b \in B)$

  $a + a' * b = a + b \qquad (\forall\, a, b \in B)$

  $a * (a + b) = a \qquad (\forall\, a, b \in B)$

  $a * (a' + b) = a * b \qquad (\forall\, a, b \in B)$

Por último, se destacarán las **leyes de De Morgan**:

$(a + b)' = a' * b'$

$(a * b)' = a' + b'$

La importancia de las *Leyes de De Morgan* radica en su capacidad para sustituir los productos lógicos de una expresión por sumas lógicas y viceversa.

Antes de concluir con este apartado, destacar que las variables que aparecen en las propiedades, teoremas y leyes descritos anteriormente pueden ser sustituidos por expresiones lógicas.

EJEMPLO 1.6

Si se tiene la función lógica $F(x, y, z) = x * y + ((z + y) * (z + x))$, se puede sustituir $a = x * y$, $b = z + y$ y $c = z + x$ de forma que $F = a + (b * c)$ y aplicar la propiedad distributiva $F = (a + b) * (a + c)$ y por último deshacer el cambio, de forma que $F(x, y, z) = (x * y + z + y) * (x * y + z + x)$

Como el lector imaginará no hace falta realizar las sustituciones de variables de forma explícita. La mayor parte de las veces se podrán hacer estos cálculos mentalmente. De esta forma, se podría simplificar la expresión anterior de la siguiente manera:

- Teorema de absorción: $F(x, y, z) = (z + y) * (z + x)$
- Propiedad distributiva: $F(x, y, z) = z + x * y$

1.2.2 PUERTAS LÓGICAS

Llegados a este punto, ¿dónde radica la potencia del álgebra de *Boole* en el diseño de circuitos digitales? Antes de contestar a esa pregunta, se debe echar un vistazo a la unidad básica de cualquier circuito digital: el **transistor**. Este dispositivo se explicará de forma más detallada en los capítulos siguientes. Lo único que por ahora el lector necesita saber es que a partir de transistores se pueden construir los tres operadores básicos del álgebra de *Boole*: el producto, la suma y la negación.

Pueden encontrase implementaciones de dichos operadores en distintas tecnologías como las unipolares (o de efecto campo: *CMOS*, *NMOS* o *PMOS*) o las bipolares (*RTL* o *TTL*). La tecnología seleccionada para implementar nuestro circuito es básica a la hora de determinar los parámetros físicos de funcionamiento del sistema. Dependiendo de dicha tecnología nuestro circuito interpretará como 0 o como 1 a distintos valores de tensión, variara el retardo del cada dispositivo, el consumo de potencia… El fabricante del dispositivo nos detallara toda esta información en la hoja de características del producto o **datasheet**.

En la ficha del libro de *www.ra-ma.es* se podrán consultar las hojas de características de componentes electrónicos y los recursos de la unidad.

¿SABÍAS QUE…?

Los circuitos de tecnología TTL se prefijan normalmente con el número 74. En ocasiones, los verás con el número 54, es la versión de la serie militar e industrial (aeroespacial). Esto implica que sus especificaciones son superiores.

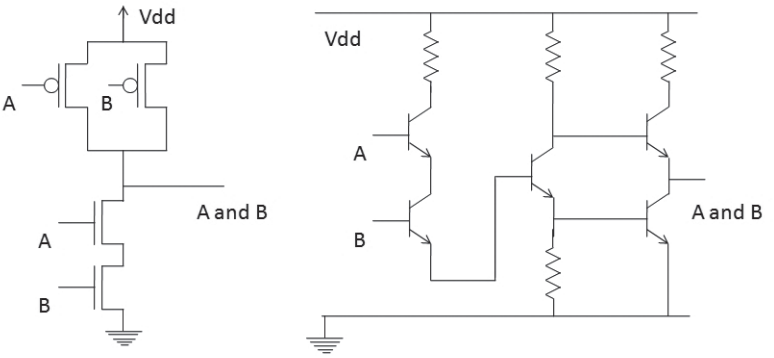
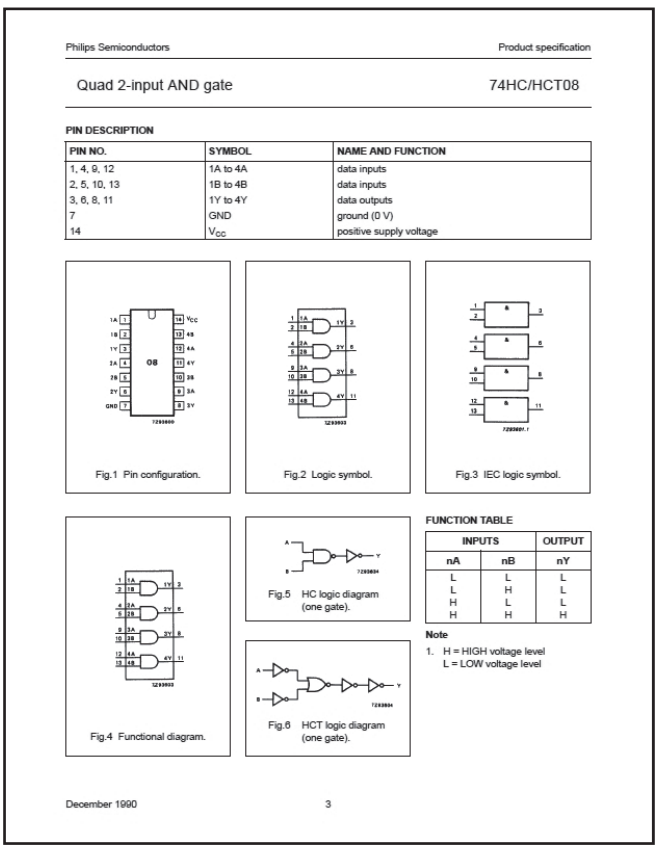


Figura 1.5. *Puerta AND de dos entradas implementada con transistores de efecto de campo (izquierda) y con transistores bipolares (derecha)*



Philips Semiconductors — Product specification

Quad 2-input AND gate — 74HC/HCT08

PIN DESCRIPTION

PIN NO.	SYMBOL	NAME AND FUNCTION
1, 4, 9, 12	1A to 4A	data inputs
2, 5, 10, 13	1B to 4B	data inputs
3, 6, 8, 11	1Y to 4Y	data outputs
7	GND	ground (0 V)
14	V_{CC}	positive supply voltage

Fig.1 Pin configuration.

Fig.2 Logic symbol.

Fig.3 IEC logic symbol.

Fig.4 Functional diagram.

Fig.5 HC logic diagram (one gate).

Fig.6 HCT logic diagram (one gate).

FUNCTION TABLE

INPUTS		OUTPUT
nA	nB	nY
L	L	L
L	H	L
H	L	L
H	H	H

Note

1. H = HIGH voltage level
   L = LOW voltage level

December 1990 — 3

Figura 1.6. *Fragmento del Datasheet de una puerta AND de dos entradas (74HC08) de Philips Semiconductors*

Mediante transistores (diodos y resistencias, dependiendo de la tecnología) no solo se pueden implementar los operadores básicos del álgebra de Boole sino también cualquier expresión lógica. En este libro no se detallará como diseñar funciones lógicas a nivel de transistor, en su lugar se utilizarán como bloques básicos implementaciones ya existentes de los operadores antes mencionados y de funciones lógicas sencillas, denominadas **puertas lógicas**.

Las puertas lógicas se caracterizan por la función lógica que realizan y por el número de entradas. De este modo se pueden encontrar puertas lógicas que realicen las tres operaciones básicas del álgebra de Boole: el producto realizado por las puertas AND, la suma realizada por las puertas OR y el complemento realizado por la el INVESOR. Se encuentran distintas puertas AND y OR dependiendo del número de entradas. Las puertas AND de dos entradas realizarán la función lógica $AND\,(a, b) = a * b$ y la puerta OR de 3 entradas realizará la función lógica $OR(a, b, c) = a + b + c$.

Además de las puertas lógicas que implementan los operadores del álgebra de *Boole* podemos encontrar otras que realizan las siguientes funciones lógicas sencillas:

- **Puerta O Negado o puerta NOR**. Como su nombre indica, implementa la función complemento de una suma lógica:

  $NOR(a, \ldots, n) = (a + \ldots + n)'$

- **Puerta Y Negado o NAND**. Como su nombre indica, implementa la función complemento de un producto lógico:

  $NAND(a, \ldots, n) = (a * \ldots * n)'$

- **Puerta O Exclusivo o XOR**. Esta puerta implementa la función lógica que toma el valor 1 si y solo una única variable de entrada toma el valor 1.

  $XOR(a, b, \ldots, n) = a * b' * \ldots * n' + a' * b * \ldots * n' + a' * b' * \ldots * n.$

- **Puerta O Exclusivo Negado o NXOR**. Esta puerta implementa la negación de la función XOR.

  $XOR(a, b, \ldots, n) = (a * b' * \ldots * n' + a' * b * \ldots * n' + a' * b' * \ldots * n)'.$

EJEMPLO 1.7

Puerta NOR de tres entradas

- $NOR(a, b, c) = (a + b + c)'$

Puerta NAND de dos entradas

- $NAND(a, b) = (a * b)'$

Puerta XOR de tres entradas

- $XOR(a, b, c) = a * b' * c' + a' * b * c' + a' * b' * c$

Esta función puede encontrases como operador en funciones lógicas representada con el símbolo $\oplus$, de esta forma:

- $XOR(a, b, c) = a \oplus b \oplus c$

Puerta NXOR de dos entradas

- $NXOR(a, b) = (a * b' + a' * b)'$
- $NXOR(a, b) = (a \oplus b)'$

ACTIVIDADES 1.4

- Buscar información sobre las familias lógicas derivadas de la serie 7400.
- Profundizar en las familias lógicas TTL y CMOS más importantes: 74LS, 74HC y 74HTC más importantes.
- Buscar la hojas de características los siguientes componentes: un bloque de 6 inversores, 4 puertas NAND de 2 entradas, 4 puertas OR de dos entradas y 3 puertas NOR de tres entradas.
- ¿Qué es y para qué se utiliza una placa de inserción?

Como el lector habrá podido adivinar, se puede sintetizar cualquier expresión lógica de forma directa utilizando puertas lógicas. En el apartado anterior, se ha podido ver que una función lógica puede representarse con distintas expresiones. Por otro lado, una expresión lógica puede implementarse con distintas puertas lógicas. Las implementaciones de una función lógica se pueden representar mediante diagramas de flujo. En estos diagramas de flujo, las líneas representan señales: o bien variables de entrada, o bien señales que conectan puertas lógicas (valores intermedios), o bien la salida del circuito. Cada puerta lógica se representa con un símbolo. En la siguiente figura se muestran los símbolos que representan cada puerta lógica:

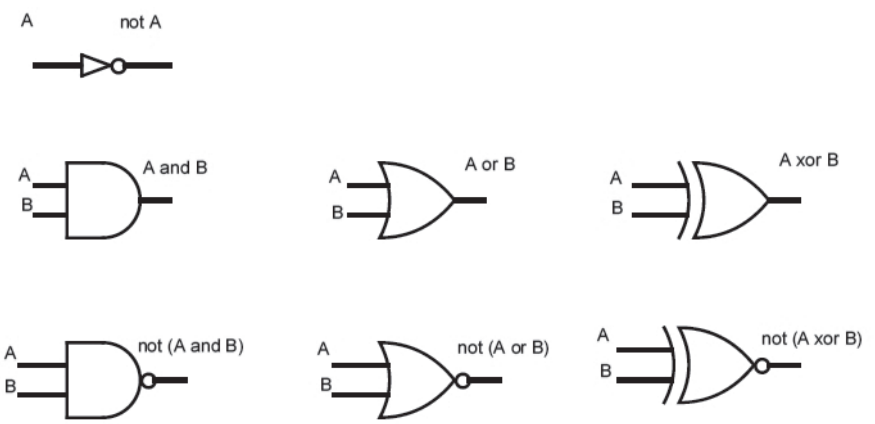


Figura 1.7. *Representación simbólica de puertas lógicas de dos entradas*

EJEMPLO 1.8

Las siguientes figuras muestran distintas implementaciones que pueden realizarse con la función lógica $F(a,b,c,d) = (a * b)' + c + d$.

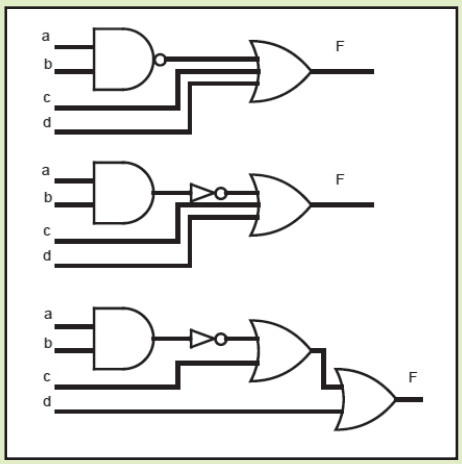


Figura 1.8. *Distintas representaciones de la función F*

En ocasiones no se dispone de todas las puertas lógicas necesarias para implementar un determinado circuito. Por ejemplo, si se desea implementar una función $F(a, b, c) = a + b + c$ bastaría con utilizar una puerta OR de tres entradas. Si no se dispone de dicha puerta podrían utilizarse dos puertas OR de dos entradas. Una puerta AND de 4 entradas puede sustituirse por cuatro puertas AND de dos entradas. Estas sustituciones son posibles gracias a las propiedades conmutativa y asociativa de la suma y producto lógicos.

EJEMPLO 1.9

$M(a, b, c, d) = (a * b) * (c * d) = a * (b * (c * d))$

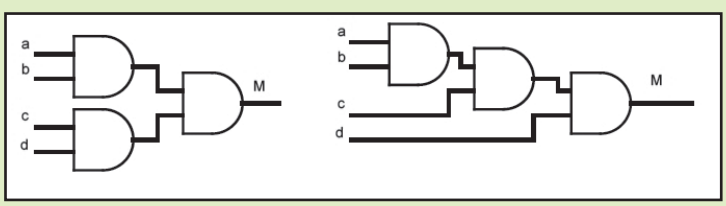


Figura 1.9*. Distintas representaciones de la función M*

Pueden darse situaciones en las que no se disponga de uno o varios tipos de puertas, por ejemplo, no se dispone de puertas AND ni de puertas NAND. En tal caso, se deberá manipular la expresión lógica para que solo aparezcan operadores que podamos implementar con las puertas disponibles.

EJEMPLO 1.10

Para resolver la situación anterior, se pueden utilizar las leyes de De Morgan, vistas anteriormente. De esta manera, la función *M* podría rescribirse de forma que solo requiera una puerta NOR de 4 entradas y 4 inversores: $M(a, b, c, d) = (a' + b' + c' + d')'$. Las leyes de De Morgan nos permiten eliminar los operadores producto o suma lógica de una expresión lógica.

1.2.3 REPRESENTACIÓN DE FUNCIONES LÓGICAS

Las funciones lógicas pueden representarse mediante expresiones lógicas que no son más que constantes y variables relacionadas a través de los operadores lógicos. Una función determinada puede describirse con múltiples expresiones lógicas. Ya se ha descrito en apartados anteriores que las expresiones lógicas pueden implementarse de forma directa mediante puertas lógicas. Esto provoca que una misma función lógica pueda implementarse mediante distintos circuitos. Las propiedades y teoremas del álgebra de Boole nos permiten operar con estas expresiones de forma que se ajusten a nuestras necesidades (minimizar el número de puertas lógicas, utilizar solo puertas lógicas de un tipo, eliminar rebotes o *gliches*…).

ACTIVIDADES 1.5

- Buscar la definición de rebote o *glich*.
- Indicar las implicaciones de los rebotes en la construcción de circuitos digitales.
- Proponer técnicas que permitan solucionarlos.

La existencia de múltiples representaciones no es el único problema cuando se usan expresiones lógicas para representar funciones. Si la función que se desea implementar es compleja, no siempre es fácil obtener una expresión que recoja el comportamiento deseado. Generalmente, es más sencillo describir una función lógica o, lo que es lo mismo, el comportamiento de un sistema utilizando una **tabla de verdad**. Las tablas de verdad indican el valor que debe tomar la salida o salidas del sistema para cada una de las combinaciones de las entradas. Esta representación es única para una determinada función lógica. A continuación se muestra la tabla de verdad de una función lógica $F(a, b, c) = a + b' * c = (a + b') * (a + c)$.

Tabla 1.1. Comparación de la tabla de verdad de dos expresiones lógicas de la misma función. Los valores intermedios se han incluido para facilitar la comprensión de las expresiones lógicas, el lector debe recordar que estos valores no forman parte de la tabla de verdad

Entradas			Valores intermedios		Salida	Valores intermedios		Salida
a	*b*	*c*	*b'*	*b'*c*	*a + b' * c*	(*a+b'*)	(*a +c*)	(*a+b'*) * (*a+c*)
0	0	0	1	0	0	1	0	0
0	0	1	1	1	1	1	1	1
0	1	0	0	0	0	0	0	0
0	1	1	0	0	0	0	1	0
1	0	0	1	0	1	1	1	1
1	0	1	1	0	1	1	1	1
1	1	0	0	0	1	1	1	1
1	1	1	0	1	1	1	1	1

Las tablas de verdad pueden utilizarse también para describir funciones lógicas de las cuales se desconoce su expresión *boolena*. De esta forma, se podría definir un sistema que tomara el valor 1 cuando hubiese un número par de unos en los cuatro bits de entrada mediante una tabla de verdad:

Tabla 1.2. La tabla de verdad contiene los valores de la salida para cada una de las combinaciones de la entrada

Entradas				Salida	Entradas				Salida
A1	A2	A3	A4	F	A1	A2	A3	A4	F
0	0	0	0	0	1	0	0	0	0
0	0	0	1	0	1	0	0	1	1
0	0	1	0	0	1	0	1	0	1
0	0	1	1	1	1	0	1	1	0
0	1	0	0	0	1	1	0	0	1
0	1	0	1	1	1	1	0	1	0
0	1	1	0	1	1	1	1	0	0
0	1	1	1	0	1	1	1	1	1

En aparados anteriores, se ha explicado como sintetizar circuitos digitales a partir de expresiones *booleanas*. Por el contrario, las tablas de verdad permiten definir el comportamiento del sistema a partir de descripciones en lenguaje natural. A continuación, se detallarán procesos para pasar de una tabla de verdad a una expresión *booleana*. De esta forma, las fases de diseño de un sistema digital son las siguientes: definir la función lógica a partir de una tabla de verdad, obtener una de las expresiones equivalentes, implementar dicha expresión mediante puertas lógicas.

A partir de una tabla de verdad pueden definirse de forma directa dos expresiones *booleanas* equivalentes: la **primera forma normal**, también conocida por primera forma canónica o forma normal disyuntiva, y la **segunda forma normal**, también conocida como segunda forma canónica o forma normal conjuntiva.

Antes de ahondar en estas dos expresiones algebraicas se definirán conceptos básicos necesarios para su comprensión:

- **Literal**: variable afirmada o negada. En la expresión $a' + b * c + c'$ los literales son a', b, c y c'.
- **Término en producto**: es un término de una expresión *booleana* compuesto por productos de literales o un único literal. La expresión *booleana* $a + b * c'$ está formada por los términos en producto: a y $b * c'$. La expresión $(a * b)' * c$ **NO** es un término en producto porque $(a * b)'$ no es un literal.
- **Minitérmino**: es un término en producto que contiene todas las variables de una determinada función. La expresión $a * b' * c + a * b' * c'$ de la función lógica $F(a, b, c)$ está forma por los minitérminos $a * b' * c$ y $a * b' * c'$; $a * c$ no es un minitérmino de dicha función porque no contiene todas las variables de la función.
- **Término en suma**: es un término de una expresión *booleana* compuesto por sumas de literales o un único literal. La expresión *booleana* $(a + b) * c'$ está formada por los términos en suma: $a + b$ y c'. La expresión $(a + b)' + c'$ **NO** es un término en suma por $(a + b)'$ no es un literal.
- **Maxitérmino**: es un término en suma que contiene todas las variables de una determinada función. La expresión $(a + b' + c) * (a + b' + c')$ de la función lógica $F(a, b, c)$ está forma por los maxitérminos $a + b' + c$ y $a * b' * c'$. El término $a + c$ no es un maxitérmino de dicha función porque no contiene todas las variables de la función.
- **Suma de productos**: es una expresión *booleana* compuesta por sumas de términos en producto ó de un único término en producto. De esta forma, las expresiones $a + b * c'$, $a + b$, $a * b$, $a * b * c'$, a', $a * b' + a' * b$… pueden considerarse sumas de productos. Por el contrario, las siguientes expresiones $(a * b)' + c$, $(a + b)'$, $(a + b) * c$ no pueden ser consideradas sumas de productos.
- **Producto de sumas**: es una expresión *booleana* compuesta por productos de términos en suma o de un único término en suma. De esta forma, las expresiones $a *(b + c')$, $a + b$, $a * b$, $a + b + c'$, a', $(a + b') * (a' + b)$… pueden considerarse sumas de productos. Por el contrario, las siguientes expresiones $(a + b)' * c$, $(a * b)'$, $(a * b) + c$… no pueden ser consideradas productos de sumas.

Conocidos los conceptos anteriores se puede definir la primera forma normal como *la expresión de una función booleana compuesta por una suma de minitérminos*. Una de las propiedades más importantes de esta expresión es que, al igual que la tabla de verdad, es única para una determinada función lógica. Más interesante que esta propiedad es que a partir de una tabla de verdad podemos extraer de forma más o menos automática la primera forma normal canónica de una función.

Como se ha indicado en su definición: la forma normal disyuntiva no es más que una suma de minitérminos. Los minitérminos de una función son un conjunto finito formado por todas las combinaciones de sus literales. Los minitérminos de cualquier función $F(a, b, c)$ de tres variables serán 8: $a' * b' * c'$, $a' * b' * c$, $a' * b * c'$, $a' * b * c$, $a * b' * c'$, $a * b' * c$, $a * b * c'$ y $a * b * c$. La característica fundamental de un minitérmino es que solo toma el valor 1 para una combinación de las variables que lo forman. De esta forma podemos asociar cada minitérmino con la fila de la tabla de verdad que le da el valor 1. A partir de ahora, denominaremos a los miniterminos con la letra m y un subíndice que nos indica para que valor de la entrada (suponiendo que ésta esté en binario puro) para el cual el término toma el valor 1. De este modo: $m_0 = a' * b' * c'$, $m_1 = a' * b' * c$… y $m_7 = a * b * c$.

Tabla 1.3. Los minitérminos se asocian a la entrada para la cual toman el valor 1

a	b	c	Minitérmino	Expresión
0	0	0	m_0	a′ * b′ * c′
0	0	1	m_1	a′ * b′ * c
0	1	0	m_2	a′ * b * c′
0	1	1	m_3	a′ * b * c
1	0	0	m_4	a * b′ * c′
1	0	1	m_5	a * b′ * c
1	1	0	m_5	a * b * c′
1	1	1	m_7	a * b * c

Esta expresión lógica recibe el nombre de primera forma normal. Recapitulando, a partir de una tabla de verdad podemos obtener la expresión lógica en primera forma normal sumando los minitérminos que se corresponden con las filas de valor 1 en la tabla de verdad. A partir de la tabla 1.2, se puede obtener la expresión de la primera forma normal sumando los minitérminos: $m(3, 5, 6, 9, 10,12, 15) = m_3 + m_5 + m_6 + m_9 + m_{10} + m_{12} + m_{15} = a'b'cd + a'bc'd + a'bcd' + ab'c'd + ab'cd' + abc'd' + abcd$.

Tabla 1.4. Para construir la función en primera forma normal se seleccionan los minitérminos asociados a un valor 1 en la tabla de verdad

Entradas				Salida	m_i	Entradas				Salida	m_i
A1	A2	A3	A4	F	m_i	A1	A2	A3	A4	F	m_i
0	0	0	0	0	0	1	0	0	0	0	8
0	0	0	1	0	1	1	0	0	1	1	9
0	0	1	0	0	2	1	0	1	0	1	10
0	0	1	1	1	3	1	0	1	1	0	11
0	1	0	0	0	4	1	1	0	0	1	12
0	1	0	1	1	5	1	1	0	1	0	13
0	1	1	0	1	6	1	1	1	0	0	14
0	1	1	1	0	7	1	1	1	1	1	15

Análogamente, la segunda forma normal es la expresión *booleana* de una función formada por el producto de maxitérminos de dicha función y al igual que la primera forma normal es única para una función determinada. Esta expresión también puede obtenerse de forma sencilla a partir de la tabla de verdad.

Los maxitérminos al igual que los minitérminos podemos obtenerlos combinado los literales de una determinada función. Los maxitérminos de cualquier función $F(a, b, c)$ de tres variables serán ocho: $a + b + c$, $a + b + c'$, $a + b' + c$, $a + b' + c'$, $a' + b + c$, $a' + b + c'$, $a' + b' + c$ y $a' + b' + c'$. A diferencia del caso anterior, los maxitérminos solo toman el valor *0* para una de las combinaciones de la entrada. De forma semejante, se puede asignar cada maxitérmino a la única combinación de valores de la entrada que hacen la expresión *0*, siendo el maxitérmino M_i aquel que se evalúa como *0* si la entrada toma el valor *i* (*i* está expresado en binario puro).

Tabla 1.5. Los maxitérminos se asocian a la entrada para la cual toman el valor 0

a	b	c	Maxitérmino	Expresión
0	0	0	M_0	a + b + c
0	0	1	M_1	a + b + c′
0	1	0	M_2	a + b′ + c
0	1	1	M_3	a + b′ + c′
1	0	0	M_4	a′ + b + c
1	0	1	M_5	a′ + b + c′
1	1	0	M_5	a′ + b′ + c
1	1	1	M_7	a′ + b′ + c′

La segunda forma normal se obtiene con el producto de los maxitérminos que hacen *0* la función. Siguiendo con el ejemplo anterior, se puede obtener la expresión *booleana* de la función expresada en la tabla 1.2 mediante el producto de los siguiente maxiterminos: $M(0, 1, 2, 4, 7, 8, 11, 13, 14) = M_0 * M_1 * M_2 * M_4 * M_7 * M_8 * M_{11} * M_{13} * M_{14} = (a + b + c + d) * (a + b + c + d') * (a + b + c' + d) * (a + b' + c + d) * (a + b' + c' + d') * (a' + b + c + d) * (a' + b + c' + d') * (a' + b' + c + d')* (a' + b' + c' + d)$.

Tabla 1.6. Para construir la función en primera forma normal se seleccionan los minitérminos asociados a un valor 1 en la tabla de verdad

Entradas				Salida	m_i	Entradas				Salida	m_i
A1	A2	A3	A4	F	m_i	A1	A2	A3	A4	F	m_i
0	0	0	0	0	0	1	0	0	0	0	8
0	0	0	1	0	1	1	0	0	1	1	9
0	0	1	0	0	2	1	0	1	0	1	10
0	0	1	1	1	3	1	0	1	1	0	11
0	1	0	0	0	4	1	1	0	0	1	12
0	1	0	1	1	5	1	1	0	1	0	13
0	1	1	0	1	6	1	1	1	0	0	14
0	1	1	1	0	7	1	1	1	1	1	15

A modo de recapitulación cabe destacar que la primera y la segunda forma normal pueden obtenerse directamente a partir de la tabla de verdad. La primera mediante la suma de los minitérminos asociados a las filas que toman el valor *1* y la segunda mediante el producto de los maxitérminos que toman el valor *0*. Resaltar también, que ambas expresiones son equivalentes. Y por lo tanto el circuito sinterizado a partir de una de ellas realizará la misma función que el sintetizado a partir de la otra.

EJEMPLO 1.11

De este modo, si tenemos la función lógica *F*(*a*,*b*,*c*) descrita por la siguiente tabla de verdad:

Tabla 1.7. Tabla de verdad de una función con 3 entradas

a	b	c	Salida
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	0

De forma que la primera forma normal viene dada por la expresión: F= *m*(2, 4, 5, 6) = *a*′ * *b* * *c*′ + *a* * *b*′ * c′ + a * b′ * c + a * b * c′; y la segunda forma normal viene dada por la expresión F= M(0, 1, 3, 7) = (*a* + *b* + *c*) * (*a* + *b* + *c*′) * (*a* + *b*′ + *c*′) * (*a*′ + *b*′ + *c*′); los circuitos que se sintetizan a partir de ambas expresiones serán equivalentes:

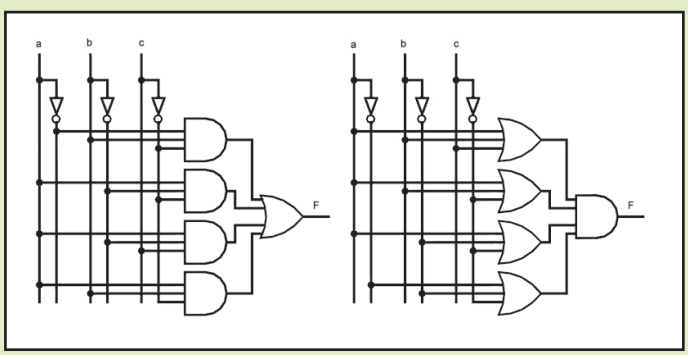


***Figura 1.10**. Los dos circuitos son equivalentes*

ACTIVIDADES 1.6

- Buscar en el apéndice información sobre los conjuntos universales de puertas lógicas. ¿Para qué sirven?

1.2.4 SIMPLIFICACIÓN DE FUNCIONES LÓGICAS

Como se ha comentado en apartados anteriores, una función lógica puede expresarse a través de distintas expresiones *booleanas*. El diseñador escogerá aquellas expresiones que se ajusten más a los requisitos del sistema. Por ejemplo, si se dispone únicamente de puertas OR e inversores, el diseñador deberá eliminar el operador lógico * de la expresión *booleana* a partir de la cual implementará el circuito.

Uno de los criterios más utilizados a la hora de diseñar la expresión *booleana* que representa la función a implementar es minimizar el número de operadores lógicos con los que se consigue minimizar el número de puertas lógicas, maximizando de este modo la integración del circuito y minimizado a su vez el coste económico y el consumo de potencia. Existen multitud de técnicas que permiten realizar este proceso. Una de las más referenciadas en la bibliografía son los mapas de **Karnaugh**. Se trata de un método visual que puede utilizarse cuando el número de variables de entrada es inferior a seis. En la práctica, no es operativo cuando el número de variables es mayor que cuatro.

Existen otras técnicas capaces de lidiar con las limitaciones de este método. En concreto destaca el método de **Quine**-**McCluskey**. Este algoritmo es fácil de implementar en un computador y garantiza que se obtiene la función *booleana* mínima.

ACTIVIDADES 1.7

- Buscar información sobre los métodos de *Quine-McCluskey* y las implicaciones por mapas de *Karnaugh*. ¿En qué casos se podrían necesitar?

En muchas ocasiones hay combinaciones de los valores de entrada que no pueden darse. Por ejemplo, si se codifican los dígitos del 0 al 9 con cuatro bits, hay combinaciones de bits que no tendrán asignado ningún valor. El diseñador deberá identificar estas situaciones. Estas combinaciones de las variables de entrada reciben el nombre de *don't care values*, puesto que podrán marcarse en la tabla de verdad del sistema como 1 o como 0, facilitándose la simplificación de la función *booleana*.

1.3 TIPOS DE CIRCUITOS DIGITALES

En apartados anteriores se ha descrito como utilizar el álgebra de *Boole* en el diseño e implementación de circuitos digitales. En el resto de apartados de este tema se estudiarán como aplicar estas técnicas a distintos sistemas digitales atendiendo a sus peculiaridades.

Los sistemas digitales pueden clasificarse en dos grupos: **sistemas combinacionales** y *sistemas secuenciales*. Los sistemas combinacionales son aquellos en los que *las salidas Z en un instante t dependen exclusivamente del valor de las entradas X en ese mismo instante.* Estos sistemas pueden caracterizarse con una función *booleana* por cada una de las salidas, $Z(t) = F\ (X(t))$. El comportamiento de estos sistemas puede describirse fácilmente por una tabla de verdad.

En los sistemas secuenciales, la *salida Z en un determinado instante de tiempo t depende de las entradas X y del estado S en ese mismo instante de tiempo t, el estado S en dicho instante dependerá del valor de las entradas X en todos los instantes anteriores*. Para permitir este comportamiento es necesario que el sistema almacene su estado. Podemos describir un sistema secuencial mediante dos funciones *booleanas*. La primera describe el valor de las salidas y depende del estado y en algunos sistemas, también de la entrada $Z(t) = G(S(t), X(t))$. La segunda describe el valor del estado en el instante siguiente y depende del estado actual y del valor de las entradas $S(t+1) = H(S(t), X(t))$.

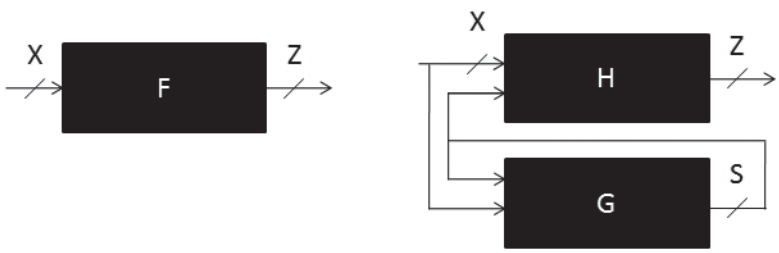


Figura 1.11. *La figura de la izquierda muestra el diagrama de bloques de un circuito combinacional y el de la derecha, el de un circuito secuencial*

Existen dos tipos de sistemas secuenciales: sistemas síncronos y asíncronos. Los circuitos secuenciales asíncronos son aquellos en los que los cambios en sistemas se producen en función de cambios en el estado o en las entradas. Por el contrario, los síncronos son sistemas secuenciales solo pueden cambiar de estado en determinados instantes de tiempo, estos instantes vienen marcados por una señal de reloj. El sistema solo hace caso de las entradas y de su estado interno en determinados instantes. Dependiendo de en qué momento se actualice el estado, podemos distinguir entre dos tipos de sistemas: activos por nivel y activos por flanco. Los sistemas activos por nivel actualizan su valor cuando el reloj está o bien a nivel alto, o bien esta a nivel bajo. Por el contrario, los sistemas activos por flanco cambian el valor de sus señales cuando se produce un flanco de bajada o cuando se produce un flanco de subida.

1.4 CIRCUITOS COMBINACIONALES

Como ya se ha enunciado anteriormente, los circuitos combinacionales son aquellos en los cuales las salidas solo dependen del valor de las entradas en dicho instante. Dichos circuitos se pueden definir utilizando una función lógica para cada una de sus entradas. Los pasos para diseñar un circuito combinacional son:

1 Definir la tabla de verdad de cada una de las salidas del sistema.

2 Crear una expresión *booleana* que defina el comportamiento de cada una de las salidas. En apartados anteriores se ha detallado como obtener la primera y la segunda forma normal a partir de una tabla de verdad.

3 Operar con la función lógica para que se ajuste a los requisitos del sistema. Utilizando los principios del álgebra de Boole o alguno de los métodos de simplificación de funciones existentes, la expresión lógica se puede transformar para ajustarse a los requisitos del sistema.

4 Implementar la función lógica obtenida mediante puertas lógicas. En apartados anteriores se describió como implementar funciones lógicas mediante puertas AND, OR y NOT y otros conjuntos de puertas lógicas universales. Recordar al lector que funciones lógicas en suma de productos y en producto de sumas pueden implementarse de forma directa mediante puertas NAND y NOR respectivamente.

La implementación de circuitos mediante puertas lógicas no es siempre la más adecuada. Cuando la complejidad del diseño es grande, se impone el uso de un diseño jerárquico y modular. Este diseño se basa en dividir el problema en bloques que realizan tareas complejas. Estos bloques, a su vez, pueden dividirse en bloques de menor complejidad hasta llegar a un nivel de puerta lógica. De esta forma, el diseño del sistema global se divide en el diseño de componentes cada vez más sencillos. Esta metodología recibe en nombre de metodología *top-down* (diseño de arriba a abajo).

A la hora de realizar el diseño de un circuito digital existen bloques combinacionales básicos que se pueden utilizar en la implementación del circuito. Además de los bloques convencionales básicos existen dispositivos formados por conjuntos de puertas lógicas y/o módulos básicos (combinacionales y/o secuenciales) cuyas conexiones pueden ser programadas, facilitando el desarrollo circuitos digitales. En los apartados siguientes se detallarán algunos de los bloques básicos más importantes así como algunos ejemplos de circuitos lógicos programables (PLD).

1.4.1 BLOQUES BÁSICOS

1.4.1.1 Multiplexores

Los **multiplexores** son circuitos combinacionales que permiten poner uno de los valores de la entrada en la salida. Dichos circuitos se caracterizan por tener 2^n entradas, 1 salida y n entradas de control. La salida tomará el valor de una de las 2^n entradas dependiendo del valor de las n señales de control. Adicionalmente, puede añadirse una entrada de control de activación o *enable*, de forma que si dicha entrada toma el valor 0 la salida tomará el valor 0 independientemente del valor de las entradas y de las señales de control. A continuación, se muestra la tabla de verdad, el diagrama de bloques de alto nivel y la implementación a nivel puerta lógica de un multiplexor 4 a 1 con señal de *enable* o activación.

Tabla 1.8. Tabla de verdad de un multiplexor 4 a 1

Entradas			Salida
Control 1	Control 0	*Enable*	MUX
0	0	0	0
0	1	0	0
1	0	0	0
1	1	0	0
0	0	1	Entrada 0
0	1	1	Entrada 1
1	0	1	Entrada 2
1	1	1	Entrada 3

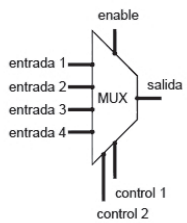


***Figura 1.12**. Diagrama de bloques de un multiplexor 4 a 1*

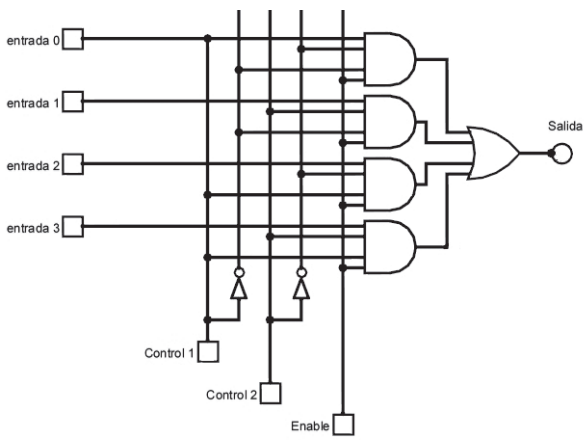


***Figura 1.13.** Implementación de un multiplexor 4 a 1*

EJEMPLO 1.12

Un ejemplo de la utilización de multiplexores (aunque no digitales como los que se ven aquí) se encuentra en las líneas telefónicas. Éstas usan exactamente el principio explicado. Transmiten varias llamadas telefónicas (señales de audio) a través de un único par cableado usando la técnica de multiplexado, de manera que cada señal de audio va únicamente al receptor al que está destinado.

Si no se dispone de un multiplexor con el número de entradas requerido, esté puede sintetizarse con multiplexores con un número de entradas menor. A continuación se muestra un ejemplo de cómo crear un multiplexor de 4 a 1 a partir de 3 multiplexores de 2 a 1. De forma análoga a partir de 5 multiplexores de 4 a uno puede crearse un multiplexor de 16 a 1.

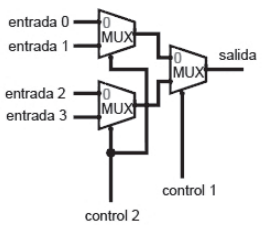


Figura 1.14. *Implementación de un multiplexor 4 a 1 a partir de multiplexores 2 a 1. Si los multiplexores 2 a 1 dispusiesen de señal de enable, se conectarían a la misma señal*

Los multiplexores pueden utilizarse para implementar circuitos digitales a partir de su tabla de verdad o de una función en primera forma normal. La salida de un multiplexor toma el valor de la entrada x_i cuando las señales de control codifican en binario el valor i. El valor i en binario puro se corresponde con la secuencia de unos y ceros que dan el valor *1* al minitérmino i. Si tomamos como señales de control las entradas del sistema, podemos asociar cada una de las entradas del multiplexor con un minitérmino. De esta forma, las entradas cuyo minitérmino tome el valor 1 en la función que se desea implementar, se colocarán a 1 y el resto de entradas a 0. La siguiente figura muestra la implementación de la función $F(a, b) = a'*b + a*b'$ utilizando un multiplexor 4 a 1.

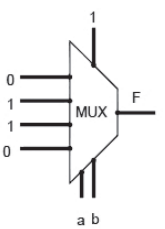


Figura 1.15*. Implementación de una función lógica utilizando un multiplexor 4 a 1 con señal de enable*

Desafortunadamente, en muchas ocasiones no se dispondrá de un multiplexor con tantas señales de control como entradas tenga el sistema. Pero aún en estos casos el podemos usar multiplexores en la implementación de nuestro circuito. Los multiplexores no son más que circuitos que realizan la suma de todos los minitérminos de las señales de

control, multiplicados por la entrada correspondiente. De esta forma, la función $F(a,b,c,d) = a * b + a * b' * (c + d') + a' * b' * d'$ puede implementarse con un multiplexor 4 a 1, donde a y b son las señales de control, reescribiendo la función como $f(a,b,c,d) = a * b * x_3 + a * b' * x_2 + a' * b * x_1 + a' * b' * x_0$, donde $x_0 = d'$, $x_1 = 0$, $x_2 = c + d'$ y $x_3 = 1$.

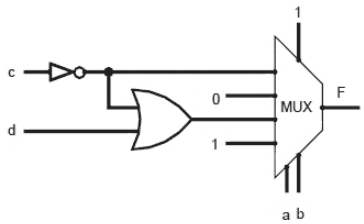


Figura 1.16. *Implementación de una función lógica de 4 variables de entrada utilizando un multiplexor 4 a 1 con señal de enable*

ACTIVIDADES 1.8

- Busca en Internet que es una ALU (Unidad Aritmético y Lógica).
- Diseña una ALU de un bit que sea capaz de hacer las operaciones AND, OR, NOT, dependiendo de unos valores de control que indiquen la operación a realizar (vea la nota lateral).

Para la realización de la actividad 15, tendrás que utilizar una puerta AND de dos entradas, una puerta OR de dos entradas y una puerta OR de una entrada. Además, para seleccionar la operación a realizar se deberá utilizar un multiplexor de tres a uno.

1.4.1.2 Demultiplexores

Los **demultiplexores** son circuitos combinacionales con 1 entrada, n señales de control y 2^n salidas. En estos circuitos todas las salidas tomarán el valor 0 excepto aquella seleccionada por el valor de la señales de control que tomará el valor de la entrada. Al igual en los multiplexores, puede añadirse una señal de activación o *enable* de forma que si esta toma el valor *0* todas las salidas toman el valor 0 y si toma el valor 1, el demultiplexor tiene el comportamiento descrito anteriormente. Puesto que este circuito tiene 2^n salidas, será necesario definir 2^n funciones lógicas para definir el comportamiento del circuito.

Tabla 1.9. Tabla de verdad de un demultiplexor 1 a 4

Entradas			Salidas			
Control 1	Control 0	*Enable*	Salida 0	Salida 0	Salida 0	Salida 0
0	0	0	0	0	0	0
0	1	0	0	0	0	0
1	0	0	0	0	0	0
1	1	0	0	0	0	0
0	0	1	Entrada	0	0	0
0	1	1	0	Entrada	0	0
1	0	1	0	0	Entrada	0
1	1	1	0	0	0	Entrada

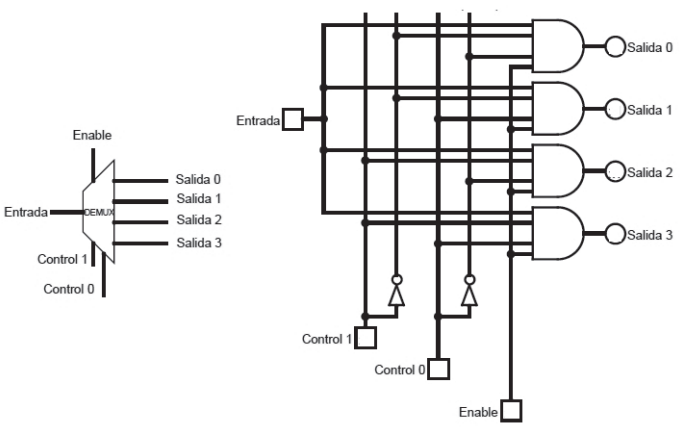


***Figura 1.17**. A la izquierda, la descripción de un demultiplexor a nivel de bloques. A la derecha, la implementación de un demultiplexor a nivel de puerta lógica*

ACTIVIDADES 1.9

- ¿Cuántas entradas de control son necesarias para seleccionar los datos presentas en un demultiplexor de seis entradas?

1.4.1.3 Decodificadores

Los **decodificadores** son circuitos combinacionales que activan una única salida dependiendo del valor de las entradas. Se caracterizan por tener n entradas y 2^n salidas. Estos circuitos activan la salida correspondiente al número en binario puro codificado en la entrada. Su comportamiento es similar al demultiplexor salvo que el valor que toma la salida activada siempre es 1. Al igual que en todos los circuitos anteriores podemos añadir una señal de activación o *enable*. Si dicha señal toma el valor 1, el circuito se comportará de la forma ya descrita y si toma el valor 0 todas las señales de salida tomarán el valor 0 también. Como sucede con los demultiplexores (y con cualquier circuito que tenga más de una salida), se deberá definir una función lógica por cada una de sus salidas.

Tabla 1.10. Tabla de verdad de un decodificador 1 a 4

Entradas			Salidas			
Control 1	Control 0	*Enable*	Salida 0	Salida 0	Salida 0	Salida 0
0	0	0	0	0	0	0
0	1	0	0	0	0	0
1	0	0	0	0	0	0
1	1	0	0	0	0	0
0	0	1	1	0	0	0
0	1	1	0	1	0	0
1	0	1	0	0	1	0
1	1	1	0	0	0	1

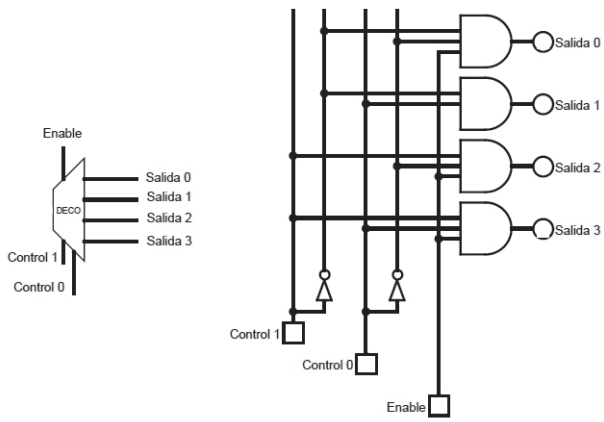


***Figura 1.18**. A la izquierda, la descripción de un decodificador a nivel de bloques. A la derecha, la implementación de un decodificador a nivel de puerta lógica*

De forma análoga a como se hacía con los multiplexores, los decodificadores pueden agruparse de forma jerárquica formando unidades con un mayor número de entradas. A continuación se muestra como construir un decodificador de 4 a 16 a partir de 2 decodificadores de 2 a 4.

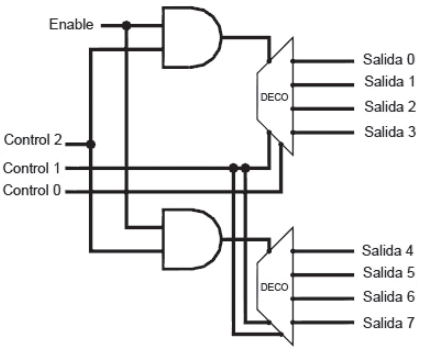


Figura 1.19. *Implementación de un decodificador 3 a 8 a partir de decodificadores 2 a 4*

Al igual que los multiplexores, los decodificadores también pueden utilizarse para implementar circuitos digitales a partir de tablas de verdad o de funciones en primera forma normal. La salida y_i de un decodificador toma el valor 1 cuando las entradas codifican en binario puro el valor de i, es decir, si conectamos las entradas del decodificador a las entradas de la función lógica, podemos asociar cada salida del decodificador con un minitérmino de la función. De esta forma solo tendremos que unir mediante una puerta OR todos los minitérminos que hagan 1 la función lógica. Si hay que implementar distintas funciones lógicas con las mismas entradas se podrá reutilizar el mismo decodificador para calcular los minitérminos de ambas funciones. La siguiente imagen muestra las implementación de las funciones $F(a, b, c) = a * b * c + a * b' * c' + a' * b' * c'$ y $G(a, b, c) = a * b * c + a * b * c'$.

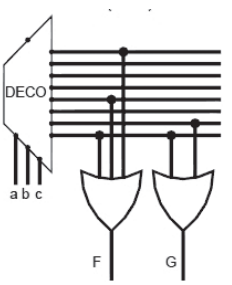


Figura 1.20. *Implementación de 2 funciones lógicas de 3 variables de entrada utilizando un decodificador de 3 a 8*

1.4.1.4 Codificadores

Un **codificador** es un dispositivo combinacional con 2^n entradas, n salidas y una señal de actividad A. Las salidas codifican en binario puro el único bit activo de las entradas. Si todas la entradas están desactivadas, todas salidas tomarán el valor 0 y la señal A también lo tomará. Si todas las señales de entrada menos una toman el valor 0, el codificador se comportará de la forma descrita y la salida A tomará el valor 1. Si están activas varias entradas, el valor de las salidas no es estable. Al igual que en el resto de componentes vistos, se puede añadir una señal *enable.* Si dicha señal toma el valor 0, todas las salidas del circuito (incluyendo la señal A) tomarán el valor 0.

Tabla 1.11. Tabla de verdad de un codificador 8 a 3 sin prioridad. La columna de las entradas indica que entrada está activa

Entradas		Salidas			
Entradas	*Enable*	Salida 0	Salida 1	Salida 2	Activación
-	0	0	0	0	0
X_0	0	0	0	0	1
X_1	0	0	0	1	1
X_2		0	1	0	1
X_3	0	0	1	1	1
X_4	1	1	0	0	1
X_5	1	1	0	1	1
X_6	1	1	1	0	1
X_7	1	1	1	1	1

La tabla de verdad anterior define el comportamiento de un decodificador de 8 a 3. A partir de la tabla anterior, se pueden obtener las siguientes funciones lógicas:

- ✓ $activación = enable * (x_0 + x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7)$
- ✓ $salida_0 = enable * (x_1 + x_3 + x_5 + x_7)$
- ✓ $salida_1 = enable * (x_2 + x_3 + x_6 + x_7)$
- ✓ $salida_2 = enable * (x_4 + x_5 + x_6 + x_7)$

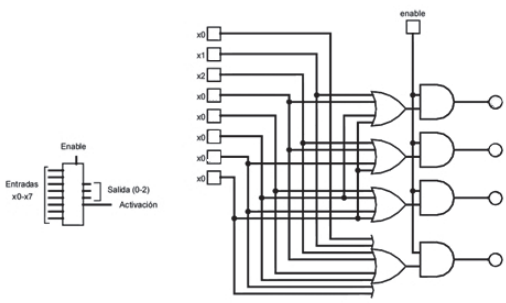


***Figura 1.21**. Implementación de un decodificador de 8 a 3 sin prioridad*

Existen tipos de codificadores que permiten que más de una señal de entrada esté activa a la vez. Dichos codificadores reciben el nombre de codificadores con prioridad. En estos circuitos, la salida toma el valor de la entrada activa con mayor peso. Por ejemplo, si las entrada x_1 y x_3 toman en valor *1*, la salida y tomará el valor 3 por tratarse de la entrada con mayor peso. El diseño de este circuito es bastante sencillo. Solo hay que implementar un circuito que resuelva las prioridades de las entradas. A continuación se muestra dicho circuito para un codificador de 4 entradas.

Los codificadores son circuitos encargados de convertir una información expresada en un código a otro código diferente, por ejemplo, la información que se genera cuando se escribe en el teclado del ordenador debe ser convertida a binario para que el procesador la pueda utilizar.

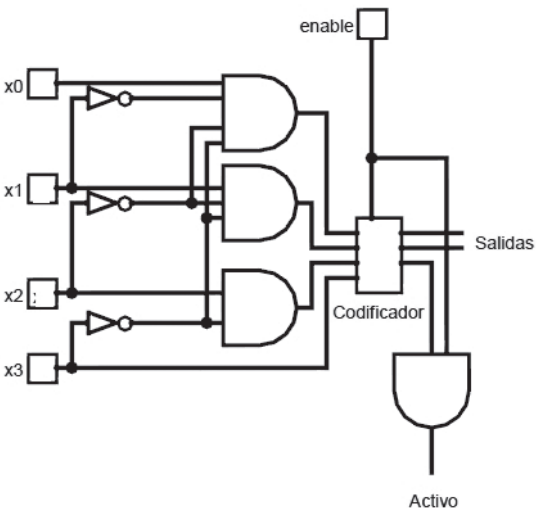


Figura 1.22. *Implementación de un decodificador de 8 a 3 con prioridad*

¿SABÍAS QUE...?

En inglés:
- Codificador: *encoder*.
- Decodificador: *descrambler.*
- Multiplexor: *multiplexer.*
- Demultiplexor: *demultiplexer.*

1.4.1.5 Otros bloques combinacionales

Además de los bloques descritos en los apartados anteriores existen numerosos más. Está fuera del ámbito de este libro describirlos todos. A continuación se presenta una lista con algunos de los ejemplos más importantes:

- **Desplazadores** o *shifters*. Son dispositivos con n+2 entradas y n salidas. Estos dispositivos permiten desplazar los bits de la entrada hacia la izquierda o hacia la derecha.

- **Comparadores**. Son dispositivos con 3 salidas que indican si dos operandos son iguales o bien si el primero es mayor que el segundo o bien si el segundo es mayor que el primero.
- **Detectores** o **generadores de paridad**. Estos dispositivos generalmente toman un valor 1 si el número de señales activas en la entrada es impar y un valor cero en el caso contrario. También existen dispositivos que funcionan de forma inversa: toman un valor 0 si el número de unos de la entrada es par y 1 si es impar. De la misma forma, podemos encontrar circuitos que detectan la paridad de las señales que tienen un valor de tensión bajo ó 0.
- **Sumador elemental** o *half adder*. Este componente suma dos entradas de un bit. Además de la suma de los dos bits genera otra salida indicando si se ha producido acarreo en la operación.
- **Sumador completo** o *full adder*. Este dispositivo suma dos entradas de un bit y un acarreo de entrada. Al igual que el *half adder* genera una salida con la suma de la operación y otra con el acarreo.

ACTIVIDADES 1.10

- ¿Cuáles son las diferencias entre un multiplexor y un demultiplexor?
- Diseñar la tabla de verdad, la función *booleana* y la descripción a nivel de puerta lógica de los siguientes circuitos combinacionales: un comparador, un detector de paridad, un sumador elemental y un sumador completo.

1.4.2 DISPOSITIVOS LÓGICOS PROGRAMABLES (PLD)

Los **dispositivos lógicos programables** se encuadran dentro de un conjunto de circuitos integrados formados por puertas lógicas y/o módulos básicos (tanto combinacionales como secuenciales), cuyas interconexiones pueden ser programas o bien por el usuario o bien por el fabricante. Este apartado se centrará en tres de los dispositivos lógicos programables más simples: las memorias ROM, los dispositivos tipo PLA y los dispositivos tipo PAL. Los distintos dispositivos se clasifican atendiendo a la flexibilidad con la que se programan y a su capacidad. Los circuitos que se verán a lo largo de este apartado se caracterizan por poseer entre 200 y 1000 puertas lógicas. Además de estos, se pueden encontrar dispositivos con mayor capacidad como pueden ser los dispositivos CPLD (matrices de dispositivos PLD interconectados) y dispositivos de tipo FPGA (compuestos de bloques lógicos sin interconexiones prefijadas).

ACTIVIDADES 1.11

- Buscar información adicional en el apéndice relacionada con los dispositivos lógicos programables.

1.5 CASO PRÁCTICO

En este caso práctico, se iniciará al lector en el montaje de circuitos reales. Para ello, se mostrarán los dispositivos y el manejo de algunos de ellos que deben emplearse a la hora de crear un circuito electrónico.

1.5.1 EL MULTÍMETRO DIGITAL

El **multímetro** digital o **tester** digital, es un instrumento electrónico de medición que generalmente calcula voltaje, corriente y resistencia. Dependiendo del modelo, también puede llegar a medir otras magnitudes como capacitancia y temperatura. Gracias a este dispositivo podemos comprobar el correcto funcionamiento de los componentes y circuitos electrónicos.

Figura 1.23 . *Multímetro digital*

Es muy importante leer el manual de operación de cada multímetro en particular, pues en él, el fabricante fija los valores máximos de corriente y tensión que puede soportar y el modo más seguro de manejo, tanto para evitar el deterioro del instrumento como para evitar accidentes al usuario.

De manera general, proveen dos terminales cuya polaridad se identifica mediante colores: negro con polaridad negativa (-) y rojo con polaridad positiva (+).

En las medidas de corriente directa (CD), la polaridad de los terminales debe ser observada para conectar apropiadamente el instrumento. Esta precaución no es necesaria para las medidas de corriente alterna (CA).

Poseen una llave selectora para elegir el tipo de medida a realizar:

- **Tensión eléctrica**: la unidad de medida es el Voltio (V). Pueden medir tanto voltajes en circuitos de corriente directa o continua, simbolizada como "DC" ó "-", como de corriente alterna, simbolizada como "AC" ó "~". Por ello, dependiendo del tipo de corriente, se debe elegir una de estas dos opciones en el correspondiente selector de funciones, también se debe escoger la escala y colocar las puntas de medición en los bornes apropiados.
- **Corriente eléctrica**: la unidad por el Sistema Internacional corresponde al Amperio, sin embargo, esta cantidad es muy grande para el tipo de mediciones a realizar. Es por ello que siempre la escala que se utiliza está en mili Amperios, (mA) la milésima parte de un amperio. Puede ser usado para medir corrientes en circuitos de *corriente directa* y de *corriente alterna*. Recuerda que se debe seleccionar la opción deseada, escoger la escala y colocar las puntas de prueba apropiadamente.
- **Resistencia**: la unidad de medida es el Ohm (W). Nunca debe conectarse a un circuito con la fuente de energía activada. En general, la resistencia debe ser aislada del circuito para medirla.

Con ayuda del profesor, coge una pila de 1,5 V algo gastada, para ver en qué estado se encuentra la misma. Para realizar la medición de voltajes, colocamos la llave selectora del multímetro en el bloque DCV (Voltaje de Corriente Continua), puesto que la pila constituye un generador de corriente continua.

Colocamos la punta roja en el electrodo positivo de la pila, la punta negra en el negativo. Apunta el voltaje resultante de la medición.

1.5.2 PLACAS DE INSERCIÓN

La **placa de inserción** es elemento sobre el que se montan todos los circuitos integrados, componentes pasivos y los cables para realizar las conexiones adecuadas entre ellos. Se utilizan para el montaje rápido de circuitos ya que no necesitan ningún tipo de soldadura.

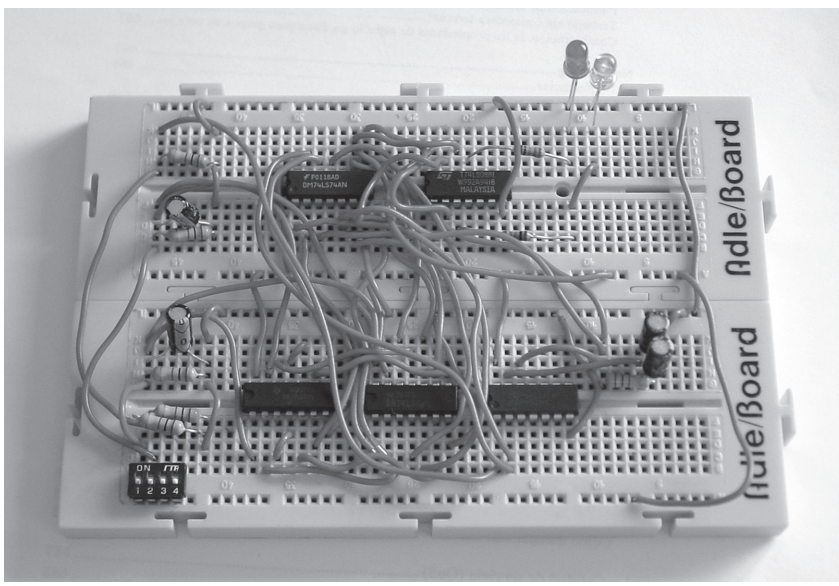


Figura 1.24. *Placa de inserción con componentes conectados*

Esta placa se forma a partir de una matriz de agujeros donde se pueden insertar los componentes por simple presión. Dichos agujeros poseen uniones eléctricas por la parte inferior de la placa de inserción, de manera que dos componentes que se pinchemos en dos agujeros unidos eléctricamente se comportarán como si se hubieran conectado entre sí.

En general, en la zona central de la placa se sitúan tiras de cinco contactos, unidos internamente entre sí para la conexión los diferentes componentes. En la parte lateral de la placa, se pueden encontrar tiras de mayor dimensión también unidas internamente entre sí. Estas tiras suelen estar reservadas para la alimentación del circuito.

Para establecer conexiones entre unas tiras y otras se utilizan cables de hilo rígido (a ser posible de diferentes colores) de un diámetro similar al de los huevos de la placa.

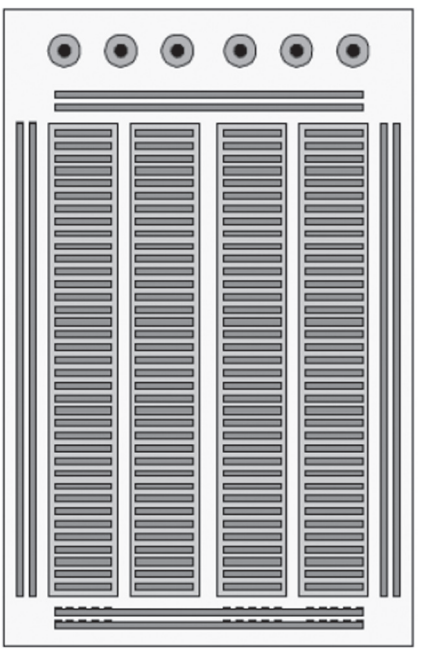


Figura 1.25. *Organización de una placa de inserción*

Los componentes integrados deben situarse sobre las divisiones entre tiras (tal y como se muestra en la figura 1.26), ya que en caso contrario los pines opuestos quedarían conectados entre sí.

Monta en el laboratorio los circuitos integrados 7408 y 7483. Comprueba su tabla de verdad utilizando el multímetro digital.

Con ayuda del profesor, coloca un componente integrado (por ejemplo, una puerta AND o una puerta OR), conéctalas de manera que la salida de una sea la entrada de la otra y analiza con el multímetro digital el resultado final. Recuerda, que la puerta lógica tomará el valor de 1 cuando le llegue corriente y la señal de 0 cuando no sea así.

Para un análisis más fácil del circuito, se recomienda la utilización de un componente integrado *Interruptor* para la entrada de señales y el uso de LED para visualizar la salida.

RESUMEN DEL CAPÍTULO

En este capítulo se introduce al lector en el diseño de sistemas electrónicos digitales definiéndolos como aquellos sistemas, construidos utilizando tecnología electrónica, capaces de tratar señales digitales. Las señales digitales son aquellas que representan la variación de una magnitud digital a lo largo del tiempo. Se pueden tratar señales analógicas utilizando circuitos digitales siempre y cuando éstas se conviertan a una señal digital utilizando un conversor A/D. La salida de estos sistemas también podrá ser digital siempre y cuando se transforme la señal de salida utilizando un conversor D/A.

En la actualidad, los circuitos digitales codifican la información que tratan de forma binaria, es decir, con dos valores: el 0 o valor falso y el 1 o valor cierto. Los distintos componentes electrónicos codifican estos valores mediante tensiones. Los valores de tensión utilizados dependerán de la tecnología.

Claude E. Shannon propuso el uso de las expresiones del álgebra de Boole en la descripción de circuitos digitales. De esta forma, el álgebra de Boole se convierte en una poderosa herramienta en la síntesis y análisis de estos circuitos. Las expresiones lógicas del álgebra de Boole pueden trasformarse directamente en circuitos digitales mediante puertas lógicas. Las puertas lógicas son componentes electrónicos que definen expresiones lógicas básicas. Las puertas lógicas se representan mediante símbolos gráficos, permitiendo describir circuitos mediante diagramas de flujo que conectan distintas puertas lógicas.

Una determinada función lógica puede representarse por medio de distintas expresiones lógicas (constantes y variables combinadas utilizando operadores). Las tablas de verdad y las expresiones lógicas permiten describir una determinad función lógica de forma única, facilitando el diseño de circuitos digitales.

Existen dos tipos de circuitos digitales. Los circuitos combinacionales y los circuitos secuenciales. En los primeros, la salida depende exclusivamente del valor de las entradas y en los segundos, también depende del estado del sistema, es decir, de los valores de las entradas en instantes anteriores.

- ✓ El diseño de los circuitos combinacionales se divide en las siguientes etapas:
- ✓ Definir la tabla de verdad de cada una de las salidas del sistema.
- ✓ Crear una expresión *booleana* que defina el comportamiento de cada una de las salidas.
- ✓ Operar con la función lógica para que se ajuste a los requisitos del sistema.
- ✓ Implementar la función lógica obtenida mediante puertas lógicas.

Las puertas lógicas pueden agruparse en componentes de más alto nivel que facilitan la síntesis de circuitos más complejos. Entre los componentes combinacionales básicos destacan: los mutiplexores, los demultiplexores, los codificadores, los decodificadores, los sumadores, los desplazadores y los detectores de paridad.

EJERCICIOS PROPUESTOS

1. Indique las ventajas e inconvenientes de los sistemas digitales frente a los analógicos.

2. Obtener el valor de las funciones lógicas se muestran a continuación para los valores de las entradas indicados:

- $F(a, b, c) = a + bc'$ si ($a = 0, b = 0$ y $c = 1$)
- $F(a, b, c) = a + (bc)'$ si ($a = 0, b = 0$ y $c = 1$)
- $F(a, b, c) = (a + b)\, c'$ si ($a = 0, b = 0$ y $c = 1$)
- $F(a, b, c, d) = (ac + db)\, (c'+d)'$ si ($a = 1, b = 0$, $c = 1$ y $d = 1$)

3. Transformar las siguientes expresiones en sumas de productos:

- $F(a, b, c, d) = (a + bc')d'$
- $F(a, b, c, d) = ((a + bc')d')'$
- $F(a, b, c, d) = (ab + bc')'+ (cd)'+ b$
- $F(a, b, c, d) = ((ad)' + bc')\, (a + d)\, (b + c)'$

4. Transformar las siguientes expresiones en productos de sumas:

- $F(a, b, c, d) = (a + bc')d'$
- $F(a, b, c, d) = ((a + bc')d')'$
- $F(a, b, c, d) = (ab + bc')'+ (cd)'+ b$
- $F(a, b, c, d) = ((ad)' + bc')\, (a + d)\, (b + c)'$

5. Simplificar las siguientes funciones utilizando los teoremas y propiedades del álgebra, el resultado deberá expresarse en suma de productos.

- $F(a, b, c) = (ab'+c)'\, (b+c)'$
- $F(a, b, c, d) = ((a+b)\, (a+c'))'+d(a+a'b+a'b')$
- $F(a, b, c, d) = ((a+c')'\, (b + d'))'+ (c'+ d')'$

6. Obtener el complemento de las siguientes funciones en forma de sumas de productos.

- $F(a, b, c) = (a+ b')\, (a'+ c)\, (c'+ b')$
- $F(a, b, c, d) = (a + b + c'+ d')\, (b'+ d)\, (b'+ c')$

7. Obtener el complemento de las siguientes funciones en forma de productos de sumas.

- $F(a, b, c) = ab'+ a'c + c'b'$
- $F(a, b, c, d) = abc'd'+ b'd + b'c'$

8. Obtener la tabla de verdad de las siguientes funciones lógicas:

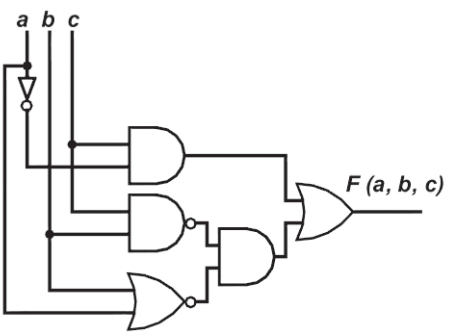
- $F(a, b, c) = a'b + (ac)'$
- $F(a, b, c) = a'(b + a)'+ c$

9. A partir de las tablas de verdad construidas en el apartado anterior obtener la primera y la segunda forma normal de las funciones *booleanas* del ejercicio 8.

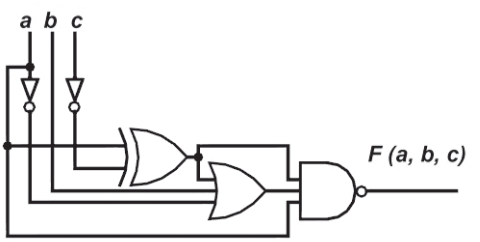
10. Implementar las siguientes funciones utilizando puertas lógicas:

- $F(a, b, c, d) = a'bd + acd' + dc'$
- $F(a, b, c, d) = a'(b + a)'d + c + d'$

11. Obtener expresiones *booleanas* equivalentes a los siguientes circuitos lógicos:



Circuito lógico A



Circuito lógico B

- **12.** Operar con las siguientes expresiones algebraicas para que puedan ser implementadas utilizando exclusivamente puertas OR e inversores:
  - F(a, b ,c) = $abc+a'b'+c'$
  - F(a, b ,c) = $(a'+b+c')(a+b)c$
- **13.** Construir la tabla de verdad de una puerta XOR de 2 entradas. A partir de la taba de verdad, obtener sus representaciones en primera y segunda forma normal. Implementar a nivel de puerta lógica las dos expresiones anteriores.
- **14.** Construir la tabla de verdad de la función lógica $F(a, b, c)$ que da el valor 1 a los minitérminos 0, 5 y 7. Expresar esa función en segunda forma normal. Implementar el circuito equivalente a la expresión anterior utilizando puertas AND y OR de dos entradas e inversores.
- **15.** Se desea diseñar un circuito con 4 bits de entrada y 5 de salida. El primer bit de salida S_0 tomará el valor uno si todos los bits de la entrada (a, b, c, d) toman el valor 0. El bit de salida S_1 tomará el valor 1 si alguno de los bits de la entrada toma el valor 1 y el resto se mantienen a 0. De la mima forma S_2 tomará el valor 1 si 2 bits de la entrada toman el valor 1 y el resto toma el valor 0 y S_3 tomará el valor 1 si y solo 3 bits de la entrada toman el valor 1. Por último, S_4 tomará el valor 1 si todos los bits de la entrada toman el valor 1.
  - Construir la tabla de verdad de cada una de las salidas.
  - Obtener las expresiones en primera y segunda forma normal de cada una de las salidas.
  - Diseñar los circuitos que implementan cada una de las salidas utilizando puertas.
- **16.** Se desea diseñar un circuito que indique si en los 3 bits de entrada los unos están salteados (combinaciones del tipo $a = 0$, $b = 1$, $c = 0$ o $a = 1$, $b = 0$, $c = 1$) o si dos o más unos son consecutivos (combinaciones del tipo $a = 0$, $b = 1$, $c = 1$ o $a = 1$, $b = 1$, $c = 0$). En el primer caso se activará una señal S_0 y en el segundo una señal S_1.
  - Construir la tabla de verdad de cada una de las salidas.
  - Obtener las expresiones en primera forma normal para cada una de las salidas.
  - Diseñar el circuito que implemente la salida S_0 utilizando decodificadores de 3 a 8 y puertas OR.
  - Diseñar el circuito que implemente la salida S_1 utilizando multiplexores de 8 a 1.

TEST DE CONOCIMIENTOS

1 Los sistemas electrónicos se dividen en analógicos y digitales:

a) Los dispositivos digitales solo procesan información digital, por este motivo su rango de aplicación es limitado.

b) Los sistemas digitales pueden utilizarse para procesar señales analógicas si éstas se discretizan utilizando conversores A/D, pero de ningún modo pueden devolver señales analógicas.

c) Los sistemas digitales pueden utilizarse para procesar señales analógicas si éstas se discretizan utilizando conversores A/D.

d) Los sistemas digitales pueden utilizarse para procesar señales analógicas si éstas se discretizan utilizando conversores D/A, pero de ningún modo pueden devolver señales analógicas.

2 Indicar cuáles de las siguientes magnitudes son digitales:

a) La diferencia de potencial entre dos puntos de un circuito.
b) La temperatura de una habitación.
c) El número de libros de una biblioteca.
d) La posición de un interruptor con dos estados (encendido y apagado).

3 Indicar cuál de los siguientes elementos tiene naturaleza binaria:

a) La posición de un interruptor con dos estados (encendido y apagado).
b) El timbre de una casa.
c) La temperatura indicada por un termómetro digital.
d) Un testigo de avería en el frontal de un coche.

4 Indicar cuál de las siguientes afirmaciones es cierta:

a) Los sistemas electrónicos codifican la información binaria con un valor de tensión. Dicho valor depende de la tecnología.
b) Los sistemas electrónicos codifican la información binaria con un valor de tensión. Valores entorno a los 5 V se reservan para el valor lógico 0 y valores en torno a los 0 V se reservan para valores lógicos de 1.
c) Los sistemas electrónicos codifican la información binaria con valores de intensidad de corriente. Dichos valores dependen de la tecnología.
d) Ninguna de las anteriores es cierta.

5 Dada la expresión $a + 0 = a$, por el teorema de dualidad se puede aseverar que:

a) $a' + 0' = a'$.
b) $a * 0 = a$.
c) $a' * 1 = a$.
d) $a * 1 = a$.

6 Indicar cuáles de los siguientes conjuntos de puertas lógicas forman conjuntos universales (información contenida en los apéndices):

a) AND, OR y NOT.
b) NAND y NOT.
c) NAND.
d) AND y OR.

7 Indicar cuáles de las siguientes afirmaciones son ciertas:

a) Las expresiones en primera forma normal son únicas para una determinada función lógica.
b) Las expresiones en suma de productos son únicas para una determinada función lógica.
c) Toda función lógica puede expresarse en forma de productos de sumas.
d) Ninguna de las anteriores es cierta.

8 Indicar cuáles de las siguientes afirmaciones son ciertas:

a) A partir de 3 multiplexores de 4 a 1 puede construirse un multiplexor de 16 a 1.
b) Los multiplexores pueden utilizarse para sintetizar funciones lógicas a partir de su tabla de verdad.
c) Un multiplexor con n entradas dispondrá de $log_2(n)$ señales de control.
d) Un multiplexor con n entradas depondrá de 2^n salidas.