Los circuitos lógicos combinacionales

Montse Peiron Guàrdia
Fermín Sánchez Carracedo

PID_00215620

Universitat Oberta de Catalunya

Índice

Introducción

Un computador es una máquina construida a partir de dispositivos electrónicos básicos, adecuadamente interconectados. Podemos decir que estos dispositivos constituyen las "piezas" o "ladrillos" con los que se construye un computador.

En este módulo conoceremos a fondo los dispositivos electrónicos básicos. Los dispositivos electrónicos más elementales son las **puertas lógicas** y los **bloques** lógicos, que forman los **circuitos lógicos**. Estos últimos se pueden ver como un conjunto de dispositivos que manipulan de una manera determinada las señales electrónicas que les llegan (las **señales de entrada**), y generan como resultado otro conjunto de señales (las **señales de salida**).

Existen dos grandes tipos de circuitos lógicos:

1) Los **circuitos combinacionales**, que se caracterizan porque el valor de las señales de salida en un momento determinado depende del valor de las señales de entrada en ese mismo momento.

2) Los **circuitos secuenciales**, en los que el valor de las señales de salida en un momento determinado depende de los valores que han llegado por las señales de entrada desde la puesta en funcionamiento del circuito. Por tanto, tienen capacidad de memoria.

En este módulo se estudian los circuitos lógicos combinacionales. Los circuitos secuenciales se estudiarán en el siguiente módulo "Los circuitos lógicos secuenciales".

Objetivos

El objetivo fundamental de este módulo es conocer a fondo los circuitos lógicos combinacionales, es decir, saber cómo están formados y ser capaces de utilizarlos con agilidad, hasta el punto de familiarizarnos totalmente con ellos.

Para llegar a este punto se tendrán que haber conseguido los siguientes objetivos:

1. Entender el álgebra de Boole y las diferentes maneras de expresar funciones lógicas.

2. Conocer las diferentes puertas lógicas, ver cómo se pueden utilizar para sintetizar funciones lógicas y ser capaces de hacerlo. Entender por qué se desea minimizar el número de puertas y de niveles de puertas de los circuitos y saberlo hacer.

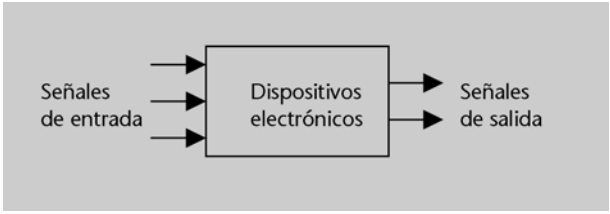
3. Conocer la funcionalidad de un conjunto de bloques combinacionales básicos y ser capaces de utilizarlos en el diseño de circuitos.

En definitiva, después del estudio de este módulo debemos ser capaces de construir fácilmente un circuito cualquiera usando los diferentes dispositivos que se habrán conocido y entender la funcionalidad de cualquier circuito dado.

1. Fundamentos de la electrónica digital

1.1. Circuitos, señales y funciones lógicas

Entendemos por circuito un sistema formado por un cierto número de **señales de entrada** (cada señal corresponde a un cable), un conjunto de **dispositivos electrónicos** que hacen operaciones sobre las señales de entrada (las manipulan electrónicamente) y que generan un determinado número de **señales de salida**. Las señales de salida, pues, se pueden considerar como funciones de las de entrada y se puede decir que los dispositivos electrónicos *computan* estas funciones.



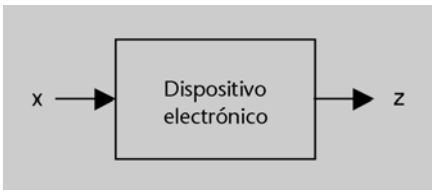
En los circuitos, los cables se pueden encontrar en dos valores de tensión (voltaje): tensión alta o tensión baja. Estos dos valores se identifican normalmente con los símbolos 1 y 0, respectivamente, de manera que se dice que una señal *vale 0* (cuando en el cable correspondiente hay tensión baja) o *vale 1* (cuando en el cable correspondiente hay tensión alta, también llamada *tensión de alimentación*); cuando una señal vale 1, se dice que está activa. Las señales que pueden tomar los valores 0 ó 1 se denominan **señales lógicas** o **binarias**. Un circuito lógico es aquél en el que las señales de entrada y de salida son lógicas. Las funciones que computa un circuito lógico son **funciones lógicas**.

La tensión alta o tensión de alimentación base puede ser de 3,3, 5 o 12 voltios, pero actualmente los circuitos tienen dispositivos para modificarla según los requerimientos de cada componente y de cada momento; por ejemplo, cuando el procesador está en una fase de actividad alta el voltaje aumenta.
La tensión baja siempre es de 0 voltios.

Hay circuitos que funcionan con lógica inversa, y en este caso se dice que una señal está activa cuando vale 0. En este curso, sin embargo, cuando decimos que una señal está activa queremos decir que vale 1.

Puesto que las señales sólo pueden tomar dos valores, diremos que uno es el contrario del otro. Así pues, podemos afirmar que cuando una señal no vale 1, entonces seguro que vale 0, y viceversa.

Tomamos un circuito que tenga sólo una señal de entrada (que llamamos *x*), un dispositivo electrónico y una señal de salida (que llamamos *z*).



Dado que tanto *x* como *z* sólo pueden valer 0 ó 1, sólo existen cuatro dispositivos electrónicos diferentes que pueden interconectar *x* y *z*:

- Un dispositivo que haga que la salida z valga siempre 0 (este dispositivo consistiría en conectar la salida con una fuente de tensión de 0 voltios).
- Un dispositivo que haga que la salida valga siempre lo mismo que la entrada *x* (este dispositivo consistiría en conectar directamente la salida con la entrada).
- Un dispositivo que haga que la salida valga siempre lo contrario de lo que vale la entrada (este dispositivo consistiría en un inversor del nivel de tensión).
- Un dispositivo que haga que la salida valga siempre 1 (este dispositivo consistiría en conectar la salida directamente con la tensión de alimentación).

En otras palabras, podemos decir que sólo hay cuatro funciones lógicas que tengan una sola variable de entrada, tal como se muestra en la figura 1:

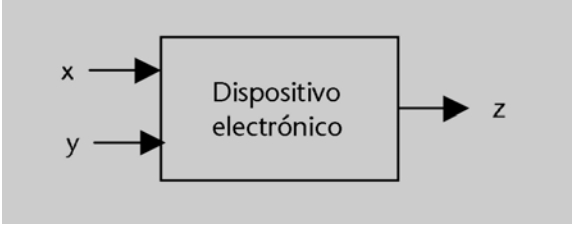
Figura 1

Función lógica	Descripción	Valor de la función cuando x = 0	Valor de la función cuando x = 1
$z = f_0(x) = 0$	función constante 0	0	0
$z = f_1(x) = x$	función identidad	0	1
$z = f_2(x) = x'$	función contrario	1	0
$z = f_3(x) = 1$	función constante 1	1	1

Nota

Introducimos aquí la nomenclatura *x'* para referirnos a la función "contrario de *x*". Más adelante la definiremos con más propiedad.

Tomamos ahora un circuito que tenga dos señales de entrada (que llamamos *x* e *y*), un dispositivo electrónico y una señal de salida (que llamamos *z*).



En este caso, existen dieciséis dispositivos electrónicos diferentes que puedan interconectar las entradas con la salida, que corresponden a las funciones lógicas que se muestran en la figura 2.

Figura 2

Función lógica	Valor de la función cuando			
	x = 0, y = 0	x = 0, y = 1	x = 1, y = 0	x = 1, y = 1
$z = g_0(x,y)$	0	0	0	0
$z = g_1(x,y)$	0	0	0	1
$z = g_2(x,y)$	0	0	1	0
$z = g_3(x,y)$	0	0	1	1
$z = g_4(x,y)$	0	1	0	0
$z = g_5(x,y)$	0	1	0	1
$z = g_6(x,y)$	0	1	1	0
$z = g_7(x,y)$	0	1	1	1
$z = g_8(x,y)$	1	0	0	0
$z = g_9(x,y)$	1	0	0	1
$z = g_{10}(x,y)$	1	0	1	0
$z = g_{11}(x,y)$	1	0	1	1
$z = g_{12}(x,y)$	1	1	0	0
$z = g_{13}(x,y)$	1	1	0	1
$z = g_{14}(x,y)$	1	1	1	0
$z = g_{15}(x,y)$	1	1	1	1

Así pues, si tenemos en cuenta los circuitos con una o dos entradas, podemos llegar a diseñar hasta 4 + 16 = 20 dispositivos electrónicos diferentes. Ahora bien, en la práctica sólo se construye un subconjunto de éstos, concretamente los que corresponden a las funciones f_2, g_1 y g_7 (y otras que veremos más adelante), porque a partir de estas funciones se pueden construir todas las demás, tal como veremos en el apartado 1.2.

A continuación, veremos que existe una correspondencia entre los elementos de un circuito lógico y la lógica intuitiva (de aquí deriva la denominación de circuitos "lógicos"). La lógica tiene cinco componentes básicos:

- Los valores *falso* y *cierto*.
- Las conjunciones *y* y *o*.
- La partícula de negación *no*.

Por ejemplo, sean las dos frases siguientes:

frase_A: "Juan estudia química",
frase_B: "Carmen estudia piano".

Cada una de estas frases puede ser verdadera o falsa. A partir de éstas, de las conjunciones y la negación construimos ahora las tres frases siguientes:

frase_Y: "Juan estudia química **y** Carmen estudia piano",
frase_O: "Juan estudia química **o** Carmen estudia piano",
frase_NO: "Juan **no** estudia química".

Según la lógica:

– La frase_Y es verdadera sólo si son ciertas la frase_A y la frase_B, simultáneamente.

– La frase_O es verdadera si lo es la frase_A o bien la frase_B, o ambas.

– La frase_NO es verdadera sólo si la frase_A es falsa.

Nota

Normalmente, cuando utilizamos la conjunción o en lenguaje natural lo hacemos de manera exclusiva. Es decir, si decimos "X o Y" nos referimos al hecho de que o bien es cierto X o bien lo es Y, pero no ambas a la vez. La *o* lógica, en cambio, incluye también la posibilidad de que las dos afirmaciones sean ciertas.

La correspondencia de la lógica con los elementos de un circuito lógico es la siguiente:

Lógica	Circuitos lógicos
falso	0
cierto	1
conjunción *y*	función g_1
conjunción *o*	función g_7
partícula *no*	función f_2

En efecto, sean dos señales lógicas x e y. Si analizamos las tablas de las figuras 1 y 2, podemos observar lo siguiente:

– $g_1(x,y)$ vale 1 si valen 1 las dos variables x e y simultáneamente;
– $g_7(x,y)$ vale 1 si x vale 1 o bien y vale 1 o bien las dos valen 1;
– $f_2(x)$ vale 1 sólo si x vale 0.

Es decir, se cumple lo mismo que en el ejemplo de las frases. Por eso, la función g_1 se llama AND (la conjunción *y* en inglés), la función g_7 se denomina OR (la conjunción *o* en inglés) y la función f_2 se llama NOT (*no* en inglés).

Así pues, los circuitos lógicos se construyen a partir del mismo fundamento que la lógica. En el siguiente apartado se estudia este fundamento.

1.2. Álgebra de Boole

Un **álgebra de Boole** es una entidad matemática formada por un conjunto que contiene dos elementos, unas operaciones básicas sobre estos elementos y una lista de axiomas que definen las propiedades que cumplen las operaciones.

George Boole

Formalizó matemáticamente la lógica en 1854, cuando definió lo que hoy se conoce como *álgebra de Boole*. De hecho, aquí presentamos un caso particular de álgebra de Boole, denominado *álgebra de Boole binaria*; en general, el conjunto de un álgebra de Boole puede tener más de dos elementos.

En 1938, Claude Shannon definió el comportamiento de los circuitos lógicos sobre el fundamento del álgebra de Boole y creó el *álgebra de conmutación*.

Los dos **elementos** de un álgebra de Boole se pueden llamar *falso* y *cierto* o, más usualmente, 0 y 1. De este modo, una variable booleana o **variable lógica** puede tomar los valores 0 y 1.

Las **operaciones booleanas** básicas son las siguientes:

- La **negación** o complementación o NOT, que corresponde a la partícula *no* y se representa con una comilla simple (’). Así, la expresión *x’* denota la negación de la variable *x*, y se lee “*no x*”.
- El **producto lógico** o **AND**, que corresponde a la conjunción *y* de la lógica y se representa con el símbolo “·”. Así, si *x* e *y* son variables lógicas, la expresión $x \cdot y$ denota su producto lógico y se lee “*x* e *y*”.
- La **suma lógica** u **OR**, que corresponde a la conjunción *o* y se representa con el símbolo “+”. Así, la expresión “$x + y$” denota la suma lógica de las variables *x* e *y*, y se lee “*x* o *y*”.

La operación de negación se puede representar también de otras maneras. Por ejemplo: $!x$ o, más usualmente, $\bar{x}$.

Estas operaciones booleanas básicas se pueden definir escribiendo el resultado que dan para cada posible combinación de valores de las variables de entrada, tal como se muestra en la figura 3. En las tablas de esta figura aparecen a la izquierda de la línea vertical todas las combinaciones posibles de las variables de entrada, y a la derecha, el resultado de la operación para cada combinación. Podemos comprobar que corresponden a las funciones f_2, g_1 y g_7 que hemos visto en el apartado anterior (figuras 1 y 2).

Atención

Es importante no confundir los operadores · y + con las operaciones *producto entero* y *suma entera* a las que estamos acostumbrados. El significado de los símbolos vendrá determinado por el contexto en el que nos encontremos. Así, si trabajamos con enteros, 1 + 1 = 2 (“uno más uno igual a dos”), mientras que si estamos en un contexto booleano, 1 + 1 = 1 (“cierto o cierto igual a cierto”, tal como se puede ver en la tabla de la operación OR).

Figura 3

x	x’
0	1
1	0

Operación NOT

x	y	x · y
0	0	0
0	1	0
1	0	0
1	1	1

Operación AND

x	y	x + y
0	0	0
0	1	1
1	0	1
1	1	1

Operación OR

Los **axiomas** que describen el comportamiento de las operaciones booleanas son los siguientes:

La formulación que se presenta aquí de los axiomas booleanos fue introducida por Huntington en 1904, a partir de la descripción original de Boole.

Sean *x*, *y* y *z* variables lógicas. Entonces se cumple lo siguiente:

a) La **propiedad conmutativa:**

$$x + y = y + x$$
$$x \cdot y = y \cdot x$$

b) La **propiedad asociativa:**

$$x + (y + z) = (x + y) + z$$
$$x \cdot (y \cdot z) = (x \cdot y) \cdot z$$

c) La **propiedad distributiva:**

$$x \cdot (y + z) = x \cdot y + x \cdot z$$
$$x + (y \cdot z) = (x + y) \cdot (x + z)$$

Observamos que la segunda igualdad de la propiedad distributiva no se cumple en el contexto de la aritmética entera.

d) Los **elementos neutros:**

$$x + 0 = x$$
$$x \cdot 1 = x$$

e) La **complementación**. Para cualquier x se cumple lo siguiente:

$$x + x' = 1$$
$$x \cdot x' = 0$$

A partir de estos axiomas, se puede demostrar una serie de leyes o teoremas muy útiles para trabajar con expresiones algebraicas booleanas.

Teoremas del álgebra de Boole

Si x, y y z son variables lógicas, se cumplen las siguientes leyes:

1) Principio de dualidad

Toda identidad deducida a partir de los axiomas continúa siendo cierta si las operaciones + y · y los elementos 0 y 1 se intercambian en toda la expresión.

Esta ley se puede comprobar, por ejemplo, examinando las parejas de expresiones que aparecen en la definición de los axiomas.

2) Ley de idempotencia

$$x + x = x$$
$$x \cdot x = x$$

3) Ley de absorción

$$x + x \cdot y = x$$
$$x \cdot (x + y) = x$$

4) Ley de dominancia

$$x + 1 = 1$$
$$x \cdot 0 = 0$$

5) Ley de involución

$$(x')' = x$$

6) Leyes de De Morgan

$$(x + y)' = x' \cdot y'$$
$$(x \cdot y)' = x' + y'$$

Un buen ejercicio para comprobar que se cumplen estas leyes es pensar en su correspondencia con la lógica, tal y como se hace en la última parte del apartado 1.1.

En las expresiones algebraicas utilizamos los paréntesis de la misma forma que estamos acostumbrados a hacerlo en aritmética entera; por ejemplo, la expresión $x + y \cdot z$ es lo mismo que la $x + (y \cdot z)$ y es diferente que $(x + y) \cdot z$.

Para negar una expresión entera la pondremos entre paréntesis y, por tanto, $x + y'$ es diferente de $(x + y)'$.

Actividades

1. Evaluad el valor de la expresión $x + y \cdot (z + x')$ para los casos:
$[x\ y\ z] = [0\ 1\ 0]$,
$[x\ y\ z] = [1\ 1\ 0]$,
$[x\ y\ z] = [0\ 1\ 1]$.

2. Demostrad las leyes de De Morgan de acuerdo con todos los posibles valores que pueden tomar las variables que aparecen.

3. Demostrad la ley 4 del álgebra de Boole a partir de la ley 2 y del axioma *c* y *e*.

1.3. Representación de funciones lógicas

Las funciones lógicas se pueden expresar de varias maneras. Las que usaremos nosotros son las **expresiones algebraicas** y las **tablas de verdad**.

1.3.1. Expresiones algebraicas

Las **expresiones algebraicas** están formadas por variables lógicas, los elementos 0 y 1, los operadores producto (·), suma (+) y negación (’), y los símbolos (,) e =.

Mediante estos elementos se puede expresar cualquier función lógica. Por ejemplo, la función g_4 de la figura 2 se puede expresar así:

$$g_4(x,y) = x' \cdot y$$

La expresión $x' \cdot y$ vale 1 (es cierta) sólo si valen 1 (son ciertas) las subexpresiones x' e y simultáneamente. La expresión x' vale 1 sólo si x vale 0. En la descripción de la función g_4 (figura 2) se puede comprobar que sólo vale 1 en el caso en que $x = 0$ e $y = 1$.

Una misma función lógica se puede expresar mediante infinitas expresiones algebraicas equivalentes.

Para saber si dos expresiones algebraicas son equivalentes (es decir, expresan la misma función) podemos analizar si una se puede derivar de la otra usando los axiomas y leyes del álgebra de Boole.

Por ejemplo, la primera ley de De Morgan nos dice que las funciones lógicas f y g son la misma:

$$f(x, y) = (x + y)',$$
$$g(x, y) = x' \cdot y'.$$

A la inversa, a partir de la expresión algebraica de una función podemos encontrar otras expresiones equivalentes aplicándole los axiomas y leyes del álgebra de Boole.

Por ejemplo, si tenemos la siguiente función:

$$f(x, y, z) = x \cdot y' \cdot z + x' \cdot y \cdot z + x \cdot y \cdot z,$$

y aplicamos el axioma c (propiedad distributiva), obtenemos la siguiente expresión equivalente:

$$f(x, y, z) = x \cdot y' \cdot z + (x' + x) \cdot y \cdot z.$$

Por el axioma e (de complementación) y después el d (de elementos neutros), obtenemos lo que expresamos a continuación:

$$f(x, y, z) = x \cdot y' \cdot z + 1 \cdot y \cdot z = x \cdot y' \cdot z + y \cdot z.$$

Por la propiedad distributiva,

$$f(x, y, z) = (x \cdot y' + y) \cdot z.$$

Y, de este modo, podríamos seguir encontrando infinitas expresiones equivalentes para la misma función.

Al igual que cuando trabajamos con ecuaciones, podemos omitir el signo "·" en los productos lógicos para simplificar la escritura de las expresiones algebraicas: si escribimos seguidos los nombres de diferentes variables, sobreentenderemos que el producto lógico se obtiene de éstas.

Así, las dos expresiones siguientes son igualmente válidas:

$$x_1' \cdot x_0 + x_2 \cdot x_1' \cdot x_0', \text{ o bien}$$
$$x_1' x_0 + x_2 x_1' x_0'.$$

Actividades

4. Encontrad una expresión algebraica para las funciones g_1, g_3, g_6, g_7 y g_{10} de la figura 2.

5. Encontrad una expresión más sencilla para esta función: $f = wx + xy' + yz + xz' + xy$.

6. Sea una función de tres variables x, y y z. Encontrad una expresión algebraica de la misma suponiendo que la función debe valer 1 cuando se cumpla alguna de las siguientes condiciones:
– $x = 1$ o $y = 0$,
– $x = 0$ y $z = 1$,
– las tres variables valen 1.

7. Se dispone de dos cajas fuertes electrónicas, A y B. Cada una de éstas tiene una señal asociada, x_A y x_B respectivamente, que vale 1 cuando la caja está abierta y 0 cuando está cerrada. Se tiene también un interruptor general que tiene una señal asociada i_g, que vale 1 si el interruptor está cerrado y 0 si no lo está.
Se quiere construir un sistema de alarma antirrobos, que generará una señal de salida s. Esta señal tiene que valer 1 cuando alguna caja fuerte esté abierta y el interruptor esté cerrado. Escribid la expresión algebraica de la función $s = f(x_A, x_B, i_g)$.

8. Juan se ha examinado de tres asignaturas. Sus amigos han visto los resultados de los exámenes y le han comentado lo siguiente:
– Has aprobado matemáticas o física, dice el primero.
– Has suspendido química o matemáticas, dice el segundo.
– Has aprobado sólo dos asignaturas, dice el tercero.

Entendemos que la "o" de estas frases es exclusiva. Es decir, la primera frase se podría sustituir por "o bien has aprobado matemáticas y has suspendido física, o bien has suspendido matemáticas y has aprobado física", y de manera similar con la segunda frase.
a) Escribid las expresiones algebraicas de las afirmaciones de cada uno de los amigos.
b) Utilizando los axiomas y teoremas del álgebra de Boole, deducid qué asignaturas ha aprobado Juan y cuál ha suspendido.

1.3.2. Tablas de verdad

Una **tabla de verdad** expresa una función lógica especificando el valor que tiene la función para cada posible combinación de valores de las variables de entrada.

Tabla de verdad

En aritmética entera es imposible escribir la tabla de verdad de una función, ya que las variables de entrada pueden tomar infinitos valores y, por tanto, la lista debería ser infinita. En álgebra de Boole es posible gracias al hecho de que las variables pueden adoptar sólo dos valores.

Concretamente, a la izquierda de la tabla hay una lista con todas las combinaciones de valores posibles de las variables de entrada y, a la derecha, el valor de la función para cada una de las combinaciones. Por ejemplo, las tablas que habíamos visto en la figura 3 eran, de hecho, las tablas de verdad de las funciones NOT, AND y OR.

Si una función tiene n variables de entrada, y dado que una variable puede tomar sólo dos valores, 0 ó 1, las entradas pueden adoptar 2^n combinaciones de valores diferentes. Por tanto, su tabla de verdad tendrá a la izquierda n columnas (una para cada variable) y 2^n filas (una para cada combinación posible). A la derecha habrá una columna con los valores de la función.

Las filas se escriben siempre en *orden lexicográfico*, es decir, si interpretamos las diferentes combinaciones como números naturales, escribiremos primero la combinación correspondiente al 0 (formada sólo por ceros), después la correspondiente al 1 y así sucesivamente, en orden creciente hasta la correspondiente al $2^n - 1$ (formada sólo por unos).

Nota

No obstante las combinaciones se podrían escribir en cualquier orden. Por ejemplo, las dos tablas siguientes expresan la misma función:

x	y	f
0	0	0
0	1	1
1	0	1
1	1	0

x	y	f
0	1	1
1	1	0
1	0	1
0	0	0

La figura 4 muestra la estructura de las tablas de verdad para funciones de una, dos y tres variables de entrada.

Figura 4

Estructura de la tabla de la verdad de una función de

1 variable

a	f
0	
1	

2 variables

a	b	f
0	0	
0	1	
1	0	
1	1	

3 variables

a	b	c	f
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

Por ejemplo la función g_5 de la figura 2 tiene la siguiente tabla de verdad:

x	y	g_5
0	0	0
0	1	1
1	0	0
1	1	1

Es importante notar que el orden en que ponemos las columnas de las variables es significativo. Por ejemplo, podíamos haber escrito la tabla de verdad de la función g_5 de la siguiente manera:

y	x	g_5
0	0	0
0	1	0
1	0	1
1	1	1

Es bastante usual denominar las variables de una función con una misma letra y diferentes subíndices; por ejemplo, x_2, x_1, x_0. Por convención, pondremos en la columna situada más a la izquierda la variable de subíndice más alto y, en la que se encuentra más a la derecha, la de subíndice más bajo.

Diremos que la variable que ponemos en la columna más a la izquierda en una tabla de verdad es la variable **de más peso**, y la que ponemos en la columna más a la derecha es la **de menos peso**. Una vez fijado el orden de las columnas y las filas, la tabla de verdad de una función es **única**.

Podemos expresar el comportamiento de diferentes funciones que tienen las mismas variables de entrada en una única tabla de verdad. En este caso, a la derecha de la línea vertical habrá tantas columnas como funciones. No estableceremos ninguna convención para ordenar las columnas correspondientes a las funciones (se pueden poner en cualquier orden). A continuación se muestra un ejemplo.

x_2	y_1	x_0	f_0	f_1	f_2
0	0	0	1	0	0
0	0	1	1	0	1
0	1	0	0	1	1
0	1	1	0	0	0
1	0	0	0	1	1
1	0	1	1	0	0
1	1	0	0	0	1
1	1	1	0	1	1

Actividades

9. Escribid la tabla de verdad de las funciones g_1, g_3, g_6, g_7 y g_{10} de la figura 2.

10. Dibujad la estructura de la tabla de verdad de la función $f(x_3, x_2, x_1, x_0)$.

11. Especificad la tabla de verdad de una función de cuatro variables, x_3, x_2, x_1 y x_0, que vale 1 sólo cuando un número par de las variables valen 1 (recordad que el 0 es un número par).

1.3.3. Correspondencia entre expresiones algebraicas y tablas de verdad

Es importante saber pasar con agilidad de una expresión algebraica de una función a su tabla de verdad, y viceversa.

Para obtener la tabla de verdad a partir de una expresión algebraica podemos actuar de dos maneras:

a) Evaluar la expresión para cada combinación de las variables y escribir en el lugar correspondiente de la tabla el resultado de la evaluación.

Por ejemplo, con la función siguiente:

$$f(a, b, c) = a + a'\, b'\, c + ac'.$$

Si evaluamos la expresión para el caso de la combinación $[a\ b\ c] = [0\ 0\ 0]$, obtenemos el siguiente resultado:

$$0 + 1 \cdot 1 \cdot 0 + 0 \cdot 1 = 0 + 0 + 0 = 0.$$

Por tanto, en la primera fila de la tabla irá un 0.

Para la combinación $[a\ b\ c] = [0\ 0\ 1]$ obtenemos el resultado que presentamos a continuación:

$$0 + 1 \cdot 1 \cdot 1 + 0 \cdot 0 = 0 + 1 + 0 = 1.$$

Por tanto, en la segunda fila irá un 1. Y así sucesivamente hasta haber evaluado la función para todas las combinaciones.

b) Analizar la expresión *a vista*, deducir para qué valores de las variables de la función vale 1 ó 0, y rellenar la tabla.

Tomamos la misma función del ejemplo anterior. Vemos que siempre que $a = 1$, la función vale 1 (por la ley 4, de dominancia). Por tanto, podemos poner un 1 en todas las filas en las que $a = 1$ (las cuatro últimas).

A continuación estudiamos los casos en que $a = 0$ (los que nos faltan por rellenar). La expresión de la función nos queda de la siguiente manera:

$$f(0, b, c) = 0 + 0' \cdot b' \cdot c + 0 \cdot c' = b' \cdot c.$$

Esta expresión vale 1 sólo en el caso $[b\ c] = [0\ 1]$. Por tanto, pondremos un 1 en la fila correspondiente a la combinación [0 0 1].

Para las combinaciones que aún nos quedan por rellenar, la función vale 0.

Observamos que el paso de expresión a tabla de verdad nos puede ser muy útil a la hora de determinar si dos expresiones algebraicas son equivalentes o no. Podemos obtener la tabla de verdad a partir de cada una de las expresiones, de modo que si las tablas resultantes son iguales, podemos concluir que las dos expresiones corresponden a una misma función.

Actividades

12. Obtened la tabla de verdad de las siguientes funciones:
a) $f(x, y, z, w) = xy'zw' + yz'w + x'z + xyw + x'yz'w'$
b) $f(x, y, z, w) = xy + z$

13. Mediante tablas de verdad, estudiad si las dos expresiones siguientes son equivalentes:
a) $f = x'y' + yw + x'w$,
b) $g = x'y'z'w' + x'z'w + yz'w + xyw + x'z$.

14. Escribid la tabla de verdad correspondiente a la actividad 7.

15. Escribid la tabla de verdad correspondiente a la actividad 8. Encontrad la solución de la actividad a partir del estudio del contenido de esta tabla.

El paso de tabla de verdad a expresión algebraica se puede hacer "a vista", si tenemos suficiente experiencia. Sin embargo, será más cómodo si lo hacemos conociendo las expresiones en suma de mintérminos, que veremos en el siguiente apartado.

1.3.4. Expresiones en suma de mintérminos

Ya hemos visto que hay infinitas expresiones algebraicas equivalentes para una función cualquiera. Ahora bien, cualquier función lógica se puede expresar con una expresión en suma de productos. Es decir, una expresión de la forma siguiente:

$$A + B + C,$$

donde A, B y C son términos producto, es decir, tienen la forma que reproducimos a continuación.

$$X \cdot Y \cdot Z,$$

donde X, Y y Z son variables lógicas, bien negadas o bien sin negar. Por ejemplo, las dos expresiones siguientes corresponden a la misma función (lo podéis demostrar), pero sólo la segunda tiene la forma de suma de productos:

$$f(x, y, z) = (x + y\ z')'z,$$
$$f(x, y, z) = x'\ y'\ z + x'\ y\ z.$$

A partir de la tabla de verdad de una función es muy sencillo obtener una expresión en suma de productos. Veamos las tablas de verdad de las cuatro funciones siguientes:

x_2	x_1	x_0	f_0	f_1	f_2	F
0	0	0	0	0	0	0
0	0	1	1	0	0	1
0	1	0	0	0	0	0
0	1	1	0	0	0	0
1	0	0	0	1	0	1
1	0	1	0	0	0	0
1	1	0	0	0	1	1
1	1	1	0	0	0	0

Si examinamos f_0, f_1 y f_2, observamos que tienen la particularidad de valer 1 sólo para una de las combinaciones de las variables de entrada, de manera que las podemos expresar fácilmente con un solo término producto, **en el que aparecen todas las variables**, negadas o sin negar. Las variables que valen 0 en la combinación que hace que la función valga 1 aparecerán negadas en el término producto; las que valen 1 aparecerán sin negar. Así, obtenemos los siguientes términos producto para las funciones anteriores:

$$f_0(x_2, x_1, x_0) = x_2' \, x_1' \, x_0.$$
$$f_1(x_2, x_1, x_0) = x_2 x_1' x_0'.$$
$$f_2(x_2, x_1, x_0) = x_2 \, x_1 \, x_0'.$$

Se denomina *término mínimo* o **mintérmino** el término producto (único) que expresa una función que sólo vale 1 para una sola combinación de las variables de entrada.

Nota

En un mintérmino figuran siempre todas las variables de la función, ya sea negadas o sin negar.

No es tan sencillo obtener "a vista" una expresión para la función F, ya que vale 1 para varias combinaciones de las variables. Ahora bien, sí podemos ver fácilmente que F vale 1 o bien f_0, o f_1 o f_2 valen 1. Es decir,

$$F = f_0 + f_1 + f_2.$$

Por tanto, obtenemos lo siguiente:

$$F(x_2,x_1,x_0) = x_2' \, x_1' \, x_0 + x_2 \, x_1' \, x_0' + x_2 \, x_1 \, x_0',$$

que es una expresión algebraica de F en forma de suma de productos.

Así pues, a partir de la tabla de verdad de una función podemos obtener una expresión en suma de productos haciendo la suma lógica de los mintérminos que la forman. Por este motivo, la expresión que obtenemos de esta forma se llama **suma de mintérminos**.

Nota

Cualquier función lógica se puede expresar también en forma de producto de sumas, pero no trataremos este asunto en este curso.

Las expresiones en suma de mintérminos o producto de sumas se llaman *formas canónicas* de una función. Los mintérminos se denominan *términos canónicos*.

En el siguiente capítulo (apartado 2.3) veremos que a partir de una expresión en suma de mintérminos se puede obtener otra expresión en suma de productos más sencilla.

Actividades

16. Obtened la expresión en suma de mintérminos de las funciones f_0, f_1, f_2 y f_3:

x_2	x_1	x_0	f_0	f_1	f_2	f_3
0	0	0	0	0	0	0
0	0	1	0	0	1	1
0	1	0	1	0	0	1
0	1	1	0	0	1	0
1	0	0	0	1	0	1
1	0	1	0	1	1	0
1	1	0	0	1	0	0
1	1	1	1	1	1	1

17. Obtened la expresión en forma de suma de mintérminos de las siguientes funciones:
a) $f(x, y, z) = xy + z$,
b) $f(x, y, z, w) = x'y'zw + x'y'z' + xy'w + yz'w$.

1.4. Otras funciones comunes

Hemos visto que existen cuatro funciones lógicas de una variable (figura 1) y dieciséis de dos variables (figura 2). Nos hemos fijado, especialmente, en las funciones f_2, g_1 y g_7, porque corresponden a las operaciones básicas del álgebra de Boole, y las hemos llamado, respectivamente, *negación* (NOT), *producto lógico* (AND) y *suma lógica* (OR). A continuación hemos visto que cualquier función lógica se puede construir a partir de estas tres.

De entre las funciones de dos variables, existen otras que también reciben un nombre propio, ya que se utilizan de manera común. Son las siguientes:

Función O-exclusiva o XOR

Esta función vale 1 cuando alguna de las dos variables de entrada vale 1, pero no cuando valen 1 las dos a la vez; por eso recibe el nombre de O-*exclusiva*. Es la función g_6 de la figura 2.

Se representa con símbolo "⊕", y presenta esta tabla de verdad:

a	b	$a \oplus b$
0	0	0
0	1	1
1	0	1
1	1	0

$a \oplus b = a'b + ab'$

La función XOR de más de dos variables se calcula de este modo:

$$a \oplus b \oplus c \oplus d = (((a \oplus b) \oplus c) \oplus d).$$

Por tanto, da 0 si un número par de las variables vale 1, y da 1 si un número impar de las variables vale 1. Fijémonos que en el caso de dos variables, este hecho se traduce en lo siguiente:

$a \oplus b = 0$ => a y b tienen el mismo valor,
$a \oplus b = 1$ => a y b tienen valores diferentes.

Por tanto, la función XOR nos permite saber si dos variables son iguales o no lo son.

Otras propiedades de la XOR:

$$0 \oplus a = a,$$
$$1 \oplus a = a'.$$

Así, si hacemos la XOR de una variable con un 0, la variable queda inalterada, mientras que si hacemos la XOR con un 1, en la salida aparece la variable negada.

Función NAND

Es la negación de la AND, es decir:

$$a \text{ NAND } b = (a \cdot b)'.$$

Por tanto, vale 1 siempre que no se cumpla que las dos variables de entrada valgan 1. Es, pues, la función g_{14} de la figura 2. Ésta es su tabla de verdad:

a	*b*	(*a*+*b*)'
0	0	1
0	1	1
1	0	1
1	1	0

Función NOR

Es la negación de la OR, es decir:

$$a \text{ NOR } b = (a + b)'.$$

Por tanto, vale 1 sólo cuando ninguna de las dos variables de entrada vale 1. Es la función g_8 de la figura 2. Ésta es su tabla de verdad:

a	*b*	(*a*+*b*)'
0	0	1
0	1	0
1	0	0
1	1	0

1.5. Funciones especificadas de manera incompleta

Existen funciones para las que algunas combinaciones de las variables de entrada no se producirán nunca. Por ejemplo, supongamos una función $f(x_2, x_1, x_0)$ en la que las variables corresponden a señales conectadas a tres aparatos de medición del sonido en una sala, de manera que tenemos lo siguiente:

$x_2 = 1$ si el sonido supera los 100 dB, $x_2 = 0$ de otro modo,
$x_1 = 1$ si el sonido supera los 200 dB, $x_1 = 0$ de otro modo,
$x_0 = 1$ si el sonido supera los 300 dB, $x_0 = 0$ de otro modo.

La combinación $[x_2, x_1, x_0] = [0\ 1\ 0]$ no se producirá nunca, ya que es imposible que el sonido esté por debajo de los 100 dB y por encima de los 200 dB simultáneamente. Tampoco se darán nunca las combinaciones [0 0 1], [0 1 1] y [1 0 1].

Las combinaciones que nunca se producirán se llaman **combinaciones "no importa"** o **combinaciones *don't care***.

En la tabla de verdad de una función escribiremos *x* en las filas correspondientes a las combinaciones "no importa". Por ejemplo, supongamos que la función anterior debe valer 1 si el sonido está entre 100 dB y 200 dB o si el sonido supera los 300 dB. Su tabla de verdad es la siguiente:

x_2	x_1	x_0	f_0
0	0	0	0
0	0	1	x
0	1	0	x
0	1	1	x
1	0	0	1
1	0	1	x
1	1	0	0
1	1	1	1

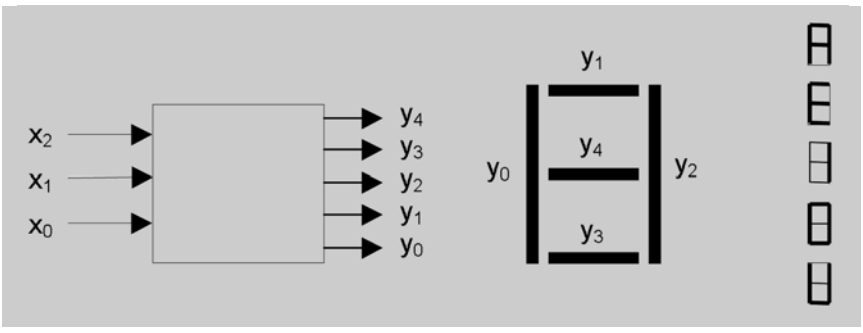
En el capítulo siguiente veremos cómo se pueden obtener expresiones algebraicas de funciones especificadas de manera incompleta.

Actividades

18. Se quiere construir un sistema que compute cinco funciones de salida (y_1, y_2, y_3, y_4 e y_5) de tres variables (x_2, x_1 y x_0). Estas tres variables codifican las cinco vocales, tal como se muestra en la siguiente tabla:

vocal	x_2	x_1	x_0
A	0	0	0
E	0	0	1
I	0	1	0
O	0	1	1
U	1	0	0

Cada una de las cinco funciones está asociada a un segmento de un *visualizador de cinco segmentos* como el que se muestra en la figura. Por ejemplo, cuando y_1 vale 1, el segmento que hay encima del visualizador se ilumina. La figura muestra también qué segmentos se deben iluminar para formar las diferentes vocales.



En cada momento el visualizador debe formar la vocal que codifiquen las variables de entrada en ese momento. Escribid la tabla de verdad de las cinco funciones.

19. Se quiere diseñar un sistema de riego de una planta con control de la temperatura y de la humedad de la tierra. El sistema tiene tres señales de entrada (variables) y dos de salida (funciones).

Entradas:

– Un sensor de temperatura (*t*): se pone a 1 si la temperatura de la tierra supera un límite prefijado *T0*.

– Dos sensores de humedad de la tierra (h_0 y h_1): se ponen en 1 cuando la humedad de la tierra supera los límites *H0* y *H1*, respectivamente. El límite *H0* es inferior al límite *H1* (*H0* < *H1*).

Salidas:

– *Regar* (*R*): cuando se pone en 1 se activa el riego de la planta.
– *Calentar* (*E*): cuando se pone en 1 se activa el calentamiento de la tierra.

Las especificaciones del sistema son las siguientes:

– La planta se riega siempre que la tierra está seca, es decir, siempre que no supera el límite *H0*.
– También se riega cuando la temperatura supera el límite *T0* y la humedad de la tierra es inferior a *H1*.
– La tierra de la planta se calienta cuando la temperatura es inferior a *T0* y la humedad superior a *H0*.

Escribid la tabla de verdad de las funciones *R* y *E*.

2. Implementación de circuitos lógicos combinacionales

2.1. Puertas lógicas. Síntesis y análisis

Ved las figuras 1 y 2 en el subapartado 1.1 de este módulo.

En las figuras 1 y 2 habíamos visto que se pueden construir hasta veinte dispositivos electrónicos diferentes para funciones de una y dos variables de entrada. Ahora bien, en la práctica sólo se construyen los que calculan las funciones NOT, AND, OR, XOR, NAND y NOR.

Los dispositivos electrónicos que calculan funciones lógicas se llaman puertas lógicas. Son dispositivos que están conectados a un cierto número de señales de entrada y a una señal de salida.

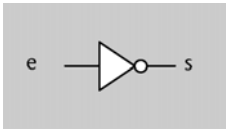
Puertas lógicas

Están formadas internamente por diferentes combinaciones de transistores, que son los dispositivos electrónicos más elementales.

A continuación presentamos el símbolo gráfico con el que se representa cada puerta lógica. En todas las figuras que aparecen a continuación, las señales de entrada quedan a la izquierda de las puertas y la señal de salida queda a la derecha.

Puerta NOT o inversor

Esta puerta se representa gráficamente con este símbolo:

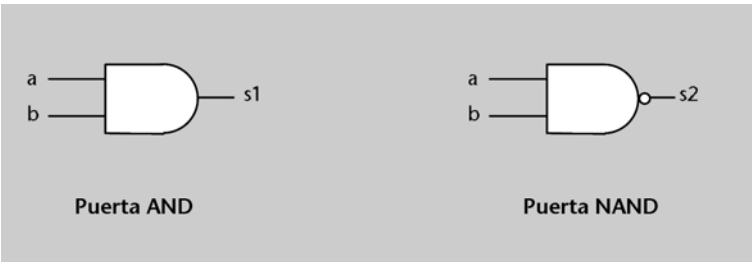


El funcionamiento es el siguiente: si en la entrada *e* hay un 0 (tensión baja), entonces en la salida *s* habrá un 1 (tensión alta). Y a la inversa, si hay un 1 en el punto *e*, entonces habrá un 0 en el punto *s*.

Por tanto, la puerta NOT implementa físicamente la función de negación: $s = e'$.

Puertas AND y NAND

Se representan gráficamente con estos símbolos:



NAND

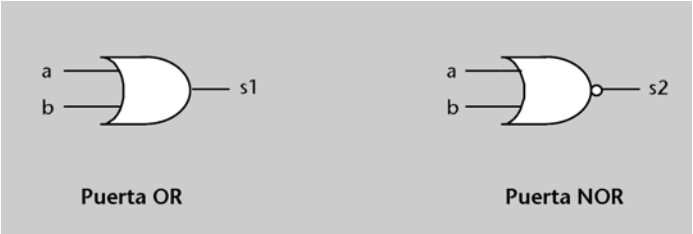
La función NAND es muy utilizada en el diseño de circuitos reales porque es muy fácil implementarla con transistores (es decir, no se implementa a partir de una puerta AND y una NOT).

La puerta AND implementa la función lógica AND, es decir, la salida $s1$ vale 1 sólo si las dos entradas a y b valen 1. Por tanto, $s1 = a \cdot b$.

La puerta NAND implementa la función lógica NAND. En el punto $s2$ hay un 1 siempre que en alguna de las dos entradas a o b haya un 0. Es decir, $s2 = (a \cdot b)'$.

Puertas OR y NOR

Se representan gráficamente con estos símbolos:



El círculo en un circuito

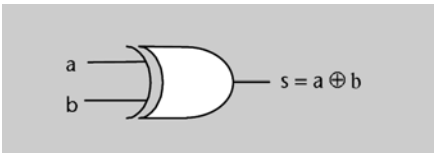
El círculo simboliza una negación en señales de salida (como en las puertas NOT, NAND y NOR). También se usa un círculo para negar señales de entrada, pero en este curso no utilizaremos esta posibilidad.

La puerta OR implementa la función lógica OR, es decir, en la salida $s1$ hay un 1 si cualquiera de las dos entradas está en 1. Por tanto, $s1 = a + b$.

La puerta NOR implementa la función lógica NOR. En el punto $s2$ encontraremos un 1 sólo cuando en las dos entradas haya un 0, es decir: $s2 = (a + b)'$.

Puerta XOR

Implementa la función lógica XOR, es decir, la salida s vale 1 si alguna de las dos entradas vale 1, pero no si valen 1 las dos a la vez. Se representa gráficamente con este símbolo:



Todas las puertas (menos la NOT) pueden tener más de dos señales de entrada. Su funcionamiento es el siguiente:

- En una puerta AND de n entradas, la salida vale 1 sólo cuando todas las n entradas valen 1.
- En una puerta OR de n entradas, la salida vale 1 cuando una o más de las n entradas valen 1.
- En una puerta XOR de n entradas, la salida vale 1 cuando hay un número impar de entradas en 1. El 0 se considera un número par y, por tanto, si todas las entradas valen 0, la salida también vale 0.

Síntesis y análisis

Cualquier función lógica se puede implementar usando estas puertas, es decir, se puede construir un circuito que se comporte como la función.

El proceso de obtener el circuito que implementa una función a partir de una expresión algebraica se denomina **síntesis**.

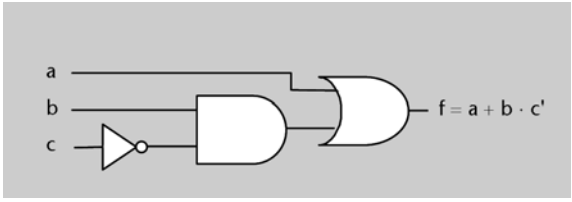
El proceso de obtener una expresión de una función a partir del circuito que la implementa se denomina **análisis**.

Dibujo de las puertas

En un circuito, las puertas se pueden dibujar en cualquier sentido: con la señal de salida a la derecha o a la izquierda, y también verticalmente con la señal de salida arriba o abajo, según convenga para la claridad del diseño.

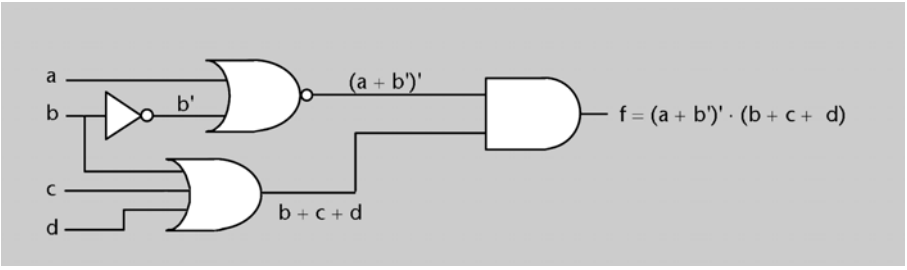
Para sintetizar (o implementar) una función a partir de su expresión algebraica, es suficiente con sustituir cada operador de la función por la puerta lógica adecuada. Por ejemplo, un término producto de tres variables se implementará con una puerta AND de tres entradas. Las puertas deben estar interconectadas entre sí y con las entradas tal como indica la expresión.

Por ejemplo, el circuito que implementa la función $f(a, b, c) = a + bc'$ es el siguiente:

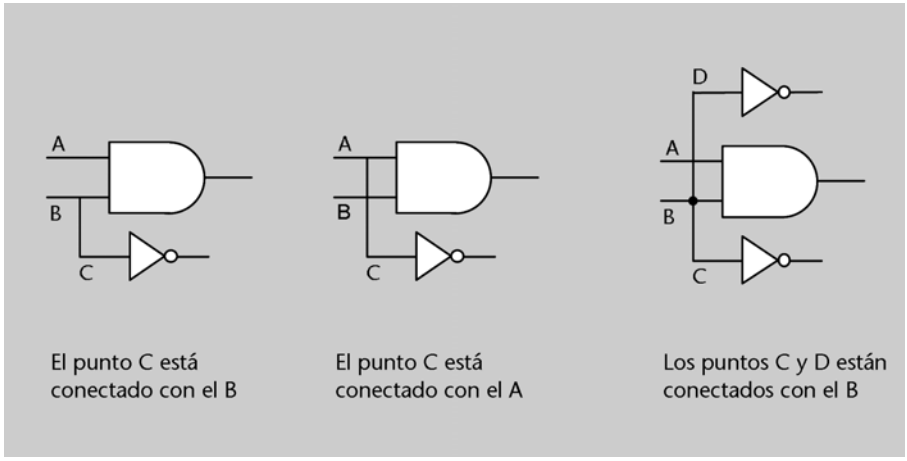


Para analizar un circuito, es necesario escribir las expresiones que corresponden a la salida de cada puerta, empezando desde las entradas del circuito hasta obtener la expresión correspondiente a la línea de salida del mismo.

Por ejemplo, el circuito siguiente implementa la función $f(a, b, c, d) = (a + b')'(b + c + d)$.



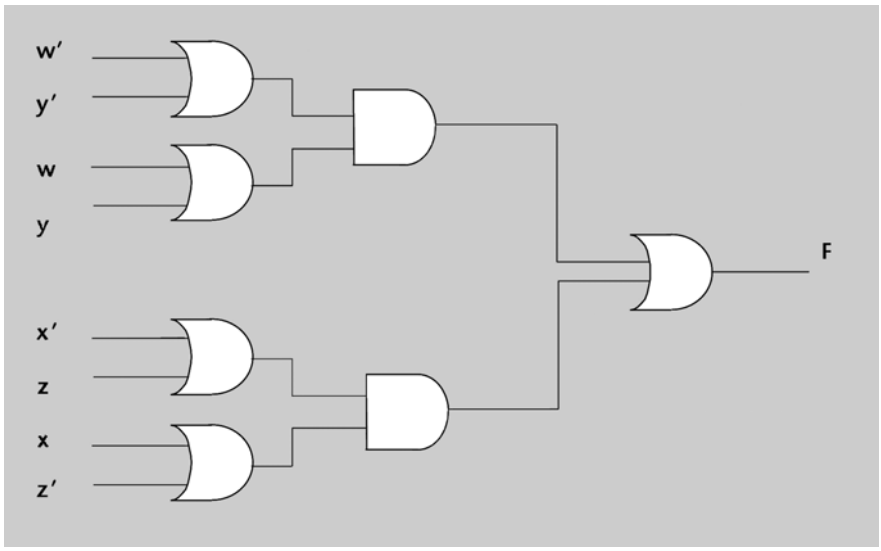
En un circuito, si una línea empieza sobre otra línea perpendicular, se entiende que las líneas están conectadas (y, por tanto, que tienen el mismo valor lógico). Si dos líneas perpendiculares se cruzan, se considera que no están conectadas. Si se pone un punto por encima de una línea, se interpreta que todas las líneas que lo tocan están conectadas. A continuación se muestran unos cuantos ejemplos.



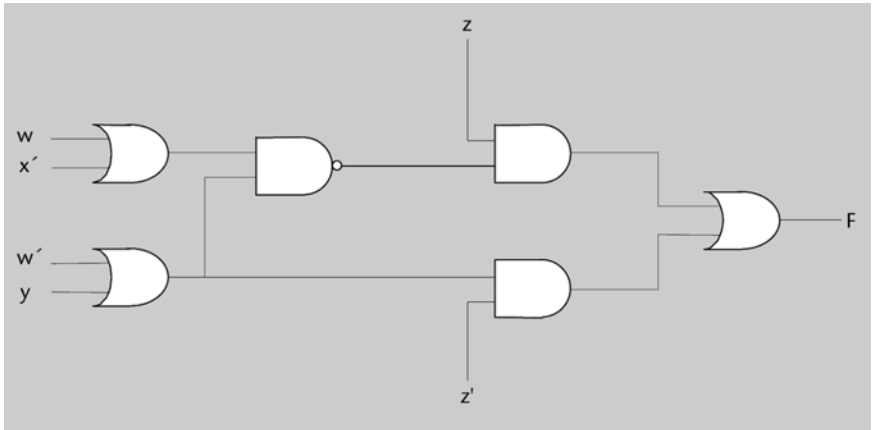
Actividades

20. Analizad los siguientes circuitos:

a)



b)



21. Sintetizad la función $f(a, b, c, d) = d \cdot (a' + a \cdot b) \cdot (b' \oplus c)$.

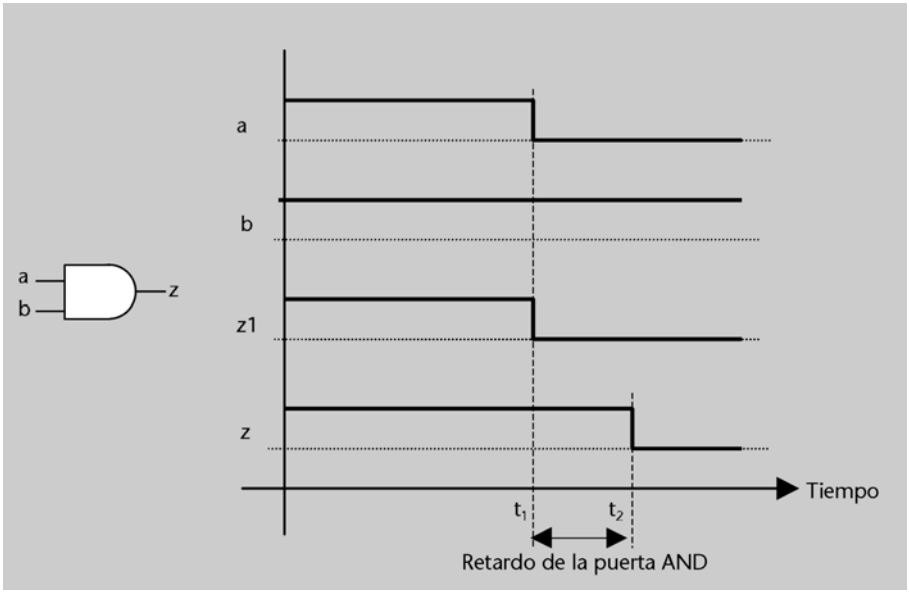
2.2. Diseño de circuitos a dos niveles

2.2.1. Retardos. Cronogramas. Niveles de puertas

Las puertas lógicas no responden instantáneamente a las variaciones en las señales de entrada, sino que experimentan un cierto **retardo**. La mejor forma de entender este concepto es observando la figura 5, en la que hay un circuito sencillo (sólo una puerta AND) y un **cronograma** de su funcionamiento.

Un **cronograma** es una representación gráfica de la evolución de las señales de un circuito a lo largo del tiempo.

Figura 5



Cronograma

Las líneas de puntos horizontales representan el valor lógico 0 para cada señal. Las líneas continuas gruesas representan el valor en que se encuentra cada señal en cada momento (0 si la línea continua está sobre la línea de puntos, 1 si está más arriba).

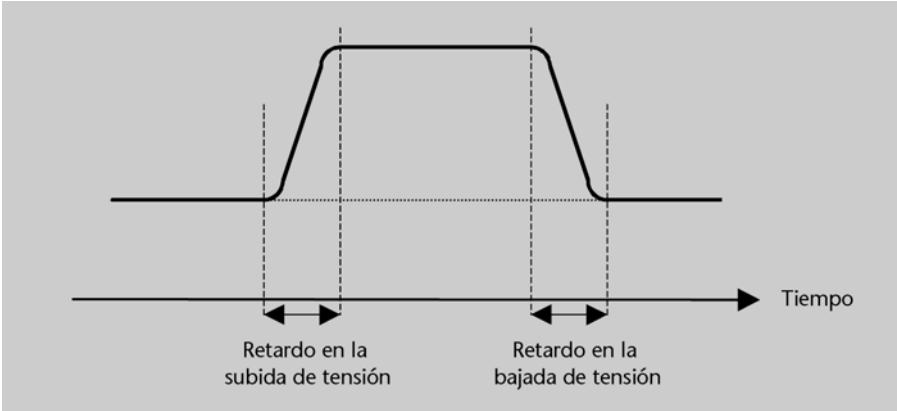
En vertical se pueden señalar con líneas discontinuas instantes determinados de tiempo (como t_1 y t_2 en el caso de la figura 5).

Supongamos que en las entradas a y b hay conectados dos interruptores que nos permiten en cada momento poner estas señales en 0 o en 1. En el ejemplo de la figura 5 hemos supuesto que, inicialmente, las dos señales están en 1 y que en el instante t_1 ponemos la señal a a 0. En este momento, la señal z se debería poner a 0, porque $0 \cdot 1 = 0$. Esta situación hipotética es la que se muestra en el cronograma con la señal $z1$. Sin embargo, en la realidad z no se pone a 0 hasta el instante t_2, a causa del hecho de que los dispositivos electrónicos internos de la puerta AND tardan un tiempo determinado en reaccionar. Diremos que la puerta AND tiene un retardo de $t_2 - t_1$.

Cada puerta tiene un retardo diferente que depende de la tecnología que se haya usado para construirla. Los retardos son muy pequeños, del orden de nanosegundos, pero es necesario tenerlos en cuenta a la hora de construir físicamente un circuito.

De hecho, la transición entre diferentes niveles de tensión de una señal tampoco es instantánea, sino que se produce tal como se muestra en la figura 6.

Figura 6



El tiempo de respuesta de una puerta depende, entre otras cosas, del número de entradas de la puerta.

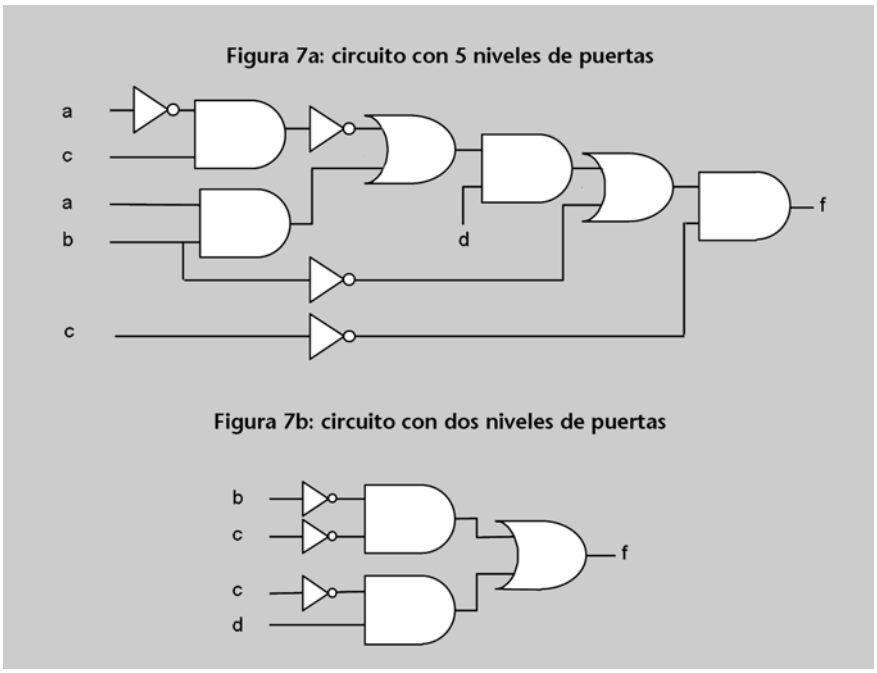
Uno de los objetivos de los ingenieros electrónicos que construyen circuitos es que el tiempo de respuesta del circuito sea lo más breve posible. Dado que cada puerta tiene un cierto retardo, un circuito será, en general, más rápido cuantos menos **niveles de puertas** haya entre las entradas y las salidas.

El número de **niveles de puertas** de un circuito es el número máximo de puertas que una señal debe atravesar consecutivamente para generar la señal de salida.

Al contabilizar el número de niveles de puertas de un circuito **no se tienen en cuenta las puertas NOT** (por razones que no veremos en este curso).

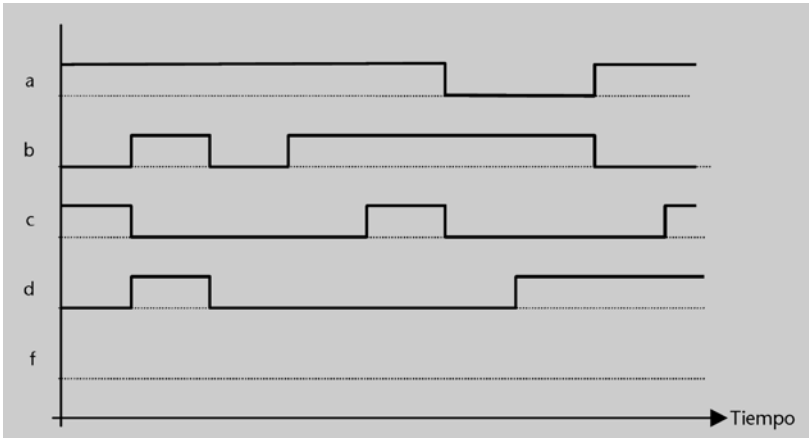
Por ejemplo, la figura 7 muestra el número de niveles de puertas de dos circuitos diferentes (podéis comprobar que la función que implementan es la misma). Un buen ingeniero elegiría el circuito de la figura 7*b* para implementar esta función, ya que es más rápido.

Figura 7



Actividades

22. Completad el siguiente cronograma, suponiendo que la señal *f* corresponde a la función de la figura 7*b* y que las entradas toman los valores dibujados. Considerad que los retardos introducidos por las puertas son 0.

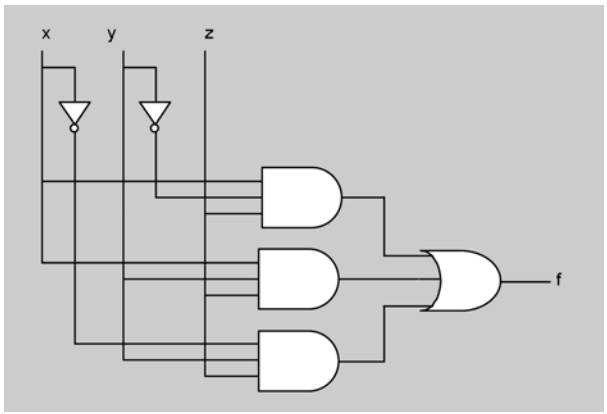


2.2.2. Síntesis a dos niveles

No existe una fórmula universal para encontrar la expresión de una función que dará lugar al circuito más rápido posible. Sin embargo, conocemos una forma de garantizar que un circuito no tenga más de dos niveles de puertas: partir de la expresión de la función en suma de mintérminos. En efecto, el circuito correspondiente a la suma de mintérminos tiene, además de las puertas NOT, un primer nivel de puertas AND (que computan los diferentes mintérminos) y un segundo nivel en el que habrá una única puerta OR, con tantas entradas como mintérminos tenga la expresión. Por ejemplo, la figura 8 muestra el circuito que se obtiene cuando se sintetiza esta función:

$$f = xy'z + xyz + x'yz.$$

Figura 8



Los circuitos que se obtienen a partir de las sumas de mintérminos se denominan **circuitos a dos niveles**. El proceso por el que los obtenemos se denomina **síntesis a dos niveles**.

Es posible que una función se implemente con un circuito que tenga menos de dos niveles. Sin embargo, no hay una manera de encontrarlo sistemáticamente. Así pues, las expresiones de las funciones en suma de mintérminos nos permiten construir los circuitos en dos niveles más rápidos posibles que podemos obtener de manera sistemática.

Por otro lado, a partir de la expresión de una función en suma de mintérminos puede resultar un circuito con menos de dos niveles: si hay un sólo mintérmino no habrá el nivel OR.

Tener en cuenta los retardos de las diferentes puertas y de las transiciones entre niveles de tensión es fundamental a la hora de construir circuitos. Sin embargo, en este curso consideraremos que los circuitos son ideales, de manera que no se tendrán nunca en cuenta los retardos, es decir, se asumirá que siempre son 0.

Hay otros métodos de síntesis que generan circuitos con más de dos niveles pero más rápidos que los que se obtienen a partir de las expresiones de las funciones en suma de mintérminos. En este curso no los veremos.

Actividades

23. Haced la síntesis a dos niveles de las siguientes funciones:
a) $f(x,y,z,w) = x'y'z'w' + x'yz'w' + xy'zw' + xyzw + x'y'z'w + x'yzw$.
b) $f(x,y,z) = xz' + y'z + x'y$.

24. Se quiere diseñar un circuito combinacional que permita multiplicar dos números naturales de dos bits.
a) Indicad el número de bits de la salida.
b) Escribid la tabla de verdad de las funciones de salida.
c) Implementad el circuito a dos niveles.

25. Sintetizad la siguiente función a dos niveles:

x_2	x_1	x_0	f
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

2.3. Minimización de funciones

Ya hemos visto que uno de los objetivos que se tiene que conseguir cuando construimos un circuito es que sea tan rápido como sea posible. La síntesis a dos niveles nos permite conseguir de manera sistemática un grado de rapidez aceptable.

También es deseable que un circuito sea pequeño y barato. Estas dos características están relacionadas con el número de puertas del circuito: cuantas menos haya, más pequeño y barato será.

2.3.1. Simplificación de expresiones

En el capítulo anterior se ha visto cómo se puede obtener la expresión de una función en suma de mintérminos. A veces, estas expresiones se pueden

simplificar, lo que nos permitirá implementar la función con un circuito de menores dimensiones.

En concreto, las simplificaciones que nos serán útiles corresponden a los casos en que **dos o más mintérminos se pueden reducir a un único término producto**, en el que aparecerán menos variables.

Por ejemplo, sea la siguiente función:

$$f(x, y, z) = x'yz' + x'yz + \ldots$$

Estos dos mintérminos nos dicen que la función vale 1 si $x = 0$, $y = 1$ y $z = 0$ o bien si $x = 0$, $y = 1$ y $z = 1$. Sin embargo, esta información se puede resumir diciendo que la función vale 1 siempre que $x = 0$ e $y = 1$, independientemente del valor de z. Esto se puede expresar algebraicamente sacando el factor común en los dos mintérminos (es decir, aplicando las propiedades distributiva, de complementación y de elemento neutro):

$$f(x,y,z) = x'yz' + x'yz + \ldots =$$
$$= x'y(z'+z) + \ldots = x'y \cdot 1 + \ldots = x'y + \ldots$$

Tomamos ahora una función que vale 1 para las siguientes combinaciones [x y z] (entre otras):

[1 0 0]
[1 0 1]
[1 1 0]
[1 1 1]

Esta función vale 1 siempre que $x = 1$, independientemente del valor de y y z. Aplicando el mismo razonamiento que antes, lo podemos expresar algebraicamente de este modo:

$$f(x,y,z) = xy'z' + xy'z + xyz' + xyz + \ldots =$$
$$= xy'(z'+z) + xy\,(z'+z) + \ldots =$$
$$= xy' + xy + \ldots = x\,(y'+y) + \ldots = x + \ldots$$

Así pues, vemos que podemos obtener un solo término producto en los siguientes casos:

1. Si en dos mintérminos todas las variables se mantienen constantes menos una, entonces podemos sacar factor común eliminando la variable que cam-

bia. En el término *producto resultante* aparecerán $n - 1$ variables, siendo n el número de variables de la función.

2. Si en cuatro mintérminos todas las variables se mantienen constantes menos dos, entonces podemos sacar factor común dos veces y eliminar las dos variables que cambian. En el término *producto resultante* aparecerán $n - 2$ variables.

También podemos detectar otros casos en los que se puede sacar factor común en una expresión algebraica. Por ejemplo, .
$f(x, y, z) = xy' + xz = x \cdot (y' + z)$
Sin embargo, en estos casos no nos interesan, ya que la expresión resultante no es una suma de productos.

3. En general, si en 2^m mintérminos todas las variables se mantienen constantes menos m, entonces podemos sacar factor común m veces y eliminar las m variables que cambian. En el término *producto resultante* aparecerán $n - m$ variables.

El hecho de obtener la expresión en suma de productos más simplificada posible de una función, sacando factor común en los casos que se acaban de describir, se denomina **minimizar la función**.

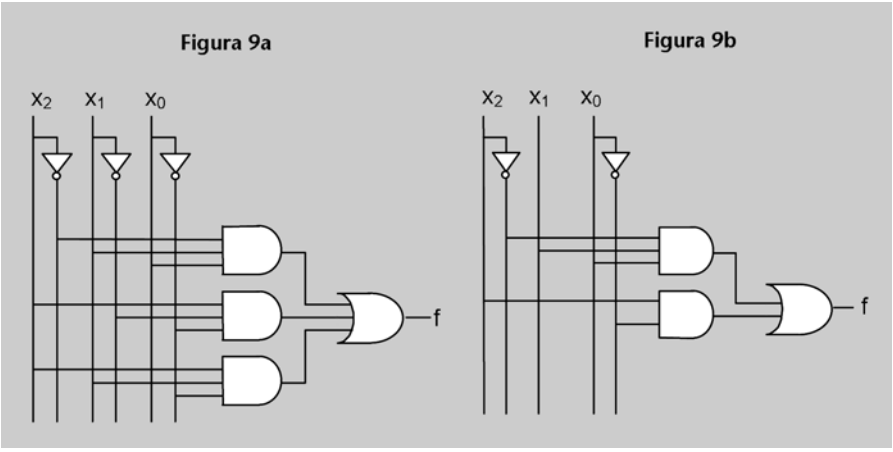
A partir de la expresión minimizada de una función, podremos construir circuitos de menores dimensiones y más baratos que los que obteníamos de la expresión en suma de mintérminos: quizá se necesitarán menos puertas NOT, algunas de las puertas AND serán de menos entradas, tendrán menos puertas AND y la puerta OR será de menos entradas.

La figura 9 muestra los circuitos correspondientes a la expresión en suma de mintérminos (figura 9*a*) y a la expresión minimizada (figura 9*b*) de esta función.

$$f(x_2,x_1,x_0) = x_2'x_1x_0 + x_2x_1'x_0' + x_2x_1x_0' =$$
$$= x_2'x_1x_0 + x_2x_0' \ (x_1'+x_1) = x_2'x_1x_0 + x_2x_0'.$$

Se puede observar que el circuito de la figura 9*a* tiene tres puertas NOT, tres AND de tres entradas y una OR de tres entradas. El circuito simplificado, en cambio, sólo tiene dos puertas NOT, dos AND (una de dos entradas y la otra de tres) y una puerta OR de dos entradas.

Figura 9



2.3.2. Síntesis mínima a dos niveles. Método de Karnaugh

Observando la tabla de verdad de la función podemos detectar los casos en que la expresión en suma de mintérminos se puede minimizar. Ahora bien, en algunos casos lo podemos ver fácilmente, pero en otros es más difícil. Por ejemplo, la figura 10*a* muestra la tabla de verdad de la función:

$$f(x_2, x_1, x_0) = x_2'x_1.$$

Sólo observando la tabla es fácil obtener esta expresión, porque las combinaciones por las que la función vale 1 (los mintérminos) se encuentran en filas consecutivas.

La función de la figura 10*b* es $f(x_2, x_1, x_0) = x_1'x_0$ porque vale 1 siempre que $x_1 = 0$ y $x_0 = 1$, independientemente de lo que valga x_2. Sin embargo, en este caso no es tan sencillo verlo a simple vista, ya que los mintérminos no están en filas consecutivas.

Figura 10

Figura 10a

x_2	x_1	x_0	f
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

Figura 10b

x_2	x_1	x_0	f
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

El **método de Karnaugh** proporciona una mecánica sencilla para detectar visualmente los casos en que se puede minimizar una expresión en suma de mintérminos. Por tanto, entre todos los circuitos a dos niveles que implementan una función, permite obtener fácilmente el menor de todos.

El método de Karnaugh consta de cuatro pasos:

1) Trasladar la tabla de verdad de la función a una estructura que se llama *mapa de Karnaugh*.

2) Detectar visualmente los casos en que se puede sacar factor común.

3) Deducir los términos producto más simples posible.

4) Obtener la expresión mínima de la función haciendo la suma lógica de los términos producto.

Veamos detalladamente cómo se lleva a cabo cada uno de estos procesos.

1) Construcción del mapa de Karnaugh

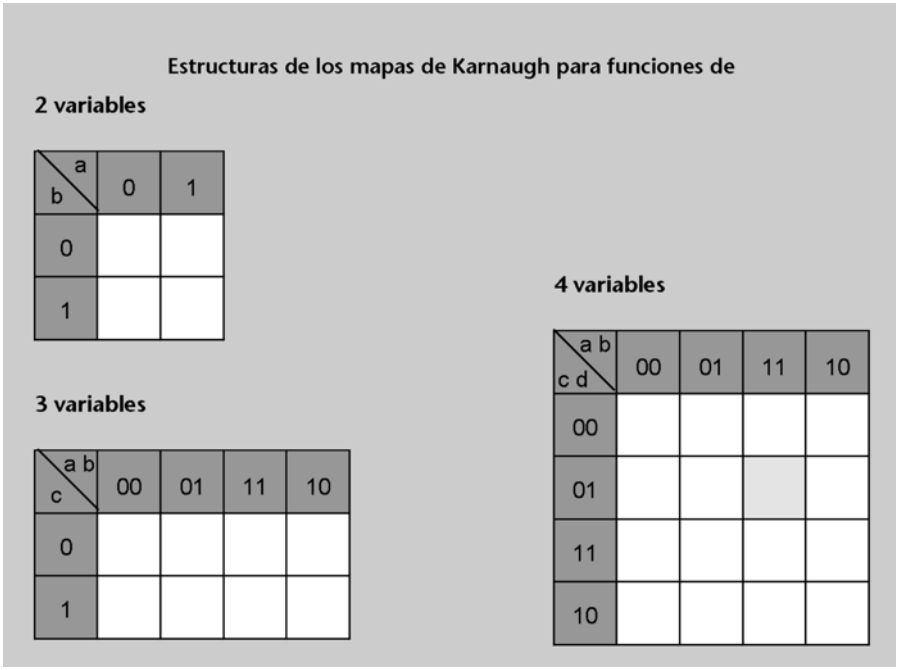
El **mapa de Karnaugh** es una transcripción de la tabla de verdad de una función a una estructura formada por casillas en la que cada una de éstas corresponde a una combinación de las variables (y, por tanto, a una fila de la tabla de verdad).

Se dice que dos casillas del mapa son **adyacentes** si corresponden a combinaciones en las que sólo cambia el valor de una variable.

La figura 11 muestra la estructura del mapa de Karnaugh para funciones de 2, 3 y 4 variables . Por ejemplo, la casilla que está sombreada con gris claro corresponde a la combinación [*a b c d*] = [1 1 0 1].

También es posible aplicar el método de Karnaugh a funciones de más de cuatro variables, pero en este curso no se estudiará. De hecho, para estos casos hay otros métodos más adecuados, que no estudiaremos.

Figura 11



Fijémonos en que en las cabeceras de las filas y las columnas de los mapas de Karnaugh las combinaciones no están en orden lexicográfico.

De esta manera se cumple que las casillas adyacentes quedan dispuestas en el mapa de la siguiente manera:

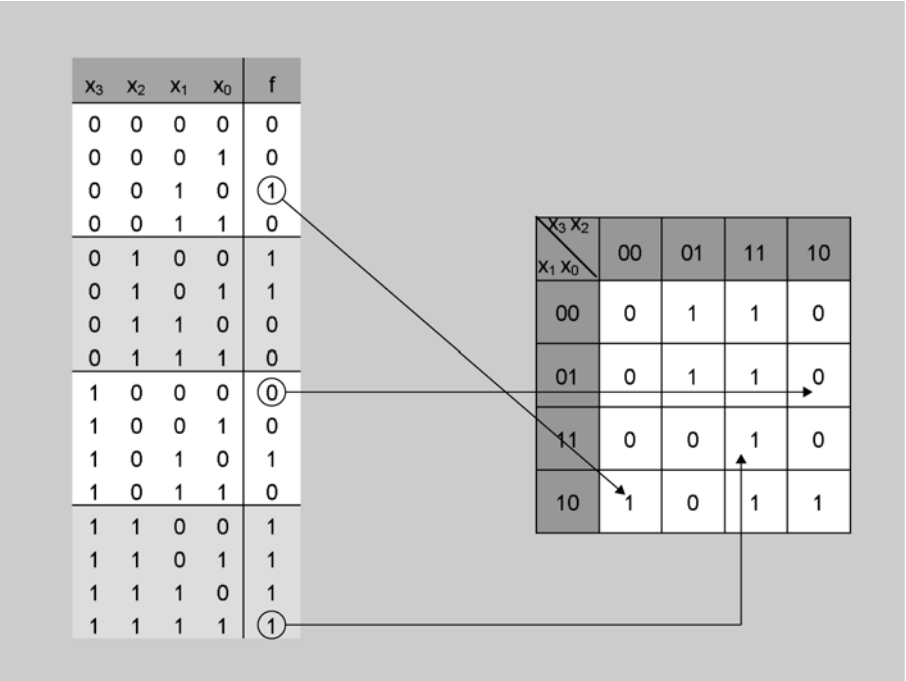
1. Dos casillas donde únicamente cambia el valor de una variable son adyacentes.

2. En los mapas de tres y cuatro variables, las casillas de la columna más a la derecha también son adyacentes con las de la columna más a la izquierda.

3. En los mapas de cuatro variables, las casillas de la fila superior también son adyacentes con las de la fila inferior.

Una vez dibujado el mapa, pondremos dentro de cada casilla el valor de la función para la combinación correspondiente de variables, a partir de la tabla de verdad. En la figura 12 se puede ver un ejemplo de ello donde se muestra explícitamente la posición de algún mintérmino en la tabla.

Figura 12



De hecho, es suficiente con rellenar las casillas para las que la función vale 1.

2) Detección de los casos en que se puede sacar factor común

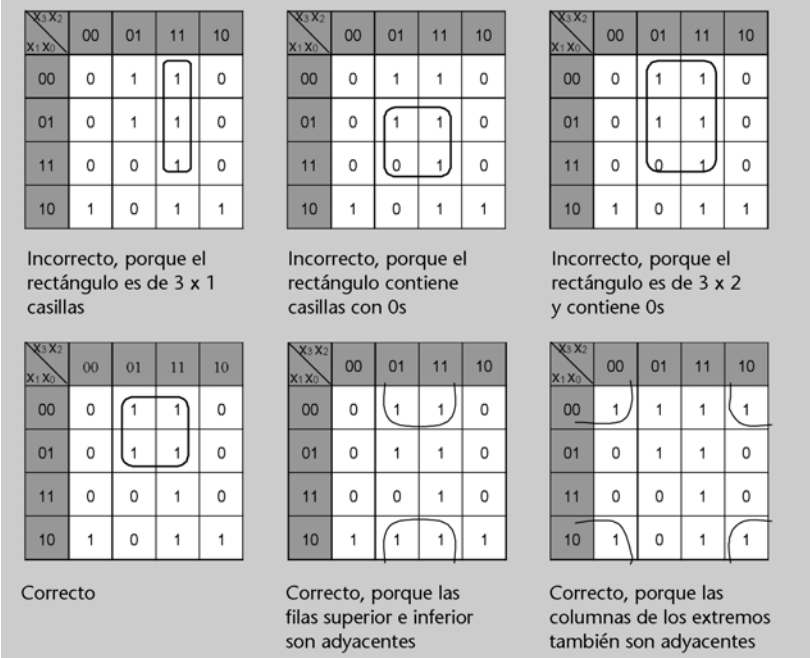
Supongamos que dos casillas adyacentes contienen unos; corresponden, pues, a dos mintérminos de la función. Sin embargo, entre éstas sólo cambia el valor de una variable (por la definición de adyacencia) y, por tanto, corresponden a un caso en el que se puede sacar factor común de los dos mintérminos.

Supongamos ahora que cuatro casillas adyacentes contienen unos. Entre éstas sólo cambia el valor de dos variables y, por tanto, corresponden a un caso en el que se puede sacar factor común dos veces.

Así, el segundo paso del método Karnaugh consiste en agrupar con rectángulos los unos que estén en casillas adyacentes, formando grupos de 1, 2, 4, 8 ó 16 unos. Los lados de estos rectángulos deben ser de un número de casillas potencia de 2, y en su interior sólo puede haber unos.

En la figura 13 se muestran varios ejemplos de rectángulos correctos e incorrectos.

Figura 13

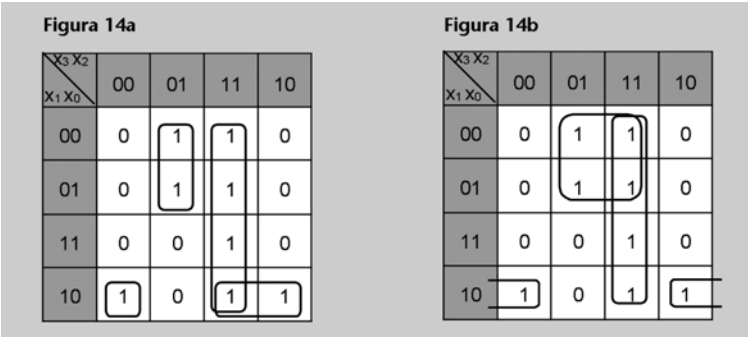


La manera de agrupar los unos de un mapa no es única. Al hacer las agrupaciones, se debe tener en cuenta lo siguiente:

- Todos los unos deben formar parte de algún grupo.
- Los grupos tienen que ser lo más grandes posible (para que en cada término aparezca el menor número posible de variables). Por este motivo, empezaremos buscando los mayores grupos que podamos hacer (recordando que dentro de un grupo sólo puede haber unos).
- Cuantos menos grupos haya, mejor (para obtener el menor número posible de términos producto). El hecho de buscar los grupos más grandes reduce el número total de grupos que se harán.
- Un mismo 1 puede formar parte de más de un grupo si eso ayuda a satisfacer los dos objetivos anteriores. Si después de hacer los grupos grandes que veíamos a primera vista, queda algún 1 disperso, vemos si lo podemos agrupar con otros unos que ya formen parte de algún grupo.

En la figura 14 se muestran dos formas de agrupar los unos del mapa del ejemplo anterior. La de la figura 14*a* no es incorrecta, pero la de la figura 14*b* es mejor. Siempre se debe procurar encontrar la agrupación óptima.

Figura 14



3) Deducción de los términos producto

Tomamos el mapa de la figura 14*b*. El grupo de los cuatro unos de la tercera columna corresponde a las combinaciones $[x_3\ x_2\ x_1\ x_0]$ =

$$[1\ 1\ 0\ 0]$$
$$[1\ 1\ 0\ 1]$$
$$[1\ 1\ 1\ 0]$$
$$[1\ 1\ 1\ 1]$$

La expresión en suma de mintérminos que se obtendría de estos cuatro unos es $x_3x_2x_1'x_0' + x_3x_2x_1'x_0 + x_3x_2x_1x_0' + x_3x_2x_1x_0$. Si sacamos factor común, obtenemos x_3x_2, porque el valor de la función es independiente de x_1 y x_0. Si observamos el mapa, podemos ver que las variables x_3 y x_2 no cambian de valor en el rectángulo, mientras que x_1 y x_0 adoptan todas las combinaciones posibles.

Así pues, obtendremos un término producto de cada grupo de la siguiente manera:

1. Sólo aparecen las variables cuyo valor es constante para todas las casillas que forman el grupo.

2. Si en todas las casillas del grupo una variable vale 1, la variable aparece en el término *producto sin negar*.

3. Si en todas las casillas del grupo una variable vale 0, ésta aparece negada en el término *producto*.

Así, en el mapa de la figura 14*b*, del otro grupo de cuatro unos obtenemos el término

$$x_2x_1'.$$

Del grupo de dos unos obtenemos el término

$$x_2'x_1x_0'.$$

Observad

A partir de un rectángulo de 2^m unos obtenemos un término producto con $n - m$ variables, siendo n el número de variables de la función. Por ejemplo, en el término $x_2'x_1x_0'$, que se obtiene a partir de un rectángulo de dos unos ($m = 1$, $n - m = 3$), hay tres variables, mientras que el término x_2x_1 tiene sólo dos variables, porque se obtiene de un rectángulo con cuatro unos ($m = 2$, $n - m = 2$).

4) Obtención de la expresión mínima de la función

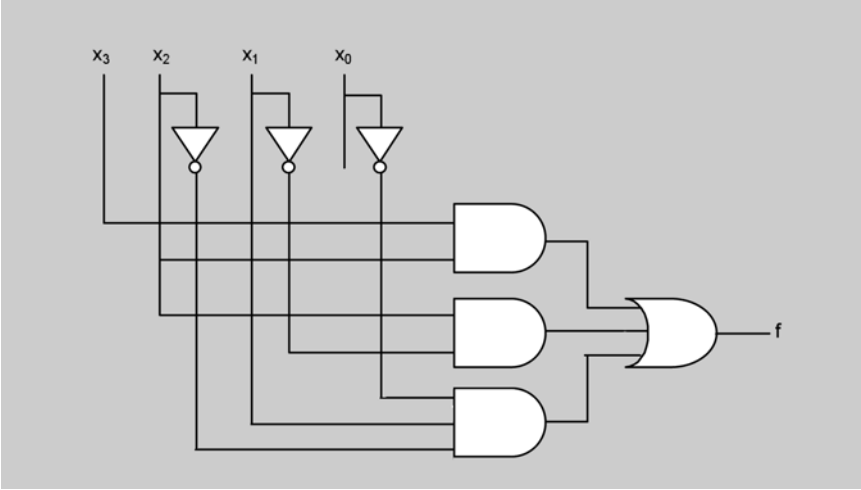
Ya sabemos que una función se puede expresar haciendo la suma lógica de todos sus mintérminos. Los términos *producto* obtenidos en el paso anterior "resumen" los mintérminos de una función. Por tanto, podemos expresar la función realizando la suma lógica de estos términos *producto*.

En el ejemplo de la figura 14*b*, la expresión minimizada de la función es la siguiente:

$$f(x_3,x_2,x_1,x_0) = x_3x_2 + x_2x_1' + x_2'x_1x_0'.$$

La figura 15 muestra el circuito a dos niveles que implementa esta función a partir de la expresión minimizada. Se puede ver que es menor y más barato que el que obtendríamos a partir de la expresión en suma de mintérminos original, ya que éste tendría cuatro puertas NOT, ocho AND de tres entradas cada una y una OR de ocho entradas. De todos los circuitos a dos niveles que implementan esta función, el de la figura 15 es el de dimensiones más reducidas y el más barato; se dice que es el **circuito mínimo a dos niveles**.

Figura 15



Actividades

26. Sintetizad mínimamente a dos niveles la función descrita en la actividad 7.

27. Sintetizad mínimamente a dos niveles la función descrita en la actividad 23*a*. Comparad la respuesta con la que habéis obtenido en la actividad 23*a*.

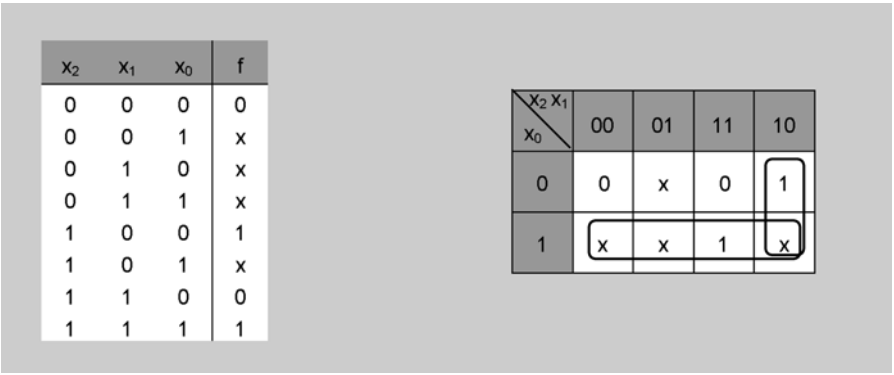
2.3.3. Minimización de funciones especificadas incompletamente

Tal como hemos visto en el apartado 1.5, el valor de las funciones especificadas incompletamente es indiferente para algunas combinaciones de las variables (las combinaciones "no importa"). Es decir, por las combinaciones "no importa" tanto podemos suponer que la función vale 1 como que vale 0.

En la tabla de verdad de la función ponemos x en las filas correspondientes a las combinaciones "no importa". En el mapa de Karnaugh también pondremos *x* en las casillas correspondientes. Dado que el valor de la función en estos casos es indiferente, supondremos que las *x* son un 1 o un 0, según lo que nos convenga más, teniendo en cuenta que siempre debemos procurar obtener el menor número posible de agrupaciones y que las agrupaciones sean tan grandes como sea posible.

Tomamos de ejemplo la función que hemos visto en el apartado 1.5. La figura 16 muestra su tabla de verdad y el mapa de Karnaugh, con las agrupaciones de casillas.

Figura 16



x_2	x_1	x_0	f
0	0	0	0
0	0	1	x
0	1	0	x
0	1	1	x
1	0	0	1
1	0	1	x
1	1	0	0
1	1	1	1

x_0 \ x_2x_1	00	01	11	10
0	0	x	0	1
1	x	x	1	x

Nos interesa suponer que las *x* de la fila inferior valen 1, ya que así obtenemos un grupo de cuatro unos y un grupo de dos unos. Por tanto, la expresión mínima de la función es la siguiente:

$$f = x_0 + x_2x_1'.$$

Fijémonos en que hemos tomado la *x* de la casilla $[x_2\ x_1\ x_0] = [0\ 1\ 0]$ como 0, ya que si no lo hiciéramos así obtendríamos un grupo adicional innecesariamente (los grupos deben cubrir todos los unos, pero no es necesario que cubran todas las *x*).

Actividades

28. Sintetizad mínimamente a dos niveles la función descrita en la actividad 19.

3. Bloques combinacionales

Un **bloque combinacional** es un circuito lógico combinacional con una funcionalidad determinada. Está construido a partir de puertas, como los circuitos que hemos visto hasta ahora.

Hasta este momento, las "piezas" que hemos utilizado para sintetizar circuitos han sido puertas lógicas. Después de estudiar este capítulo, podremos utilizar también los bloques combinacionales como piezas para diseñar circuitos más complejos, como por ejemplo un computador, que no se pueden pensar a nivel de puertas pero sí a nivel de bloques.

Existen muchos bloques combinacionales, en este curso veremos sólo los más básicos.

3.1. Multiplexor. Multiplexor de buses. Demultiplexor

Imaginemos que en una ciudad hay tres calles que confluyen en otra calle de un solo carril. Será necesario un urbano o algún tipo de señalización para controlar que en cada momento circulen hacia la calle de salida los coches provenientes de una única calle confluente.

Un multiplexor es un bloque que cumple la función de guardia urbano en circuitos electrónicos. Tiene un determinado número de señales de entrada que "compiten" para conectarse a una señal única de salida, y unas señales de control que sirven para determinar qué señal de entrada se conecta en cada momento con la salida.

Más concretamente, las entradas y salida de un multiplexor son las siguientes:

1. 2^m entradas de *datos*, identificadas por la letra *e* y numeradas desde 0 hasta $2^m - 1$. Diremos que la entrada de datos numerada con el 0 es la de *menos peso* y, la numerada con el $2^m - 1$, la de *más peso*.

2. Una salida de datos, *s*.

3. *m* entradas *de control* o *de selección*, identificadas por la letra *c* y numeradas desde 0 hasta *m* – 1. Diremos que la entrada de control numerada con el 0 es la *de menos peso*, y la numerada con *m* – 1, la *de más peso*.

Para especificar el tamaño de un multiplexor, diremos que es un multiplexor 2^m-1; por ejemplo, un multiplexor 4-1 es un multiplexor de cuatro entradas de datos y dos de control.

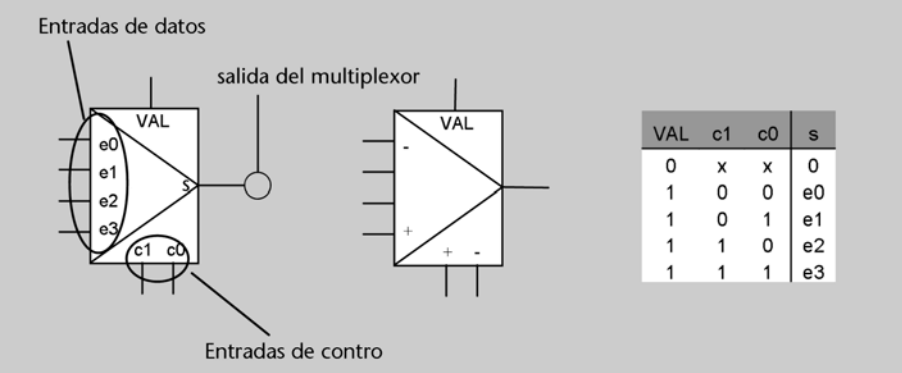
4. Una entrada de *validación*, que denominamos VAL.

La figura 17 muestra dos posibles representaciones gráficas de un multiplexor 4-1, es decir, con $m = 2$ (podéis ver el primer recuadro al margen). La diferencia entre éstas es que en la primera ponemos los nombres de las entradas, mientras que en la segunda sólo indicamos con los signos "+" y "–" cuáles son las de más y las de menos peso (se asume que el resto está ordenado en orden de peso). Las dos representaciones son igualmente válidas.

Algunos de los bloques combinacionales que se describen en este capítulo, además de las puertas lógicas descritas en el capítulo 2, pueden encontrarse en el mercado en forma de chips. Cada una de las entradas o salidas de los circuitos se corresponden con una "pata" (llamada *pin*) del chip. Entonces, una puerta AND de dos entradas necesita 3 pins de un chip, 2 para sus entradas y 1 para la salida. Por otro lado, un multiplexor de 4-1 necesita 7 pins: 4 para las entradas de datos, 2 para las entradas de control y 1 para la salida de datos. Además, como las puertas y los bloques son circuitos electrónicos, necesitan conectarse a una fuente de alimentación (los circuitos lógicos funcionan con corriente continua). Por este motivo, todos los chips tienen 2 pins adicionales, uno para conectarse al positivo de la fuente de alimentación y otro para conectarse a masa (0 voltios).

En general, no escribiremos el nombre de la salida (tal y como se hace en la segunda representación) porque la podemos identificar de manera inequívoca por el hecho de que está dibujada en el punto del multiplexor donde se juntan las dos líneas oblicuas interiores. También, en el caso de los multiplexores 2-1, que sólo tienen una entrada de control, podemos identificarla por la letra "c" (sin número) o bien no ponerle el nombre, ya que la podemos identificar inequívocamente por el hecho de que estará dibujada en un lado corto del multiplexor.

Figura 17



VAL	c1	c0	s
0	x	x	0
1	0	0	e0
1	0	1	e1
1	1	0	e2
1	1	1	e3

Nota

En esta figura la entrada de datos de más peso del multiplexor se encuentra en la parte inferior y la de menos peso, en la parte superior. Podemos invertir el orden si es más conveniente para la claridad de un circuito. Igualmente, las entradas de control pueden girarse (más peso en la derecha, menos en la izquierda).

La figura 17 también muestra la tabla de verdad que describe el funcionamiento del multiplexor (en este caso un multiplexor 4-1), que es el siguiente:

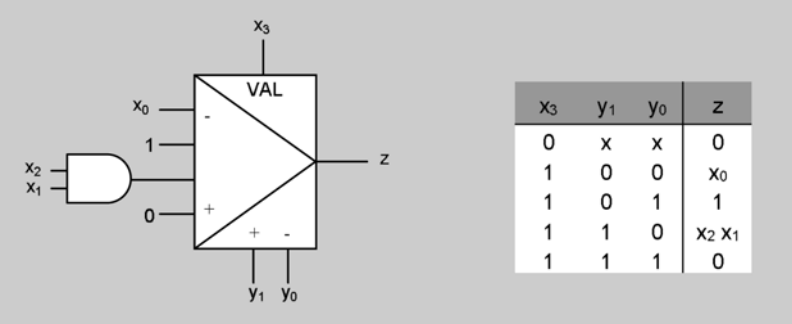
- Cuando la entrada de validación vale 0, la salida del multiplexor se pone en 0 independientemente del valor de las entradas de datos. En la tabla, esto lo reflejamos poniendo *x* en las columnas correspondientes a las entradas de datos, en la fila en la que VAL = 0.

- Cuando la entrada de validación está activa (vale 1), entonces las entradas de control determinan cuál de las entradas de datos se conecta con la salida de la forma siguiente: se interpretan las variables conectadas en las entradas de control (c_1 y c_0 en el ejemplo) como un número codificado en binario con m bits (la entrada de más peso corresponde al bit de más peso); si el número codificado es i, la entrada de datos que se conecta con la salida es la numerada con el número i.

Si no dibujamos la entrada de validación en un multiplexor, asumiremos que ésta está en 1.

En las entradas y salida de un multiplexor conectaremos las señales lógicas que convengan para el funcionamiento de los circuitos. Por ejemplo, si conec-

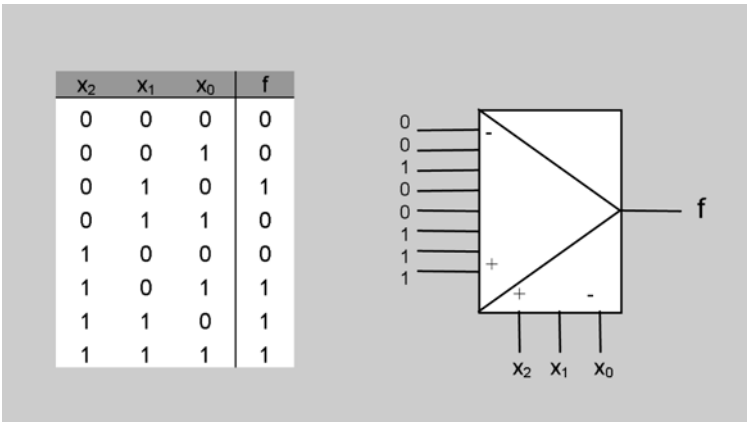
tamos las señales de entrada x_3, x_2, x_1, x_0, y_1, y_0 a un multiplexor 4-1 tal y como se muestra en la figura siguiente, la señal de salida z tomará los valores que se describen en la tabla de verdad de la derecha.



x_3	y_1	y_0	z
0	x	x	0
1	0	0	x_0
1	0	1	1
1	1	0	x_2 x_1
1	1	1	0

Una posible aplicación de los multiplexores es la implementación de funciones lógicas. La figura 18 muestra la implementación de una función con un multiplexor que tiene tantas entradas de control como variables tiene la función. La salida del multiplexor valdrá 1 si las variables conectadas a las señales de control construyen las combinaciones 2, 5, 6 ó 7, que corresponden a los casos en que la función f vale 1.

Figura 18



x_2	x_1	x_0	f
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Multiplexor de buses

Una **palabra de *n* bits** es una agrupación de n bits, usualmente con un significado semántico conjunto (por ejemplo, un número codificado en binario con n bits).

Por convención, usaremos las letras mayúsculas para dar nombre a una palabra o variable de n bits. Cada uno de los bits que la forman se identifica por la misma letra en minúscula, junto a un subíndice para identificar su peso. Por ejemplo,

$$A = a_7a_6a_5a_4a_3a_2a_1a_0 = a_{7..0}$$

Para referirnos a un subconjunto de los bits de una palabra escribiremos los subíndices correspondientes. Por ejemplo, $a_{4..1}$.

Un **bus** es una agrupación de un cierto número de cables, por cada uno de los cuales circula un bit. Así, una palabra de *n* bits circulará por un bus de *n* bits. Diremos que es su *ancho* o *tamaño* del bus.

Los buses se representan gráficamente tal como se muestra en la figura 19*a*. En la línea que representa la señal ponemos una línea transversal acompañada de un número que indica la cantidad de bits del bus.

Al dibujar circuitos, es fundamental indicar cuántos bits son todos los buses que aparecen. Una línea sin especificación de número de bits corresponderá siempre a una señal de un solo bit.

A veces interesa disgregar los bits que forman un bus o agregarle bits adicionales. Las figuras 19*b*, 19*c* y 19*d* muestran unos cuantos ejemplos de cómo lo representaremos gráficamente.

Figura 19

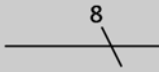


Figura 19a: bus de 8 bits

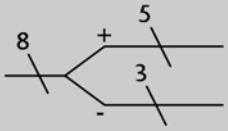


Figura 19b: el bus de 8 bits se disgrega en dos buses. uno formado por los 5 bits de mayor peso y otro formado por los 3 de menor peso

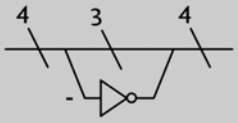


Figura 19c: la línea inferior corresponde al bit de menos peso del bus de 4 bits. Este bit se niega y después se agrega de nuevo al bus original

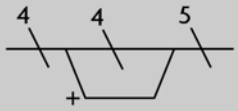
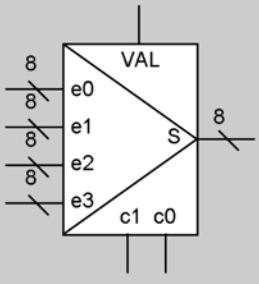


Figura 19d: la línea inferior corresponde a una replicación del bit de más peso del bus de 4 bits, porque el bus original continúa siendo de 4 bits. Este nuevo bit se agrega posteriormente al bus original, de modo que finalmente queda un bus de 5 bits

Un **multiplexor de buses** funciona igual que un multiplexor de bits, pero en este caso las entradas de datos y la salida son buses de un determinado número de bits. La figura 20 muestra la representación gráfica de un multiplexor 4-1 de buses de ocho bits.

Figura 20



VAL	c_1	c_0	S
0	x	x	0
1	0	0	e0
1	0	1	e1
1	1	0	e2
1	1	1	e3

Actividades

29. Obtened la expresión algebraica de la salida de un multiplexor 4-1. Haced su implementación interna mediante puertas lógicas.

30. Implementad la función $f(a, b, c) = abc' + a'c$ con un multiplexor 8-1.

31. Implementad un circuito combinacional que permita multiplicar dos números enteros de dos bits representados en complemento a 2 utilizando sólo un multiplexor.

32. Diseñad un multiplexor de 16-1 utilizando dos multiplexores 8-1 y las puertas lógicas que hagan falta.

33. Construid un circuito con una entrada X de 8 bits por la que llega un número natural representado en binario y una salida Z también de 8 bits, de manera que $Z = X$ si X es par y $Z = 0$ si X es impar.

Demultiplexor

Un **demultiplexor** hace la función inversa de un multiplexor: dada una señal de entrada, determina con qué señal de salida se debe conectar.

Los multiplexores tienen las señales siguientes:

Para especificar el tamaño de un demultiplexor, diremos que es un demultiplexor $1–2^m$; por ejemplo, un demultiplexor 1-4 tiene cuatro salidas de datos y dos entradas de control.

1. Una entrada de datos de n bits, e (n puede ser igual a 1).

2. 2^m salidas de datos de n bits, identificadas por la letra s y numeradas desde 0 hasta $2^m - 1$. Diremos que la salida de datos numerada con el 0 es la de menos peso, y la numerada con el $2^m - 1$, la de más peso.

3. m entradas de control o de selección, de un bit, identificadas por la letra c y numeradas desde 0 hasta $m - 1$. Diremos que la entrada de control numerada con el 0 es la de menos peso, y la numerada con $m - 1$, la de más peso.

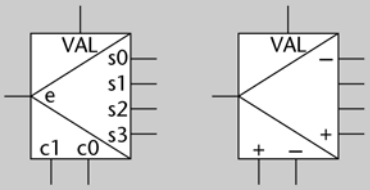
4. Una entrada de validación de un bit, que denominamos VAL.

El funcionamiento del demultiplexor es el siguiente:

- Cuando la entrada de validación vale 0, todas las salidas valen 0.
- Cuando la entrada de validación vale 1, entonces las entradas de control determinan cuál de las salidas de datos se conecta con la entrada, de la misma manera que en el caso del multiplexor (si el número que codifican las entradas de control es i, la entrada se conecta con la salida numerada con el número i). Las salidas de datos no seleccionados se ponen a 0.

La figura siguiente muestra las dos posibilidades para la representación gráfica de un demultiplexor 1-4 y la tabla que describe su funcionamiento. Al igual que en el caso de los multiplexores, podemos usar los signos "+" y "–" para indicar el peso de las entradas de control y de las salidas de datos. También, al

igual que en el caso de los multiplexores, en general no escribiremos el nombre de la entrada *e*, y si no dibujamos la entrada VAL, asumiremos que está activa.



VAL	c1	c0	s3	s2	s1	s0
0	x	x	0	0	0	0
1	0	0	0	0	0	e
1	0	1	0	0	e	0
1	1	0	0	e	0	0
1	1	1	e	0	0	0

Actividades

34. Se quiere diseñar un circuito que controle el acceso a una sala de conciertos que tiene en la entrada dos luces, una verde y otra roja, de manera que en cada momento sólo está encendida una de las dos (la roja si la sala está llena, la verde si todavía cabe gente). El circuito recibe como entrada una señal *lleno* que vale 1 cuando la sala está llena y 0 cuando todavía no. Para encender la luz verde, se debe activar la señal *verd***e**, y para encender la roja se tiene que activar la señal *rojo*. Implementad el circuito usando sólo un demultiplexor.

3.2. Codificadores y descodificadores

La función de un **codificador** es generar la codificación binaria de un número.

Los codificadores tienen las siguientes señales:

1. Una entrada de validación, VAL, que funciona igual que en el caso de los multiplexores: si vale 0, todas las salidas valen 0 (cuando no dibujemos la entrada de validación, asumiremos que vale 1).

2. 2^m entradas de datos (de un bit), identificadas por la letra *e* y numeradas de 0 a $2^m - 1$ (la de número más alto es la de más peso).

3. *m* salidas de datos (de un bit), identificadas por la letra *s* y numeradas de 0 a $m - 1$, que se interpretan como si formaran un número codificado en binario con *m* bits (la salida de más peso corresponde al bit de mayor peso).

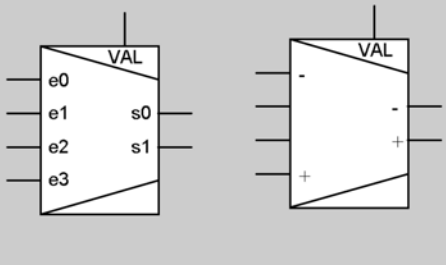
El funcionamiento de un codificador es el siguiente:

Cuando la entrada de validación vale 1, si la entrada de mayor peso de entre las que están en 1 es la numerada con el número *i*, entonces las salidas codifican en binario el número *i*. O, dicho de otro modo: para que un codificador genere la representación binaria del número *i*, es preciso que la entrada activa de más peso sea la numerada con el número *i*.

La figura 21 muestra la representación gráfica de un codificador 4-2 y la tabla de verdad que explica su funcionamiento (observamos que, como en el caso de los multiplexores, podemos utilizar los símbolos "+" y "–" en lugar de los nombres de las entradas y salidas). Vemos que, por ejemplo, cuando $e3 = 0$ y $e2 = 1$, las salidas codifican el número 2, independientemente del valor de $e1$ y $e0$, porque la entrada de más peso que está activa es $e2$. Se dice que, cuanto más peso tiene una entrada, más prioridad tiene.

Para especificar el tamaño de un codificador, diremos que es un codificador $2^m - m$; por ejemplo, un codificador 4-2 es un codificador de cuatro entradas y dos salidas.

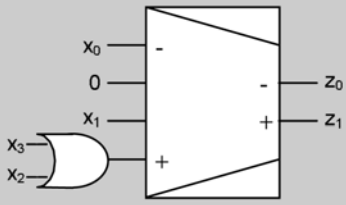
Figura 21



VAL	e3	e2	e1	e0	s1	s0
0	x	x	x	x	0	0
1	0	0	0	0	0	0
1	0	0	0	1	0	0
1	0	0	1	x	0	1
1	0	1	x	x	1	0
1	1	x	x	x	1	1

Fijémonos en que las salidas del codificador valen 0 tanto si no hay ninguna entrada en 1 como si se encuentra en 1 la entrada de menos peso (y también cuando la entrada VAL es 0).

La siguiente figura muestra un ejemplo de uso de un codificador y la tabla de verdad que describe el funcionamiento del circuito.



x_3	x_2	x_1	x_0	z_1	z_0
0	0	0	0	0	0
0	0	0	1	0	0
0	0	1	x	1	0
0	1	x	x	1	1
1	x	x	x	1	1

Recordemos que si en un codificador no dibujamos la entrada de validación, asumiremos que ésta se encuentra en 1.

Los **descodificadores** cumplen la función inversa a los codificadores: dada una combinación binaria presente en la entrada, indican a qué número decimal corresponde.

Los descodificadores tienen las siguientes señales:

1. Una entrada de validación.

2. m entradas de datos, identificadas por la letra e y numeradas de 0 a $m - 1$, que se interpretan como si formaran un número codificado en binario (la entrada de más peso corresponde al bit de mayor peso).

3. 2^m salidas, identificadas por la letra s y numeradas de 0 a $2^m - 1$, de las cuales sólo una vale 1 en cada momento (si la entrada de validación está en 0, entonces todas las salidas valen 0).

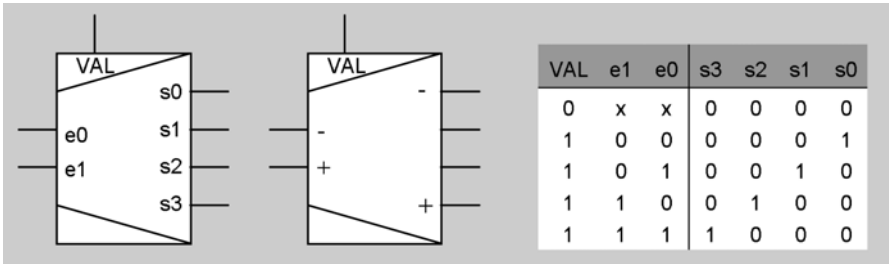
El funcionamiento de un descodificador es el siguiente:

Para identificar el tamaño de los descodificadores utilizaremos la misma convención que en el caso de los codificadores; por ejemplo, descodificador 3-8.

> Cuando la entrada de validación vale 1, si las entradas codifican en binario el número *i*, entonces se pone en 1 la salida numerada como *i*.

La figura 22 muestra la representación gráfica de un descodificador 2-4 y la tabla de verdad que describe su funcionamiento.

Figura 22



VAL	e1	e0	s3	s2	s1	s0
0	x	x	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

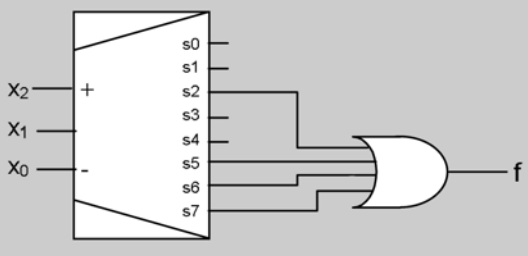
Los descodificadores también se pueden utilizar para implementar funciones lógicas. Si la función tiene n variables, usaremos un descodificador $n - 2^n$ y conectaremos las variables en las entradas en orden de peso. De este modo, la salida i del descodificador se pondrá en 1 cuando las variables, interpretadas como bits de un número binario, codifiquen el número i. Por ejemplo, si conectamos las variables $[x_1\ x_0]$ a las entradas $[e_1\ e_0]$ del descodificador de la figura 22, la salida $s2$ valdrá 1 cuando $[x_1\ x_0] = [1\ 0]$.

Por tanto, para implementar la función será suficiente con conectar las salidas correspondientes a las combinaciones que hacen que la función valga 1 en una puerta OR. Cuando alguna de estas combinaciones se encuentre presente en la entrada del descodificador, la salida correspondiente se pondrá en 1 y, por tanto, de la puerta OR saldrá un 1. Cuando las variables construyan una combinación que haga que la función valga 0, todas las entradas de la puerta OR valdrán 0 y, por tanto, también su salida.

La figura 23 muestra un ejemplo de esto. Podemos comprobar que la salida de la puerta OR valdrá 1 cuando valga 1 la salida $s2$ del descodificador, o la $s5$, la $s6$ o la $s7$. Es decir, cuando las variables de entrada construyan alguna de las combinaciones que hagan que la función valga 1.

Figura 23

x_2	x_1	x_0	f
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1



Si no dibujamos la entrada de validación en un descodificador, asumiremos que está en 1.

Cuando diseñamos un circuito usando bloques, podemos dejar una o más salidas de los bloques sin conectar a ningún sitio. Sin embargo, no podemos dejar ninguna entrada sin conectar, puesto que el comportamiento sería indeterminado.

Actividades

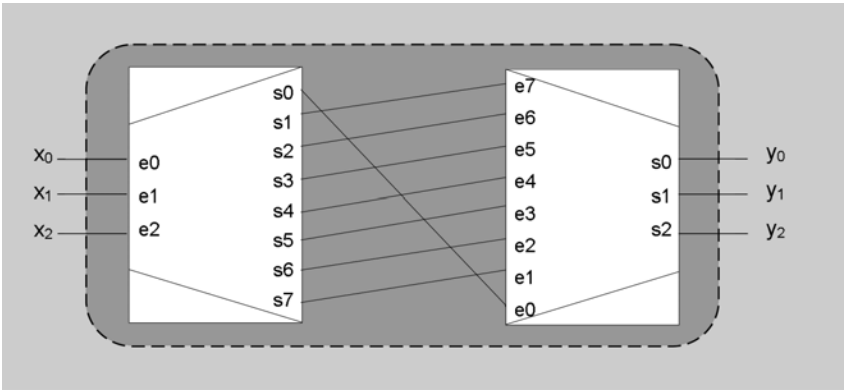
35. Deducid la implementación interna (mediante puertas lógicas) de un codificador 4-2.

36. Deducid la implementación interna (mediante puertas lógicas) de un descodificador 2-4.

37. Implementad la función $f(a, b, c) = abc' + a'c$ con un descodificador 3-8.

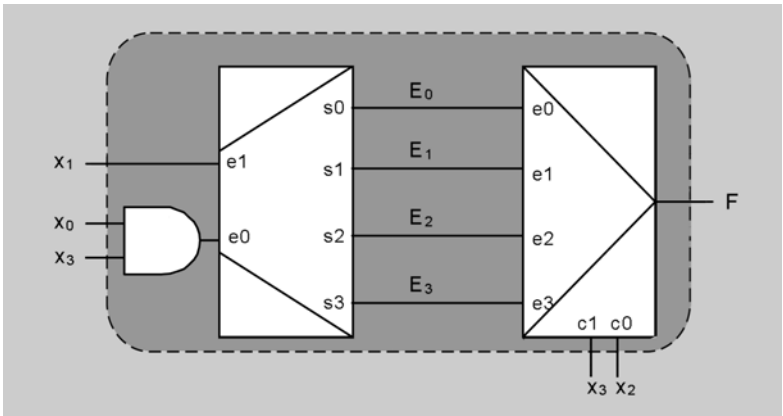
38. Diseñad un circuito que genere la representación en signo y magnitud de números en el rango [–7, 7]. El circuito debe tener estas entradas y salidas:
- Ocho entradas de un bit, *e*7, *e*6, ... *e*0, de las cuales como mucho una estará activa en cada momento. Si la que está a 1 es *ei*, la magnitud del número que hay que representar es *i*.
- Otra entrada de un bit *sg*, que indica el signo del número que hay que representar (0 positivo, 1 negativo).
- Cuatro salidas de un bit, *s*3... *s*0, que contendrán la representación en signo y magnitud del número que se quiere representar. El número 0 se representará siempre con el bit de signo a 0.
- Otra salida de un bit, *null*, que valdrá 1 sólo cuando no se indique ninguna magnitud para ser representada. En este caso, las salidas *s*3... *s*0 también valdrán 0.

39. ¿Qué hace el siguiente circuito suponiendo que $X = (x_2, x_1, x_0)$ y $Y = (y_2, y_1, y_0)$ son números enteros codificados en complemento a 2?



40. Construid un circuito que implemente la función $Y = (X + 3) \bmod 4$, siendo X e Y números naturales representados en binario con 2 bits. El circuito sólo puede contener dos bloques y ninguna puerta.

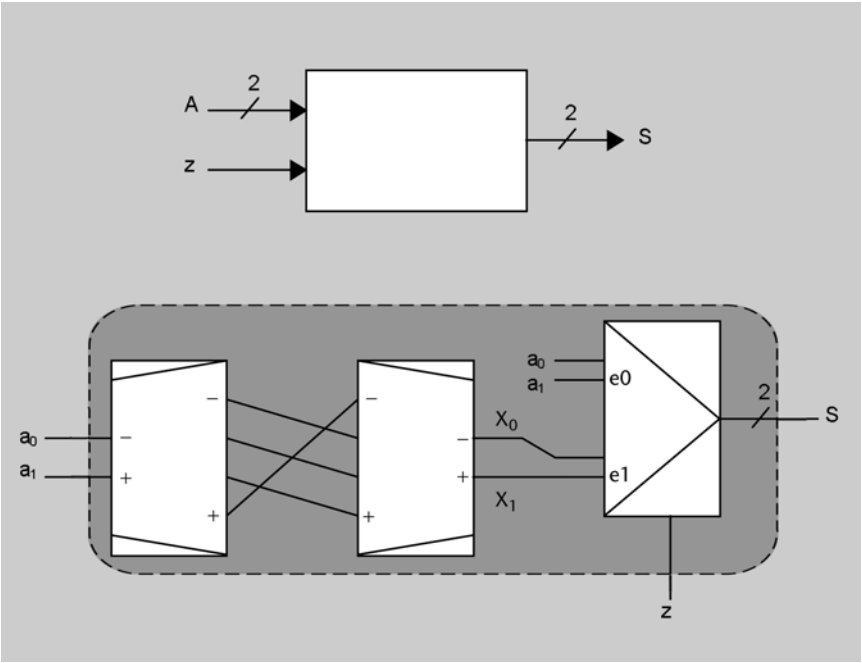
41. Dado el siguiente circuito:



a) Encontrad las expresiones algebraicas de E_i ($i = 0, 1, 2, 3$) en función de x_3, x_1 y x_0.
b) Escribid la tabla de verdad de $F(x_3, x_2, x_1, x_0)$.

42. Diseñad un descodificador 3-8 utilizando dos descodificadores 2-4 y las puertas lógicas que hagan falta.

43. ¿Qué función cumple el siguiente circuito si interpretamos las entradas y la salida como números codificados en binario natural?



3.3. Desplazadores lógicos y aritméticos

Un **desplazador** es un bloque combinacional que tiene la función de desplazar bits hacia la izquierda o hacia la derecha.

Los desplazadores tienen una señal de entrada y una señal de salida, las dos de n bits. La señal de salida se obtiene desplazando los bits de entrada m veces hacia la derecha o hacia la izquierda.

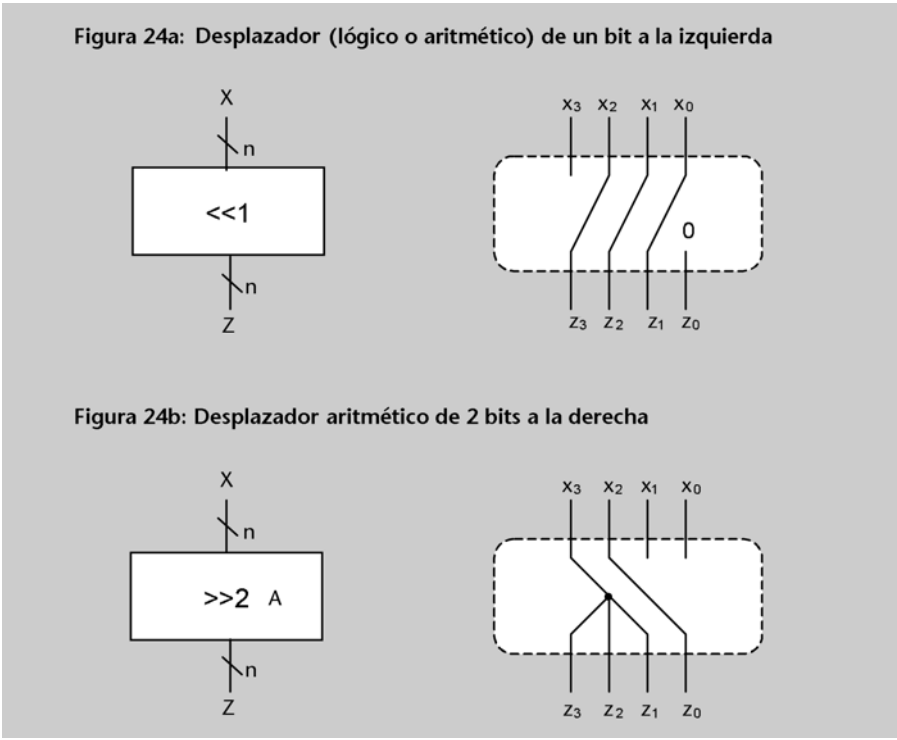
1. Si el desplazamiento es hacia la izquierda, los m bits de menos peso de la salida se ponen a 0.

2. Si el desplazamiento es hacia la derecha, existen dos posibilidades para los m bits de más peso de la salida:

- En los **desplazadores lógicos** se ponen a 0.
- En los **desplazadores aritméticos** toman el valor del bit de más peso de la entrada. Fijémonos en que, si interpretamos las entradas y salidas como números codificados en complemento a 2 con n bits, los desplazadores aritméticos mantienen en la salida el signo de la entrada.

Las figuras 24*a* y 24*b* muestran la representación gráfica y la implementación interna de un desplazador de un bit a la izquierda y de un desplazador aritmético de dos bits a la derecha, respectivamente, de señales de cuatro bits. En el caso de los desplazadores a la derecha, usaremos la letra *L* para indicar que son lógicos y la letra *A* para indicar que son aritméticos.

Figura 24



Si interpretamos las entradas y salidas como codificaciones de números naturales, podemos decir que los desplazadores cumplen las funciones de multiplicar y dividir por potencias de 2 (división entera). Un desplazador de m bits a la izquierda multiplica por 2^m, y un desplazador lógico de m bits a la derecha realiza la división entera por 2^m. Si interpretamos los números como codificados en complemento a 2, entonces para dividir por 2^m hay que usar desplazadores aritméticos.

Actividades

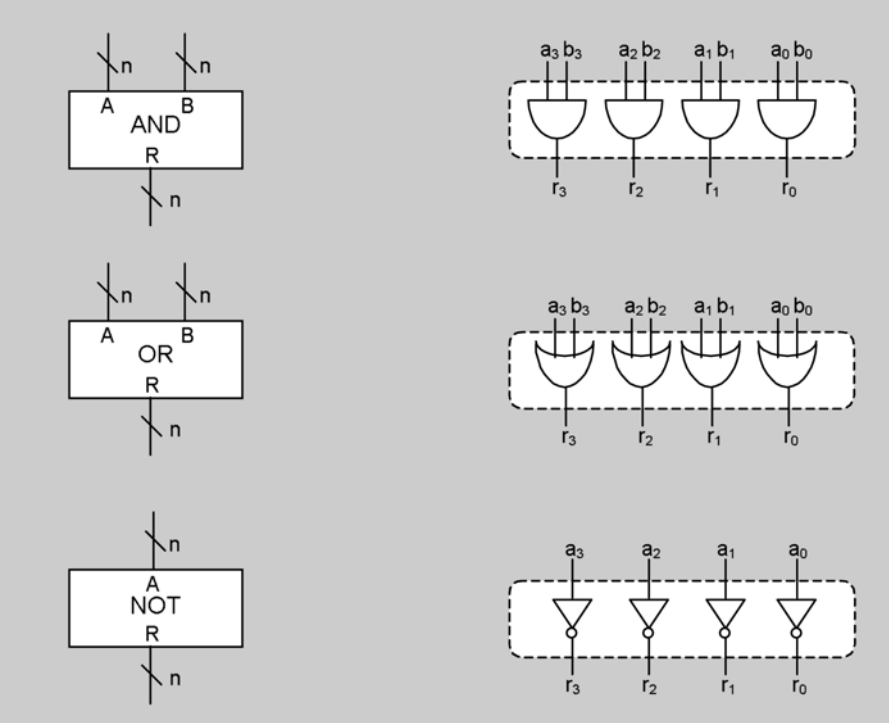
44. Contestad los siguientes apartados:
a) Implementad un circuito que pueda actuar como un desplazador de un bit a la izquierda de señales de cuatro bits usando un multiplexor 2-1 de buses de cuatro bits. Una señal de control de un bit, *d*, determina si el número de entrada se debe desplazar o no.
b) ¿A qué operación aritmética equivale este desplazamiento?
c) Indicad cuándo se produciría desbordamiento en los casos de interpretar la señal de entrada como un número natural codificado en binario o bien como un número entero codificado en complemento a 2.

3.4. Bloques AND, OR y NOT

Estos bloques hacen las operaciones AND, OR y NOT bit a bit sobre entradas de n bits.

Tienen dos entradas de datos (sólo una en el caso del bloque NOT) y una salida, todas de n bits. La figura 25 muestra su representación gráfica y la implementación interna para el caso $n = 4$.

Figura 25



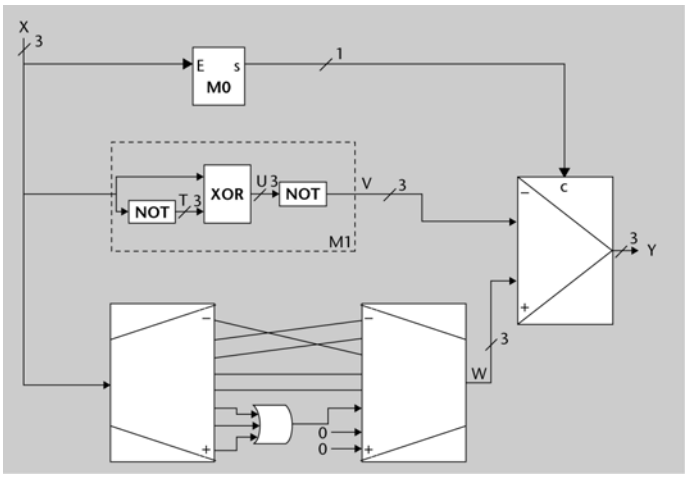
Podemos construir bloques análogos a estos, por ejemplo un bloque XOR sería aquel que hace la XOR bit a bit de las entradas.

Actividades

45. Diseñad un circuito con una entrada X por la que llegan números naturales representados en binario con 8 bits y una salida Z de 8 bits, de manera que $Z = X'$ si X es múltiple de 4, y $Z = x_3x_200$ si no lo es.

46. Sea el circuito lógico combinacional siguiente, en el que la entrada X y la salida Y codifican números naturales en binario y 3 bits y el bloque $M0$ tiene este comportamiento:

$$s = x_2x_1x_0 + x_2x_1x_0' + x_2x_1'x_0' + x_2x_1'x_0 + x_2'x_1x_0$$



a)Implementad el bloque *M*0 con el mínimo número de bloques combinacionales y, si fuera necesario, puertas lógicas.

b)¿Qué función hace el subcircuito identificado como *M*1?

c)Escribid la tabla de verdad del circuito completo.

3.5. Memoria ROM

La **memoria ROM** es un bloque combinacional que permite guardar el valor de 2^m palabras de n bits.

ROM

La denominación ROM deriva del inglés *read only memory*, porque se refiere a las memorias en las que no se pueden hacer escrituras, sino sólo lecturas.

Podemos ver una memoria ROM como un archivador con cajones que guardan bits. Cada cajón tiene capacidad para guardar un cierto número de bits (todos tienen la misma) y el archivador tiene un número determinado de cajones, que es siempre una potencia de dos.

La memoria ROM tiene los siguientes elementos:

1. 2^m palabras o datos de n bits, cada una en una posición (cajón) diferente de la memoria ROM. Las posiciones que contienen los datos están numeradas desde el 0 hasta el $2^m - 1$; estos números se llaman **direcciones**.

2. Una entrada de direcciones de m bits, que se identifica con símbolo @. Los m bits de la entrada de direcciones se interpretan como números codificados en binario (y, por tanto, es necesario determinar el peso de cada bit).

3. Una salida de datos de n bits.

El funcionamiento de la ROM es el siguiente:

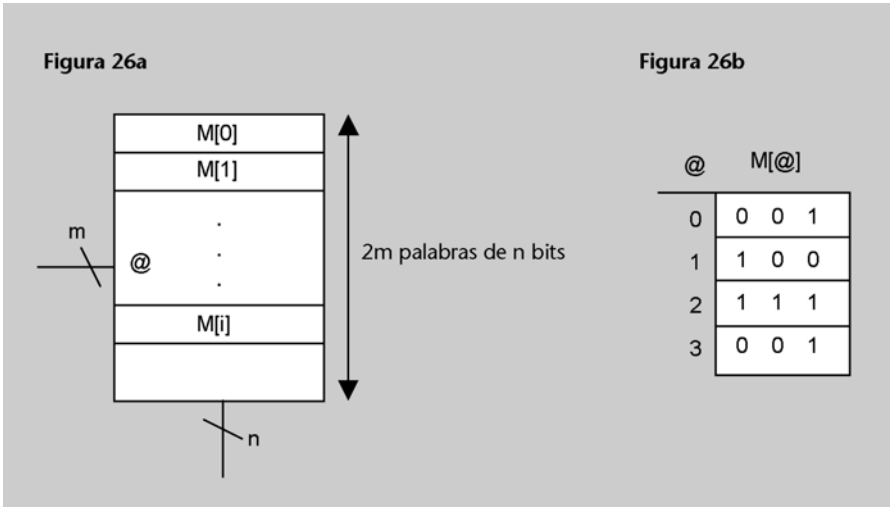
Cuando los m bits de la entrada de direcciones (interpretados en binario) codifican el número i, entonces la salida toma el valor del dato que hay almacenado en la dirección i. Para referirnos a este dato usaremos la notación **M[*i*]**, y diremos que **leemos** el dato de la dirección i.

Así pues, sólo se puede acceder al valor de una palabra (leerla) en cada instante (es como si en cada momento sólo se pudiera abrir un cajón).

La figura 26*a* muestra cómo representaremos la memoria ROM. La figura 26*b* muestra un posible contenido de una memoria ROM de cuatro palabras de 3 bits. En este ejemplo,

M[0] = 001
M[1] = 100
M[2] = 111
M[3] = 001.

Figura 26



La memoria ROM también se puede utilizar para sintetizar funciones lógicas. Por ejemplo, si conectamos las variables x_1 y x_0 (en orden de peso) a las entradas de direcciones de la ROM de la figura 26b, entonces podemos interpretar los tres bits de salida como la implementación de las tres funciones siguientes:

x_1	x_0	f_2	f_1	f_0
0	0	0	0	1
0	1	1	0	0
1	0	1	1	1
1	1	0	0	1

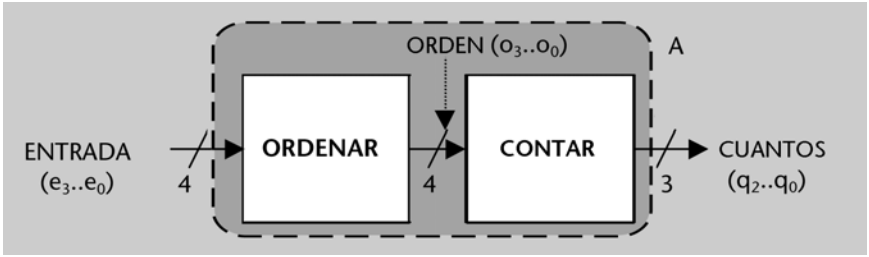
Actividades

47. Indicad el tamaño y el contenido de una memoria ROM que implemente la función descrita en la actividad 24.

48. Indicad el tamaño y el contenido de una memoria ROM que implemente la función descrita en la actividad 19.

49. Utilizando sólo una memoria ROM, se quiere diseñar un circuito que, dado un número X representado en signo y magnitud con 8 bits, dé a la salida el mismo número representado en complemento a 2 y 8 bits. Describid el contenido que debe tener la memoria ROM (sin escribir todo su contenido) e indicad el tamaño mínimo que debe tener. Escribid el contenido de las palabras de las direcciones 50, 100 y 150.

50. El circuito combinacional de la figura cuenta el número de unos que tiene una palabra de entrada de cuatro bits.



El bloque ORDENAR ordena la palabra ENTRADA y coloca todos los unos a la derecha y todos los ceros a la izquierda. Por ejemplo:

Si ENTRADA = 0101, entonces ORDEN = 0011.
Si ENTRADA = 0000, entonces ORDEN = 0000.

El bloque CONTAR genera en la salida CUANTOS la codificación binaria de la cantidad de unos de la palabra ORDEN. Por ejemplo:

Si ORDEN = 0011, entonces CUANTOS = 010 (2).
Si ORDEN = 0000, entonces CUANTOS = 000 (0).

Resolved los siguientes apartados:

a) Escribid la tabla de verdad del bloque ORDENAR.
b) Diseñad el bloque CONTAR utilizando sólo bloques combinacionales (exceptuando memoria ROM), del tamaño que haga falta.
c) Diseñad el circuito combinacional completo A con una memoria ROM e indicad su tamaño y contenido.

3.6. Comparador

El **comparador** es un bloque combinacional que compara dos números codificados en binario e indica qué relación existe entre éstos.

Un comparador tiene las siguientes señales:

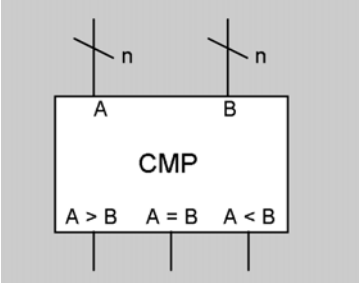
1. Dos entradas de datos de n bits, que reciben los nombres A y B. Se interpretan como números naturales codificados en binario.

2. Tres salidas de 1 bit, de las que sólo una vale 1 en cada momento:

- La salida $A > B$ vale 1 si el número que llega por la entrada A es mayor que el que llega por la entrada B.
- La salida $A = B$ vale 1 si los dos números de entrada son iguales.
- La salida $A < B$ vale 1 si el número que llega por la entrada A es menor que el que llega por la entrada B.

La figura 27 muestra la representación gráfica de un comparador.

Figura 27



Actividades

51. Deducid las expresiones algebraicas de las funciones de salida $A = B$ y $A < B$ de un comparador de números de 2 bits y haced su implementación mediante puertas lógicas.

52. Diseñad un circuito que obtenga en su salida el máximo de dos números naturales, X e Y, de 4 bits.

53. Utilizando cualesquiera de los bloques y puertas que se han visto, diseñad un circuito con dos entradas X e Y que codifican números naturales en binario y 3 bits y una salida S de 2 bits, que funcione de la manera siguiente:

- Si $X > Y$, S debe valer 01.
- Si $X < Y$, S debe valer 10.
- Si $X = Y$ y son diferentes de 0, S debe valer 00.
- Si $X = Y$ y son iguales a 0, S debe valer 11.

54. Se dispone de dos hornos, cada uno de los cuales viene equipado con un termómetro digital para medir la temperatura interior. Los termómetros generan una señal de 8 bits que codifica la temperatura en binario (los hornos siempre están entre 0 °C y 255 °C).

Utilizando cualesquiera de los bloques y puertas que se han visto, diseñad un circuito lógico combinacional que, recibiendo por las entradas A y B la temperatura de los dos hornos, informe mediante la salida R, de 2 bits, en qué rango de temperaturas se encuentra el horno que está más caliente de los dos; si los dos están a la misma temperatura, indicad en qué rango se encuentra esta. La correspondencia entre rangos de temperaturas y valor de la señal R es como sigue:

Temperatura del horno que está más caliente (o igual)	*R*
[11000000, 11111111]	11
[10000000, 10111111]	10
[01000000, 01111111]	01
[00000000, 00111111]	00

3.7. Sumador

El **sumador** es un bloque combinacional que realiza la suma de dos números codificados en binario o bien en complemento a 2.

Recordad

Tal como se ha estudiado en el módulo "Representación de la información numérica", la representación de los enteros en complemento a 2 permite hacer las sumas utilizando la misma mecánica que en las sumas de números codificados en binario. Por tanto, si un bloque realiza la suma de números codificados en binario, su salida también será la suma correcta si interpretamos los números de entrada como enteros codificados en complemento a 2. Por eso decimos que el sumador suma números codificados en binario o bien en complemento a 2.

La figura 28 muestra la representación gráfica de un sumador de n bits. Las señales que tiene son las siguientes:

1. Dos entradas de datos de n bits, que llamaremos A y B, por donde llegarán los números que se tienen que sumar.

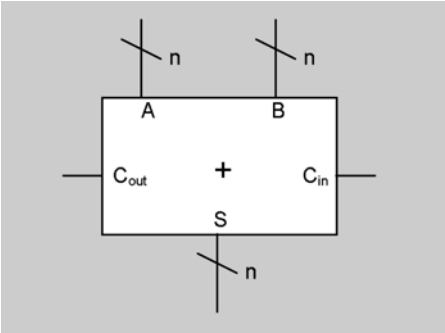
2. Una salida de n bits, llamada S, que tomará el valor de la suma de los números A y B.

3. Una salida de 1 bit, C_{out}, que valdrá 1 si cuando se haga la suma se produce acarreo en el bit de más peso.

La nomenclatura de las señales de entrada y salida de acarreo (C) deriva de *carry*, que es la palabra inglesa que se utiliza para acarreo.

4. Una entrada de 1 bit, C_{in}, por donde llega un acarreo de entrada.

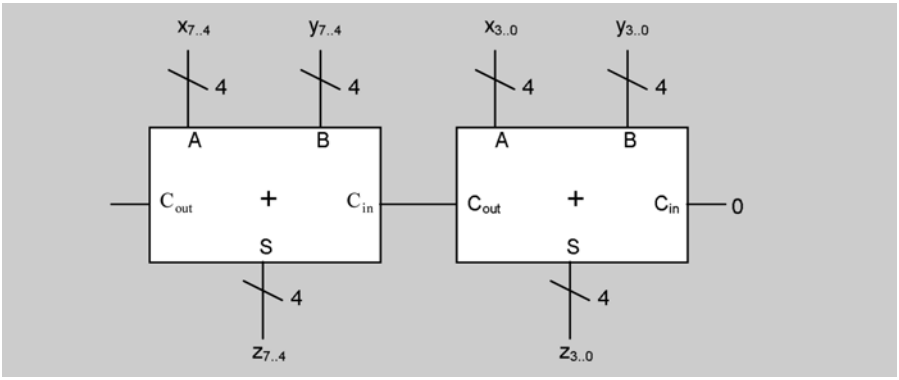
Figura 28



La entrada C_{in} es útil cuando se quieren sumar números de $2 \cdot n$ bits y sólo se dispone de sumadores de n bits. En este caso se encadenan dos sumadores: el primero suma los n bits más bajos de los números y el segundo, los n bits más altos. La salida de C_{out} del primer sumador se conecta con la entrada C_{in} del segundo para que el resultado sea correcto. La figura 29 ofrece un ejemplo para el caso $n = 4$, en el que hacemos la suma $Z = X + Y$, siendo X, Y y Z números de 8 bits. Fijémonos en que el sumador que suma los bits más bajos lo dibujamos a la derecha, porque así los bits quedan ordenados de la manera en que estamos acostumbrados a verlos.

Si no necesitamos tener en cuenta un acarreo de entrada, conectaremos un 0 a la entrada C_{in}.

Figura 29



Como ya sabemos, el hecho de limitar la longitud de los números a un determinado número de bits (n) tiene como consecuencia que el resultado de una suma no sea siempre correcto (es incorrecto cuando se produce exceso de capacidad, es decir, cuando el resultado requiere más de n bits para ser codificado). En las sumas binarias, sabemos que si se produce acarreo en el bit de más peso entonces el resultado es incorrecto. En cambio, en las sumas en complemento a 2 no existe ninguna relación entre el acarreo y el desbordamiento.

Recordad

El concepto de desbordamiento que se ha estudiado en el módulo "Representación de la información numérica".

Por tanto, la salida C_{out} de un sumador nos indica que se ha producido un desbordamiento si interpretamos las entradas en binario, pero no nos dice nada sobre si el resultado es correcto si las interpretamos en complemento a 2.

Dado que $X - Y = X + (-Y)$, los sumadores se pueden utilizar también para hacer la resta de dos números, si cambiamos el signo del segundo operando antes de conectarlo al sumador.

Recordad

La operación de cambio de signo que se ha estudiado en el módulo "Representación de la información numérica".

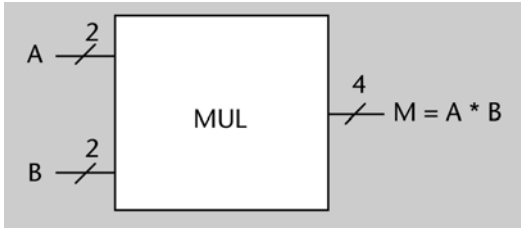
Actividades

55. Suponed que X es un número natural codificado con 3 bits e implementad la función $Z = (3 \cdot X) \bmod 8$ usando sólo un sumador de cuatro bits y un desplazador.

56. Diseñad un circuito que haga la operación $Z = X - Y$ siendo X, Y y Z números codificados en complemento a 2 con n bits.

57. Usando bloques combinacionales, diseñad un circuito que dado un número entero X codificado en complemento a 2 y 3 bits obtenga su valor absoluto, también en complemento a 2 y 3 bits. El circuito debe indicar si se produce desbordamiento en el cálculo poniendo la salida V (de un bit) a 1.

58. Se dispone de un circuito combinacional que implementa un multiplicador de dos números enteros, representados en complemento a 2 con dos bits. La salida de un circuito multiplicador de números enteros siempre tiene el doble de bits que las entradas.



a) Escribid la tabla de verdad del bloque MUL.
b) Implementad con bloques combinacionales un circuito que calcule la siguiente operación:

$$Z = A^4 + A^3 + A^2,$$

donde A es un número entero de dos bits representado en complemento a 2. Podéis usar bloques de cualquier número de bits (especificad de cuántos y justificad la respuesta), incluso el del apartado *a*, que no es necesario diseñar. Una posible solución sólo requiere un sumador y dos multiplicadores.

59. A, B, C y D son señales de 8 bits que codifican números naturales. Se quiere diseñar un circuito que implemente la función Y, que se describe así:

$Y = 2 \times C$	si	$C = D$
$Y = (A + B + C) / 4$	si	$C \neq D$

Los buses de entrada y salida de cada bloque deben tener el menor ancho posible para conseguir que en ningún momento se produzcan desbordamientos.

3.8. Unidad aritmética y lógica (UAL)

Una **unidad aritmética y lógica** es un aparato capaz de efectuar un determinado conjunto de operaciones aritméticas y lógicas sobre dos números de entrada codificados en binario o en complemento a 2.

Las unidades aritméticas y lógicas se llaman *UAL* o, también, *ALU* (del inglés *arithmetic and logic unit*).

Para diseñar una UAL es necesario especificar el conjunto de operaciones que queremos que haga. Por ejemplo, una UAL puede hacer la suma, la resta, la AND y la OR de los operandos de entrada. En cada momento se especificará en la UAL cuál es la operación que debe hacer.

En este curso sólo estudiaremos UAL que operen con números naturales y enteros. Los procesadores tienen, además, UAL para operar con números reales codificados en coma flotante.

Las señales que tiene una UAL son las siguientes:

1. Dos entradas de datos de n bits, A y B, por donde llegarán los números sobre los que se deben hacer las operaciones.

2. Una salida de datos de n bits, R, donde se obtendrá el resultado de la operación.

3. Un cierto número de entradas de control, c_i, de un bit cada una. Si la UAL es capaz de hacer 2^m operaciones diferentes, debe tener m entradas de control. Cada combinación de las entradas de control indicará a la UAL que haga una operación concreta. Por ejemplo, la UAL de la figura 30 puede hacer cuatro operaciones.

4. Un cierto número de salidas de un bit, que llamamos ***bits de estado***, y tienen la función de indicar algunas circunstancias que se pueden haber producido durante el cálculo.

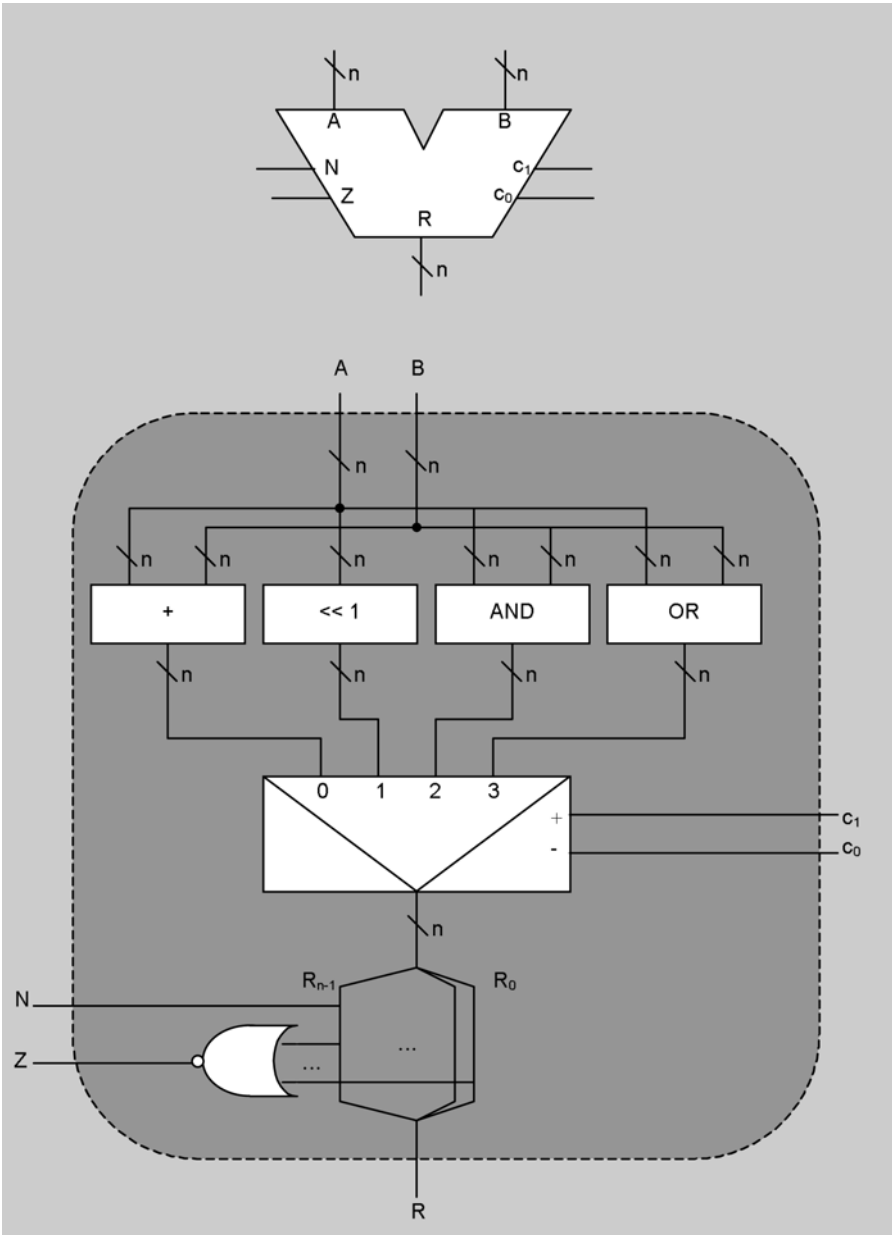
El bit de desbordamiento se suele identificar con las letras *O* o *V*, que derivan de la palabra inglesa ***overflow***, que significa 'desbordamiento'.

Los bits de estado más habituales son los que indican que se ha producido acarreo en el último bit (C), si se ha producido desbordamiento (V), si el resultado de la operación ha sido negativo (N) o si el resultado de la operación ha sido 0 (Z).

La figura 30 muestra la representación gráfica y la implementación interna de una UAL que genera los bits de estado N y Z y que realiza las siguientes operaciones:

c_1	c_0	R
0	0	$A + B$
0	1	$2 * A$
1	0	a AND b
1	1	a OR b

Figura 30



Nota

En esta figura hemos disgregado los bits del bus correspondiente a la señal de salida R para dibujar la implementación de los bits *N* y *Z*. Se tiene que interpretar que todos los bits de *R* se conectan a la puerta NOR (de *n* entradas) que computa *Z*.

Actividades

60. Contestad los siguientes apartados:

a) Diseñad una UAL que, a partir de dos entradas de control c_1 y c_0, realice las siguientes operaciones sobre dos números A y B de cuatro bits:

- $[c_1\ c_0] = [0\ 0] : R = A + B$.
- $[c_1\ c_0] = [0\ 1] : R = A - B$.
- $[c_1\ c_0] = [1\ 0] : R = A$.
- $[c_1\ c_0] = [1\ 1] : R = -A$.

b) Añadid a la UAL diseñada en el apartado anterior los circuitos necesarios para calcular los siguientes bits de condición:

- *Vb*: desbordamiento si se interpreta que los números de entrada están codificados en binario.
- *V*: desbordamiento si se interpreta que los números de entrada están codificados en complemento a 2.
- *N:* $N = 1$ si el resultado de la operación interpretado en complemento a 2 es negativo, y 0 en el caso contrario.
- *Z:* $Z = 1$ si el valor de la salida es 0, y $Z = 0$ en el caso contrario.

Resumen

En este módulo se han estudiado los fundamentos de los circuitos electrónicos digitales. Se ha visto que se construyen a partir de los mismos elementos que la lógica natural, formalizada matemáticamente por el álgebra de Boole: los elementos básicos son los valores 0 y 1 y las operaciones suma lógica, producto lógico y negación. A partir de aquí se han introducido las variables y funciones lógicas.

Se han expuesto dos maneras de representar funciones lógicas: las expresiones algebraicas y las tablas de verdad. Se ha visto que son especialmente cómodas las expresiones en suma de productos.

Después se ha visto cómo se implementan físicamente las operaciones lógicas básicas para construir circuitos mediante las puertas lógicas. Se ha aprendido la manera de minimizar circuitos a dos niveles, mediante el método de minimización de Karnaugh.

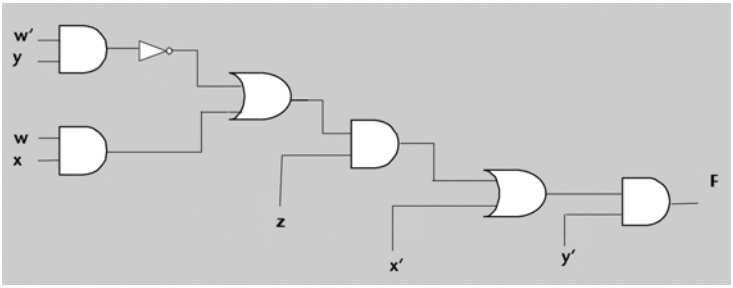
Finalmente, se han conocido varios bloques combinacionales, y se ha aprendido a utilizar la funcionalidad de cada uno de ellos para diseñar y entender circuitos complejos.

Ejercicios de autoevaluación

Entre paréntesis se indica el apartado de los apuntes que se debe haber estudiado para resolver cada ejercicio.

1. (1.2) Demostrad que las leyes de De Morgan también se cumplen para tres y cuatro variables. Es suficiente con demostrar que $(xyz)' = x' + y' + z'$ y que $(xyzw)' = x' + y' + z' + w'$, porque las otras leyes se obtienen directamente aplicando el principio de dualidad.

2. (2.1) Analizad el siguiente circuito combinacional y expresad algebraicamente en forma de suma de productos la función que implementa:



3. (2.2) Diseñad a dos niveles un circuito combinacional que permita multiplicar dos números enteros de dos bits representados en complemento a 2.

4. (2.3) Sintetizad de manera mínima a dos niveles el circuito descrito en la actividad 24.

5. (3.1) Diseñad un multiplexor 16-1 utilizando únicamente multiplexores 4-1 (sin utilizar ninguna puerta lógica adicional).

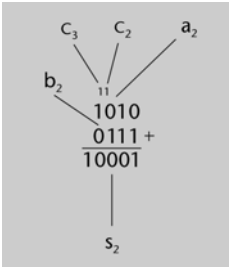
6. (3.2) Diseñad un descodificador 4-16 utilizando únicamente cinco descodificadores 2-4.

7. (3.2) Suponed que X es un número natural codificado con tres bits. Implementad la función $(3 \cdot X)$ mod 8 usando sólo un descodificador 3-8 y un codificador 8-3.

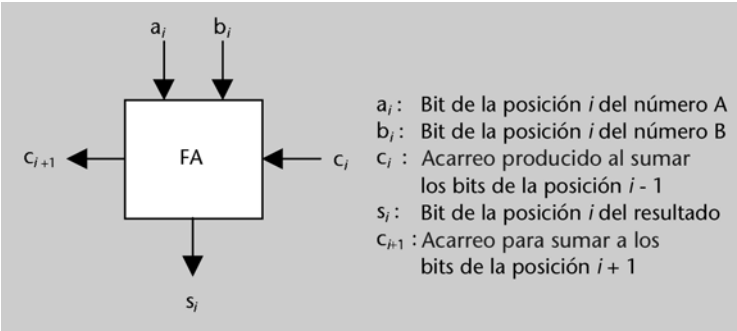
8. (3.3) Resolved los siguientes apartados:
a) Implementad un *desplazador programable a la izquierda* de números de ocho bits. El desplazador tiene una entrada de control de tres bits, C, que indica el número de bits del desplazamiento.
b) Si C codifica en binario el valor n, ¿a qué operación aritmética equivale este desplazamiento?
c) Suponed que $C = 010$ e indicad cuándo se produciría desbordamiento en los casos de interpretar la entrada como un número natural o bien como uno entero representado en complemento a 2.
Supongamos ahora que queremos hacer la operación $X / 2^n$ usando un desplazador en la derecha.
d) ¿Qué tipo de desplazador tenemos que utilizar si X es un número natural representado en binario? ¿Y si es un número entero representado en complemento a 2?
e) ¿En qué casos el resultado de la división no es exacto?

9. (3.6) Sintetizad un comparador de números enteros de 4 bits codificados en complemento a 2, utilizando comparadores de números naturales de cuatro bits y las puertas lógicas necesarias.

10. (3.7) Cuando hacemos una suma de números binarios, la hacemos bit a bit (de la misma manera que en decimal la hacemos dígito a dígito). Para obtener el bit de la posición i necesitamos conocer los bits que se encuentran en la posición i en los dos operandos y el acarreo que se genera en la suma de los bits de la posición $i - 1$.
La suma de estos 3 bits da como resultado 2 bits: el bit s_i, que corresponde al bit de la posición i de la suma, y el bit $c_i + 1$, que es el acarreo que se genera para la suma de los bits de la posición $i + 1$.
A continuación se muestra un ejemplo para $A = 1010$ y $B = 0111$. El ejemplo ofrece concretamente la suma de los bits de la posición 2 (recordad que el bit más a la derecha es el de la posición 0).



El sumador de números de un bit tiene tres entradas (a_i, b_i y c_i) y dos salidas (s_i y c_{i+1}). Este sumador recibe el nombre de *full adder* (sumador completo), y se abrevia FA. La figura siguiente muestra la estructura.



a) Escribid la tabla de verdad y realizad la implementación interna (mediante puertas lógicas) de un FA como el descrito.

b) Utilizad cuatro FA para construir un sumador de dos números de cuatro bits. ¿Qué hay que conectar a la entrada que corresponde al bit c_0?

11. (3.7) Diseñad un circuito que sea capaz de sumar o restar dos números codificados en complemento a 2 con 4 bits de acuerdo con lo que valga la señal de entrada *s'/r*: si vale 0, el circuito debe realizar la suma, y si vale 1, debe realizar la resta. El circuito debe generar además una señal de salida *C* que indique si se ha producido acarreo en el último bit.

12.(3.7) Queremos diseñar un circuito que controle los ascensores de un edificio de cinco plantas y planta baja. El edificio tiene dos ascensores, *A* y *B*, cada uno con dos señales asociadas:

- Una señal que indica en binario en qué planta está el ascensor en cada momento (*plantaA* y *plantaB*).
- Una señal de activación (*actA*, *actB*).

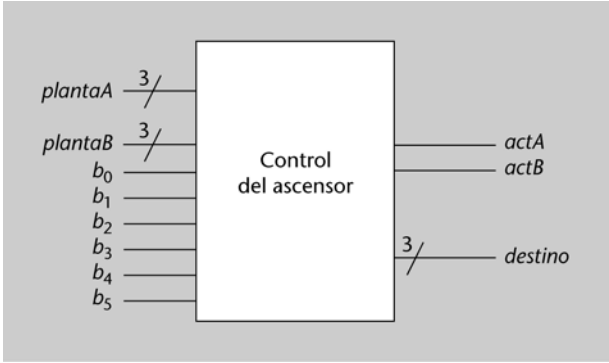
Cuando una señal de activación se pone a 1, el ascensor correspondiente se pone en movimiento; la señal *destino* le indica, en binario (número natural), hacia qué planta debe ir. Después de que el ascensor se ponga en movimiento, la señal de activación vuelve a 0 automáticamente (no es preciso que nos preocupemos de hacerlo).

En cada planta *i* ($i = 0, 1, ..., 5$) hay un botón para llamar al ascensor, conectado a la señal b_i. Cuando alguien lo llama desde la planta *i*-ésima, la señal b_i se pone a 1 y vuelve a 0 cuando el ascensor ha llegado a la planta correspondiente.

Mientras no haya ninguna llamada, los ascensores deben permanecer detenidos. Cuando alguien llame al ascensor desde una planta, se debe mover el que esté más cerca. Si los dos están a la misma distancia, irá el ascensor *A*.

Para facilitar el diseño, supondremos algunas simplificaciones en el sistema:

- Nunca se pulsará más de un botón de llamada al mismo tiempo.
- En el momento de pulsar un botón de llamada, los dos ascensores están detenidos.



Diseñad el circuito de control de los ascensores utilizando los bloques y las puertas que sean necesarios, y teniendo cuidado de dar a todos los buses el ancho mínimo necesario. Si queréis, podéis utilizar como bloques los circuitos que se diseñan en las actividades 56 y 57, dimensionando la amplitud de las entradas y salidas según os sea necesario.

13. (3.8) Diseñad una UAL con salida R de 4 bits que, a partir de tres entradas de control c_2, c_1 y c_0, realice las operaciones siguientes sobre dos números A y B de 4 bits:

- $[c_2\, c_1\, c_0] = [0\ x\ x] : R = B$.
- $[c_2\, c_1\, c_0] = [1\ 0\ 0] : R = A + B$.
- $[c_2\, c_1\, c_0] = [1\ 0\ 1] : R = A - B$.
- $[c_2\, c_1\, c_0] = [1\ 1\ 0] : R = B >> 1$.
- $[c_2\, c_1\, c_0] = [1\ 1\ 1] : R = A$ AND B.

Solucionario

Actividades

1. Para evaluar la expresión $x + y \cdot (z + x')$ sustituiremos las variables para cada combinación de valores:

Para el caso de $[x\ y\ z] = [0\ 1\ 0]$, tenemos lo siguiente:
$$0 + 1 \cdot (0 + 1) = 0 + 1 \cdot 1 = 0 + 1 = 1.$$
Para el caso $[x\ y\ z] = [1\ 1\ 0]$, tenemos lo siguiente.
$$1 + 1 \cdot (0 + 0) = 1 + 1 \cdot 0 = 1 + 0 = 1.$$
Para el caso $[x\ y\ z] = [0\ 1\ 1]$, tenemos lo siguiente:
$$0 + 1 \cdot (1 + 1) = 0 + 1 \cdot 1 = 0 + 1 = 1.$$

2. Para la demostración, sustituiremos las variables de entrada por todas las combinaciones de valores que puedan tomar. Las igualdades se deben cumplir para todos los casos.

- Igualdad $(x + y)' = x' \cdot y'$
  - Caso $x = 0$ y $y = 0$:
    $(x + y)' = (0 + 0)' = 0' = 1$,
    $x' \cdot y' = 0' \cdot 0' = 1 \cdot 1 = 1$.
  - Caso $x = 0$ y $y = 1$:
    $(x + y)' = (0 + 1)' = 1' = 0$,
    $x' \cdot y' = 0' \cdot 1' = 1 \cdot 0 = 0$.
  - Caso $x = 1$ y $y = 0$:
    $(x + y)' = (1 + 0)' = 1' = 0$,
    $x' \cdot y' = 1' \cdot 0' = 0 \cdot 1 = 0$.
  - Caso $x = 1$ y $y = 1$:
    $(x + y)' = (1 + 1)' = 1' = 0$,
    $x' \cdot y' = 1' \cdot 1' = 0 \cdot 0 = 0$.

Puesto que las dos expresiones valen lo mismo para todos los casos posibles, concluimos que son equivalentes.

- Igualdad $(x \cdot y)' = x' + y'$
  - Caso $x = 0$ y $y = 0$:
    $(x \cdot y)' = (0 \cdot 0)' = 0' = 1$,
    $x' + y' = 0' + 0' = 1 + 1 = 1$.
  - Caso $x = 0$ y $y = 1$:
    $(x \cdot y)' = (0 \cdot 1)' = 0' = 1$,
    $x' + y' = 0' + 1' = 1 + 0 = 1$.
  - Caso $x = 1$ y $y = 0$:
    $(x \cdot y)' = (1 \cdot 0)' = 0' = 1$,
    $x' + y' = 1' + 0' = 0 + 1 = 1$.
  - Caso $x = 1$ y $y = 1$:
    $(x \cdot y)' = (1 \cdot 1)' = 1' = 0$,
    $x' + y' = 1' + 1' = 0 + 0 = 0$.

Puesto que las dos expresiones valen lo mismo para todos los casos posibles, concluimos que son equivalentes.

3. La ley 4 de álgebra de Boole dice que $x + 1 = 1$ y que $x \cdot 0 = 0$.

Demostraremos sólo la primera igualdad, la segunda quedará demostrada por el principio de dualidad.

El axioma *e* dice que $x + x' = 1$. Por tanto,

$$x + 1 = x + x + x'.$$

La ley 2 (de idempotencia) dice que $x + x = x$. Por tanto,

$$x + x + x' = x + x' = 1.$$

Hemos obtenido la última igualdad aplicando de nuevo el axioma *e*. Por tanto, $x + 1 = 1$, tal como queríamos demostrar.

4.

- La función g_1 vale 1 sólo cuando x vale 1 e y vale 1. Por tanto, una posible expresión lógica para esta función es $g_1(x, y) = x \cdot y$.
- La función g_3 vale 1 cuando x vale 1 e y vale 0, o bien cuando x vale 1 e y vale 1. Una posible expresión por función es, por tanto, $x \cdot y' + x \cdot y$. También podemos apreciar que la

función vale 1 cuando la variable x vale 1, independientemente del valor que tome la variable y. Por tanto, una posible expresión lógica para esta función es $g_3(x, y) = x$.
- La función g_6 vale 1 cuando x vale 1 e y vale 0, o bien cuando x vale 0 e y vale 1. Por tanto, otra posible expresión lógica para esta función es $g_6(x, y) = xy' + x'y$.
- La función g_7 vale 1 cuando x vale 0 e y vale 1, cuando x vale 1 e y vale 0, o bien cuando x vale 1 e y vale 1. Por tanto, una posible expresión lógica para esta función es $g_7(x, y) = x'y + xy' + xy$. También podemos decir que la función vale 1 siempre que no se cumpla que x = 0 e y = 0. Por tanto, otra expresión para la función es $g_7(x, y) = (x'y')'$. Si aplicamos la segunda ley de De Morgan sobre esta expresión, y después de la ley de involución, obtenemos la expresión $g_7(x, y) = x'' + y'' = x + y$.
- La función g_{10} vale 1 cuando x vale 0 e y vale 0, o bien cuando x vale 1 e y vale 0. Es decir, la función vale 1 cuando la variable y vale 0, independientemente del valor de x. Por tanto, la función vale lo contrario de lo que valga y, y una posible expresión lógica para esta función es $g_{10}(x, y) = y'$.

5. Tenemos la expresión:

$$wx + xy' + yz + xz' + xy.$$

Para simplificarla aplicaremos la propiedad distributiva. Agrupamos el segundo término con el último, y obtenemos lo siguiente:

$$wx + x(y' + y) + yz + xz'.$$

Por el axioma de complementación sabemos que $y + y' = 1$. Si hacemos la sustitución en la expresión y aplicamos el axioma de los elementos neutros, obtenemos lo siguiente:

$$w{\cdot}x + x \cdot 1 + yz + xz' = wx + x + yz + xz'.$$

Por la ley de absorción, $wx + x = x$. Por tanto,

$$wx + x + yz + xz' = x + yz + xz'.$$

Aplicamos de nuevo la ley de absorción a $x + xz'$, y obtenemos lo siguiente:

$$x + yz + xz' = x + yz.$$

Por tanto, concluimos con lo que reproducimos a continuación:

$$wx + xy' + yz + xz' + xy = x + yz.$$

6.
- Para la primera condición debemos encontrar una expresión que valga 1 cuando $x = 1$ o $y' = 1$. La expresión que vale 1 cuando se cumple esta condición es $x + y'$.
- La segunda condición es $x = 0$ y $z = 1$. Esto es lo mismo que decir que $x' = 1$ o $z = 1$. La expresión que vale 1 en este caso es $x' \cdot z$.
- La tercera condición, $x = 1$, $y = 1$ y $z = 1$, sólo se cumple por la expresión $x \cdot y \cdot z$.
- Dado que la función debe valer 1 en cualquiera de los tres casos, es decir, cuando se cumple la condición 1, la condición 2 o la 3, tenemos que la expresión de la función es la siguiente:

$$F = x + y' + x'z + xyz.$$

7. La función debe valer 1 cuando se tenga que activar la alarma. Según nos dice el enunciado, la alarma se activa cuando el interruptor general está cerrado ($ig = 0$) y alguna de las dos cajas fuertes está abierta (es decir, $x_A = 1$ o $x_B = 1$).

Una expresión para la función S que vale 1 cuando $ig = 0$ y ($x_A = 1$ o $x_B = 1$) es la siguiente:

$$s = ig'(x_A + x_B).$$

8. Para saber qué asignaturas ha aprobado Juan y cuál ha suspendido, plantearemos las afirmaciones de sus tres amigos de manera algebraica.

El resultado de cada asignatura se puede representar con una variable booleana. Esta variable valdrá 1 si la asignatura está aprobada y 0 en el caso contrario.

Las variables que utilizaremos para cada una de las asignaturas serán m para matemáticas, f para física y q para química.

Entendemos que la "o" de estas frases es exclusiva. Es decir, la primera frase se podría sustituir por "o bien has aprobado matemáticas y has suspendido física, o bien has suspendido matemáticas y has aprobado física", y de manera similar con la segunda frase.

– Has aprobado matemáticas o física: $mf' + m'f$.
– Has suspendido química o matemáticas: $qm' + q'm$.
– Has aprobado dos asignaturas: $m'fq + mf'q + mfq'$.

Dado que se deben cumplir las tres afirmaciones a la vez, tenemos lo siguiente:

$$(mf' + m'f)\ (qm' + q'm)\ (m'fq + mf'q + mfq') = 1.$$

Simplificaremos la expresión aplicando los axiomas y teoremas del álgebra de Boole hasta que obtengamos la respuesta:

$$(mf' + m'f)(qm' + q'm)(m'fq + mf'q + mfq') =$$

(propiedad distributiva sobre los dos primeros paréntesis)

$$(mf'qm' + mf'q'm + m'fqm' + m'fq'm)(m'fq + mf'q + mfq') =$$

(complementación e idempotencia)

$$(0 + mf'q' + m'fq + 0)\ (m'fq + mf'q + mfq') =$$

(elementos neutros y distributiva sobre los dos paréntesis)

$$mf'q'm'fq + mf'q'mf'q + mf'q'mfq' + m'fqm'fq + m'fqmf'q + m'fqmfq' =$$

(complementación e idempotencia)

$$0 + 0 + 0 + m'fq + 0 + 0 = m'fq.$$

Hemos obtenido, por tanto, la siguiente expresión:

$$m'fq = 1.$$

Esta expresión sólo vale 1 cuando m vale 0 y f y q valen 1, es decir, que Juan ha suspendido matemáticas y ha aprobado física y química.

9. Las tablas de verdad de estas funciones son las siguientes:

x	y	g_1
0	0	0
0	1	0
1	0	0
1	1	1

x	y	g_3
0	0	0
0	1	0
1	0	1
1	1	1

x	y	g_6
0	0	0
0	1	1
1	0	1
1	1	0

x	y	g_7
0	0	0
0	1	1
1	0	1
1	1	1

x	y	g_{10}
0	0	1
0	1	0
1	0	1
1	1	0

10. Pondremos las variables a la izquierda de la tabla, en orden decreciente de subíndices. Las cuatro variables pueden tomar dieciséis combinaciones de valores diferentes; las escribiremos en orden lexicográfico (si las interpretáramos como números naturales, corresponderían a los números desde el 0 hasta el 15). A la derecha de la tabla se encontrará la columna correspondiente a f.

x_3	x_2	x_1	x_0	f
0	0	0	0	
0	0	0	1	
0	0	1	0	
0	0	1	1	
0	1	0	0	
0	1	0	1	
0	1	1	0	
0	1	1	1	
1	0	0	0	
1	0	0	1	
1	0	1	0	
1	0	1	1	
1	1	0	0	
1	1	0	1	
1	1	1	0	
1	1	1	1	

11. La tabla de verdad es la siguiente:

x_3	x_2	x_1	x_0	f
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

12. Para obtener estas tablas, escribiremos una columna para cada uno de los términos producto, y después obtendremos la columna correspondiente a f haciendo la OR de las columnas parciales.

a)

x	y	z	w	$x \cdot y' \cdot z \cdot w'$	$y \cdot z' \cdot w$	$x' \cdot z$	$x \cdot y \cdot w$	$x' \cdot y \cdot z' \cdot w'$	f
0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0
0	0	1	0	0	0	1	0	0	1
0	0	1	1	0	0	1	0	0	1
0	1	0	0	0	0	0	0	1	1
0	1	0	1	0	1	0	0	0	1
0	1	1	0	0	0	1	0	0	1
0	1	1	1	0	0	1	0	0	1
1	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0
1	0	1	0	1	0	0	0	0	1
1	0	1	1	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0
1	1	0	1	0	1	0	1	0	1
1	1	1	0	0	0	0	0	0	0
1	1	1	1	0	0	0	1	0	1

b)

x	y	z	w	$x \cdot y$	$x \cdot y + z$
0	0	0	0	0	0
0	0	0	1	0	0
0	0	1	0	0	1
0	0	1	1	0	1
0	1	0	0	0	0
0	1	0	1	0	0
0	1	1	0	0	1
0	1	1	1	0	1
1	0	0	0	0	0
1	0	0	1	0	0
1	0	1	0	0	1
1	0	1	1	0	1
1	1	0	0	1	1
1	1	0	1	1	1
1	1	1	0	1	1
1	1	1	1	1	1

13. Escribiremos las tablas de verdad de las dos funciones:

x	y	z	w	$x' \cdot y'$	$y \cdot w$	$x' \cdot w$	f
0	0	0	0	1	0	0	1
0	0	0	1	1	0	1	1
0	0	1	0	1	0	0	1
0	0	1	1	1	0	1	1
0	1	0	0	0	0	0	0
0	1	0	1	0	1	1	1
0	1	1	0	0	0	0	0
0	1	1	1	0	1	1	1
1	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0
1	0	1	0	0	0	0	0
1	0	1	1	0	0	0	0
1	1	0	0	0	0	0	0
1	1	0	1	0	1	0	1
1	1	1	0	0	0	0	0
1	1	1	1	0	1	0	1

x	y	z	w	$x' \cdot y' \cdot z' \cdot w'$	$x' \cdot z' \cdot w$	$y \cdot z' \cdot w$	$x \cdot y \cdot w$	$x' \cdot z$	g
0	0	0	0	1	0	0	0	0	1
0	0	0	1	0	1	0	0	0	1
0	0	1	0	0	0	0	0	1	1
0	0	1	1	0	0	0	0	1	1
0	1	0	0	0	0	0	0	0	0
0	1	0	1	0	1	1	0	0	1
0	1	1	0	0	0	0	0	1	1
0	1	1	1	0	0	0	0	1	1
1	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0
1	0	1	1	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0
1	1	0	1	0	0	1	1	0	1
1	1	1	0	0	0	0	0	0	0
1	1	1	1	0	0	0	1	0	1

Deducimos que las funciones no son equivalentes porque sus tablas de verdad son diferentes.

14. La tabla de verdad es la siguiente:

ig	x_A	x_B	s
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

15. Para resolver este problema, asignaremos una función a cada una de las afirmaciones de los amigos de Juan. La función valdrá 1 para las combinaciones que hacen que la afirmación sea cierta, y 0 en el caso contrario. Las funciones que representan las afirmaciones de los amigos corresponden a las funciones F_1, F_2 y F_3, respectivamente.
Llamaremos m, f y q las variables que representan las asignaturas. Estas variables valdrán 1 cuando la asignatura esté aprobada, y 0 si está suspendida.

La solución representada por la función F se obtiene haciendo un AND de las funciones F_1, F_2 y F_3, ya que las tres frases son ciertas simultáneamente. La combinación por la que F vale 1 corresponderá a la solución del problema.

m	f	q	F_1	F_2	F_3	F
0	0	0	0	0	0	0
0	0	1	0	1	0	0
0	1	0	1	0	0	0
0	1	1	1	1	1	1
1	0	0	1	1	0	0
1	0	1	1	0	1	0
1	1	0	0	1	1	0
1	1	1	0	0	0	0

La función F indica que Juan ha aprobado física y química y ha suspendido matemáticas.

16.
$f_0 = x_2'x_1x_0' + x_2x_1x_0$,
$f_1 = x_2x_1'x_0' + x_2x_1'x_0 + x_2x_1x_0' + x_2x_1x_0$,
$f_2 = x_2'x_1'x_0 + x_2'x_1x_0 + x_2x_1'x_0 + x_2x_1x_0$,
$f_3 = x_2'x_1'x_0 + x_2'x_1x_0' + x_2x_1'x_0' + x_2x_1x_0$.

17. Escribiremos la tabla de verdad y a partir de ésta obtendremos la expresión en suma de mintérminos.

a) La tabla de verdad de la función es la siguiente:

x	y	z	f
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

La expresión de f en suma de mintérminos es ésta:

$$f = x'y'z + x'yz + xy'z + xyz' + xyz.$$

b) La tabla de verdad de la función es la siguiente:

x	y	z	w	f
0	0	0	0	1
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	0
1	1	1	1	0

La expresión de f en suma de mintérminos es la que reproducimos a continuación:

$$f = x'y'z'w' + x'y'z'w + x'y'zw + x'yz'w + xy'z'w + xy'zw + xyz'w.$$

18. Las combinaciones de entrada 101, 110 y 111 no se pueden producir y, por tanto, no nos importará el valor que tomen las salidas en estos casos.
La tabla de verdad de las funciones de este sistema es la siguiente:

x_2	x_1	x_0	y_4	y_3	y_2	y_1	y_0
0	0	0	1	0	1	1	1
0	0	1	1	1	0	1	1
0	1	0	0	0	1	0	0
0	1	1	0	1	1	1	1
1	0	0	0	1	1	0	1
1	0	1	x	x	x	x	x
1	1	0	x	x	x	x	x
1	1	1	x	x	x	x	x

19.
a) A partir del enunciado, obtenemos que las expresiones de las funciones R y E son las siguientes:

$$R = h_0' + h_1't$$
$$E = t'h_0$$

Ahora bien, las combinaciones con $h_1 = 1$ y $h_0 = 0$ no se pueden producir (debemos suponer que los sensores funcionan bien). Por tanto, la tabla de verdad del sistema es la siguiente:

t	h_1	h_0	R	E
0	0	0	1	0
0	0	1	0	1
0	1	0	x	x
0	1	1	0	1
1	0	0	1	0
1	0	1	1	0
1	1	0	x	x
1	1	1	0	0

20. Para realizar el análisis, es necesario obtener la expresión de la función a partir del circuito. A partir de la expresión obtendremos la tabla de verdad. Para rellenarla cómodamente, encontraremos previamente una expresión simplificada de la función.

a) La expresión que obtenemos directamente a partir del circuito es la siguiente:

$$F = (y' + w')\ (y + w) + (x' + z)\ (x + z').$$

Si simplificamos esta expresión, obtenemos lo siguiente:

$$F = y'w + yw' + x'z' + xz.$$

La tabla de verdad es la siguiente:

x	y	z	w	f
0	0	0	0	1
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0

x	y	z	w	f
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	0
1	1	1	0	1
1	1	1	1	1

b) La expresión que obtenemos directamente a partir del circuito es la siguiente:

$$F = ((x' + w)\ (y + w'))'z + (y + w')z'.$$

Si simplificamos esta expresión, obtenemos lo que reproducimos a continuación:

$$F = ((x' + w)' + (y + w')')z + (y + w')z' =$$
$$= (xw' + y'w)z + (y + w')z' =$$
$$= xzw' + y'zw + yz' + z'w'.$$

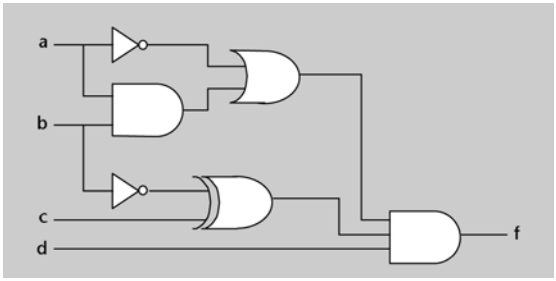
La tabla de verdad es la siguiente:

x	y	z	w	f
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	1
0	1	0	1	1
0	1	1	0	0
0	1	1	1	0
1	0	0	0	1
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

21. Para hacer la síntesis, simplemente dibujamos una línea para cada variable de la función y sustituimos las operaciones lógicas de la expresión por la puerta lógica correspondiente. La función es ésta:

$$f(a, b, c, d) = d{\cdot}(a' + a \cdot b){\cdot}(b' \oplus c).$$

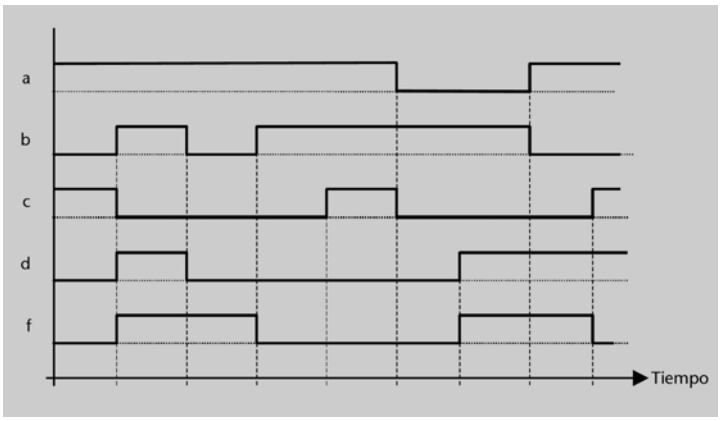
La siguiente figura muestra el circuito.



22. En la figura se muestra el cronograma. Dado que se trata de un circuito combinacional en el que no se consideran los retardos (tal como haremos habitualmente en esta asignatura), las variaciones en la salida se producen en el mismo momento en que se produce alguna

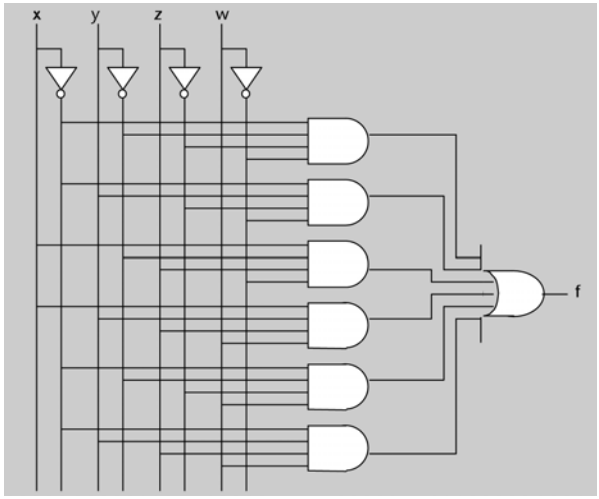
variación de las entradas (líneas punteadas verticales). Sin embargo, no todas las variaciones de las entradas producen una variación de la salida, como se puede ver en el cronograma.

La función que implementa el circuito es $f = b'c' + c'd$.

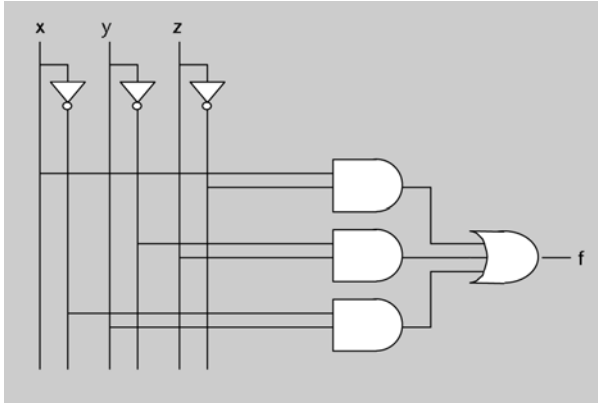


23. En las figuras se muestra la síntesis a dos niveles de cada circuito. Cada término *producto* está implementado por una puerta AND de tantas entradas como variables tiene el término *producto*. La suma lógica está implementada por una puerta OR de tantas entradas como términos *producto* tiene la expresión.

a)



b)



24.

a) El rango de valores que puede tomar un número natural de 2 bits es 0..3. Por tanto, el rango de la multiplicación de dos números será 0..9. Para representar el 9 necesitamos 4 bits. Por tanto, la salida tendrá 4 bits.

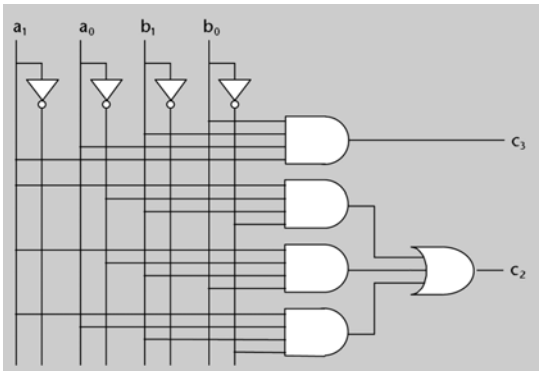
b) Denominaremos A y B, respectivamente, a los números de entrada, y C a su multiplicación. Para obtener la tabla de verdad pasamos A y B a decimal, calculamos $C = A \cdot B$ en decimal y, finalmente, codificamos C en binario, con los bits c_3 ... c_0. Por ejemplo, tomamos la fila $[a_1\ a_0\ b_1\ b_0] = [1\ 0\ 1\ 1]$. Tenemos que $A = 2$, $B = 3$. Por tanto, $C = 2 \cdot 3 = 6$, que se codifica [0 1 1 0] en binario. La tabla de verdad completa es la siguiente:

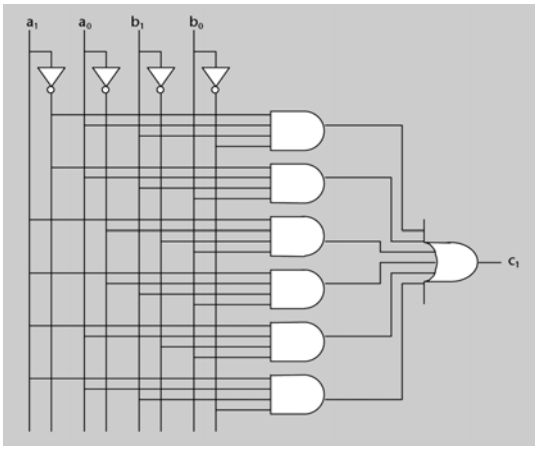
a_1	a_0	b_1	b_0	c_3	c_2	c_1	c_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0
0	0	1	0	0	0	0	0
0	0	1	1	0	0	0	0
0	1	0	0	0	0	0	0
0	1	0	1	0	0	0	1
0	1	1	0	0	0	1	0
0	1	1	1	0	0	1	1
1	0	0	0	0	0	0	0
1	0	0	1	0	0	1	0
1	0	1	0	0	1	0	0
1	0	1	1	0	1	1	0
1	1	0	0	0	0	0	0
1	1	0	1	0	0	1	1
1	1	1	0	0	1	1	0
1	1	1	1	1	0	0	1

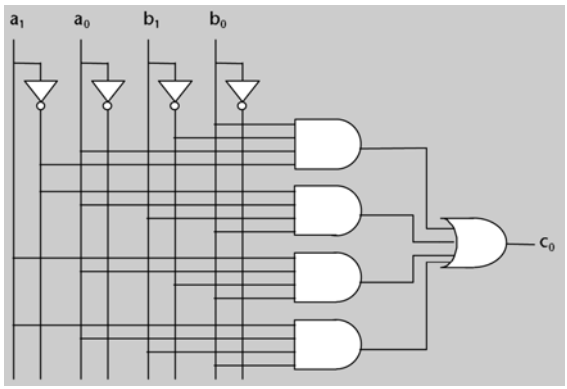
c) Las expresiones en suma de productos de las cuatro funciones de salida son las siguientes:

$c_3 = a_1a_0b_1b_0$,
$c_2 = a_1a_0'b_1b_0' + a_1a_0'b_1b_0 + a_1a_0b_1b_0'$,
$c_1 = a_1'a_0b_1b_0' + a_1'a_0b_1b_0 + a_1a_0'b_1'b_0 + a_1a_0'b_1b_0 + a_1a_0b_1'b_0 + a_1a_0b_1b_0'$,
$c_0 = a_1'a_0b_1'b_0 + a_1'a_0b_1b_0 + a_1a_0b_1'b_0 + a_1a_0b_1b_0$.

El circuito correspondiente a la función C_3 se puede implementar con una puerta AND. A continuación, se muestran los circuitos a dos niveles correspondientes a las tres funciones de salida restantes.



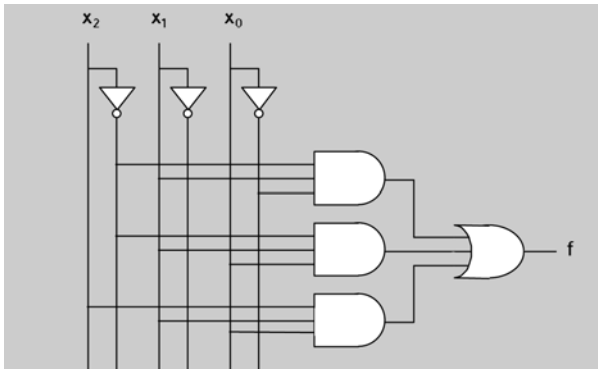




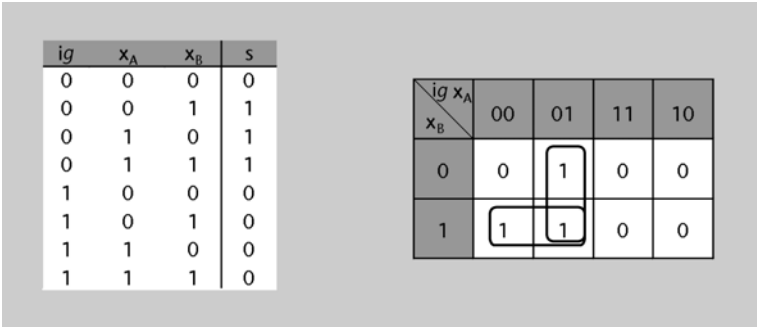
25. La expresión en suma de mintérminos de la función es la siguiente:

$$f = x_2'x_1x_0' + x_2'x_1x_0 + x_2x_1x_0$$

A continuación se muestra la síntesis a dos niveles de la función.



26. A continuación se muestra la tabla de verdad de la función y su minimización utilizando el método de Karnaugh.



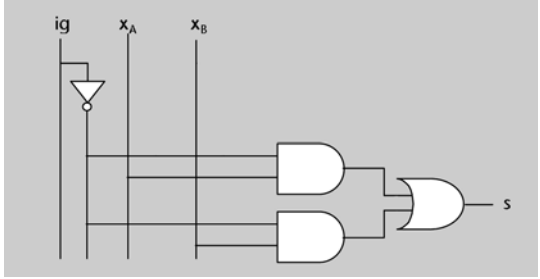
ig	x_A	x_B	s
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

ig \ x_A / x_B	00	01	11	10
0	0	1	0	0
1	1	1	0	0

Del rectángulo vertical se obtiene el término producto $ig' \cdot x_A$, y del rectángulo horizontal el término $ig' \cdot x_B$. Por tanto, la expresión de la función es ésta:

$$s = ig'x_A + ig'x_B.$$

A continuación se muestra la síntesis mínima a dos niveles de la función.



Fijaos en que si sacamos ig' factor común en la expresión mínima de s, obtenemos $s = ig'(x_A + x_B)$, que es la expresión que hemos deducido en la actividad 7 (pero no es una suma de productos).

27. A continuación se muestran la tabla de verdad de la función y su minimización por Karnaugh.

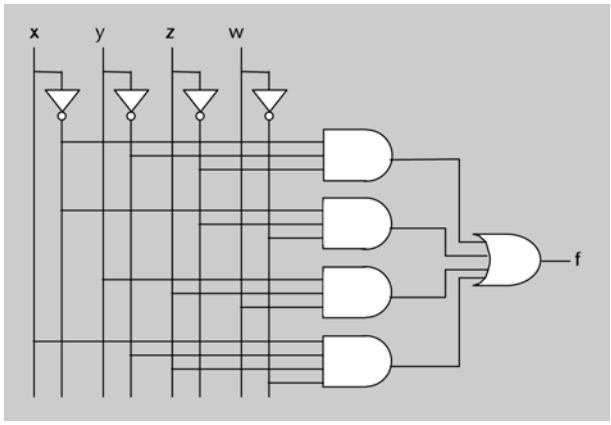
x	y	z	w	f
0	0	0	0	1
0	0	0	1	1
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

z w \ x y	00	01	11	10
00	1	1	0	0
01	1	0	0	0
11	0	1	1	0
10	0	0	0	1

Del mapa de Karnaugh se deduce esta expresión:

$$f = x'y'z' + x'z'w' + yzw + xy'zw'.$$

El circuito mínimo a dos niveles se presenta en la siguiente figura. Se puede comprobar que es más sencillo que el que se ha obtenido en la actividad 23.



28. En la siguiente figura se muestra la tabla de verdad de las funciones y su minimización por Karnaugh.

t	h_1	h_0	R	E
0	0	0	1	0
0	0	1	0	1
0	1	0	x	x
0	1	1	0	1
1	0	0	1	0
1	0	1	1	0
1	1	0	x	x
1	1	1	0	0

R

h_0 \ t h_1	00	01	11	10
0	1	x	x	1
1	0	0	0	1

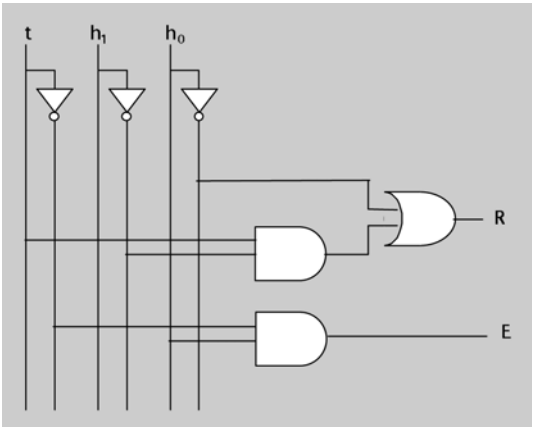
E

h_0 \ t h_1	00	01	11	10
0	0	x	x	0
1	1	1	0	0

Las funciones obtenidas son las siguientes:

$$R = h_0' + th_1',$$
$$E = t'h_0.$$

En la siguiente figura se muestra el circuito minimizado.

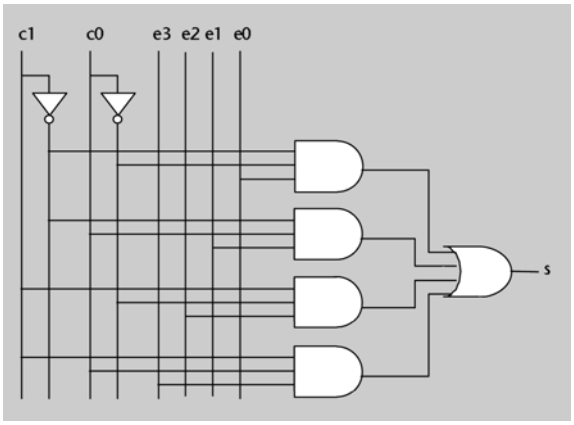


29. El valor de la salida *s* será el valor de e_3, e_2, e_1 o e_0 de acuerdo con lo que valgan las entradas de control c_1 y c_0.

Por ejemplo, si [*c*1 *c*0] [1 0], la salida *s* valdrá 1 si $e_2 = 1$, y valdrá 0 si $e_2 = 0$ (es decir, valdrá e_2). En este caso podríamos escribir que la expresión de la salida es c1c0'e2. En cada momento, las entradas c1 y c2 valen alguna de estas cuatro combinaciones: [0,0], [0,1], [1,0] o bien [1 1]. Por tanto, sólo uno de los productos *c*1'*c*0', *c*1*c*0', *c*1'*c*0 y *c*1*c*0 puede valer 1 en cada momento. Podemos concluir que una expresión válida para *s* es la siguiente:

$$s = c1'c0'e0 + c1'c0e1 + c1c0'e2 + c1c0e3.$$

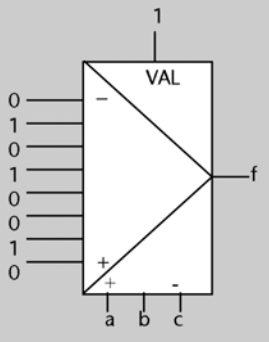
La implementación con puertas de esta expresión es la siguiente:



30. Para implementar esta función construiremos su tabla de verdad. A partir de ésta sabremos qué debemos conectar a las entradas del multiplexor para que en la salida nos aparezca un 0 o un 1 según indique la función.

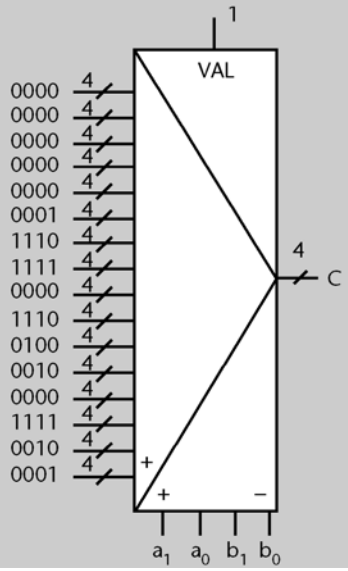
En definitiva, se trata de traspasar la columna *f* de la tabla de verdad a las entradas del multiplexor.

a	b	c	f
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0



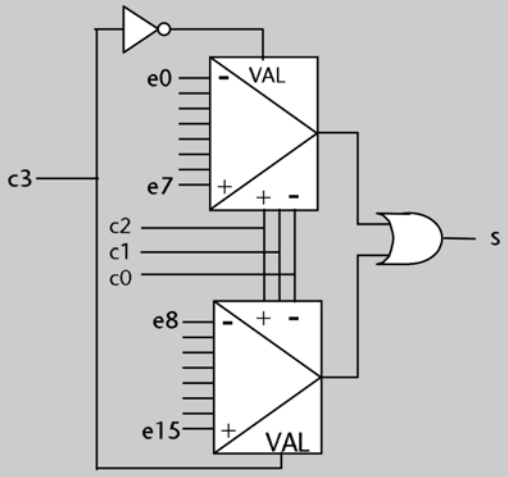
31. En primer lugar, construiremos la tabla de verdad del circuito.
El rango de un número entero representado en complemento a 2 con dos bits es [–2..1]. Por tanto, el rango de producto de dos de estos números es [–2..4.]. Para representar números dentro de este rango es suficiente con cuatro bits, ya que con éstos podemos representar números en el rango [–8..7].
Por tanto, la tabla de verdad tiene cuatro entradas y cuatro salidas. Las entradas son los dos bits de cada uno de los números *A* y *B*, y las salidas son los cuatro bits del resultado *C*.
Para implementar el circuito utilizaremos un multiplexor de buses de cuatro bits de dieciséis entradas de datos. La figura muestra esta implementación.

a_1	a_0	b_1	b_0	c_3	c_2	c_1	c_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0
0	0	1	0	0	0	0	0
0	0	1	1	0	0	0	0
0	1	0	0	0	0	0	0
0	1	0	1	0	0	0	1
0	1	1	0	1	1	1	0
0	1	1	1	1	1	1	1
1	0	0	0	0	0	0	0
1	0	0	1	1	1	1	0
1	0	1	0	0	1	0	0
1	0	1	1	0	0	1	0
1	1	0	0	0	0	0	0
1	1	0	1	1	1	1	1
1	1	1	0	0	0	1	0
1	1	1	1	0	0	0	1

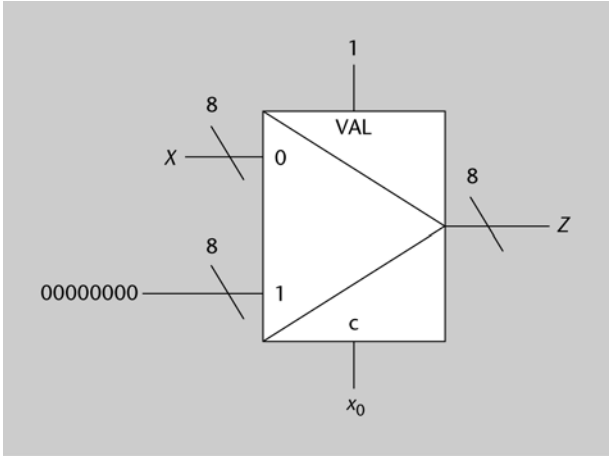


32. Escribimos primero la tabla de verdad correspondiente a un multiplexor 16-1. Éste tiene cuatro entradas de control (*c*3..*c*0) y dieciséis entradas de datos (*e*15..*e*0).
La línea discontinua en la tabla muestra que se puede descomponer el funcionamiento del circuito como si se tratara de dos multiplexores 8-1, los dos controlados por las señales *c*2, *c*1 y *c*0. La señal *c*3 determina cuál de los dos multiplexores funciona en cada momento.

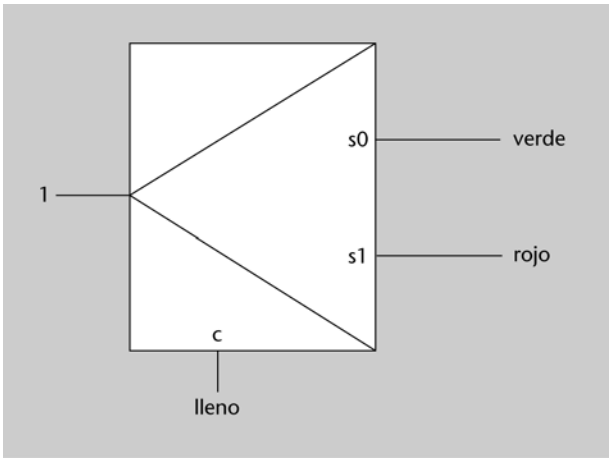
c3	c2	c1	c0	f
0	0	0	0	e0
0	0	0	1	e1
0	0	1	0	e2
0	0	1	1	e3
0	1	0	0	e4
0	1	0	1	e5
0	1	1	0	e6
0	1	1	1	e7
1	0	0	0	e8
1	0	0	1	e9
1	0	1	0	e10
1	0	1	1	e11
1	1	0	0	e12
1	1	0	1	e13
1	1	1	0	e14
1	1	1	1	e15



33. Puesto que *Z* debe valer uno de dos posibles valores, la podemos implementar mediante un multiplexor 2-1. El bit de menos peso de *X* nos indica si es par (0) o impar (1). Podemos, por lo tanto, conectar este bit a la entrada de control del multiplexor, y esto nos lleva a conectar *X* a la entrada de datos 0 y 00000000 a la entrada de datos 1.



34. Basta con poner un 1 a la entrada del demultiplexor, y hacer que la señal *llena* controle hacia cuál de las dos luces llega este 1.



35. Deduciremos las expresiones lógicas de cada una de las salidas y construiremos el circuito a partir de éstas. Para resolver la actividad, es conveniente tener delante la representación gráfica del codificador y su tabla de verdad (figura 21 de los apuntes).

Puesto que la salida del codificador debe codificar en binario la entrada de más peso que valga 1, podemos hacer el siguiente razonamiento:

1) La salida $s1$ se pondrá a 1 cuando estén en 1 las entradas $e3$ o $e2$ (en estos casos se debe codificar un 11 o un 10, respectivamente).

2) La salida $s0$ se pondrá a 1 cuando esté a 1 la entrada $e3$ (en este caso se tiene que codificar un 11, independientemente de los valores de las otras entradas) o bien cuando lo esté la entrada $e1$ y no lo esté ni la $e2$ ni la $e3$ (en este caso, $[e3\ e2\ e1] = [0\ 0\ 1]$, se tiene que codificar un 01).

3) Por otro lado, la entrada de validación debe estar activa para que el codificador funcione como tal.

Por tanto, las expresiones algebraicas de las salidas son las siguientes:

– $s1$ valdrá 1 cuando VAL = 1 y $e3$ o $e2$ sean 1 (apartados 1 y 3):

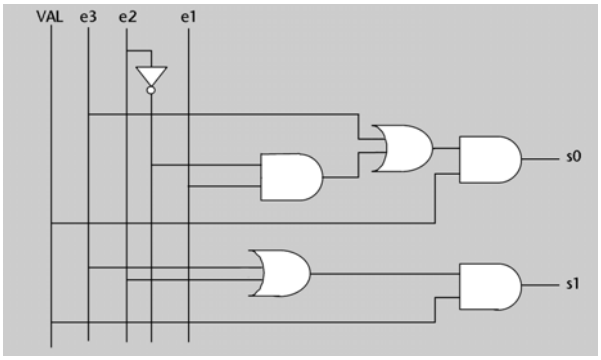
$$s1 = \text{VAL}\ (e3 + e2).$$

– $s0$ valdrá 1 cuando VAL = 1 y $e3$ sea 1, o cuando $[e3\ e2\ e1] = [0\ 0\ 1]$ (apartados 2 y 3):

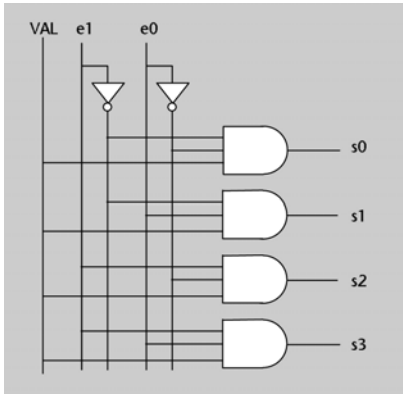
$$s0 = \text{VAL}\ (e3 + e3'e2'e1) = \text{VAL}\ (e3 + e2'e1).$$

A continuación se muestra una posible implementación de este codificador. Como se puede ver en la figura, el valor de la entrada $e0$ no tiene influencia en las salidas. Éste es el motivo

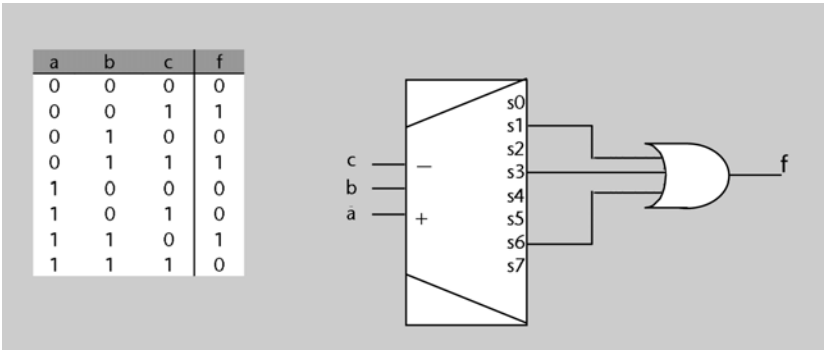
que hace que un codificador tenga un 0 en su salida tanto cuando tiene que codificar un 0 como cuando ninguna de sus entradas tiene un 1.



36. Cuando la entrada de validación está activa, cada salida de un descodificador se pone en 1 sólo cuando se produce una combinación determinada de las entradas. Por tanto, la implementación de cada una de las salidas será un circuito que detecte si esta combinación está presente en la entrada, tal como se muestra en la figura.



37. Primero haremos la tabla de verdad y, a partir de ésta, utilizaremos un descodificador para sintetizar el circuito. La puerta OR hace la suma lógica de los casos en que la función vale 1.



a	b	c	f
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

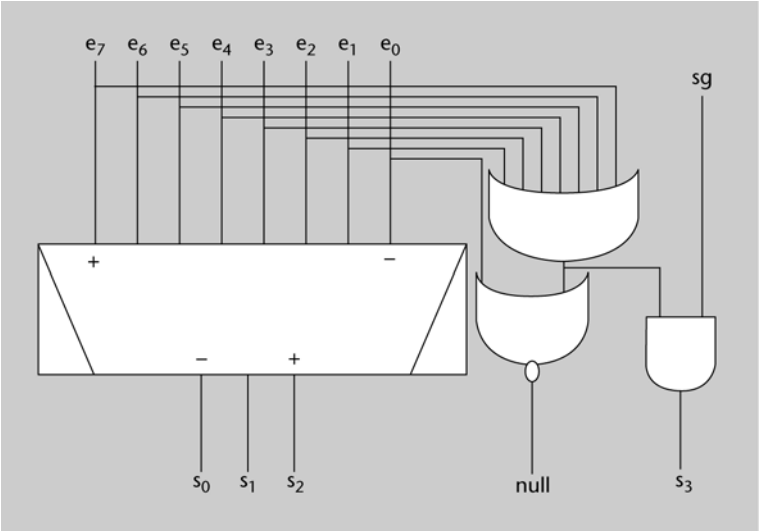
38. Podemos generar la magnitud del número usando un codificador 8-3, a cuyas entradas conectaremos *e*7, *e*6, ... *e*0. Las salidas de este codificador, por lo tanto, serán directamente las salidas *s*2, *s*1 y *s*0. Observemos que esto también hará que [*s*2 *s*1 *s*0] = [0 0 0] cuando no haya ninguna entrada *ei* a 1 (tal y como nos pide el enunciado).

A primera vista, podríamos pensar que el bit *s*3 (signo del número que hay que representar) lo podemos obtener directamente de la entrada *sg*, pero hay que tener en cuenta estas dos excepciones:

- cuando e0 = 1, entonces *s*3 debe valer 0 independientemente del valor de *sg*.
- cuando no haya ninguna entrada *ei* a 1, *s*3 también debe valer 0 independientemente del valor de *sg*.

Por lo tanto, *s*3 tiene que valer 1 cuando alguna de las entradas *e*7...e1 sea 1 y *sg* = 1.

La salida *null* debe valer 1 sólo cuando no haya ninguna entrada *ei* a 1.



39. Para deducir qué hace el circuito, construiremos su tabla de verdad. Ésta muestra también el valor de la entrada y la salida cuando las interpretamos como números representados en complemento a 2 (columnas *X* e *Y*).

X	x_2	x_1	x_0	y_2	y_1	y_0	Y
0	0	0	0	0	0	0	0
1	0	0	1	1	1	1	−1
2	0	1	0	1	1	0	−2
3	0	1	1	1	0	1	−3
−4	1	0	0	1	0	0	−4
−3	1	0	1	0	1	1	3
−2	1	1	0	0	1	0	2
−1	1	1	1	0	0	1	1

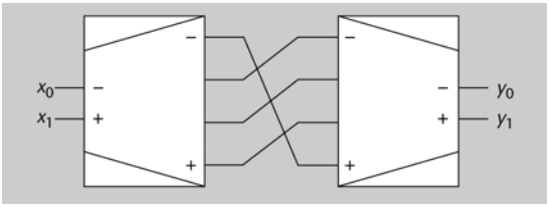
Como se puede apreciar en la tabla, lo que hace el circuito es cambiar el signo del número de la entrada. En el caso particular $X = -4$ esto no es posible, porque el 4 no se puede representar en complemento a 2 con sólo tres bits (el resultado es erróneo en este caso: se produce desbordamiento).

40. A partir de la codificación en binario de un número, debemos generar la codificación en binario de otro número. Lo podemos conseguir conectando las salidas de un descodificador con las entradas de un codificador, de manera que la salida *i* del descodificador se conecte con la entrada *j* del codificador, siendo *j* el valor que debe tener la salida cuando la entrada vale *i* (es decir, aplicando la misma técnica que en la actividad anterior).

La correspondencia entre valores de la entrada *X* y valores de la salida *Y* viene expresada por esta tabla:

X	Y
0	3
1	0
2	1
3	2

Por lo tanto, las conexiones se deben hacer tal y como se muestra en la siguiente figura.



41.

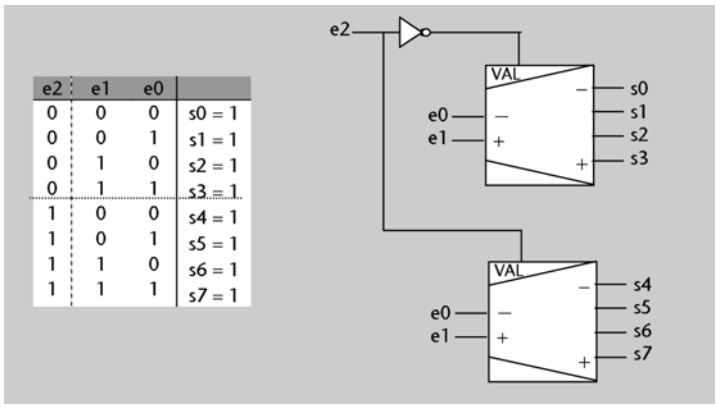
a) E_0 es 1 cuando las dos entradas del descodificador son 0, es decir, cuando $x_1 = 0$ y $x_0x_3 = 0$. Por tanto, la expresión algebraica para E_0 es $E_0 = x_1'(x_0x_3)'$. Si seguimos el mismo razonamiento para el resto de las salidas, obtenemos lo siguiente:

$E_0 = x_1'(x_0x_3)'$,
$E_1 = x_1'(x_0x_3) = x_1'x_0x_3$,
$E_2 = x_1(x_0x_3)'$,
$E_3 = x_1(x_0x_3) = x_1x_0x_3$.

b) La tabla de verdad se encuentra en la siguiente figura. Para obtener la columna correspondiente a F hemos dado un paso intermedio: hemos puesto en una columna el valor de F en términos de las entradas de datos del multiplexor (E_i), y en la columna final hemos sustituido las variables E_i por el valor que toman para cada combinación de x_i, de acuerdo con las expresiones obtenidas en el apartado **a**.

x_3	x_2	x_1	x_0	F	F
0	0	0	0	E_0	1
0	0	0	1	E_0	1
0	0	1	0	E_0	0
0	0	1	1	E_0	0
0	1	0	0	E_1	0
0	1	0	1	E_1	0
0	1	1	0	E_1	0
0	1	1	1	E_1	0
1	0	0	0	E_2	0
1	0	0	1	E_2	0
1	0	1	0	E_2	1
1	0	1	1	E_2	0
1	1	0	0	E_3	0
1	1	0	1	E_3	0
1	1	1	0	E_3	0
1	1	1	1	E_3	1

42. Esta actividad es muy similar a la actividad 32. En la siguiente figura se muestra la tabla de verdad del descodificador 3-8, que tiene tres entradas ($e2..e0$) y ocho salidas ($s0..s7$). Para simplificar la tabla, se ha puesto una única columna de salidas en la que hemos indicado cuál de éstas vale 1; todas las demás valen 0. Como se puede observar en la tabla de verdad, la entrada $e2$ determina cuál de los dos descodificadores 2-4 funciona en cada momento (y, por tanto, controla su entrada de validación). En cada descodificador 2-4, las entradas $e1$ y $e0$ determinan qué salida tiene que valer 1.



e2	e1	e0	
0	0	0	s0 = 1
0	0	1	s1 = 1
0	1	0	s2 = 1
0	1	1	s3 = 1
1	0	0	s4 = 1
1	0	1	s5 = 1
1	1	0	s6 = 1
1	1	1	s7 = 1

43. Para resolver este ejercicio construiremos su tabla de verdad y después interpretaremos los números como naturales.
Llamamos X al número que sale del codificador: $X = [x_1\ x_0]$..

A	a_1	a_0	z	x_1	x_0	X	s_1	s_0	S
0	0	0	0	0	1	1	0	0	0
1	0	1	0	1	0	2	0	1	1
2	1	0	0	1	1	3	1	0	2
3	1	1	0	0	0	0	1	1	3
0	0	0	1	0	1	1	0	1	1
1	0	1	1	1	0	2	1	0	2
2	1	0	1	1	1	3	1	1	3
3	1	1	1	0	0	0	0	0	0

Tal como podemos ver en la tabla,

$$X = (A + 1) \bmod 4.$$

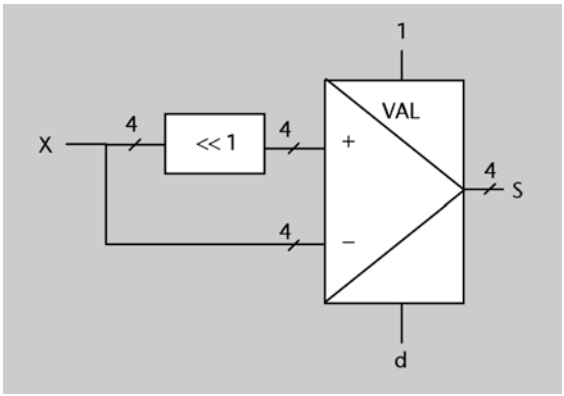
La variable z selecciona entre las entradas A y X, y, por tanto, la salida se puede expresar con la siguiente tabla:

z	S
0	A
1	(A+1) mod 4

Esto se puede resumir en la siguiente expresión:

$$S = (A + z) \bmod 4.$$

44.
a) En la figura se muestra la implementación del circuito. La señal d determina si la salida S vale lo mismo que la entrada X ($d = 0$) o X desplazada un bit a la izquierda ($d = 1$).



b) Este desplazamiento de un bit a la izquierda es equivalente a multiplicar la entrada por 2. Por tanto, $S = 2 \cdot X$.

c) Se produce desbordamiento cuando el resultado no se puede representar con 4 bits.

– Si interpretamos los números X y S como números naturales codificados en binario, el rango de X y S es [0..15]. Por tanto, cuando X sea mayor que 7, S no podrá representar el resultado y se producirá desbordamiento.

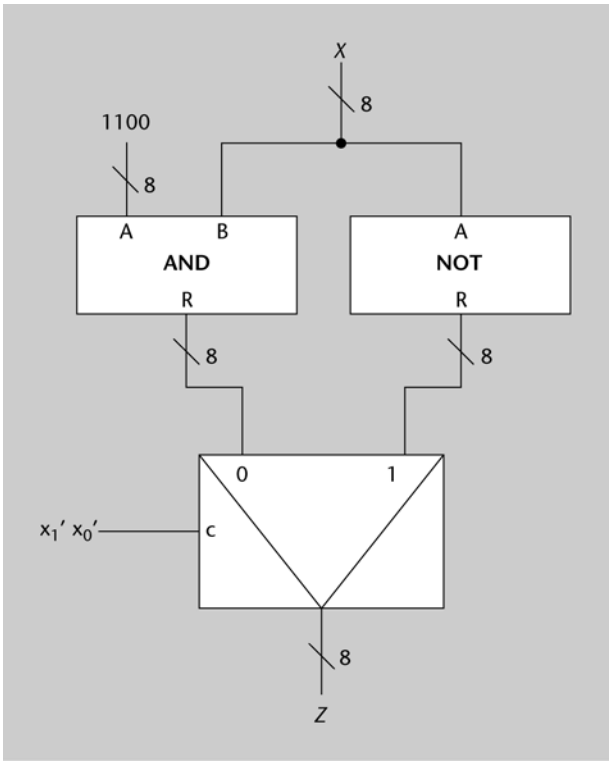
Desbordamiento si $X > 7$.

– Si interpretamos los números como enteros codificados en complemento a dos, el rango de X y S es [–8..7]. Por tanto, se producirá desbordamiento cuando X sea menor que –4 o mayor que 3.

Desbordamiento si $X < -4$ o $X > 3$.

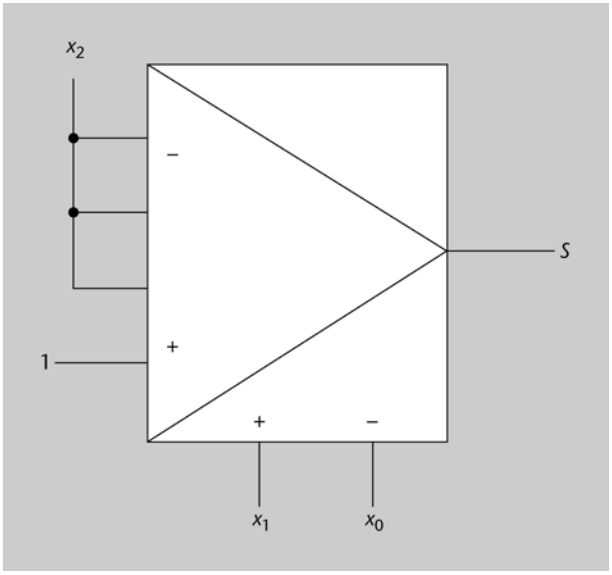
45. La salida *Z* debe tomar uno de dos valores; esto lo podemos conseguir mediante un multiplexor 2-1. Cuál de los dos valores tome depende de si *X* es múltiplo de 4 o no. Un número representado en binario es múltiplo de 4 si los dos bits de menor peso valen 0; por lo tanto, a la entrada de control del multiplexor debemos conectar $x_1'x_0'$.

Para obtener los valores que debemos conectar a cada una de las entradas de datos del multiplexor, podemos usar los bloques que se describen en el apartado 3.4, tal y como se muestra en la figura.



46.

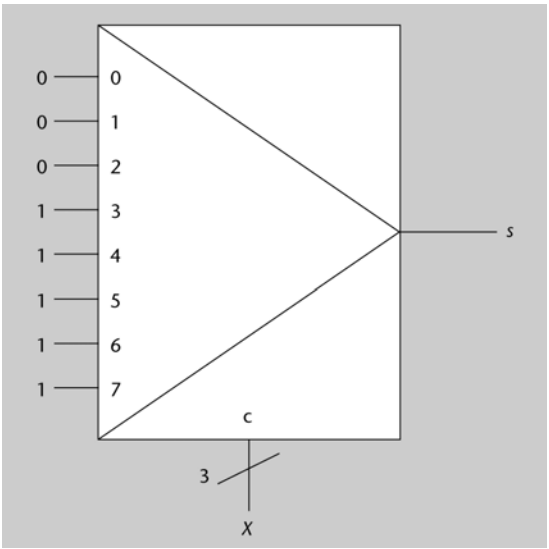
a) Vemos que para cualquier combinación de valores de x_1 y x_0 la señal de salida *s* vale x_2, excepto en el caso $x_1 = x_0 = 1$ en el que *s* vale 1 ($x_2 + x_2'$). Por lo tanto, podemos implementar *M*0 por ejemplo con un multiplexor 4-1 y sin ninguna puerta adicional. De esta manera:



Otra opción es implementar la función a partir de su tabla de verdad, tal y como se hace en la figura 18 del apartado 3.1:

$x_2x_1x_0$	s
000	0
001	0
010	0
011	1
100	1
101	1
110	1
111	1

En este caso utilizaremos un multiplexor 8-1, a las que conectaremos las variables de la función, y copiaremos los 1 y 0 en las entradas de datos del multiplexor.



b) El valor U es una XOR de X con la negación de sí misma. Dado que a XOR $a' = 1$, obtenemos que U valdrá siempre 111, y por lo tanto V valdrá siempre 000. Deducimos entonces que la función del circuito $M1$ es generar la combinación 000.
c) Analizamos los valores que toma Y en función de los valores que toma X.

Como hemos visto en el apartado *a)* analizando la salida del bloque $M0$, s vale 0 si $X = 0$, 1 o 2 y vale 1 siempre que $X > 2$. Por lo tanto, si $X \leq 2$ la salida Y del circuito vale 0 (que es lo que vale V en todo momento) y si $X > 2$ vale W.
Analicemos ahora el valor de W:

a) Si $X \leq 2$, no es necesario que lo analicemos porque sabemos que en este caso, Y vale 0.
b) Si X vale 3 o 4, W vale 3 y 4 respectivamente.
c) Si $X \geq 5$, W vale 5.

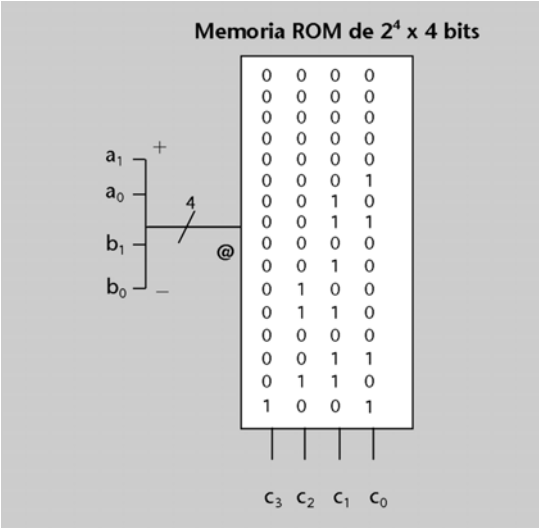
Juntándolo todo, obtenemos esta tabla de verdad:

$x_2x_1x_0$	$y_2y_1y_0$
000	000
001	000
010	000
011	011
100	100
101	101
110	101
111	101

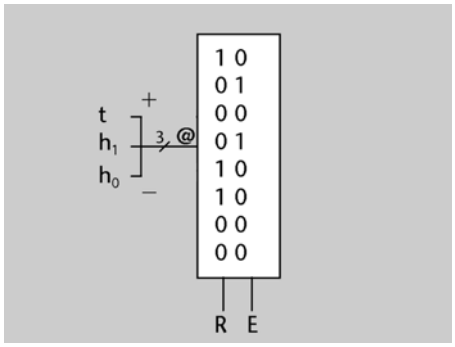
47. El circuito multiplicador de dos números naturales de dos bits tiene cuatro entradas (a_1 a_0 b_1 b_0) y cuatro salidas (c_3 c_2 c_1 c_0), tal como se explica en la actividad 24.

Cuatro variables pueden tomar $2^4 = 16$ combinaciones diferentes; por tanto, el tamaño de la ROM es de dieciséis palabras de 4 bits (1 para cada función de salida).

En la siguiente figura se muestra su contenido, que coincide con la parte derecha de la tabla de verdad de la actividad 24. La variable a_1 está conectada al bit de mayor peso del bus de direcciones y la variable b_0, al bit de menor peso (es muy importante indicar en el bus de direcciones a qué señal de entrada se conecta el bit de mayor peso para que el resultado sea correcto. Si los bits de entrada estuvieran en otro orden, la tabla de verdad sería diferente y, por tanto, también el contenido de la ROM).



48. Tal como se explica en la actividad 19, este circuito tiene tres señales de entrada y dos de salida y, por tanto, el tamaño de la memoria ROM es de $2^3 \times 2$ bits (ocho palabras de 2 bits). En el contenido de la memoria se han sustituido las x de la tabla de verdad por ceros, ya que el valor de cualquier bit en una memoria ROM debe estar definido. No obstante, las *x* se podrían haber sustituido por cualquier valor binario.



49. Conectaremos la entrada *X* a la entrada de direcciones de la memoria ROM, para que la salida nos dé la representación en complemento a 2 deseada. Por lo tanto, la entrada de direcciones será de 8 bits y la memoria tendrá 256 palabras de 8 bits cada una.

A la palabra de la dirección *i* le pondremos la representación en complemento a 2 del número que en signo y magnitud se codifica con la secuencia de bits correspondiente a la representación binaria del número *i*. De este modo, las direcciones 00000000..01111111 se accederán cuando el número *X* sea positivo, que en complemento a 2 se codifican igual que en signo y magnitud, de manera que en la palabra de cada una de estas direcciones habrá la misma combinación de bits que la de la dirección correspondiente. Las direcciones 10000000..11111111 se accederán cuando el número *X* sea negativo, de modo que en la palabra de cada una de estas direcciones habrá la representación en complemento a 2 del número $-A$, siendo A el número que codifican los 7 bits más bajos de la dirección.

La dirección 50 es, expresada en binario, la dirección 00110010. Si la leemos como un número expresado en signo y magnitud, vemos que es positivo y, por lo tanto, el contenido de esta dirección será 00110010.

La dirección 150 es, expresada en binario, la dirección 10010110. Si la leemos como un número expresado en signo y magnitud, vemos que es negativo. La magnitud del número es 0010110. Antes de cambiarle el signo, lo tenemos que expresar en 8 bits, ya que estamos buscando una representación en complemento a 2 y 8 bits: 00010110. Ahora cambiamos 1 por 0 y viceversa, sumamos 1 al resultado y obtenemos 11101010. Este será el contenido de la dirección 150.

La dirección 200 es, expresada en binario, la dirección 11001000. Si la leemos como un número expresado en signo y magnitud, vemos que es negativo. La magnitud del número es 1001000. De nuevo, antes de cambiarle el signo lo tenemos que expresar en 8 bits: 01001000. Ahora cambiamos 1 por 0 y viceversa y sumamos 1 al resultado, y obtenemos 10111000. Este será el contenido de la dirección 200.

50.
a) La tabla de verdad del circuito es la siguiente:

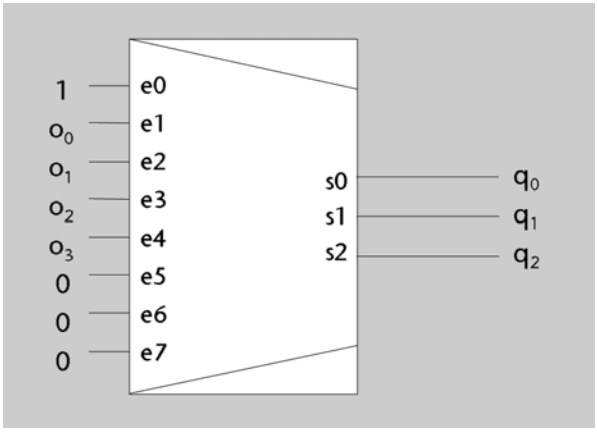
e_3	e_2	e_1	e_0	o_3	o_2	o_1	o_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	0	1
0	0	1	1	0	0	1	1
0	1	0	0	0	0	0	1
0	1	0	1	0	0	1	1
0	1	1	0	0	0	1	1
0	1	1	1	0	1	1	1
1	0	0	0	0	0	0	1
1	0	0	1	0	0	1	1
1	0	1	0	0	0	1	1
1	0	1	1	0	1	1	1
1	1	0	0	0	0	1	1
1	1	0	1	0	1	1	1
1	1	1	0	0	1	1	1
1	1	1	1	1	1	1	1

b) Las salidas del bloque ORDENAR tienen todos los unos a la derecha, tal como se muestra en la tabla siguiente (recordemos que CUANTOS = $[q_2\ q_1\ q_0]$:

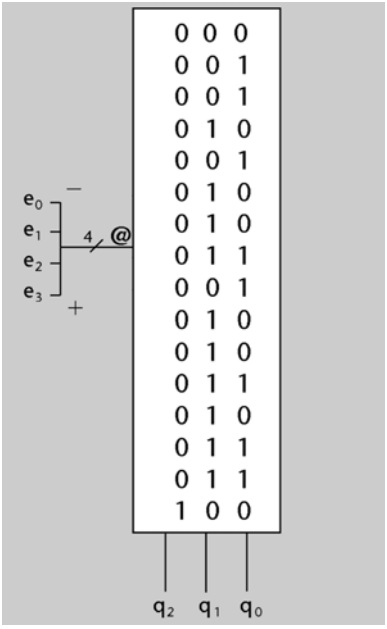
o_3	o_2	o_1	o_0	**CUANTOS**	q_2	q_1	q_0
0	0	0	0	0 (no hay 1 en la entrada)	0	0	0
0	0	0	1	1 (hay un 1 en la entrada)	0	0	1
0	0	1	1	2 (hay dos 1 en la entrada)	0	1	0
0	1	1	1	3 (hay tres 1 en la entrada)	0	1	1
1	1	1	1	4 (hay cuatro 1 en la entrada)	1	0	0

Fijémonos en que la salida CUANTOS tiene que formar la codificación binaria de uno de los números 0, 1, 2, 3 ó 4. Para sintetizarla, por tanto, podemos usar un codificador 8-3 (son necesarios tres bits para codificar el cuatro en binario). Concretamente, tenemos que si el bit más alto de ORDEN que está en 1 es o_i, entonces CUANTOS tiene que valer $i + 1$. Por tanto, conectaremos la señal o_i a la entrada $i + 1$ del codificador, tal como se muestra en la siguiente figura.

En la entrada 0 del codificador puede haber tanto un 1 como un 0 (en los dos casos, CUANTOS valdrá 0 si no hay ningún bit de ORDEN en 1).



c) El tamaño de la ROM debe ser $2^4 \times 3$ bits, y su contenido es el siguiente.



51. Primero, deduciremos la expresión algebraica de cada una de las funciones. Denominaremos a_1, a_0, b_1 y b_0 a los bits de los números de entrada. Para que los números sean iguales, lo deben ser bit a bit. Por tanto, se tiene que cumplir que $a_0 = b_0$ y $a_1 = b_1$.

La puerta XOR permite detectar la desigualdad entre 2 bits (recordad su tabla de verdad). Por tanto, una negación de su salida detectará la igualdad. Así pues, la expresión para la salida $A = B$ es la siguiente:

$$(a_1 \oplus b_1)'(a_0 \oplus b_0)'.$$

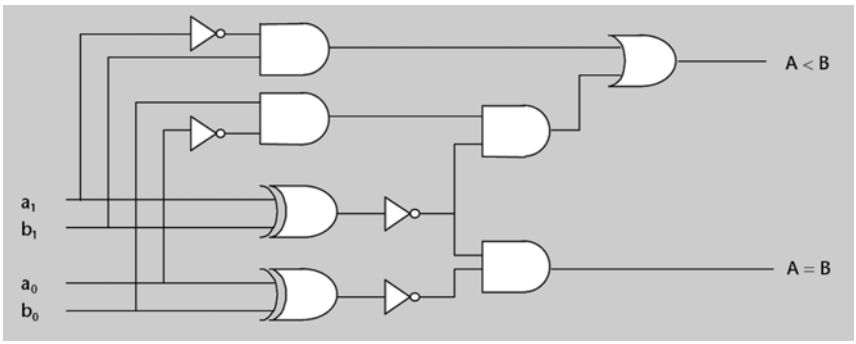
Para hacer la salida $A < B$ tendremos en cuenta lo siguiente:

- A es menor que B si lo es su bit de más peso: $b_1 = 1$ y $a_1 = 0$.
- Si los dos bits de más peso de A y B son iguales, entonces A es menor que B si lo es su bit de menos peso $a_1 = b_1$ y $b_0 = 1$ y $a_0 = 0$.

Por lo tanto, la expresión por la salida $A < B$ es la que mostramos a continuación:

$$b_1 a_1' + (b_1 \oplus a_1)' b_0\, a_0'.$$

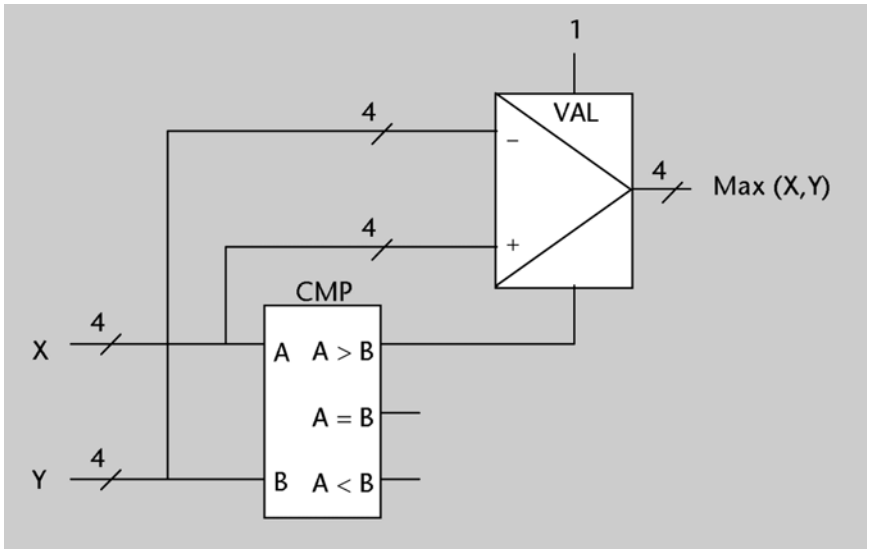
En la siguiente figura se encuentra la implementación de los dos circuitos.



52. Para deducir cuál de los dos números es mayor utilizaremos un comparador de cuatro bits. Conectaremos el número *X* a la entrada *A* y el número *Y* a la entrada *B* del comparador. La salida *A* > *B* de este comparador valdrá 1 si *X* > *Y*, y 0 en el caso contrario.

Esta salida controlará un multiplexor de buses que seleccionará entre *X* e *Y*. Así, cuando *A* > *B* vale 1 seleccionaremos el número *X*, y en cualquier otro caso (*X* < = *Y*) seleccionaremos el número *Y*.

La siguiente figura muestra este diseño.

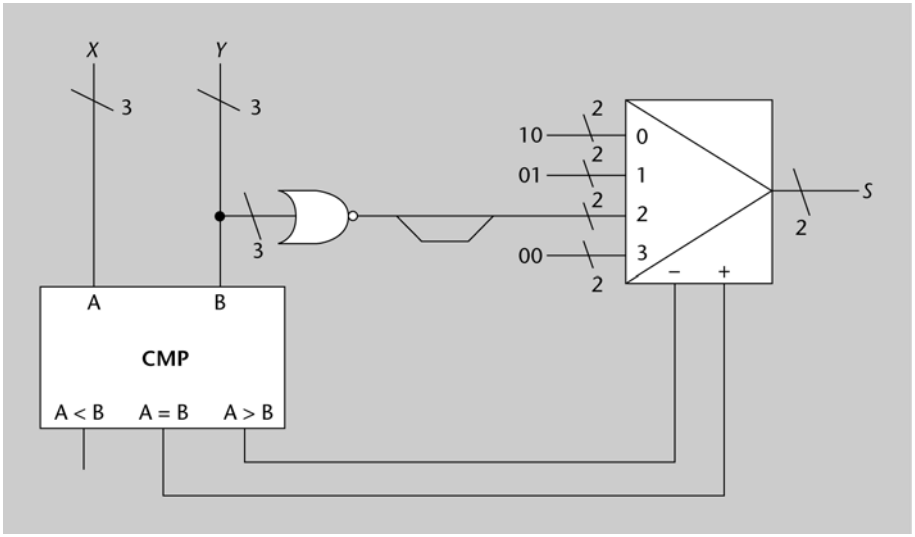


53. La salida *S* vale en todo momento alguno de cuatro valores. Una manera de implementarla es, por lo tanto, mediante un multiplexor 4-1.

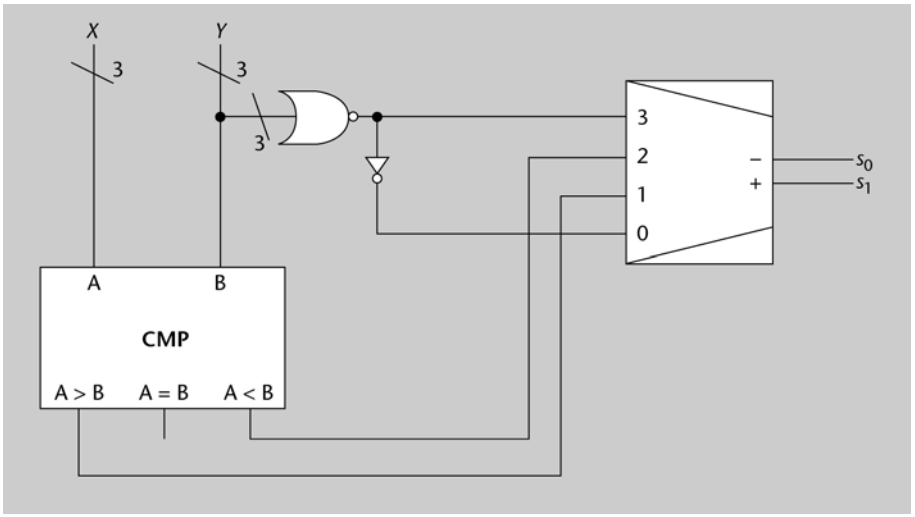
Cuál de estos valores deba elegirse en cada momento depende del resultado de la comparación entre *X* e *Y*, lo que nos lleva a usar un comparador, i, en caso de que *X* = *Y*, de si *X* (o *Y*) = 0; esto lo podemos saber usando una puerta NOR a cuyas entradas deberemos conectar los 3 bits de *X* o de *Y*.

Hay muchas posibilidades para controlar qué valor sale en cada momento del multiplexor a partir de las salidas del comparador y de la puerta NOR. Una es la que se muestra en la figura, según la cual:

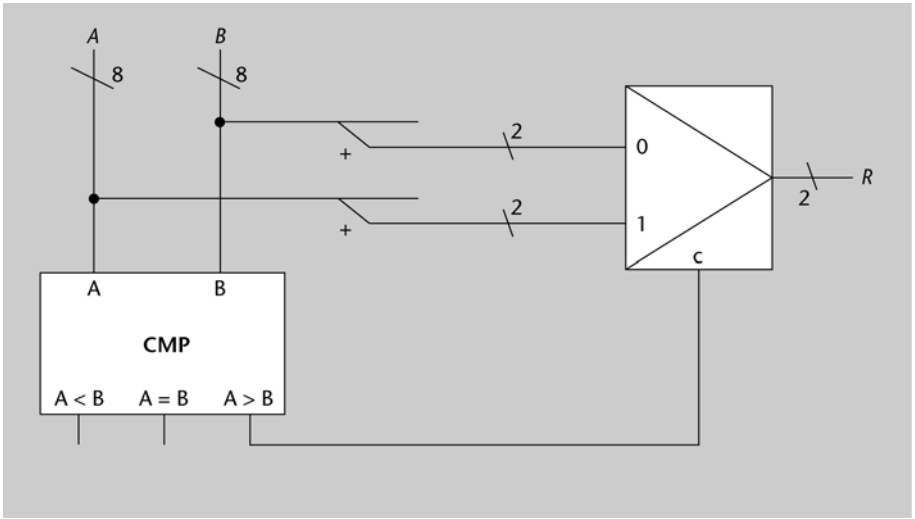
- La entrada de datos 0 pasará hacia la salida cuando *X* < *Y* (ni *X* = *Y* ni *X* > *Y*).
- La entrada de datos 1 pasará hacia la salida cuando *X* > *Y*.
- La entrada de datos 2 pasará hacia la salida cuando *X* = *Y*. En este caso, si la salida de la puerta NOR es 1 la salida *S* debe valer 11, y si es 0, *S* debe valer 00. Por lo tanto, podemos duplicar la salida de la puerta NOR para obtener el bus de dos bits que conectamos a la entrada de datos 2.
- La entrada de datos 3 no pasará nunca hacia la salida porque nunca se dará la combinación [1 1] en las entradas de control. Aquí podemos conectar por lo tanto cualquier cosa, por ejemplo 00.



También podemos diseñar el circuito sin multiplexor, generando los valores que puede tomar la salida con un codificador 4-2, ya que los valores que puede tomar son las cuatro combinaciones que se pueden hacer con dos bits. Observemos que si $X > Y$ o $X < Y$ la entrada 0 del codificador estará a 1, pero en la salida no se generará la combinación [0 0] porque habrá otra entrada del codificador también a 1.

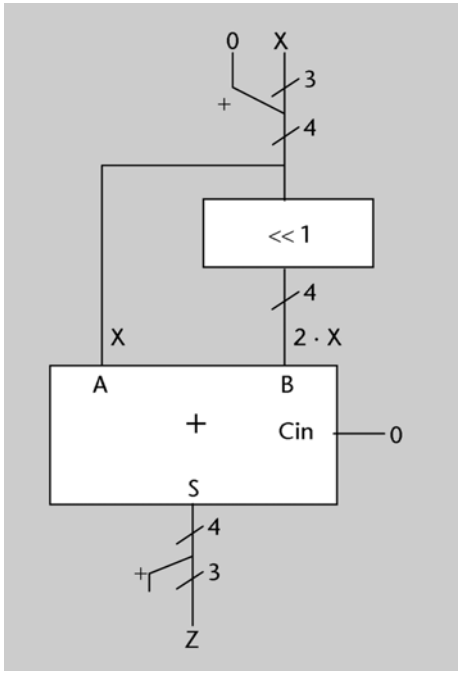


54. Para saber cuál de los dos hornos está más caliente, usaremos un comparador. Observemos que el valor de R coincide en todo momento con los 2 bits más altos de la temperatura del horno que está más caliente (o igual), de manera que podemos generar R a partir de estos bits. Un multiplexor selecciona si se deben tomar de la temperatura A o de la B.



55. Hacer la operación X mod 2n en binario consiste a quedarse con los n bits de menos peso de X. Por lo tanto, para implementar esta función tendremos que sumar $X + 2 * X$ (X desplazado un bit a la izquierda) y quedarse con los tres bits de menos peso de la salida del sumador ([$z2$ $z1$ $z0$]).

La siguiente figura muestra este diseño.

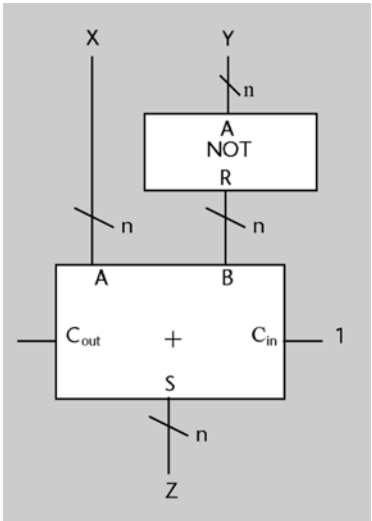


56. Para realizar esta operación aprovecharemos la siguiente igualdad, válida cuando los números están representados en complemento a 2:

$$Z = X - Y = X + (-Y) = X + Y' + 1.$$

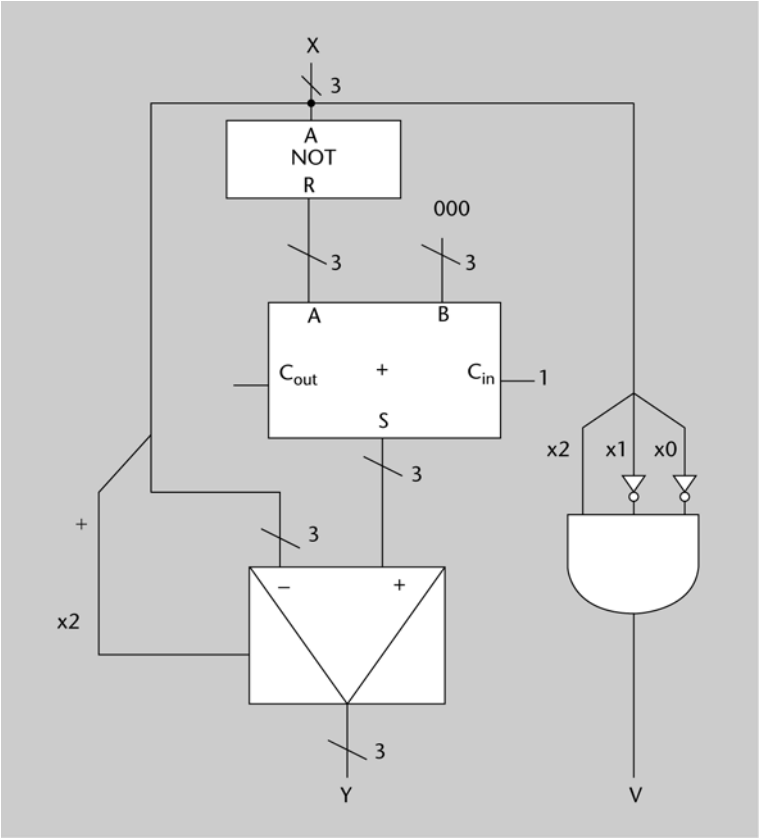
Por tanto, para implementar el circuito necesitamos un bloque NOT de n bits para obtener Y', y un sumador de n bits para hacer la suma $X + Y'$. El 1 que aún queda por sumar se puede conectar a la entrada de acarreo del sumador.

La siguiente figura muestra el diseño propuesto.



57. En caso de que el número sea positivo, la salida tiene que ser igual a la entrada, y en caso de que sea negativo es preciso cambiar el signo de la entrada; esto lo conseguimos negando todos los bits y sumando 1 al resultado. Para seleccionar una opción u otra, utilizamos el bit de más peso de la entrada.

Sólo se puede producir desbordamiento en el caso de que la entrada valga –4, ya que en este caso el valor de la salida (4) no es representable con 3 bits en complemento a 2. Por lo tanto, $V = x_2x_1'x_0'$.



58.

a) La tabla de verdad es la siguiente (aquí escribimos también la interpretación de los números en complemento a 2 para hacerla más comprensible). Para calcular el resultado de la multiplicación pasamos las entradas a decimal como en la actividad 24.

A	B	a_1	a_0	b_1	b_0	m_3	m_2	m_1	m_0	$M = A*B$
0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	–2	0	0	1	0	0	0	0	0	0
0	–1	0	0	1	1	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	0
1	1	0	1	0	1	0	0	0	1	1
1	–2	0	1	1	0	1	1	1	0	–2
1	–1	0	1	1	1	1	1	1	1	–1
–2	0	1	0	0	0	0	0	0	0	0
–2	1	1	0	0	1	1	1	1	0	–2
–2	–2	1	0	1	0	0	1	0	0	4
–2	–1	1	0	1	1	0	0	1	0	2
–1	0	1	1	0	0	0	0	0	0	0
–1	1	1	1	0	1	1	1	1	1	–1
–1	–2	1	1	1	0	0	0	1	0	2
–1	–1	1	1	1	1	0	0	0	1	1

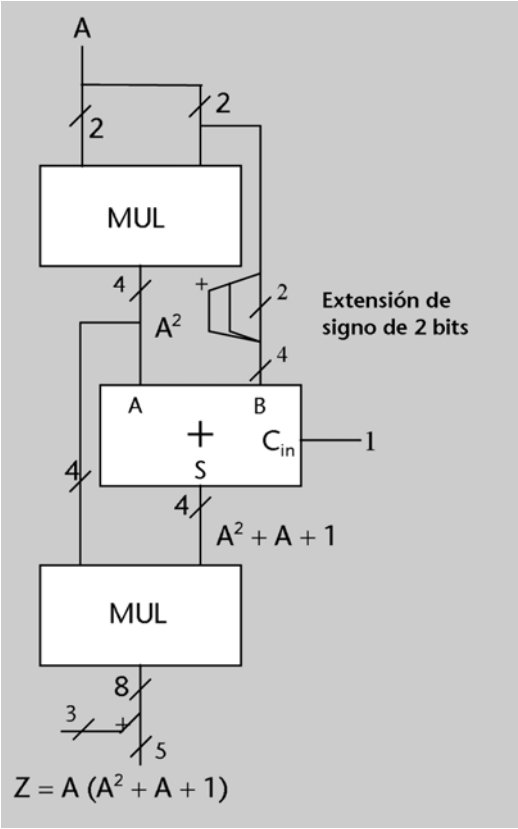
b) Para obtener la solución que se describe en el enunciado, con un sumador y dos multiplicadores se debe utilizar la siguiente igualdad:

$$A^4 + A^3 + A^2 = A^2\,(A^2 + A + 1).$$

A continuación haremos un estudio del rango que pueden tener cada una de las expresiones para decidir el número de bits de cada bloque..

$a_1\ a_0$	A	A^2	$A^2 + A + 1$	$A^2 \cdot (A^2 + A + 1) = A^4 + A^3 + A^2$
00	0	0	1	0
01	1	1	3	3
10	–2	4	3	12
11	–1	1	1	1
se necesitan:		4 bits	3 bits	5 bits

A^2 se obtendrá mediante un bloque MUL de 2 bits, conectando A a las dos entradas. A^2, pues, tendrá 4 bits (el doble de bits que las entradas).
$A^2 + A + 1$ se obtendrá mediante un sumador, conectando el 1 a la entrada C_{in}. Aunque si nos fijamos en el rango del resultado de esta suma con 3 bits sería suficiente, el sumador debe ser de 4 bits porque A^2 tiene 4 bits. Por tanto, será necesario ampliar a 4 bits la entrada A (haciendo una extensión de signo) para conectarla a la otra entrada del sumador, tal como se muestra en la figura.
Otro bloque MUL de 4 bits calculará $A^2\,(A^2 + A + 1)$. La salida del multiplicador será, pues, de 8 bits, aunque el rango del resultado sólo necesita 5. Éste está formado, por tanto, por los 5 bits de menor peso de la salida de este segundo bloque MUL.



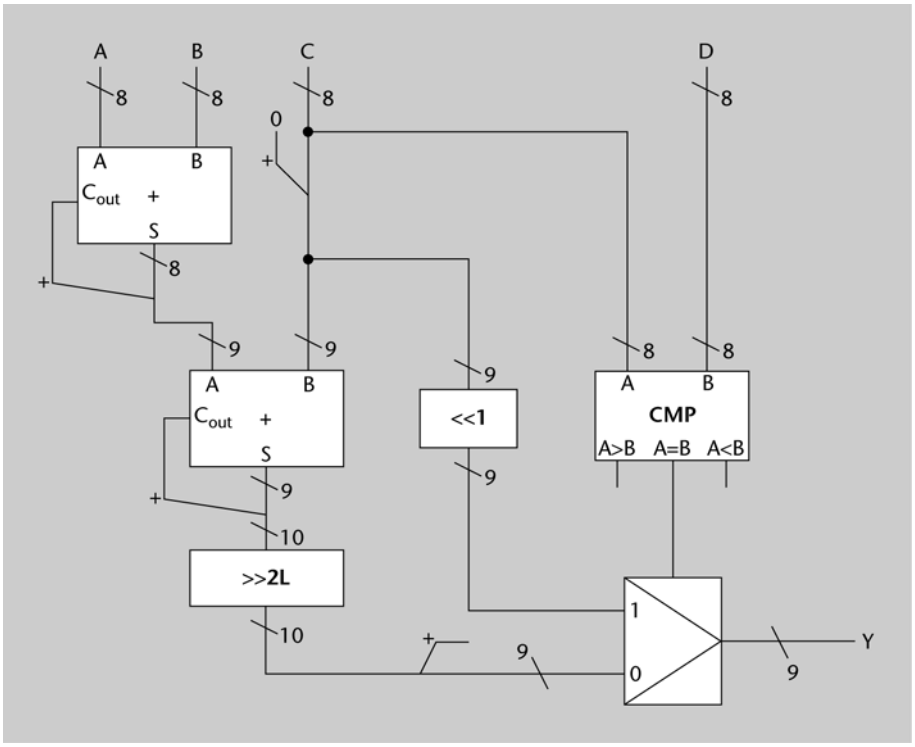
59. Puesto que A, B, C y D codifican números naturales con 8 bits, pueden valer entre 0 y 255.

Multiplicaremos C por 2 mediante un desplazador a la izquierda de 9 bits, porque el resultado será un valor entre 0 y 510. Por lo tanto, extendemos C a 9 bits antes de conectarlo a la entrada del desplazador.

Por otro lado, para obtener la suma $A + B + C$ calcularemos primero $A + B$ con un sumador de 8 bits. El resultado valdrá entre 0 y 510, y por lo tanto se tiene que representar con 9 bits: los alcanzaremos agregando al resultado el C_{out} del sumador. Después sumamos a este resultado el valor de C extendido a 9 bits, y obtendremos un valor entre 0 y 765, que se debe representar con 10 bits. De nuevo los obtendremos agregando al resultado el C_{out} del sumador.

Para dividir la suma entre 4 se tiene que desplazar 2 bits a la derecha, lo cual podemos hacer con un desplazador de 10 bits. Puesto que estamos trabajando con números naturales, el desplazador debe ser lógico. El resultado de la división es un valor entre 0 y 191, que requiere 8 bits para ser representado. Sin embargo, puesto que Y debe tener 9 bits para representar cualquiera de los dos posibles resultados (ya hemos visto que $2 \times C$ requiere 9 bits), descartaremos sólo uno de los bits de salida del desplazador.

Finalmente, comparamos C y D para saber cuál de los dos cálculos tenemos que dar como resultado. Elegimos entre los mismos mediante un multiplexor.

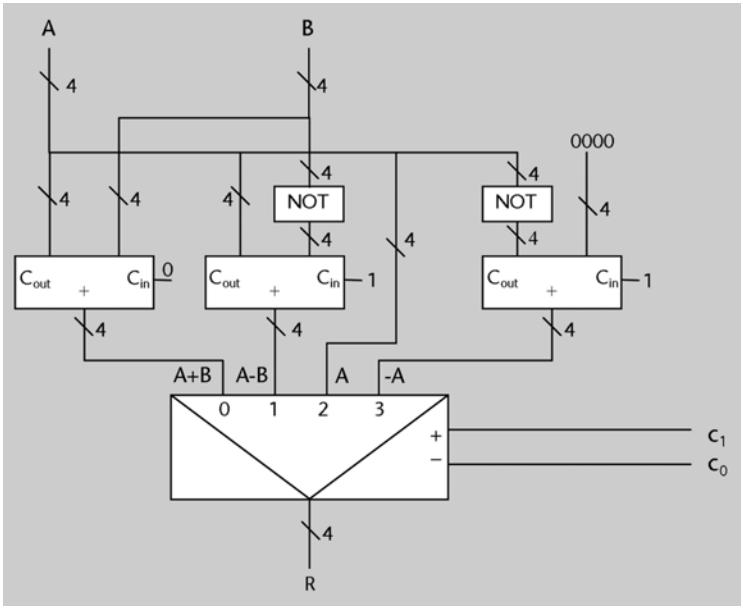


60.

a) Debemos diseñar una UAL que pueda realizar cuatro operaciones. Algunas de éstas se pueden hacer directamente usando bloques que ya hemos estudiado en la teoría de este módulo; sin embargo, otras requieren la utilización de circuitos que combinen bloques con puertas. Analizaremos una por una las operaciones para decidir cómo haremos su implementación. Tal como hemos visto en la teoría, una vez se han implementado los circuitos que hacen todas las operaciones, seleccionaremos la operación que se debe realizar mediante un multiplexor de buses de 4 bits de cuatro entradas.

- $R = A + B$ Esta suma se puede hacer directamente con un sumador de 4 bits.
- $R = A - B$ Tal como se ha visto en la actividad 56, $A - B = A + B' + 1$. Por tanto, para implementar esta resta harán falta un sumador y un bloque NOT de 4 bits.
- $R = A$ La entrada A, de 4 bits, estará conectada directamente a una de las entradas del multiplexor.
- $R = -A$ Como hemos hecho en el caso de la resta, podemos usar la igualdad $-A = A' + 1$ y, por tanto, para implementar esta operación necesitaremos un sumador y un bloque NOT de 4 bits.

En la siguiente figura se puede ver la implementación descrita.



b) Analizamos cómo hay que calcular cada uno de estos bits. Fijémonos en que la operación $R = A$ nunca generará un desbordamiento.

- *Vb*: para estudiar el valor de este bit se deben interpretar las entradas (y, por tanto, también la salida) como números naturales codificados en binario.

– Operación $R = A + B$: en este caso el desbordamiento es el acarreo generado en el último bit.
– Operación $R = A – B$: se producirá desbordamiento cuando el resultado sea negativo (no se podrá codificar en binario). Teniendo en cuenta que para restar sumamos el complementario más 1, el resultado es negativo cuando no se produce acarreo (podéis verificar esta afirmación probando unos cuantos ejemplos).
– Operación $R = –A$: se produce desbordamiento siempre, porque el resultado siempre es un número negativo.

La siguiente tabla resume el valor de *Vb*:

c_1	c_0	*Vb*
0	0	C_{out}
0	1	C_{out}'
1	0	0
1	1	1

- *V*: para estudiar el valor de este bit es necesario que nos pongamos en el caso de interpretar las entradas y, por tanto, también la salida, como números enteros codificados en complemento a 2.

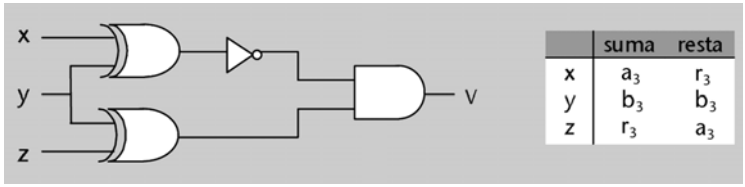
– Operación $R = A + B$: se produce desbordamiento si los dos sumandos son del mismo signo y el resultado es de signo contrario. El signo de los operandos y del resultado viene determinado por su bit de mayor peso. Como en la actividad 51, podemos usar la operación XOR para detectar igualdad o desigualdad entre estos bits y, de este modo, obtener la siguiente expresión:

$$V = (a_3 \oplus b_3)' \, (b_3 \oplus r_3).$$

– Operación $R = A – B$: se produce desbordamiento cuando los dos operandos son de signo contrario y el resultado es del mismo signo que B. Por tanto,

$$V = (a_3 \oplus b_3) \, (b_3 \oplus r_3)'.$$

Como se puede observar en las dos expresiones anteriores, se puede utilizar el mismo circuito para calcular el desbordamiento para $A + B$ y para $A – B$, aunque las entradas se deben conectar de manera diferente. Este circuito tiene tres entradas, x, y y z, y calcula la expresión $s = (x \oplus y)' \cdot (y \oplus z)$. Lo hemos llamado *detector V*, y su diseño interno se presenta a continuación.



– Operación $R = -A$. Se produce desbordamiento sólo en el caso $A = -8$ y, por tanto:

$$V = a_3\ a_2'\ a_1'\ a_0'.$$

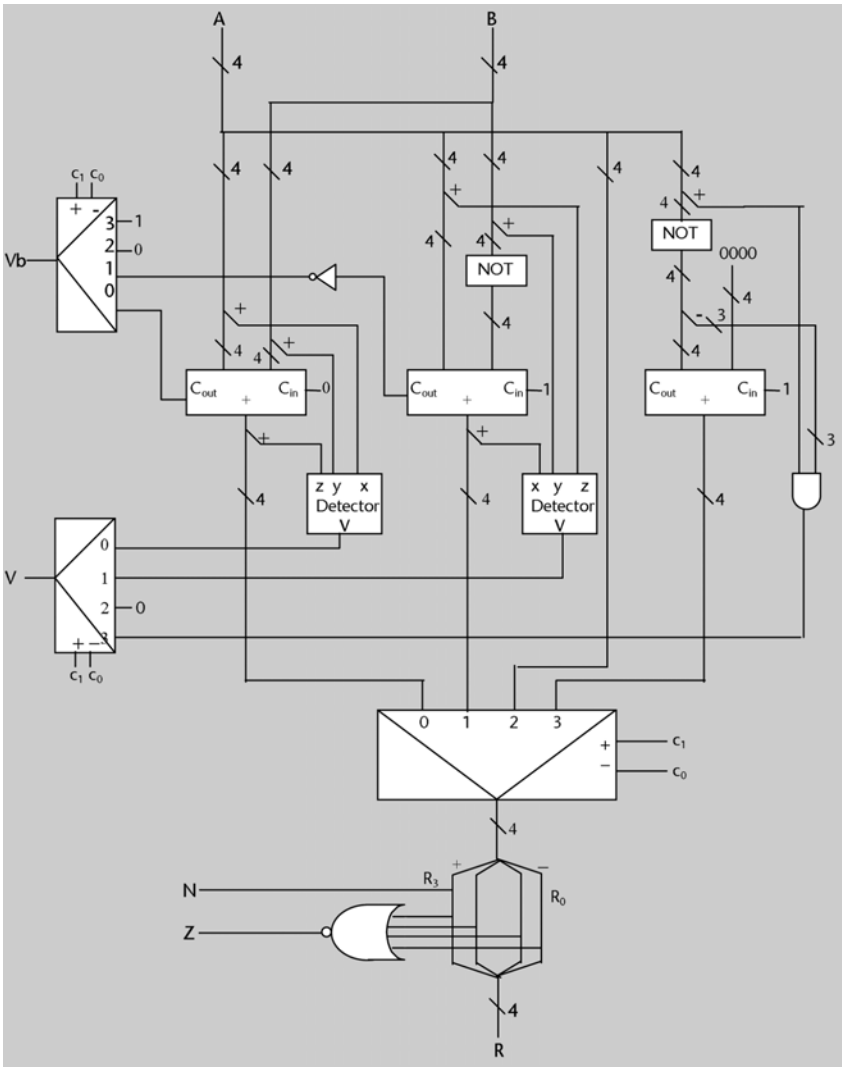
La siguiente tabla resume el valor del bit V:

c_1	c_0	V
0	0	$(a_3 \oplus b_3)'(b_3 \oplus r_3)$
0	1	$(r_3 \oplus b_3)'(b_3 \oplus a_3)$
1	0	0
1	1	$a_3\ a_2'\ a_1'\ a_0'$

Para implementar los dos bits de desbordamiento utilizaremos dos mutiplexores, los dos controlados por los bits c_1 y c_0. Los multiplexores seleccionarán en cada momento el valor correcto de V y Vb según la operación que se realice.

- N: tal como se ha visto en la teoría, el signo del resultado es el bit de más peso.
- Z: tal como se ha visto en la teoría, para calcular si el resultado es 0 se hace con una puerta NOR de cuatro entradas.

A continuación, se muestra el circuito.



Ejercicios de autoevaluación

1. Debemos demostrar que $(xyz)' = x' + y' + z'$. En primer lugar aplicamos la propiedad asociativa y, después, el teorema de De Morgan para dos variables, dos veces:

$$(xyz)' = ((xy)z)' = (xy)' + z' = x' + y' + z'.$$

Haremos lo mismo para cuatro variables, habiendo demostrado ya que se cumple para tres variables:

$$(xyzw)' = ((xyz)w)' = (xyz)' + w' = x\ '+ y' + z' + w'.$$

2.

$F = (((w'\ y)' + wx)\ z + x')\ y' = ((w'' + y' + wx)\ z + x')\ y' = ((w + y' + wx)\ z + x')\ y' = ((w + y') + z + x')\ y' =$
$= (wz + y'z + x')\ y' = wzy' + y'y'z + x'y' = wzy' + y'z + x'y'.$

3. Primero haremos la tabla de verdad. Después implementaremos el circuito a dos niveles.

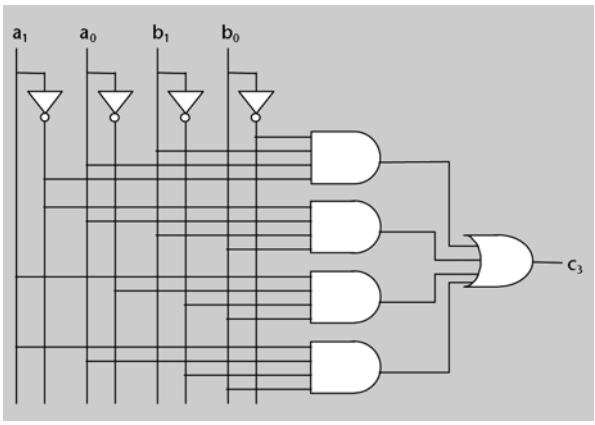
El rango de valores que puede tomar un número entero de dos bits representado en complemento a dos es [–2,1]. Por tanto, el rango de la multiplicación de dos números será [–2..4]. Para representar el 4 en complemento a 2 necesitamos 4 bits. Por tanto, la salida tendrá 4 bits.

Llamaremos $[a_1\ a_0]$ y $[b_1\ b_0]$ respectivamente a los bits de los dos números de entrada, y $[c_3\ c_2\ c_1\ c_0]$ a los bits de su multiplicación. La tabla de verdad de este multiplicador es la siguiente:

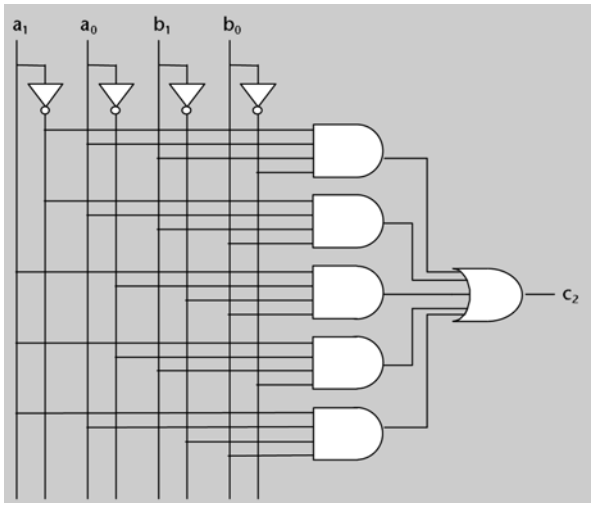
a_1	a_0	b_1	b_0	c_3	c_2	c_1	c_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0
0	0	1	0	0	0	0	0
0	0	1	1	0	0	0	0
0	1	0	0	0	0	0	0
0	1	0	1	0	0	0	1
0	1	1	0	1	1	1	0
0	1	1	1	1	1	1	1
1	0	0	0	0	0	0	0
1	0	0	1	1	1	1	0
1	0	1	0	0	1	0	0
1	0	1	1	0	0	1	0
1	1	0	0	0	0	0	0
1	1	0	1	1	1	1	1
1	1	1	0	0	0	1	0
1	1	1	1	0	0	0	1

A continuación, se muestran los circuitos a dos niveles correspondientes a las cuatro funciones de salida.

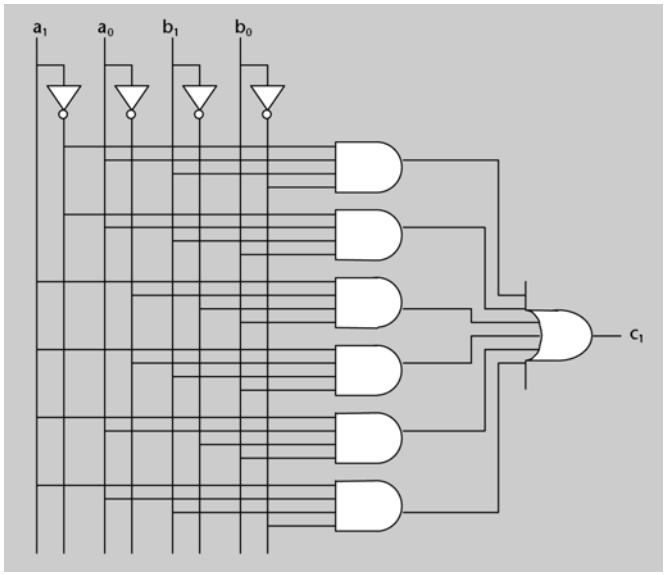
$c_3 = a_1'\ a_0\ b_1\ b_0' + a_1'\ a_0\ b_1\ b_0 + a_1\ a_0'\ b_1'\ b_0 + a_1\ a_0\ b_1'\ b_0.$



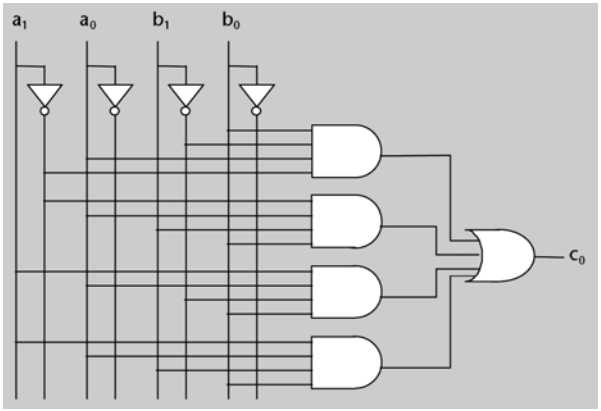
$c_2 = a_1' \, a_0 \, b_1 \, b_0' + a_1' \, a_0 \, b_1 \, b_0 + a_1 \, a_0' \, b_1' \, b_0 + a_1 \, a_0' \, b_1 \, b_0' + a_1 \, a_0 \, b_1' \, b_0.$



$c_1 = a_1' \, a_0 \, b_1 \, b_0' + a_1' \, a_0 \, b_1 \, b_0 + a_1 \, a_0' \, b_1' \, b_0 + a_1 \, a_0' \, b_1 \, b_0 + a_1 \, a_0 \, b_1' \, b_0 + a_1 \, a_0 \, b_1 \, b_0'.$



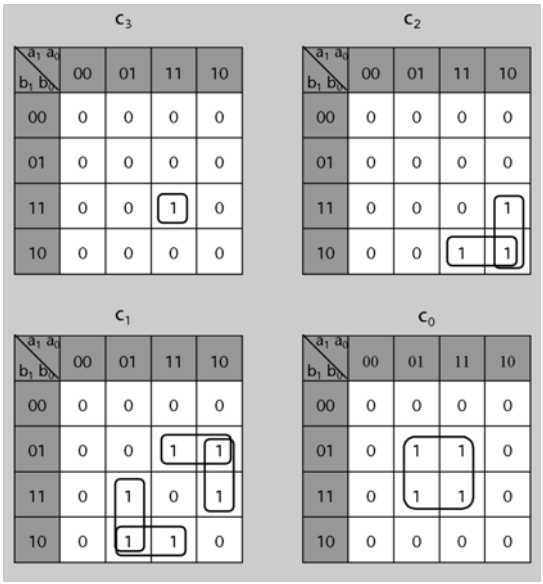
$c_0 = a_1' \, a_0 \, b_1' \, b_0 + a_1' \, a_0 \, b_1 \, b_0 + a_1 \, a_0 \, b_1' \, b_0 + a_1 \, a_0 \, b_1 \, b_0.$



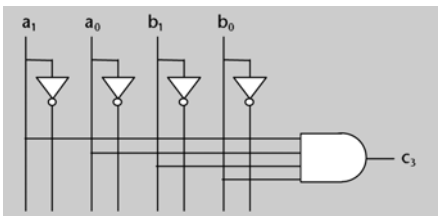
4. Tenemos la siguiente tabla de verdad:

a_1	a_0	b_1	b_0	c_3	c_2	c_1	c_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0
0	0	1	0	0	0	0	0
0	0	1	1	0	0	0	0
0	1	0	0	0	0	0	0
0	1	0	1	0	0	0	1
0	1	1	0	0	0	1	0
0	1	1	1	0	0	1	1
1	0	0	0	0	0	0	0
1	0	0	1	0	0	1	0
1	0	1	0	0	1	0	0
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	1	0
1	1	1	1	1	0	0	1

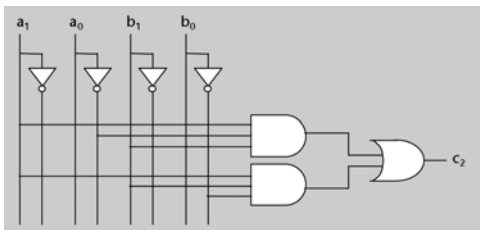
A continuación, simplificaremos por Kanaugh cada una de las funciones de salida y, después, implementaremos el circuito.



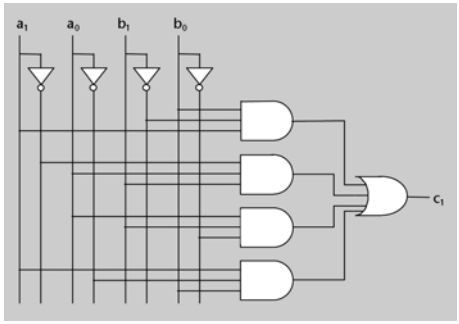
$c_3 = a_1a_0b_1b_0$.



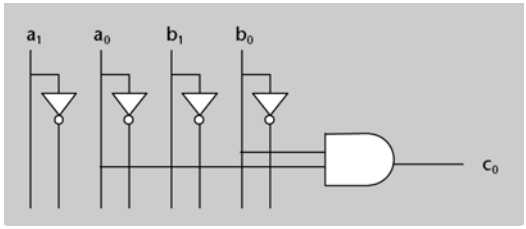
$c_2 = a_1a_0'b_1 + a_1b_1b_0'$.



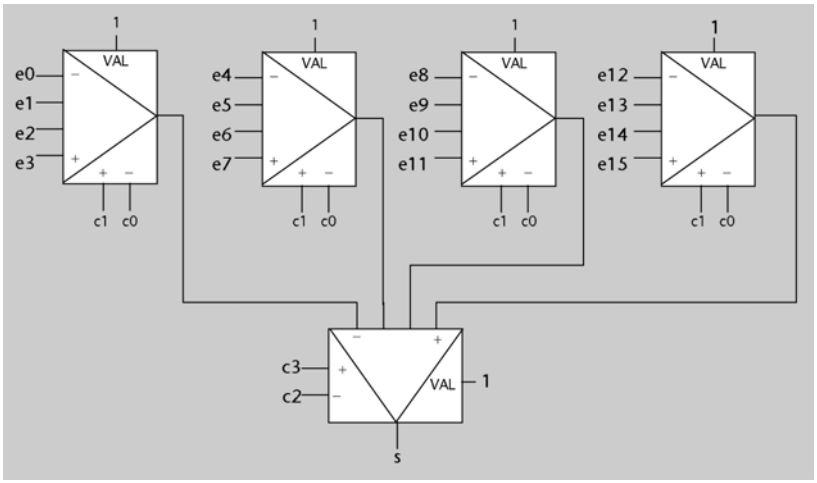
$c_1 = a_1 b_1' b_0 + a_1' a_0 b_1 + a_0 b_1 b_0' + a_1 a_0' b_0$.



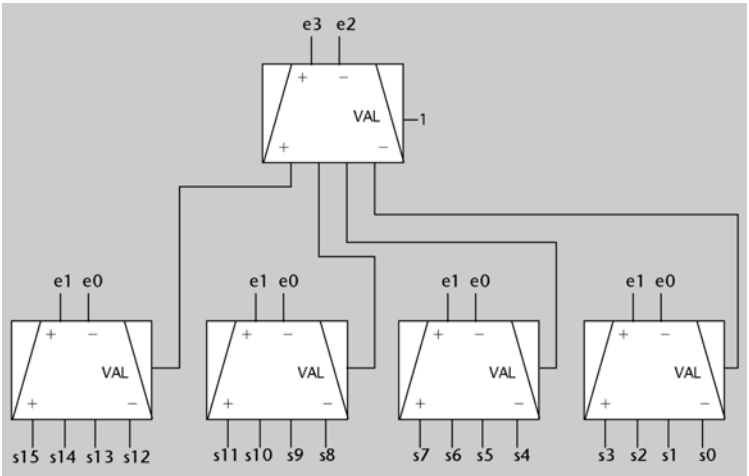
$c_0 = a_0 b_0$.



5.



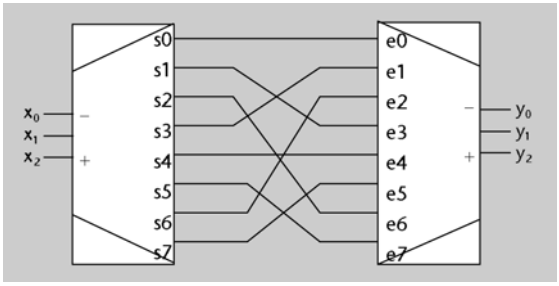
6.



7. La tabla de verdad es la siguiente::

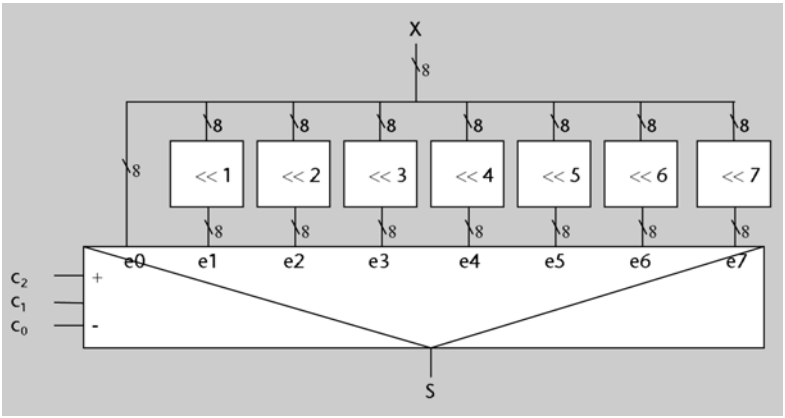
X	x_2	x_1	x_0	y_2	y_1	y_0	$Y = 3X \bmod 8$
0	0	0	0	0	0	0	0
1	0	0	1	0	1	1	3
2	0	1	0	1	1	0	6
3	0	1	1	0	0	1	1
4	1	0	0	1	0	0	4
5	1	0	1	1	1	1	7
6	1	1	0	0	1	0	2
7	1	1	1	1	0	1	5

A continuación se muestra su diseño.



8.

a)



b) Teniendo en cuenta que una operación de desplazamiento a la izquierda de un bit es equivalente a multiplicar por 2, si desplazamos X n bits, esta operación equivale a $S = X \cdot 2n$.

c) Si $C = 010$, el desplazamiento corresponde a la operación de multiplicar por 4. Si X es natural, el rango representable con 8 bits es [0..255]. Por tanto, se produce desbordamiento cuando $X \geq 256 / 4 = 64$.

Si X es entero, el rango de número representables es [–128..127]. Por tanto, se produce desbordamiento cuando $X \geq 128 / 4 = 32$ o bien cuando $X < -128 / 4 = -32$.

d) Si X es un número natural, los bits de más peso que se añaden al número cuando éste se desplaza a la derecha deben valer 0. Por tanto, se debe usar un desplazador lógico.

Si X es un número entero, los bits de más peso que se añaden al número cuando éste se desplaza a la derecha deben valer lo mismo que el bit de mayor peso del número para conservar el signo. Por tanto, se debe usar un desplazador aritmético.

e) Un desplazador a la derecha no computa la función $X / 2^n$, sino la función $\lfloor X / 2^n \rfloor$. Por tanto, si se usa para calcular $X / 2^n$, el resultado no será exacto si alguno de los bits que se pierden al desplazar a la derecha vale 1. Dicho de otra manera, si el desplazamiento es de k bits ($C = k$), el resultado es inexacto cuando tenemos lo siguiente:

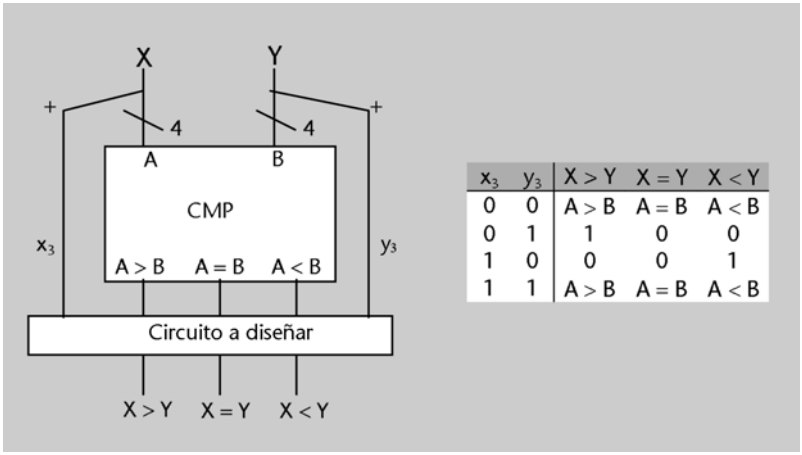
$$X \bmod 2^k \neq 0.$$

9. El resultado de la comparación depende de los signos de los dos números (que vienen determinados por su bit de más peso: 1 para los negativos, 0 para los positivos o cero).

Si los dos números son de signos diferentes, entonces el mayor es positivo.

Cuando los dos números son del mismo signo, entonces los podemos comparar usando un comparador de números naturales. Fijémonos en que el resultado será correcto también en el caso de que los dos números sean negativos, porque si $X > Y$, interpretando X e Y en complemento a dos, entonces también se cumple que $X > Y$ si los interpretamos en binario. Por ejemplo, si comparamos $X = -1$ (1111) con $Y = -2$ (1110) el comparador sacaría un 1 para la salida $X > Y$.

A continuación se muestra un primer esquema del circuito y la tabla de verdad que describe el comportamiento de la parte del circuito que queda por implementar, de acuerdo con el signo de los dos números.

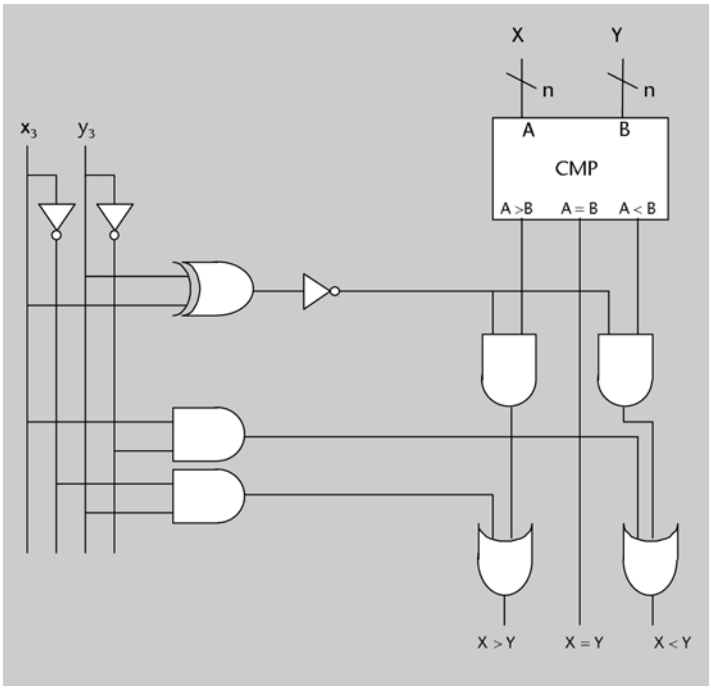


x_3	y_3	X > Y	X = Y	X < Y
0	0	A > B	A = B	A < B
0	1	1	0	0
1	0	0	0	1
1	1	A > B	A = B	A < B

De estas tablas se deducen las siguientes igualdades:

$[X > Y] = x_3' y_3 + (x_3 \oplus y_3)' \cdot [A > B]$,
$[X < Y] = x_3 y_3' + (x_3 \oplus y_3)' \cdot [A < B]$,
$[X = Y] = [A = B]$.

A continuación se muestra el circuito completo:



10. a) La tabla de verdad de un *full adder* es la siguiente:

a_i	b_i	c_i	c_{i+1}	s_i
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

La expresión en suma de productos de c_{i+1} es la siguiente:

$$c_{i+1} = a_i' \, b_i \, c_i + a_i \, b_i' \, c_i + a_i \, b_i \, c_i' + a_i \, b_i \, c_i$$

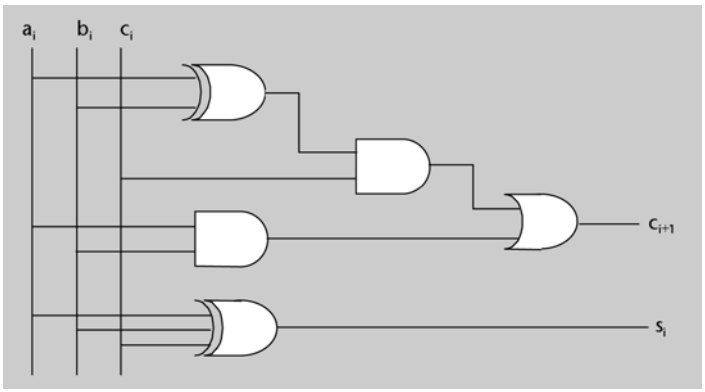
Podemos sacar factor común en los dos primeros y los dos últimos términos, y obtenemos lo siguiente:

$$c_{i+1} = (a_i' \, b_i + a_i \, b_i') \, c_i + a_i \, b_i \, (c_i' + c_i) = (a_i \oplus b_i) \, c_i + a_i \, b_i$$

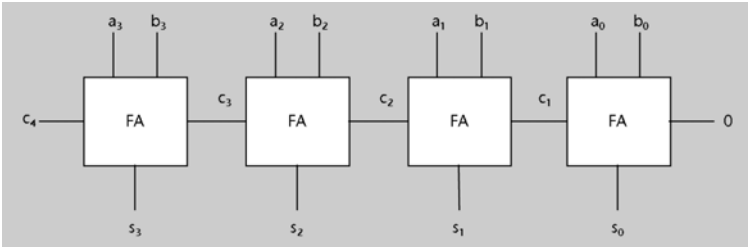
Por lo que se refiere a s_i, vemos que cuando $a_i = 0$ vale $(b_i \oplus c_i)$ y cuando $a_i = 1$ vale $(b_i \oplus c_i)'$. Esto se puede expresar de la siguiente manera:

$$s_i = a_i' \, (b_i \oplus c_i) + a_i \, (b_i \oplus c_i)' = a_i \oplus b_i \oplus c_i$$

El circuito interno de un *full adder*, que implementa estas expresiones, se muestra en la siguiente figura:



b) A continuación, se muestra cómo se encadenan cuatro *full adders* para conseguir un sumador de números de 4 bits. Como se puede ver en la figura, en el bit de acarreo de entrada, c_0, se debe conectar un 0 para que la suma sea correcta.



11. Se trata de diseñar un circuito que haga lo siguiente:

Si $s'/r = 0$, entonces $R = A + B$.
Si $s'/r = 1$, entonces $R = A - B = A + B' + 1$.

Podemos utilizar un sumador de números de cuatro bits para hacer este circuito.

– Conectaremos el número A a una entrada.

– En la otra entrada conectaremos el número *B* o el número *B'*, dependiendo de si la señal *s'/r* vale 0 o 1 respectivamente. Recordemos esta propiedad de la función XOR:

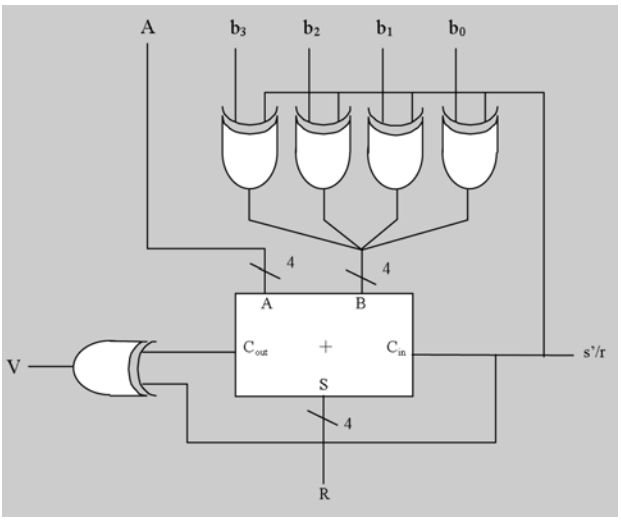
$$0 \oplus x = x,\ 1 \oplus x = x'$$

Por tanto, para obtener *B* o *B'* según *s'/r*, podemos hacer una XOR de *s'/r* con cada uno de los bits de *B*.

– Conectaremos un 0 a la entrada de acarreo si *s'/r* = 0, o bien un 1 si *s'/r* = 1. Por tanto, aquí podemos conectar directamente la señal *s'/r*.

Otra opción para seleccionar *B* o *B'* con *s'/r* sería utilizando un multiplexor.

El circuito sumador/restador que se describe en los párrafos anteriores se muestra a continuación.



Cuando se realiza una suma, la salida *V* coincide con el último bit de acarreo.

Cuando se hace la resta mediante la suma del complementario más 1, el acarreo del último bit es siempre la negación de lo que obtendríamos si se hubiera hecho directamente la resta.

Por tanto, cuando *s'/r* = 0 se cumple que $V = C_{out}$ y cuando *s'/r* = 1 se cumple que $V = C_{out}'$. Así pues, si utilizamos la misma propiedad de la función XOR que hemos utilizado antes, obtenemos lo siguiente:

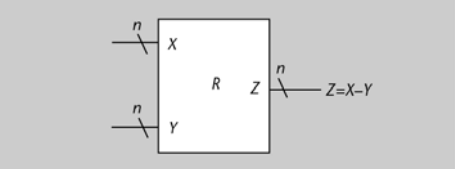
$$V = s'/r \oplus C_{out}$$

12. Para saber hacia qué planta se tienen que mover los ascensores (es decir, qué valor tenemos que dar a la señal *destino*), necesitamos codificar en binario (número natural) la planta desde la que se ha hecho la llamada. Esto lo podemos conseguir mediante un codificador de 8 entradas y 3 salidas, con b_i conectado a la entrada e_i, las entradas e_7 y e_6 conectadas a cero y con los 3 bits de salida juntos formando el bus destino.

Para saber qué ascensor está más cerca de la planta de destino, podemos restar la planta donde está cada uno de los ascensores (*plantaA* y *plantaB*) de la planta *destino*. Para hacer la resta, podemos utilizar un sumador y aplicar la igualdad siguiente:

$$X - Y = X + (-Y) = X + Y' + 1$$

tal y como se hace en la actividad 56. Puesto que X e Y se deben representar en complemento a 2, añadiremos un 0 como bit de más peso en todos los operandos de las restas. Representamos el circuito que se diseña en esta actividad mediante el bloque R, de la manera siguiente:



La resta de dos valores en el rango [0, 5] dará un valor en el rango [–5, 5]. Tenemos suficiente con 4 bits para codificar este rango en complemento a 2, y por lo tanto n puede ser 4.

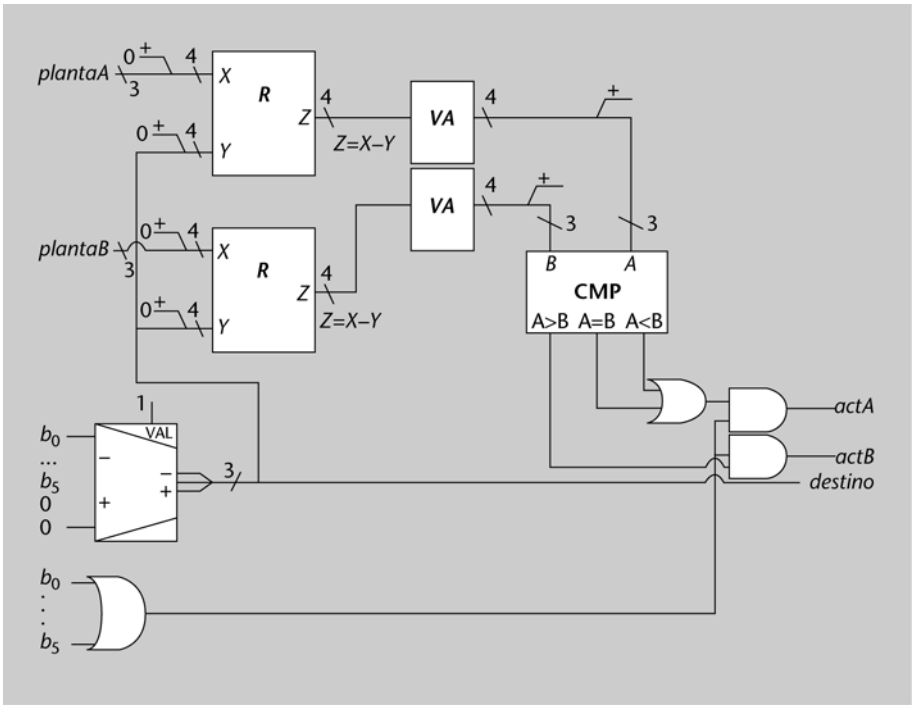
Las diferencias *plantaX – destino* pueden ser tanto positivas como negativas, porque el ascensor X puede estar más arriba o más abajo que la planta desde donde se hace la llamada. La distancia que buscamos será el valor absoluto de la resta. Lo haremos utilizando un bloque *VA*, que contendrá el circuito que se ha diseñado en la actividad 57 (pero de 4 bits). Este valor absoluto nunca será mayor que 5, de manera que se puede representar con 3 bits. Por lo tanto, no se producirá nunca desbordamiento en este bloque *VA*, y es más, descartamos uno de sus bits de salida.

Una vez tenemos ya las dos distancias, podemos saber cuál de las dos es mayor mediante un comparador. Es suficiente con un comparador de números de 3 bits.

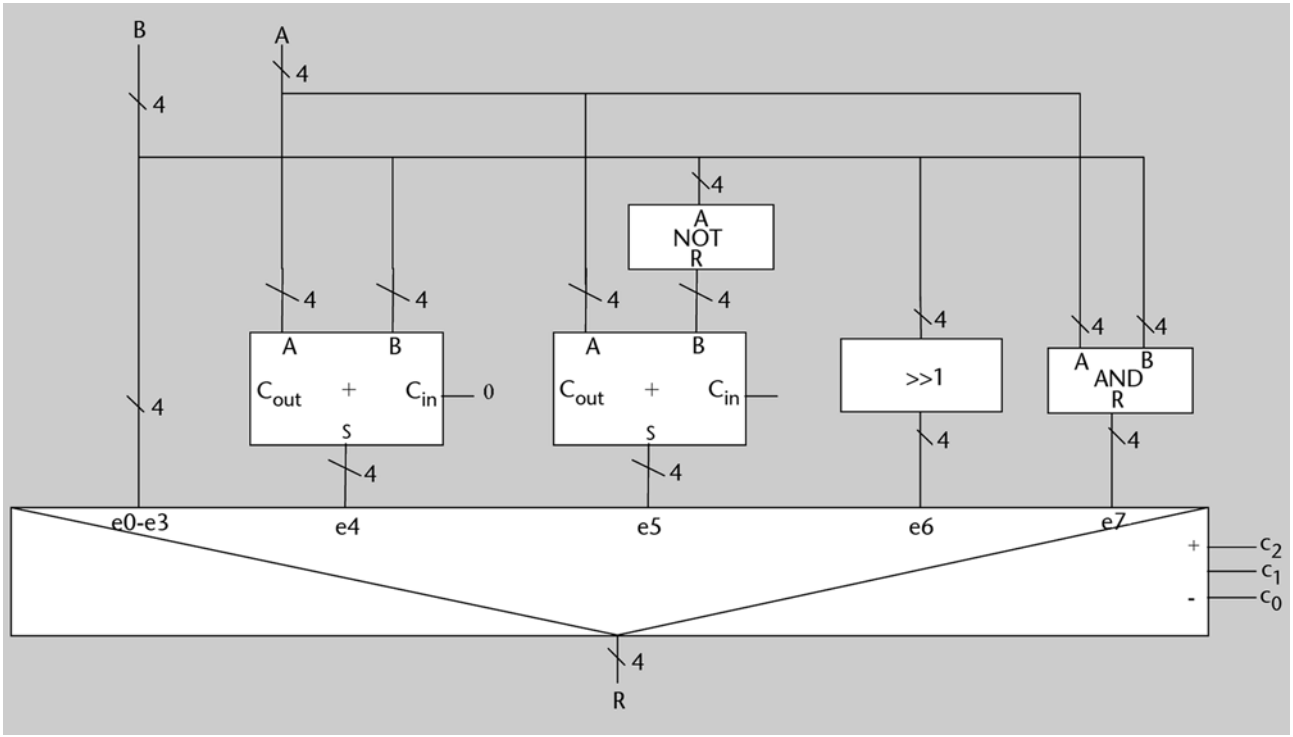
Si ponemos la distancia a la que se encuentra el ascensor A en la entrada A del comparador y la otra distancia en la entrada B, sabemos que cuando la salida $A < B$ del comparador sea 1 deberemos poner a 1 la señal *actA*. De manera análoga, cuando $A > B$ sea 1, tendremos que poner a 1 la señal *actB*. En caso de que $A = B$ sea 1, es decir, que los dos ascensores estén a la misma distancia, tenemos que poner a 1 *actA* (es decir, $actA = (A < B) + (A = B)$.

Puesto que las señales *actA* y *actB* se deben mantener a 0 mientras no se pulse ningún botón, hay que hacer un producto lógico (AND) entre estas salidas del comparador y una señal que valga 0 siempre que no se haya pulsado ningún botón. Lo podemos obtener, por ejemplo, haciendo una suma lógica (OR) con todos los bits b_i.

El circuito completo queda de la manera siguiente:



13. El circuito de esta UAL se muestra en la figura siguiente. Observamos que la entrada B se conecta a las cuatro entradas de menos peso del multiplexor.



Bibliografía

Gajsky, D.D. (1997). *Principios de Diseño Digital*. Prentice-Hall.

Hermida, R.; Corral, A. Del; Pastor, E.; Sánchez, F. (1998). *Fundamentos de Computadores*. Madrid: Síntesis.