DISEÑO COMBINACIONAL

Muchos sistemas electrónicos se manejan con señales discretas (N° fijo de estados posibles). Trabajar con estas señales vuelve más sencillo transmitir, procesar y guardar las magnitudes digitales que las analógicas, aunque una magnitud analógica tiene mejor definición. Adicionalmente, las señales digitales se ven menos afectadas por la presencia de ruido.

Los microprocesadores han hecho posible acceder a capacidades de procesamiento, almacenamiento y cálculo dentro de un solo circuito integrado. Su bajo costo y flexibilidad los vuelven viables en múltiples aplicaciones de automatización o procesamiento.

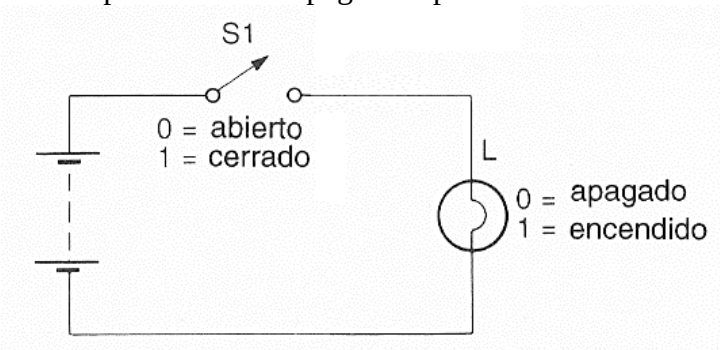
Son circuitos electrónicos que contienen gran densidad de transistores (en el orden de 10^5-10^6), organizados en partes menores (compuertas) que utilizan unos pocos transistores y componentes pasivos

Estados lógicos

Las señales discretas de mayor interés en este punto son las que toman dos estados: On-Off (un interruptor presionado o libre, una luz encendida o apagada, etc.).

Estas magnitudes tienen dos valores posibles – se las representa con variables binarias (un nombre simbólico que puede tomar dos valores).

Ejemplo: Switch para encender-apagar lámpara



Es común usar el 1 para representar el estado *encendido* o *un conmutador/interruptor cerrado* o un estado *verdadero* y el 0 para el estado *apagado* o un *conmutador abierto* o un estado *falso*.

La tabla que relaciona estados lógicos de la/las entradas y la/las salidas es una **tabla de verdad**. El orden en el que se escriben las variables de entrada es un *orden binario ascendente*.

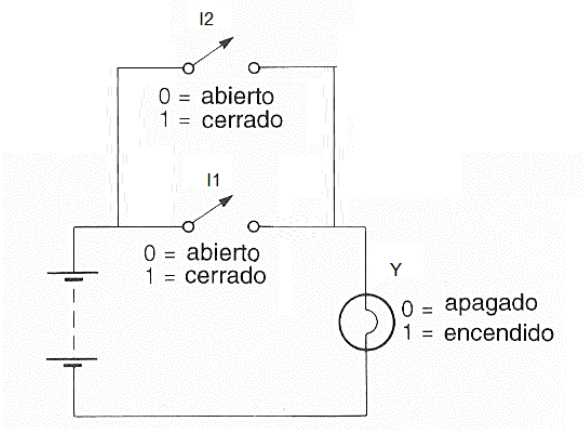
Switch	Lámpara
Abierto	Apagado
Cerrado	Encendido

⇨

S1	L
0	0
1	1

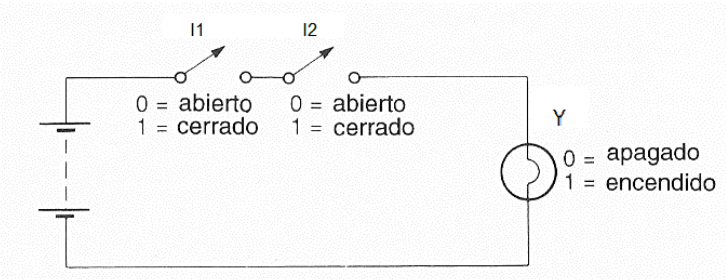
Si se consideran los estados lógicos de la entrada (S1) y de la salida (L), se puede decir para este caso que L = S1 describe el funcionamiento lógico del circuito (independientemente de su forma física).

Otras funciones lógicas simples pueden representarse de manera similar:



I1	I2	Y
0	0	0
0	1	1
1	0	1
1	1	1

Y=I1 or I2
cuando I1, I2 o las dos son 1, la salida es 1



I1	I2	Y
0	0	0
0	1	0
1	0	0
1	1	1

Y=I1 and I2
solo cuando I1 e I2 sean 1, Y vale 1

Las funciones lógicas básicas son

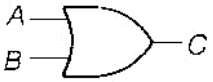
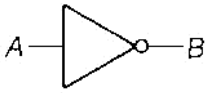
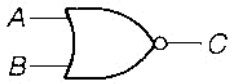
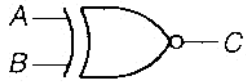
AND
OR (se llama OR INCLUSIVO porque incluye el caso donde dos entradas son 1)
NOT (inversor)

La conversión a niveles lógicos independiza al análisis de la implementación, en una etapa inicial. Normalmente no se conoce previamente el sistema en el cual se implementa la función lógica, sino que lo que se pide es obtener un circuito que cumpla una función lógica determinada (tabla de verdad) y se espera diseñar el circuito con N entradas que lo logre – caja negra con N entradas y M salidas. El bloque de construcción inicial de los circuitos lógicos es la **compuerta lógica**.

Compuertas y circuitos lógicos.

Compuerta lógica - Sistema que genera salida binaria apropiada a partir de una o más entradas lógicas. Cada una tiene un símbolo lógico correspondiente, y pueden tener más de dos entradas. Es posible construir funciones más complejas combinando compuertas sencillas en un **diagrama lógico.** Se las implementa con transistores y algunos componentes pasivos (resistencias, etc.).

Tabla 9.1 Compuertas lógicas.

Función	*Símbolo*	*Representación booleana*	*Tabla de verdad*
AND		$C = A \cdot B$	A B | C 0 0 | 0 0 1 | 0 1 0 | 0 1 1 | 1
OR		$C = A + B$	A B | C 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 1
NOT		$B = \overline{A}$	A | B 0 | 1 1 | 0
NAND		$C = \overline{A \cdot B}$	A B | C 0 0 | 1 0 1 | 1 1 0 | 1 1 1 | 0
NOR		$C = \overline{A + B}$	A B | C 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 0
OR exclusiva		$C = A \oplus B$	A B | C 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 0
NOR exclusiva		$C = \overline{A \oplus B}$	A B | C 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 1

Compuertas básicas AND OR INCLUSIVO NOT

Compuertas Compuestas NAND NOR inclusivo OR exclusivo

NOR exclusivo (compuerta equivalencia o de igualdad)

Algebra booleana, lógica combinacional

Proporciona **notación** para describir las funciones lógicas y define operaciones para simplificar su forma.

Constantes booleanas 0 o 1 – falso o verdadero

Variables booleanas Pueden tomar diferentes valores 0 y 1

Se las nombra A,B,… para entrads, y X,Y,Z … para salidas, por ejemplo

Puede variar

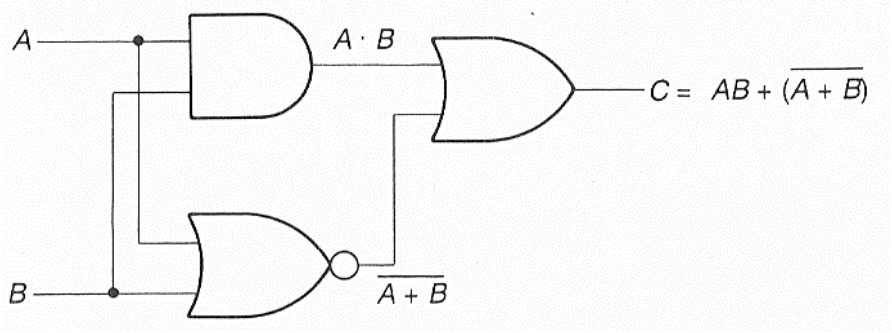
Funciones booleanas - Notación

A and B	A . B	
A or B	A + B	(or inclusivo)
Not(A)	$\overline{A}$ o A’	

La notación para las demás compuertas se encuentra en la tabla 9.1

La expresión como producto y suma abre la puerta para plantear |una analogía con algebra convencional

Es posible construir diagramas/circuitos lógicos con N entradas y M salidas aprovechando los símbolos de las compuertas, y relacionarlos con las expresiones lógicas (ecuaciones booleanas o de Boole) que vinculan entrada y salida.



Teoremas booleanos

Por analogía con el álgebra convencional, se pueden plantear identidades y demostrar teoremas del álgebra booleana.

Leyes	**Conmutativa**	AB=BA	A+B=B+A
	Distributiva	A(B+C) = AB +AC	A+BC=(A+B)(A+C)
	Asociativa	A(BC)=(AB)C	A+(B+C)=(A+B)+C
	Absorción	A+AB=A	A(A+B) = A
	Ley de De Morgan	$\overline{A+B}=\overline{A}.\overline{B}$	$\overline{A.B}=\overline{A}+\overline{B}$
	además	$A+\overline{A}.B=A+B$	$A.(\overline{A}+B)=A.B$

Con esta notación, es posible expresar un **diagrama lógico** o una **tabla de verdad** con una **expresión booleana o ecuación de Boole**. Por ejemplo, a partir de la tabla de verdad

A	B	C
0	0	0
0	1	1
1	0	1
1	1	1

Se puede decir que C será 1 cuando

A=0 **y** B=1 **o si** A=1 **y** B=0 **o si** A=1 **y** B=1

Traduciendo esto a una expresión booleana, queda

$$C = (\bar{A}.B) + (A.\bar{B}) + (A.B) \qquad (1)$$

Comparando con la tabla 9.1, también se puede decir que

$$C = A + B \qquad (2)$$

Se puede observar que las expresiones booleanas no son únicas

Dado que llevar a la práctica la función lógica implica obtener los componentes que la permitan implementar, la segunda ecuación parece ser una expresión más beneficiosa que la primera, en términos de costo y eficiencia.

La expresión (1), más compleja que (2), es una de las formas canónicas para la expresión lógica que se puede obtener a partir de la tabla de verdad, llamada **suma de minitérminos** (o suma de productos). Para obtener esta expresión:

- Generar un minitérmino por cada fila de la tabla en la que aparezca un 1 en la salida
- El minitérmino es un producto (and) que contiene cada variable de entrada en orden, no negada si está en 1, y negada si está en 0
- La expresión final es la suma (or) de estos minitérminos

Dado que la forma de la tabla, para cualquier serie de entradas, es igual del lado izquierdo (entradas), es posible numerar las filas y describir la expresión lógica como la suma de los minitérminos (m..) correspondientes a las filas de interés.

Así, la tabla de verdad de la compuerta OR

	A	B	C
m0	0	0	0
m1	0	1	1
m2	1	0	1
m3	1	1	1

Puede expresarse también como la suma de los minitérminos de las filas 1, 2 y 3

$$C = \sum_{AB} m(1,2,3) = (\bar{A}.B) + (A.\bar{B}) + (A.B)$$

En algunas notaciones la **m** de minitermino es opcional, porque la suma implica usar miniterminos. Tambien la indicacion de las variables en juego luego de la suma puede no figurar

para la compuerta NAND

	A	B	C
m0	0	0	1
m1	0	1	1
m2	1	0	1
m3	1	1	0

Puede expresarse como

$$C = \sum_{AB} m(0,1,2) = (\overline{A}.\overline{B}) + (\overline{A}.B) + (A.\overline{B})$$

Por último, por complementariedad con la suma de minitérminos o suma de productos, es posible plantear la expresión lógica a partir de la tabla de verdad como un **producto de maxitérminos o producto de sumas**. En este caso:

- Generar un maxitérmino por cada fila de la tabla en la que aparezca un 0 en la salida
- El maxitérmino (M..) es una suma (or inclusivo) que contiene cada variable de entrada en orden, **negada si está en 1**, **y no negada si está en 0**
- La expresión final es el producto (and) de estos maxitérminos

Así, la tabla de verdad de la compuerta OR

	A	B	C
M0	0	0	0
M1	0	1	1
M2	1	0	1
M3	1	1	1

Puede expresarse también como la producto del maxitérmino de la fila 0, resultando en la expresión real mínima de la compuerta

$$C = \prod_{AB} M(0) = (A + B)$$

Y la compuerta NAND

	A	B	C
M0	0	0	1
M1	0	1	1
M2	1	0	1
M3	1	1	0

Puede expresarse como

$$C = \prod M(3) = (\overline{A} + \overline{B})$$

Aplicando la ley de De Morgan, queda la expresión de la compuerta NAND

$$C = \prod M(3) = (\overline{A} + \overline{B}) = \overline{(A.B)}$$

Las expresiones como suma de minitérminos o producto de maxitérminos constituyen las alternativas de formas canónicas para la expresión lógica.

Se llama **lógica combinacional** a la lógica de sistemas en los cuales la/las salida/s dependen del estado actual de las entradas, sin importar los estados previos del sistema. En este punto se cuenta con tres formas de expresar un sistema discreto combinacional:

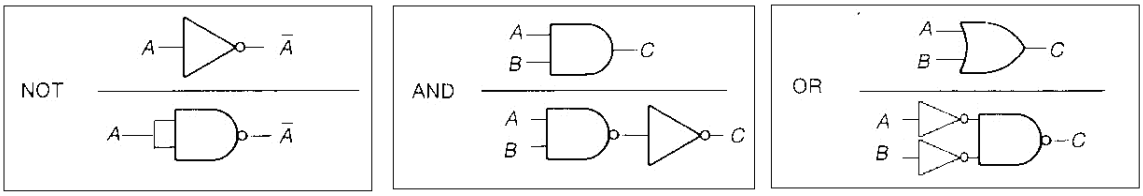
- **Tabla de verdad**
- **Diagrama lógico de compuertas conectadas**
- **Expresión booleana o ecuación de Boole**

En algunos casos es conveniente construir circuitos con una compuerta lógica de un tipo específico, dependiendo de la tecnología y del número de compuertas disponibles. Para esto, es posible buscar analogías entre estas compuertas, aprovechando los teoremas del álgebra booleana.

Construcción únicamente con compuertas NAND

- Si a una compuerta NAND se conecta la misma señal A en sus dos terminales, cuando A sea 1, los dos terminales verán un 1, de modo que la salida será 0. Asimismo, cuando A sea 0, los dos terminales de la compuerta NAND verán un 0, de modo que la salida será un 1. Se tiene entonces un inversor construido a partir de una compuerta NAND
- Por otro lado, una compuerta NAND seguida de un NOT es una compuerta AND.
- Con la Ley de De Morgan, es posible sintetizar una compuerta OR como una AND negada, negando sus entradas.

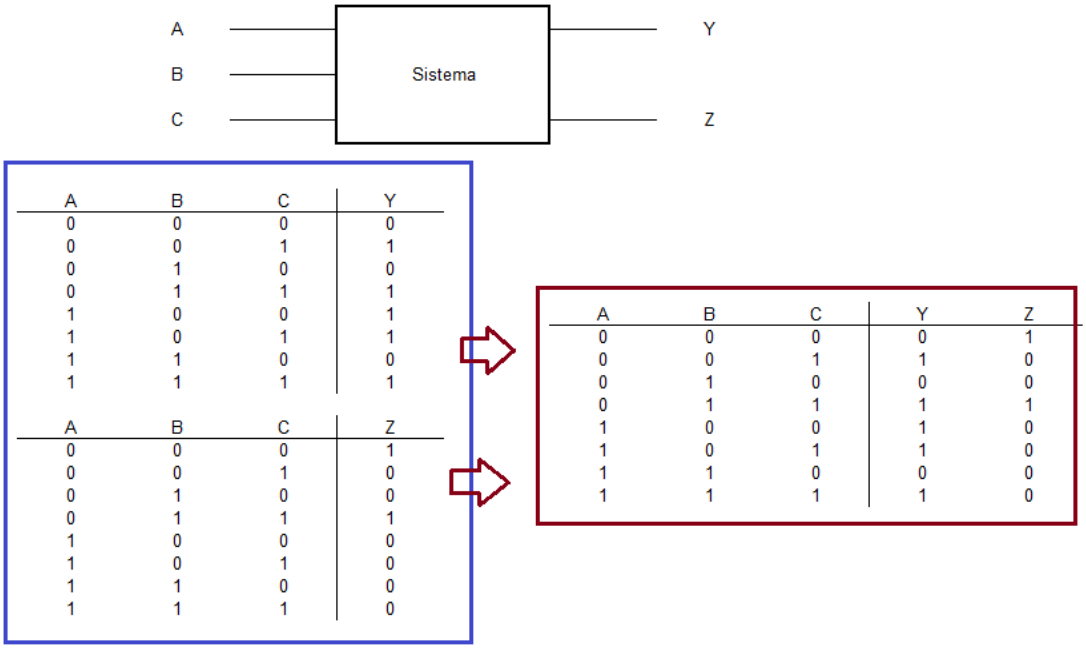
De esta forma, se ha reemplazado a las tres funciones básicas booleanas con alternativas que pueden lograrse con compuertas NAND. ES POSIBLE HACER LO MISMO CON COMPUERTAS NOR



Con manipulaciones similares, se puede utilizar compuertas NOR para sintetizar las demás compuertas

Sistemas con más de una salida

Cuando un sistema de lógica combinacional tiene más de una salida afectada por las mismas variables, es posible utilizar una tabla de verdad con cada salida, o combinarlas en una sola tabla. A partir de allí, se obtienen dos expresiones lógicas separadas. El siguiente es un ejemplo para un sistema desconocido, con 3 entradas (A, B y C) y dos salidas (Y, Z), con sus tablas de verdad, combinadas o separadas.



A	B	C	Y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

A	B	C	Z
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

A	B	C	Y	Z
0	0	0	0	1
0	0	1	1	0
0	1	0	0	0
0	1	1	1	1
1	0	0	1	0
1	0	1	1	0
1	1	0	0	0
1	1	1	1	0

Notar que se mantiene el orden binario ascendente en la tabla de verdad, que con tres entradas pasa a tener 8 filas (2^3).

La extracción de la expresión lógica se puede realizar de igual forma, como suma de minitérminos o producto de maxitérminos, o a partir de un análisis diferente de la tabla de verdad, y quedarán dos expresiones lógicas, pero esta representacion podria revelar complementariedades entre las funciones de salida:

Y = f(A,B,C)

Z = f(A,B,C)

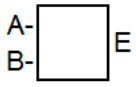
Concepto de simplificación de funciones lógicas.

Las expresiones booleanas no son únicas, de modo que encontrar la mejor o más simple puede resultar de interés. Si bien las virtudes de un circuito combinacional no se miden únicamente con su número de compuertas (el tipo de compuertas elegidas también influye entre otros factores), es un buen paso reducir el número de compuertas requeridas para implementar el circuito combinacional deseado a un mínimo. La forma más común de expresar las funciones lógicas es la suma de minitérminos. A partir de esta expresión, es posible realizar una simplificación algebraica, implementando teoremas o identidades booleanas. Si bien este método puede dar resultados, no siempre se pueden implementar métodos directos de simplificación para optimizar la expresión.

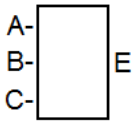
Simplificación de funciones lógicas mediante mapas de Karnaugh.

El mapa de Karnaugh es un método gráfico para representar la información presente en la tabla de verdad. A partir de esta representación, es posible implementar estrategias para obtener las expresiones mínimas lógicas de un sistema combinacional.

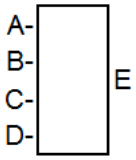
Se trata de una reestructuración de la tabla de verdad, en forma de una rejilla, en la cual cada caja interna representa unas combinaciones de valores 1 o 0 entre las entradas del sistema. El valor que se coloca dentro de esa caja (1 o 0) es el nivel de la salida correspondiente a esa combinación de entradas (dada por el lugar de la caja en la rejilla). Ejemplos de mapas de Karnaugh para 2, 3 y 4 variables se muestran a continuación:



	E	A=0	A=1
B	0	0	1
	1	0	0



	E	AB=00	AB=01	AB=11	AB=10
C	0	0	1	1	0
	1	0	0	1	0



	E	AB=00	AB=01	AB=11	AB=10
CD	00	1	0	1	1
	01	1	0	1	1
	11	0	0	1	0
	10	0	0	1	1

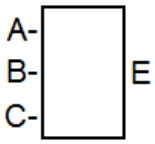
- Debe notarse que **la secuencia de valores posibles para las entradas que se ponen de a dos (AB, CD) no es binaria ascendente, sino que es una secuencia en la cual el par cambia un bit por vez**, pasando en secuencia 00-01-11-10… Esta secuencia se denomina Código Gray. ***Los extremos visibles del mapa no son límites, es posible recorrerlo entre los extremos***, pasando del extremo izquierdo al derecho, del superior al inferior, etc..

Si cada celda representa el valor de la salida E para las combinaciones de las entradas, **existe una equivalencia entre cada celda del mapa y cada fila de la columna**

de salida de la tabla de verdad. Así, para cada celda que se encuentre en 1, es posible escribir un minitérmino. En el caso del primer mapa de la figura, con 2 entradas y una salida, se puede decir que $E = A.\overline{B}$, porque la única celda que tiene un 1 corresponde a A en 1 y B en 0.

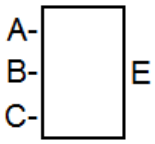
Sin embargo, la potencialidad del mapa se observa cuando se combinan celdas en 1 adyacentes, de modo que se pueden agrupar en rectángulos o cuadrados, aún a través de los extremos (no en diagonal).

Ejemplo



	E	AB 00	01	11	10
C	0	0	1	1	0
	1	0	0	1	0

Por ejemplo, en este caso se pueden encontrar tres minitérminos, pero el hecho de que los dos 1 en la fila superior corresponden a



	E	AB 00	01	11	10
C	0	0	1	1	0
	1	0	0	1	0

C=0 y A=0 y B=1
C=0 y A=1 y B=1

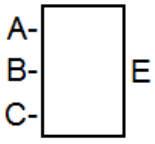
indica que sería posible ver que el valor 1 de la salida para estos dos casos es independiente de A, y solo basta con que

C=0 y B=1, para E=1

De este modo, se obtiene un término reducido automáticamente

$\overline{C}.B$, una expresión menor a los dos minitérminos que contenían información dependiente de A.

De manera similar, para los dos unos en la tercera columna (AB=11), se puede demostrar que el valor de la salida E depende de



	E	AB 00	01	11	10
C	0	0	1	1	0
	1	0	0	1	0

C=0 y A=1 y B=1
C=1 y A=1 y B=1

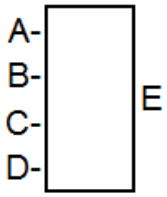
De modo que estos dos minitérminos aislados implican que E será 1 cuando A y B sean 1, independientemente del valor de C. Se obtiene un segundo término $A.B$

En conjunto, la expresión reducida de E queda como la suma de los dos términos reducidos obtenidos

$$E = \overline{C}.B + A.B$$

Agrupamientos de 2, 4, 8 y hasta 16 celdas son posibles para 4 entradas. Cuanto mayor sea el número de celdas agrupadas, menor será la expresión del término correspondiente (requerirá menos entradas para definirlo). Cuando se pueden agrupar las celdas en 1, se obtienen términos menores porque se deshace de condiciones que en verdad no importan. **Es necesario notar que los extremos del mapa no son límites, es posible cruzarlos.**

Ejemplo:



E / CD	AB = 00	AB = 01	AB = 11	AB = 10
CD = 00	1	0	1	1
CD = 01	1	0	1	1
CD = 11	0	0	1	0
CD = 10	0	0	1	1

Los recuadros en verde representan un agrupamiento de 4 celdas, que corresponden a las condiciones

C=0, D=0, A=0 y B=0
C=0, D=1, A=0 y B=0
C=0, D=0, A=1 y B=0
C=0, D=1, A=1 y B=0

Esto quiere decir que la salida E será 1, en este caso, cuando C y B sean 0, independientemente de A y D. El término correspondiente a este agrupamiento mayor es, entonces, $\overline{C}.\overline{B}$

El rectángulo azul agrupa 4 celdas, que están en 1 cuando A y B son 1, independientemente de C y D (no importa C y D si valen 0 o 1, la salida es 1 en este agrupamiento). El término correspondiente a este recuadro es $A.B$

Por último, el recuadro en rojo corresponde a la condición en que C es 1, D es 0 y A es 1, independientemente de B. El último término es entonces $A.C.\overline{D}$

La expresión final, mínima para este mapa, es la suma de las tres obtenidas de los agrupamientos

$$E = \overline{C}.\overline{B} + A.B + A.C.\overline{D}$$

- Una **suma minima** de una funcion logica es una expresion de suma de productos para F de modo que ninguna otra expresion de suma de productos para F tiene menos terminos de producto y que cualquier expresion de suma de productos con el mismo numero de terminos de producto tiene al menos tantas literales
- Se llama **Implicantes primos "implicando"** lógicamente a la función completa, de modo que si el implicante primo expresado como funcion logica P vale 1, la funcion total F tambien vale 1 para toda combinacion de entradas en P. En este caso F **incluye o cubre** a P

En terminos de Karnaugh ***el implicante primo de una funcion logica F*** es un termino de producto normal que implica a F, de modo que si cualquier variable es removida de P, entonces el termino de producto no implica F, o sea **es un conjunto encerrado de celdas en 1 que satisfacen las reglas de combinacion, tal que si se intenta hacerlo mas grande de acuedo con esas reglas se cubre uno o mas ceros**.

Las dimensiones de los implicantes válidos son 2^n x 2^m, con n y m mayor o igual a 0

Teorema del implicante primo: Una suma minima es una suma de implicantes primos

Es posible que no todos los implicantes primos sean necesarios, en la expresion minima, si son en efecto cubiertos por otros implicantes primos

Una **celda 1 distinguida** es una celda del mapa que esta en 1 y es cubierta por un solo implicante primo

Un **implicante primo esencial o termino esencial** es un implicante primo que cubre una o mas celdas distinguidas

Implicante primo esencial o Término esencial es un implicante primo que tiene uno o más 1 que no están en otro implicante – es necesario incluirlo para lograr la expresión de la función lógica.

Puede haber más de una manera de unir las celdas. Se recomienda

- Primero construir implicante más grande posible
- Agregar grupos cada vez más pequeños, hasta incluir cada celda que tenga 1
- Eliminar grupos redundantes para evitar duplicación

Una forma ordenada de buscar la suma minima consiste en:

1) Identificar las celdas 1 distinguidas, e incluir en la suma minima sus correspondientes implicantes primos esenciales

2) Si no todas las celdas 1 del diagrama son cubiertas por los implicantes primos esenciales, **armar un mapa reducido sin las celdas 1 cubiertas por los implicantes primos esenciales**

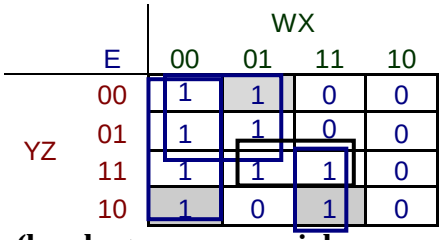
3) Para incorporar estas celdas 1 en la suma minima definitiva, encontrar los implicantes primos mas "economicos" que cubran estas celdas 1 hasta cubrir todas las celdas 1 del diagrama

Esto lleva a estrategias alternativas de solucion, como se muestra en los siguientes casos:

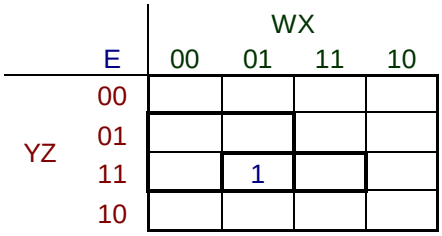
Caso a)

E	WX 00	01	11	10
YZ 00	1	1	0	0
01	1	1	0	0
11	1	1	1	0
10	1	0	1	0

a) Mapa de Karnaugh



b) Implicantes primos (borde grueso, esenciales en azul) y celdas 1 distinguidas (sobreado gris)

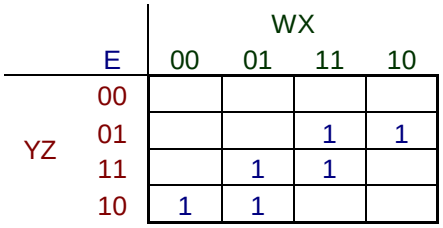


c) Mapa reducido despues de la eliminacion de implicantes primos esenciales y celdas 1 cubiertas

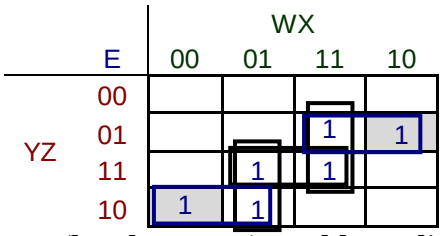
Al remover implicantes primos esenciales, queda una sola celda 1 en el mapa reducido, y dos implicantes primos (considerar todos los 1 que habia en el mapa original) que la cubren. En este caso, la opcion mas economica es elegir el termino mas pequeño, correspondiente al recuadro mas grande ($\overline{W}Z$)

Otro Caso

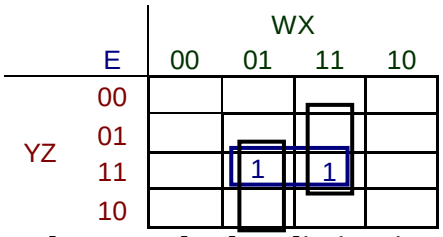
Se dice que dados dos implicantes primos P y Q, en un mapa reducido, **P eclipsa a Q** si P cubre por lo menos todas las celdas 1 cubiertas por Q, de modo que eliminar a Q de la expresion no impide hacer una suma minima



a) Mapa de Karnaugh



b) Implicantes primos (borde grueso) y celdas 1 distinguidas (sobreado gris)



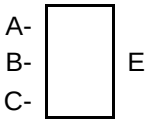
c) Mapa reducido despues de la eliminacion de implicantes primos esenciales y celdas 1 cubiertas

Se muestran todos los 3 implicantes que cubren las dos celdas 1 restantes (dos cubren cada uno una celda 1 de un implicante primo esencial ya considerada, y el tercero XYZ cubre las dos celdas 1 que quedan, de modo que, en lo que hace a este diagrama reducido, XYZ eclipsa los otros dos implicantes primos y los vuelve innecesarios. XYZ se denomina ***implicante primo esencial secundario***, que debe incluirse en la suma minima

Con más de cuatro variables el mapa de Karnaugh se vuelve difícil de manejar visualmente. En estos casos se implementan soluciones automáticas que implementan estas estrategias de minimización a muchas entradas.

Condición irrelevante

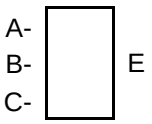
En algunos sistemas, no hace falta definir la salida para todas las combinaciones posibles de entradas. Es posible que nunca pueda ocurrir en la implementación física del sistema que dos de sus entradas se encuentren en 1 al mismo tiempo, por ejemplo. El hecho de que estas condiciones sean irrelevantes no quiere decir que se las deba ignorar. Ya que se entiende que nunca ocurrirán, o que no importa si ocurren, se las puede aprovechar para aumentar el número y tamaño de los implicantes en el mapa de Karnaugh. Las condiciones irrelevantes o redundancias se expresan con una X, queriendo decir que es indistinto si valen 0 o 1.



	E	AB 00	01	11	10
C	0	0	1	1	0
	1	X	X	1	0

En este caso, hay dos condiciones irrelevantes. Es indistinto si la salida vale 0 o 1 cuando ABC = 001 y ABC=011, porque se asume que eso nunca va a ocurrir o porque en ese caso da igual el valor que tome la salida

Para minimizar la expresión lógica de este circuito secuencial se prefiere obtener el menor número de terminos que sea posible. Si se asume un 1 en la condición irrelevante de la segunda columna (ABC=011), se puede obtener un implicante de 2x2. Sin embargo, agregar un 1 en la otra condición irrelevante no aportaría utilidad, ya que implicaría sumar otro término más. La estrategia que debe implementarse es aprovechar las redundancias, dándoles valores de 1 o 0 según convenga. Para este caso, según lo elegido quedará



	E	AB 00	01	11	10
C	0	0	1	1	0
	1	0	1	1	0

y la expresión lógica del sistema es E = B, independiente de A y C

Las redundancias pueden ser expresadas como lo son las sumas de minitérminos que constituyen la salida. Para un circuito lógico con 3 entradas (8 estados posibles), y redundancias en las combinaciones m3 y m6, la expresión como suma de minitérminos y de redundancias es:

A	B	C	Y
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	X
1	0	0	0
1	0	1	0
1	1	0	X
1	1	1	1

$$Y = \sum_{ABC} m(1,2,7)$$

$$R = \sum_{ABC} m(3,6)$$

Tambien se puede expresar la redundancia como conjunto d (don't care)

$$Y = \sum_{ABC} m(1,2,7) + d(3,6)$$

Riesgos-Condición de carrera

Las compuertas lógicas no actualizan su salida en forma instantánea: tardan en responder a sus entradas. Si bien normalmente se consideran los valores estables de las señales en lógica combinacional, cuando existen cambios en los niveles lógicos de las señales, estos retardos pueden producir errores temporarios, o glitches en los sistemas, con consecuencias variables (que pueden no impactar al sistema en absoluto o hacerlo fallar de manera inaceptable).

El caso de las figuras (a) y (b) es un circuito lógico donde $C = A.\overline{A} = 0$. Sin embargo, como la entrada negada tarda más en actualizarse que la no negada, durante un corto tiempo la salida pasa a 1. Se llama **profundidad lógica**, al número máximo de compuertas sencillas a través de las cuales pasa una señal entre la entrada y la salida.

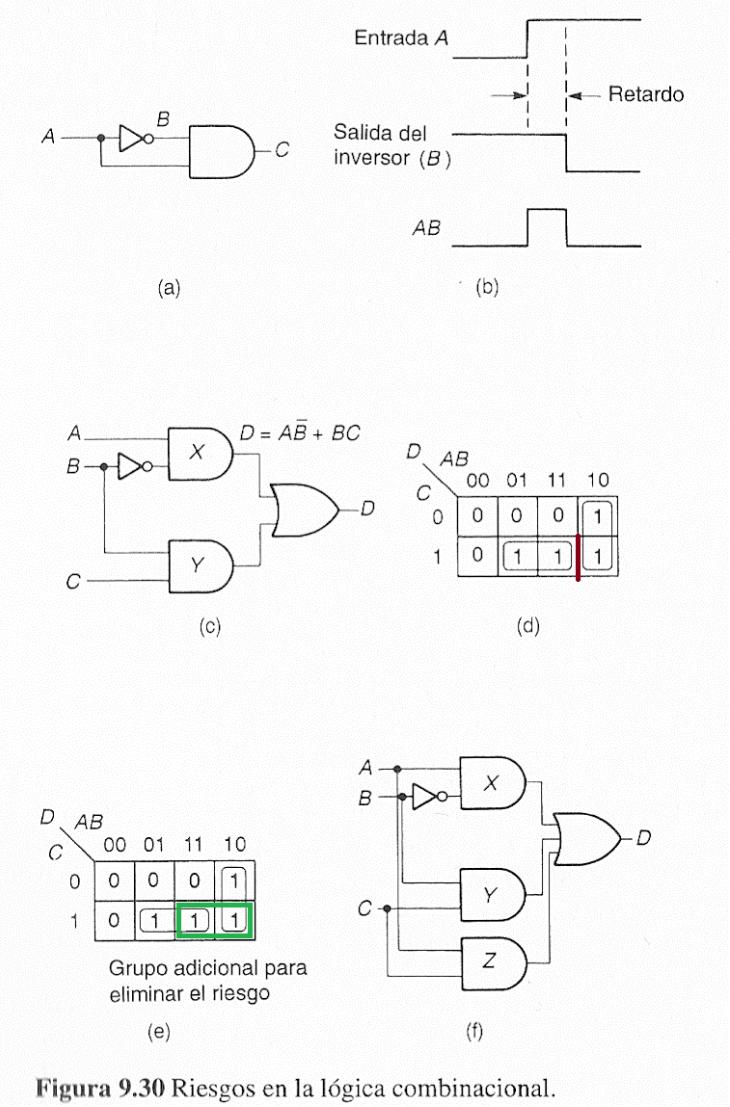


Figura 9.30 Riesgos en la lógica combinacional.

Es posible agregar compuertas extra para resolver estas situaciones y equiparar los retardos de todos los caminos. Estas compuertas serían redundantes pero no cambiarían la función lógica del sistema.

Bajo el mismo concepto, es posible visualizar estos riesgos en el mapa de Karnaugh, llamados **condiciones de carrera**, cuando algún cambio en una de las entradas permite saltar entre implicantes adyacentes pero separados del mapa de Karnaugh (los implicantes a considerar son los elegidos para expresar la función).

E	AB 00	01	11	10
CD 00	1	0	1	1
01	1	0	1	1
11	0	0	1	0
10	0	0	1	0

Los dos implicantes elegidos para obtener la expresión mínima del circuito lógico son adyacentes, y se puede pasar de uno al otro con un cambio en el valor de B, cuando C es 0 y A es 1. Esto implica un riesgo de carrera cuando se pase del implicante verde al azul.

E	AB 00	01	11	10
CD 00	1	0	1	1
01	1	0	1	1
11	0	0	1	0
10	0	0	1	0

La forma de evitar el efecto de este salto es agregar un término más (redundante) a la expresión mínima, que conecte las regiones separadas pero adyacentes. Cuando hay más de un salto posible, se agregan más términos.

Los tres términos resultantes serían la expresion no minima

$E = \overline{C}.\overline{B} + A.B + A.\overline{C}$, y el último implicante es redundante pero protege contra la condición de carrera, mientras se mantiene la misma función lógica del sistema.

Representación binaria

Las magnitudes que se ponen en juego en los circuitos discretos mencionados son binarias. Los números binarios se construyen de manera similar a los decimales, salvo por que su base son potencias de 2

$19 = \mathbf{1}x10^1 + \mathbf{9} \times 10^0$

$130 = \mathbf{1}x10^2 + \mathbf{3}x10^1 + \mathbf{0}x10^0$

La construcción de los mismos números en binario se realiza con potencias de 2.

$19 = \mathbf{1}x2^4+\mathbf{0}x2^3+\mathbf{0}x2^2+\mathbf{1}x2^1+\mathbf{1}x2^0 = 16+2+1=19$

La representación binaria de 19 es, entonces, el número binario **10011**

De manera similar, la representación en binario del número 130 es **10000010**

También se pueden representar partes fraccionarias, con las potencias negativas de 2, separadas por un punto binario. La representacion digital de enteros con signo en binario no se realiza igual que en decimal, en cambio. En cualquier caso, cada número en estas representaciones es un dígito binario, o **bit**. Entonces, un número binario de 8 dígitos es un número de 8 bits. Para números enteros, el bit de la extrema derecha corresponde a la potencia 0 de 2, es el **bit menos significativo**. El **bit más significativo** es el de mayor orden, a la extrema izquierda.

Aplicaciones de lógica combinacional

Sumadores binarios

Trabajar con representación binaria simplifica la automatización de operaciones aritméticas. De manera similar a la suma decimal, una suma binaria involucra sumar bit a bit, y llevar un acarreo al bit de orden inmediato superior si los dos elementos a sumar valen más de 1 (como cuando la suma de dígitos vale 10 o más en la suma decimal).

- *Semisumador*

Un semisumador calcula la suma de 2 números binarios, A y B, de 1 bit. La salida en una suma es el resultado o suma (S) y el acarreo al siguiente digito (Carry, C).

A	B	Suma decimal		Suma binaria	
0	0	0	>>	0	0
0	1	1	>>	0	1
1	0	1	>>	0	1
1	1	2	>>	1	0

La salida de la suma de dos números que pueden ser 0 o 1 es 0, 1 o 2. Cuando la salida es 2, hay un acarreo de 1 bit. La tabla de verdad para esta relación entrada-salida es:

A	B	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Es importante analizar si existe diferencia entra la profundidad lógica de una salida y de la otra. Con esta tabla de verdad, C=A and B, y S= A xor B. La compuerta XOR no es simple, se construye con otras compuertas, aumentando la profundidad lógica de S respecto de la de C, de modo que se S tardará más en actualizarse que C, pudiendo causar fallas o glitches.

Sumador completo (considera carry del anterior)

Un sumador completo es un semisumador con una entrada más, que considera si este sumador recibe un acarreo previo. Con esta estructura es posible crear un sumador de N bits, con un semisumador en el bit menos significativo (no recibe acarreo de nadie), y N-1 sumadores completos para los bits de orden superior. Se muestra un diagrama de conexiones de un sumador de 4 bits, y la tabla de verdad y los mapas de Karnaugh de las dos salidas de un sumador completo (C y S, como en el semisumador)

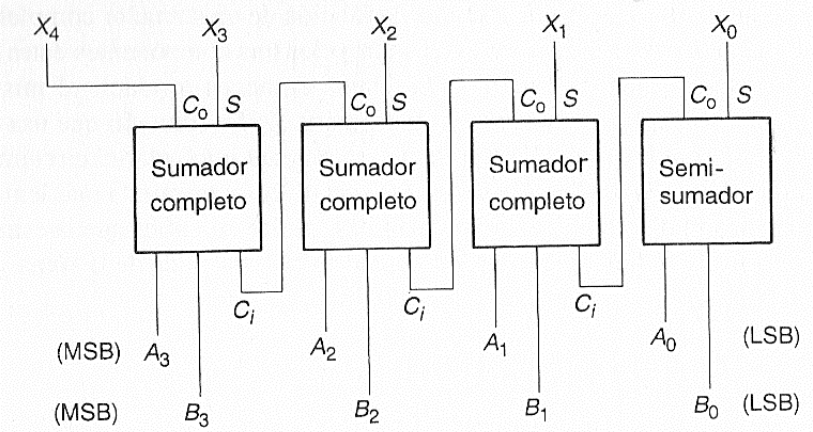
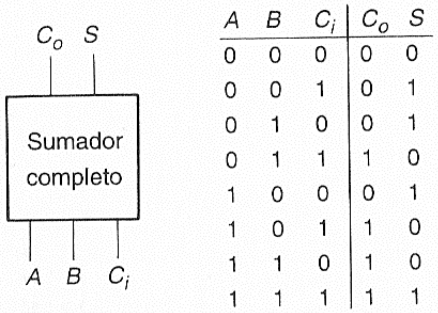


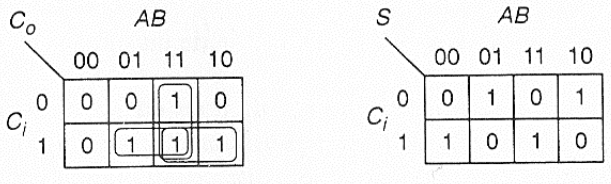
Figura 9.33 Circuito para sumar dos números de 4 bits.



A	B	C_i	C_o	S
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

(a) (b)

Figura 9.34 Sumador completo.



De forma similar se puede desarrollar un restador, y otras operaciones aritméticas binarias, aunque el producto y la división requieren gran número de compuertas.

Codificador-Decodificador

El método más común para representar información numérica en sistemas digitales es el sistema binario mencionado. Sin embargo existen otras representaciones de codificación de números en valores binarios.

- **Código Gray**

El código Gray es una representación alternativa de números, de modo que la diferencia entre números adyacentes sea de un bit.

Numero	Binario			Codigo Gray		
0	0	0	0	0	0	0
1	0	0	1	0	0	1
2	0	1	0	0	1	1
3	0	1	1	0	1	0
4	1	0	0	1	1	0
5	1	0	1	1	1	1
6	1	1	0	1	0	1
7	1	1	1	1	0	0

El cambio de un bit entre números adyacentes no es trivial, mejora la respuesta de algunos sistemas. Si debe cambiar más de un bit, por pasar de 3 a 4 en binario, por ejemplo, y estas dos salidas no cambian al mismo tiempo, se produce un salto corto a otro número, por ejemplo pasar de 011 → 111→ 101 → hasta llegar a 100. El código Gray resuelve este problema, y es implementado en contadores asíncronos y en codificadores de posición absoluta.

En la tabla anterior, se puede crear un codificador de código Gray, con 3 entradas en binario, y 3 salidas, que respondan según la tabla de verdad de arriba. Obteniendo la expresión de las 3 salidas individualmente, se puede construir un codificador Gray de 3 bits.

- Binary Coded Digit (BCD)

Otra forma de representar números decimales en bits se puede obtener convirtiendo cada dígito decimal de un número decimal a binario de manera individual. Así, el número 9450 en decimal (se expresa 9450_{10}) puede convertirse a BCD con la siguiente relación:

$$9450_{10} = (1001)(0100)(0101)(0000)_{BCD}$$

9 4 5 0

Esto no es igual a la representacion binaria del número completo

Como los dígitos decimales pueden valer entre 0 y 9, se utiliza un número fijo de 4 bits para representarlos individualmente. Estos 4 bits permiten representar 2^4=16 valores diferentes. Los valores correspondientes a 10, 11, 12, 13, 14 y 15 no se utilizan, y al utilizar circuitos que utilicen este código, pueden utilizarse como condiciones irrelevantes.

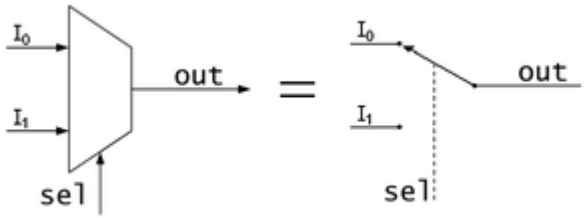
Por ejemplo, un sistema que reciba un número BCD de un dígito, y que identifique los impares, puede tener la siguiente tabla de verdad. 4 entradas, una salida.

	BCD				
Numero	A	B	C	D	Salida
0	0	0	0	0	0
1	0	0	0	1	1
2	0	0	1	0	0
3	0	0	1	1	1
4	0	1	0	0	0
5	0	1	0	1	1
6	0	1	1	0	0
7	0	1	1	1	1
8	1	0	0	0	0
9	1	0	0	1	1
10	1	0	1	0	X
11	1	0	1	1	X
12	1	1	0	0	X
13	1	1	0	1	X
14	1	1	1	0	X
15	1	1	1	1	X

Estas condiciones irrelevantes podrían servir para completar el mapa de Karnaugh y simplificar la expresión final del circuito, ya que las entradas BCD nunca pueden corresponder a las últimas 6 combinaciones.

Multiplexor de 2 canales

El equivalente de un multiplexor es una llave de muchos terminales, que permite elegir una entre más señales, a partir de señales de control.



Un multiplexor es un sistema que tiene una salida out y N entradas (I_0, I_1,…). Agregando entradas de control (S_1,S_2,…) se permite seleccionar una de las entradas.

Para el circuito de la figura, se espera que, cuando sel valga 0, la salida replique a I_0, y que cuando sel valga 1, la salida replique a I_1. La tabla de verdad para esta relación entrada-salida es la siguiente:

Sel	**I_0**	**I_1**	out
0	**0**	0	0
0	**0**	1	0
0	**1**	0	1
0	**1**	1	1
1	0	**0**	0
1	0	**1**	1
1	1	**0**	0
1	1	**1**	1

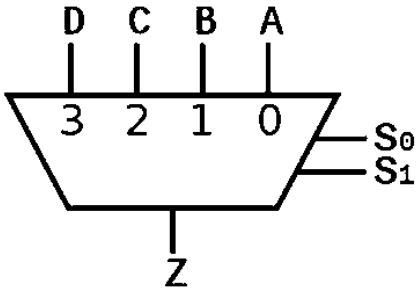
La expresión lógica puede obtenerse con el mapa de Karnaugh, o deducirse por el funcionamiento del sistema.

		I_0 I_1			
	out	00	01	11	10
S	0	0	0	1	1
	1	0	1	1	0

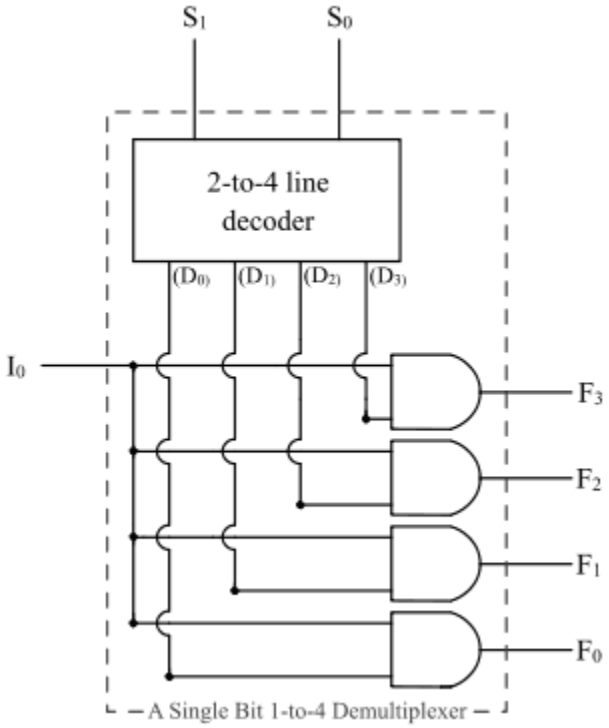
		I_0 I_1			
	out	00	01	11	10
S	0	0	0	1	1
	1	0	1	1	0

$$out = S.I_1 + \overline{S}.I_0$$

En esta combinación basta con una señal de control para elegir entre 2 señales. Para elegir una entre 4 señales basta con 2 señales de control (que en conjunto pueden tomar cuatro estados, 00, 01, 11, 10). En general basta con N señales de control para elegir entre 2^N entradas.



También existe la inversa, el demultiplexor, que tiene 1 entrada, N señales de control y 2^N salidas. Un ejemplo se muestra a continuación.



S_1 S_0	I_0	F_3 F_2 F_1 F_0
0 0	0	0 0 0 0
	1	0 0 0 1

S_1 S_0	I_0	F_3 F_2 F_1 F_0
0 1	0	0 0 0 0
	1	0 0 1 0

S_1 S_0	I_0	F_3 F_2 F_1 F_0
1 0	0	0 0 0 0
	1	0 1 0 0

S_1 S_0	I_0	F_3 F_2 F_1 F_0
1 1	0	0 0 0 0
	1	1 0 0 0