

Arquitectura de Computadoras

Unidad 0 Lógica Combinacional

ASTEC INTERNATIONAL

UM 1082 LA2/3

8225

FERRANTI
ULA 2C184E
8214

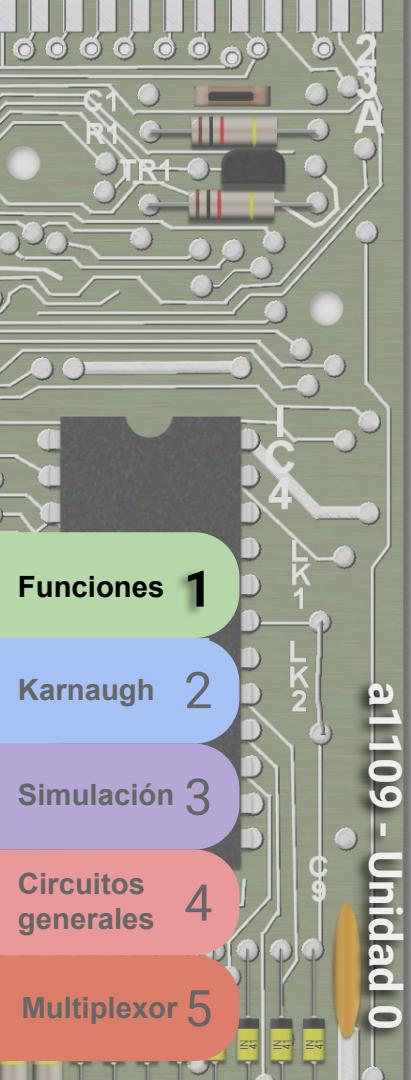
ZILLOG
Z8400A PS
Z80A CPU
8220

SINCLAIR
RESEARCH 8223Pg
D2364C 649 © 1981

Toshiba
TMM2016P
2-EE4

Edgardo Gho
Carlos Rodríguez

Lógica de Boole



Funciones 1

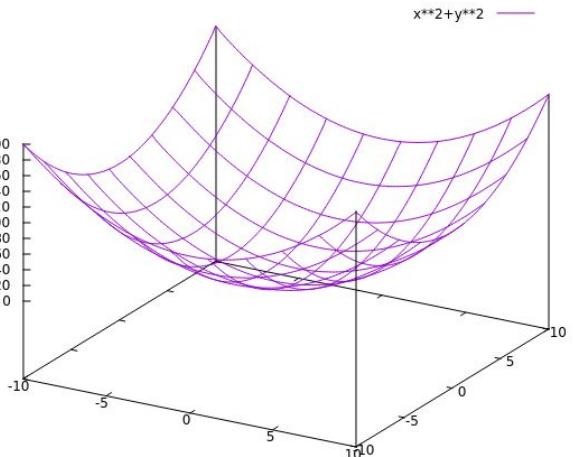
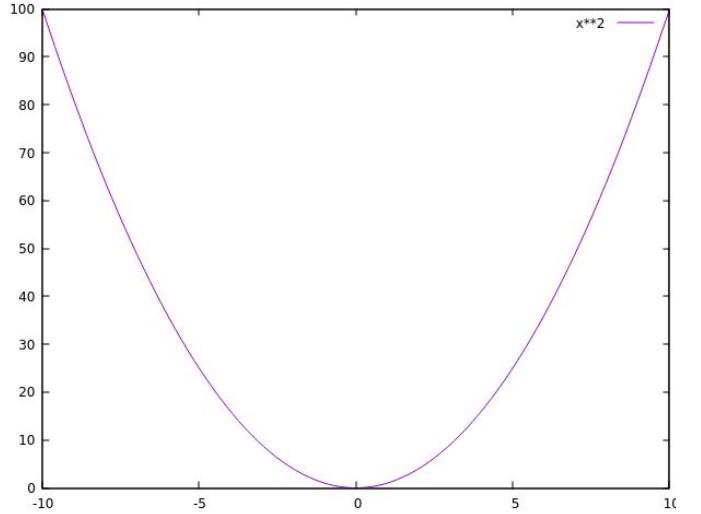
Karnaugh 2

Simulación 3

Circuitos generales 4

Multiplexor 5

En cálculo... funciones

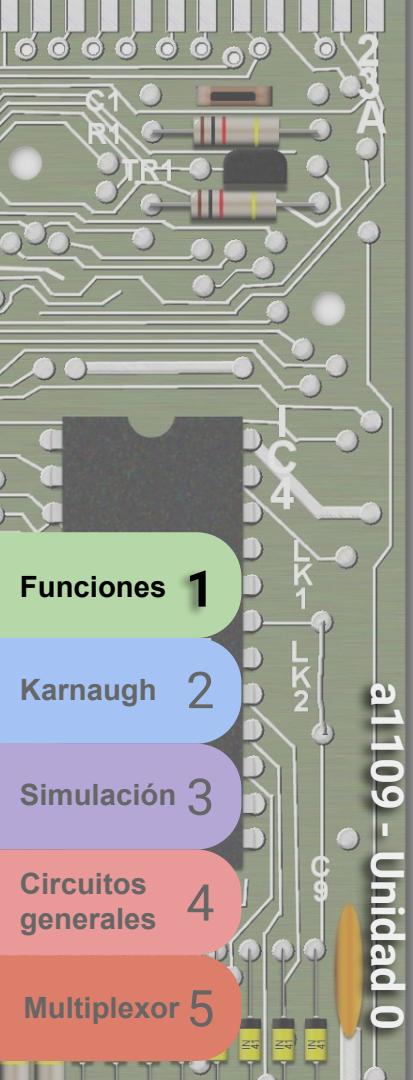


$$f : \mathbb{R} \rightarrow \mathbb{R}$$

$$f(x) = x^2$$

$$f : \mathbb{R}^2 \rightarrow \mathbb{R}$$

$$f(x, y) = x^2 + y^2$$

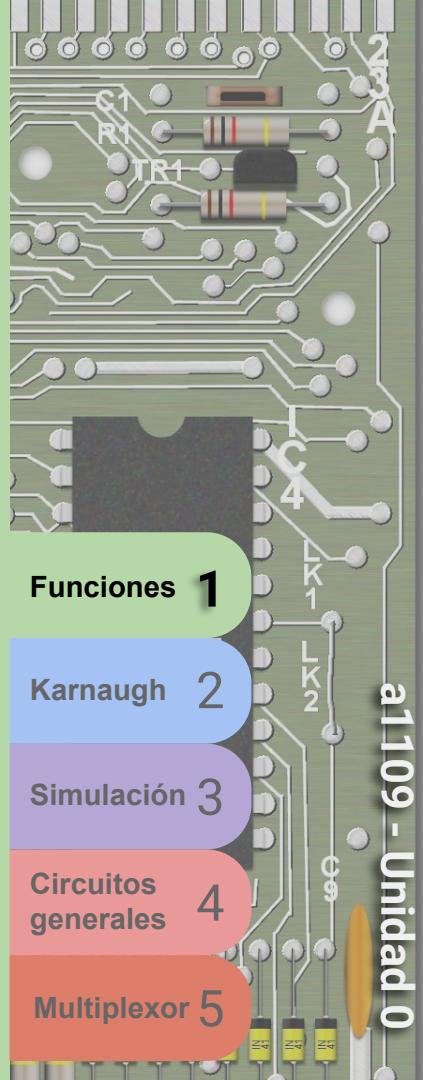
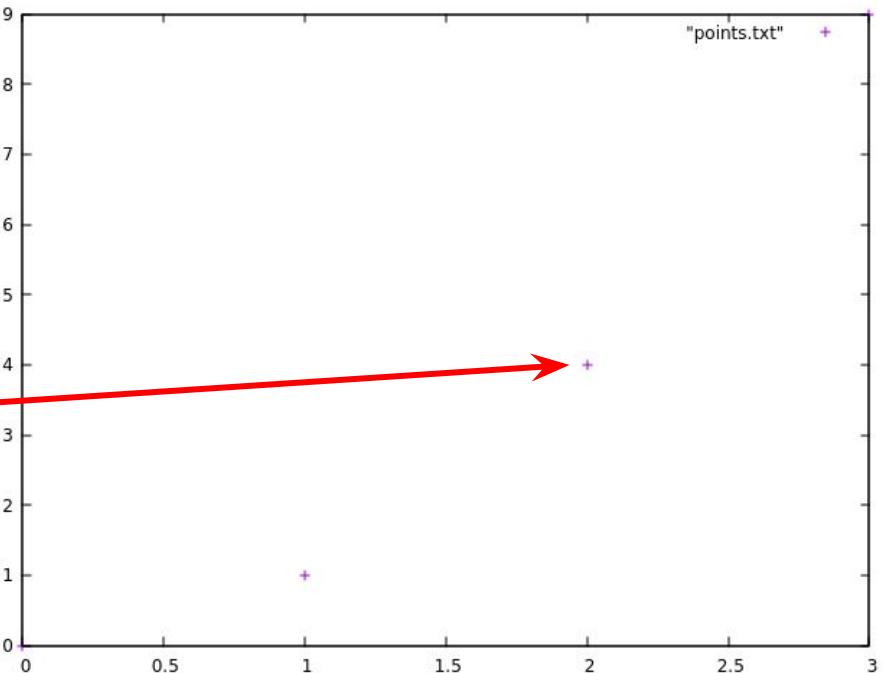


Funciones por extensión

$f : [0, 1, 2, 3] \rightarrow \mathbb{R}$

$$f(x) = x^2$$

x	f(x)
0	0
1	1
2	4
3	9

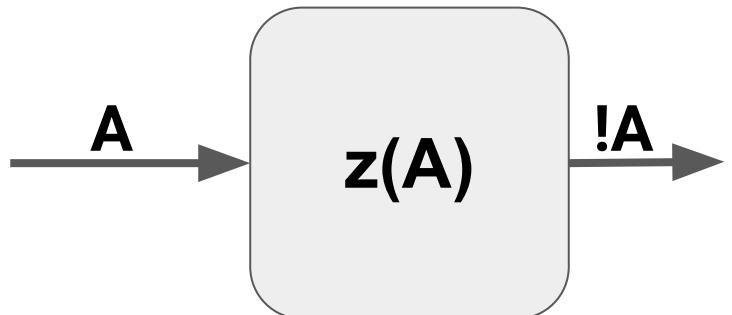


Funciones Booleanas de una variable

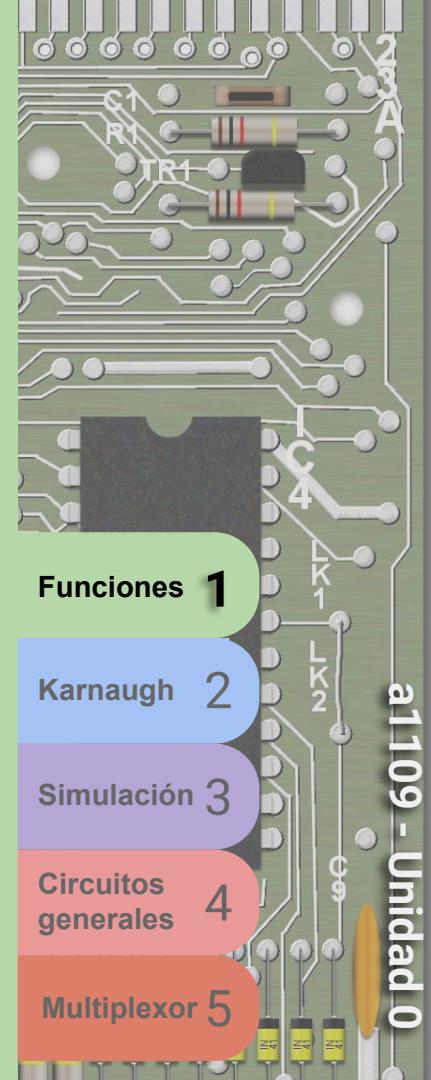
Definimos una función booleana sobre una variable a una función con dominio en los booleanos y codominio también en los booleanos. O sea la variable puede valer 0 , 1 y como resultado puede valer también 0 o 1.

$$z : B \rightarrow [0, 1]$$

$$z(A) = not(A) = \neg A = \sim A = \bar{A} = !A$$



A	!A
0	1
1	0



Funciones Booleanas de una variable

Para una variable, existen dos valores posibles de entrada. Eso quiere decir que existen $2^2 = 4$ posibles funciones booleanas cuando solo tenemos una variable de entrada.

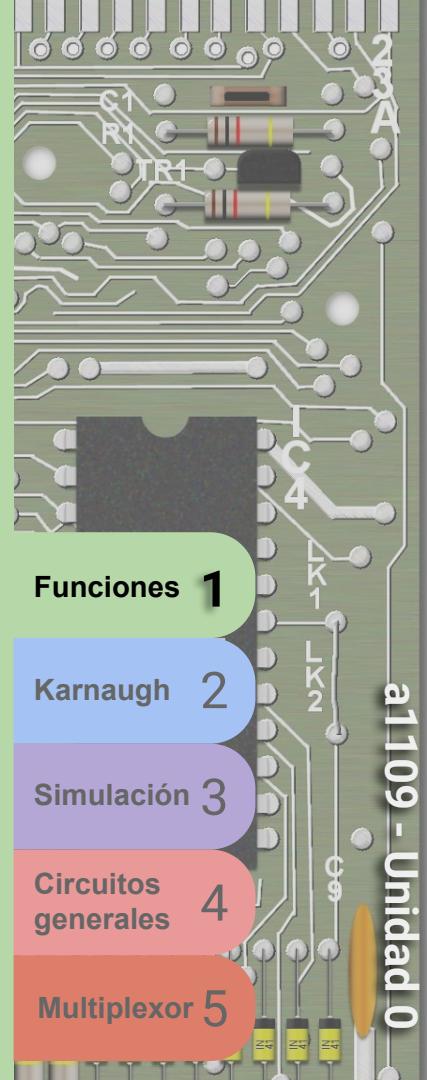
A	$\neg A$
0	1
1	0

A	A
0	0
1	1

A	0
0	0
1	0

A	1
0	1
1	1

Vemos que sólo la primera ($\neg A$) es realmente útil. La función $A=A$ es simplemente un cable. Luego las ultimas dos son constantes (0 y 1) que no dependen del valor de la variable, siempre generan el mismo valor a la salida.



Funciones Booleanas de dos variables

Definimos funciones booleanas de dos variables a aquellas que tienen como entrada dos variables booleanas y producen como resultado un valor booleano. Dos de ellas en particular son operaciones en el álgebra booleana. La suma booleana se define como OR y el producto booleano se define como AND.

$$y : B^2 \rightarrow [0, 1]$$

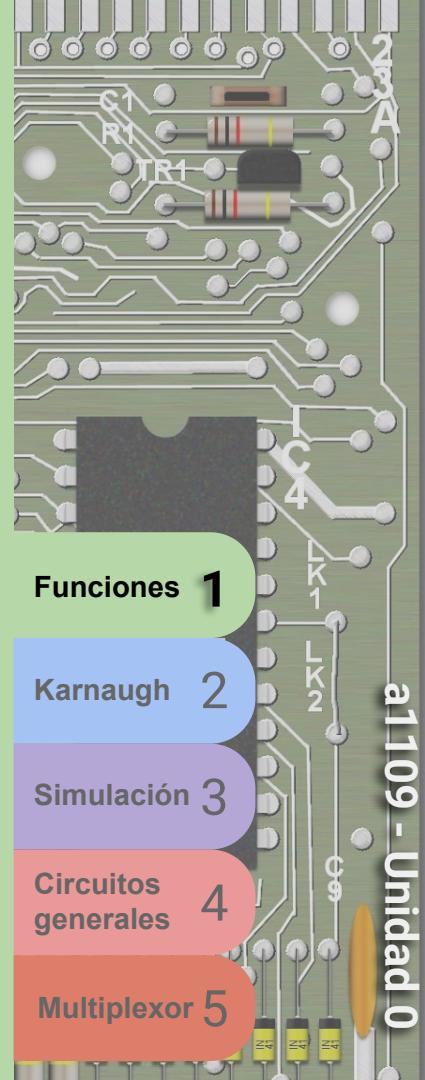
$$y(A, B) = A \cdot B = A \text{ and } B$$

A	B	And
0	0	0
0	1	0
1	0	0
1	1	1

$$y : B^2 \rightarrow [0, 1]$$

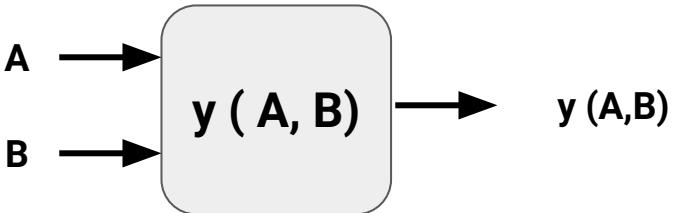
$$y(A, B) = A + B = A \text{ or } B$$

A	B	Or
0	0	0
0	1	1
1	0	1
1	1	1

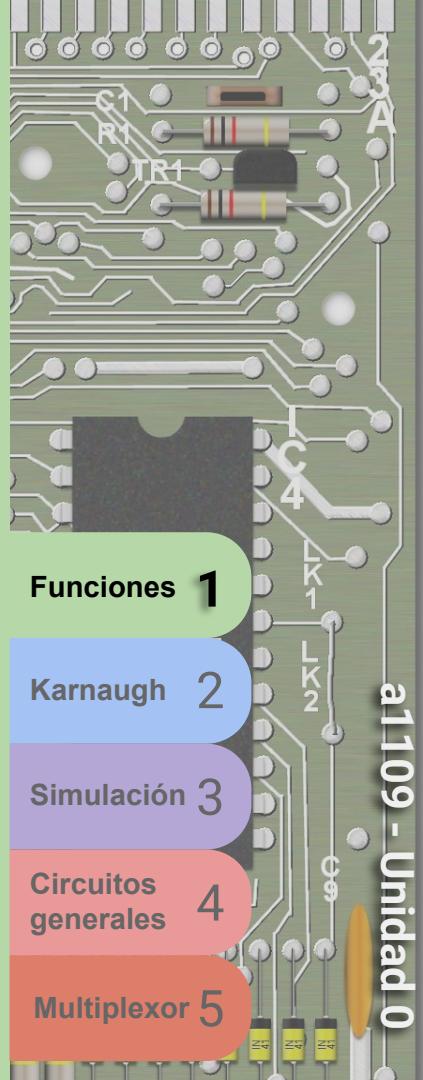
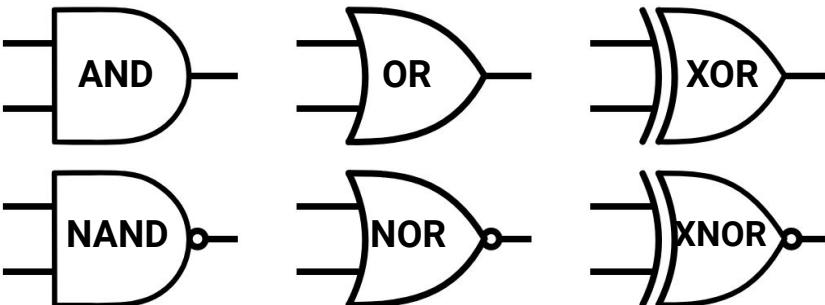


Funciones Booleanas de dos variables

En general, una función booleana de dos variables la podemos pensar como una caja negra, donde entran dos señales y sale una.



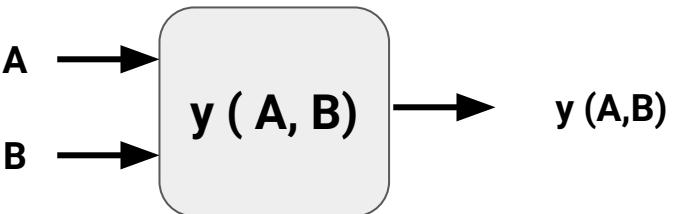
Alguna de estas funciones de dos variables son tan usadas que tienen nombres y dibujos particulares



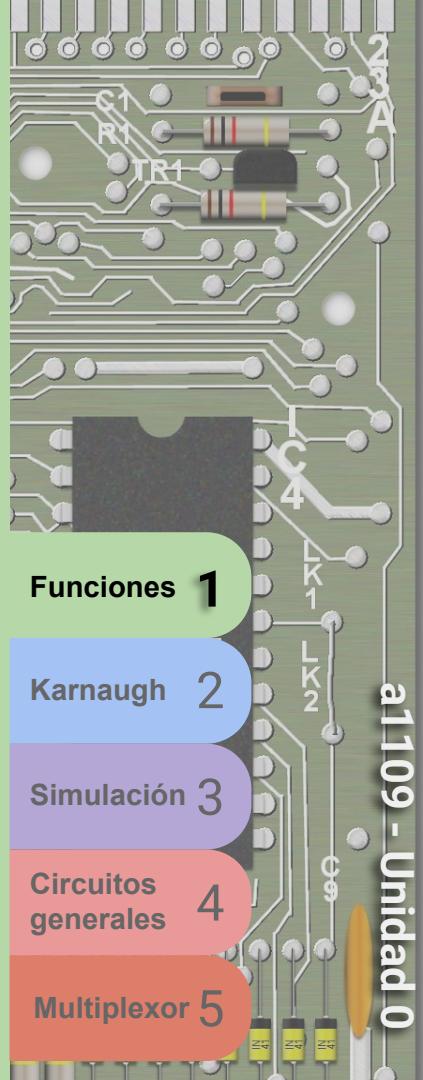
Funciones Booleanas de dos variables

Piense cuántas funciones de dos variables pueden existir. Si tenemos 2 entradas booleanas entonces una función booleana debe estar definida para 4 valores de entrada posible.

A	B	Y
0	0	?
0	1	?
1	0	?
1	1	?

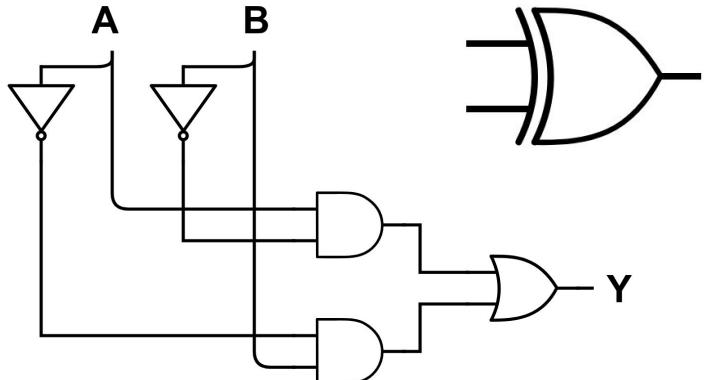


Esos 4 valores pueden tomar $2^4 = 16$ combinaciones distintas. O sea, existen 16 funciones booleanas de 2 variables. Algunas de ellas, se implementan físicamente en un circuito y las llamamos **compuertas**.



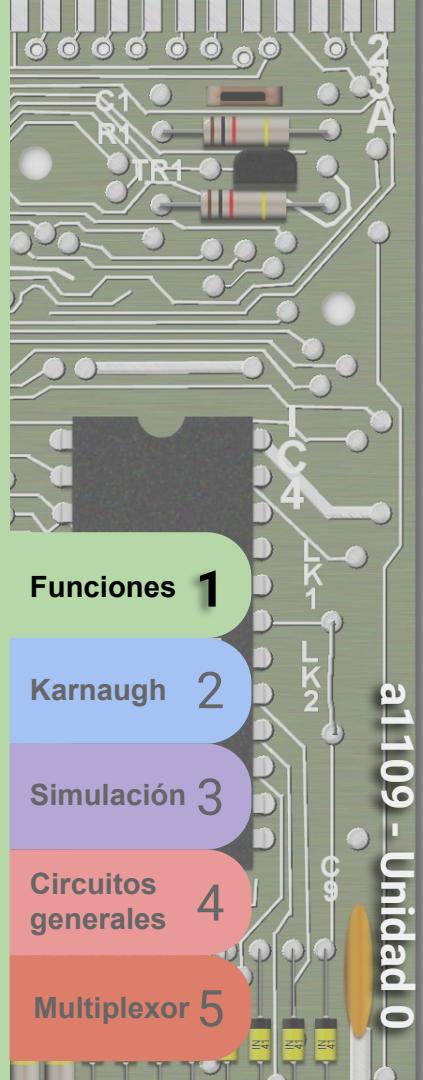
Funciones Booleanas de dos variables

Utilizando las operaciones de suma, producto y negación (or, and y not respectivamente) podemos implementar utilizando compuertas cualquier función booleana de dos variables. Vemos que la compuerta XOR puede ser construida utilizando las compuertas AND, OR y NOT. Esto vale para funciones de mayor cantidad de variables. En el caso de XOR, notemos como cuando $A=0$, $Y=B$, y cuando $A=1$ entonces $Y=\neg B$. Esto resulta muy útil.



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

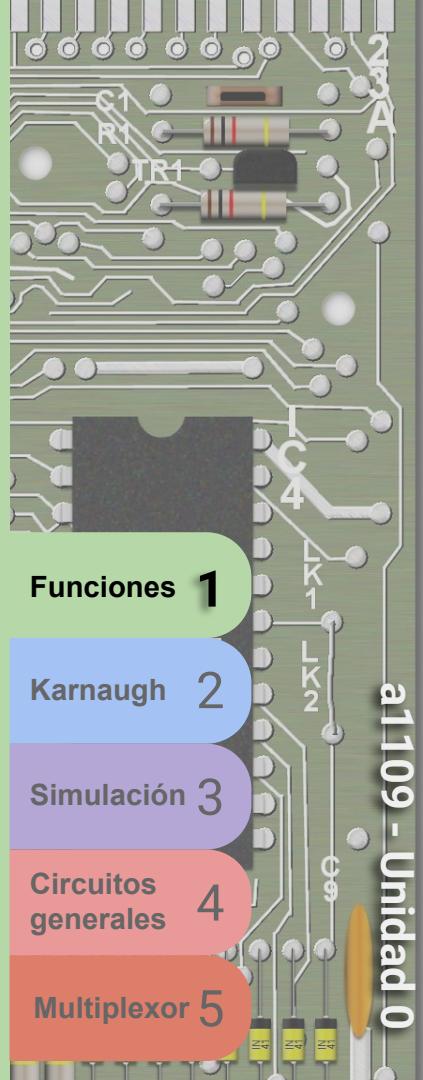
A	B	Y
0	0	B
0	1	B
1	0	!B
1	1	!B



Funciones más complejas

Veamos un ejemplo de una función de 3 variables. Esta función se llama mayoría. Produce un uno si la cantidad de unos en la entrada es mayor que la cantidad de ceros.

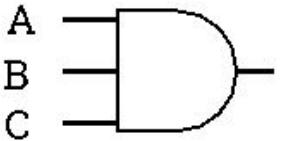
A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1



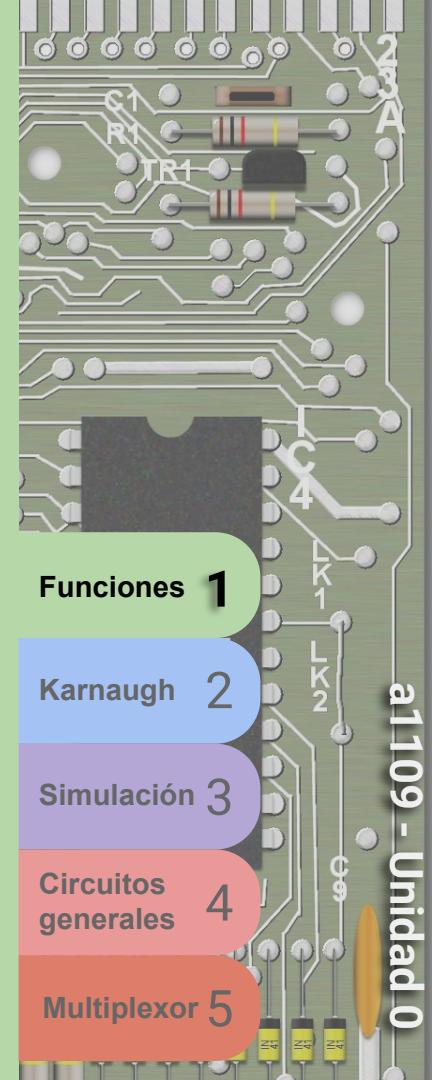
Funciones más complejas

Notemos que para cada uno existe una combinación única de valores de entrada para A,B y C. Por ejemplo, cuando A=1, B=1, C=1 la función vale 1.

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



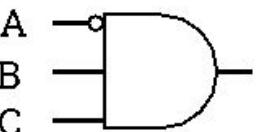
Una compuerta AND de 3 entradas solo produce un uno en su salida cuando sus tres entradas están en uno. Si conectamos A, B y C a una compuerta AND de 3 entradas detectará sin problemas el caso A=1, B=1, C=1. Sin embargo los otros casos (011, 101 y 110) no serán detectados.



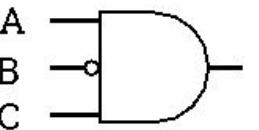
Funciones más complejas

Podemos negar ciertas entradas en una compuerta. Esto lo representamos con un círculo en esa entrada en particular. El valor de la variable se invierte.

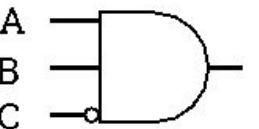
A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1



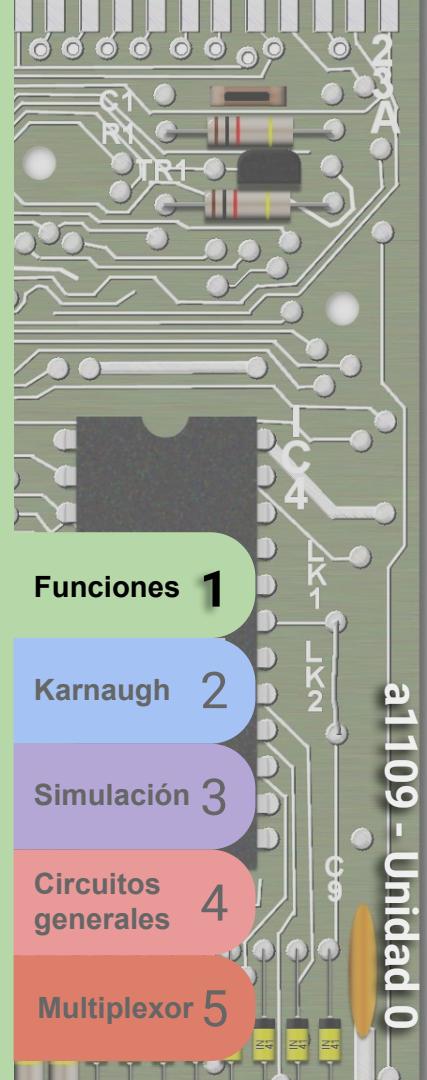
Negando A, detectamos el caso A=0, B=1 y C=1.



Luego negando B detectamos el caso A=1, B=0 y C=1



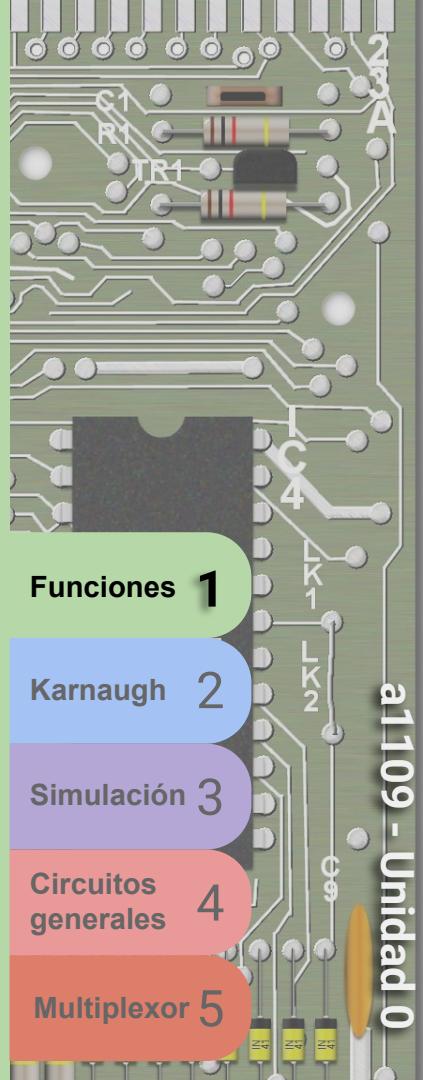
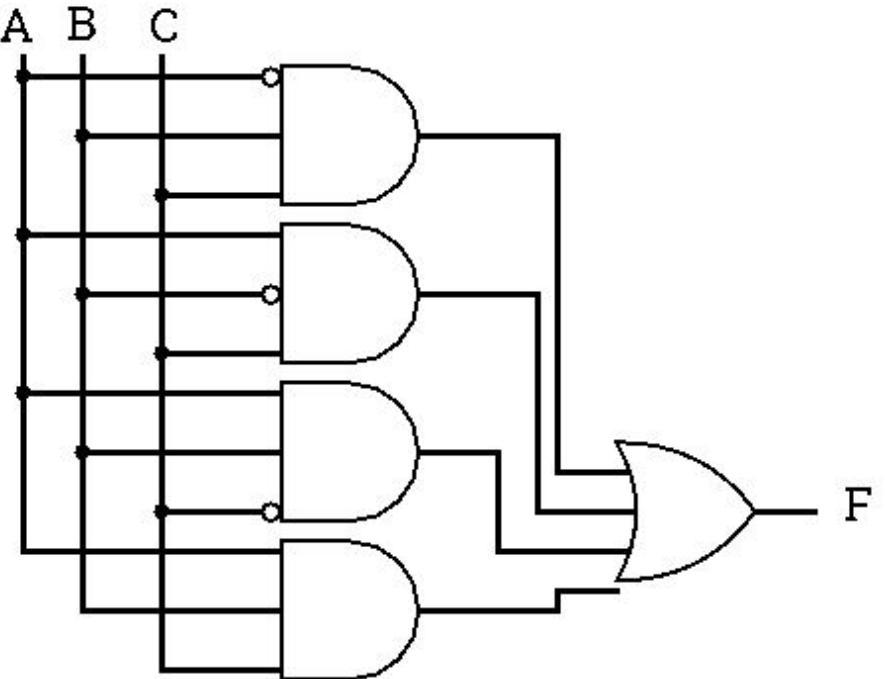
Por último negando C detectamos el caso A=1, B=1 y C=0.



Funciones más complejas

Solo nos queda “unir” todas las AND que detectan cada una de las combinaciones que generan un uno con una OR y tenemos el circuito que detecta mayoría de unos.

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

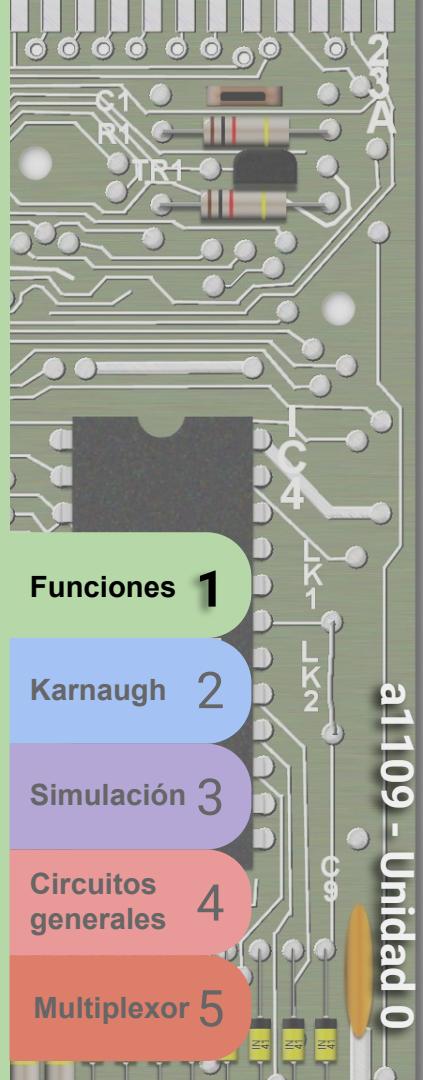
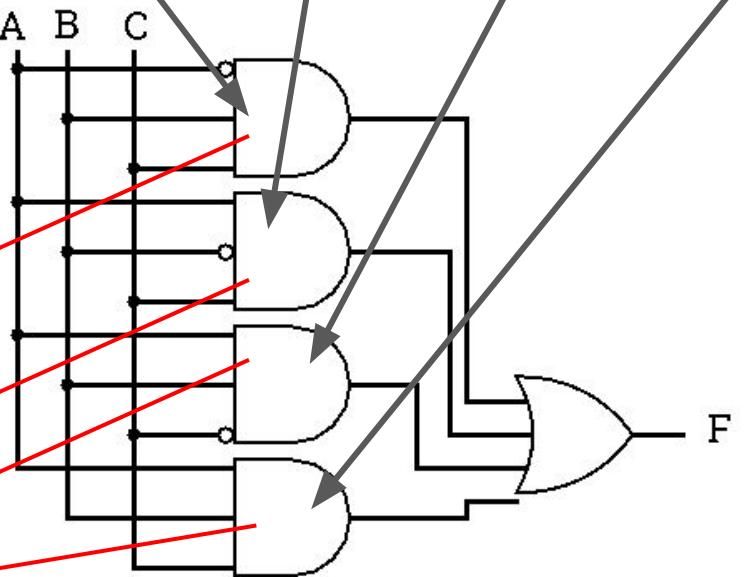


Funciones más complejas

Podemos escribir la función F utilizando la operación producto y suma booleana (más negadores).

$$F(A, B, C) = \overline{A} \cdot B \cdot C + A \cdot \overline{B} \cdot C + A \cdot B \cdot \overline{C} + A \cdot B \cdot C$$

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

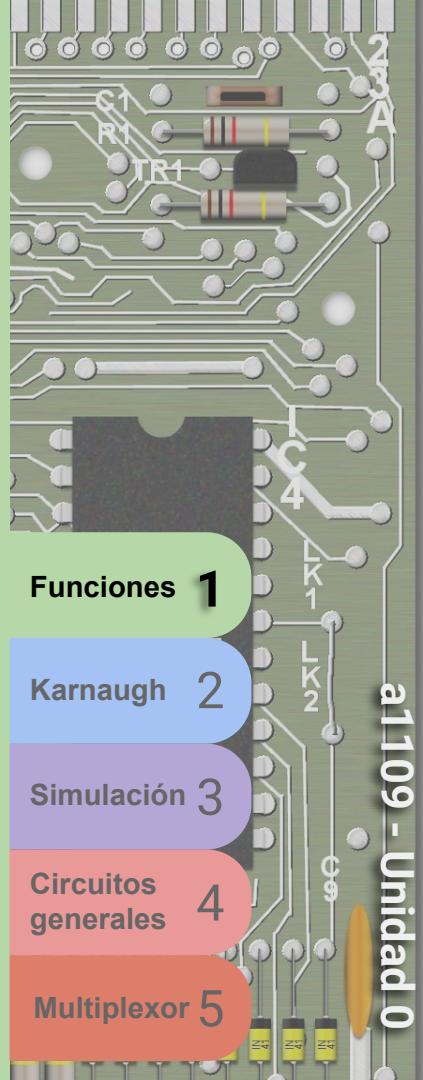
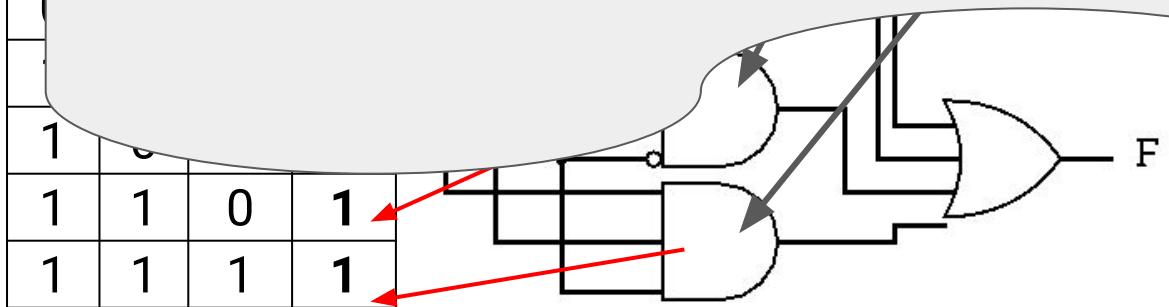


Funciones más complejas

Podemos escribir la función F utilizando la operación de producto y suma booleana (máximo común divisor)

$$F = C \cdot \overline{A} \cdot D$$

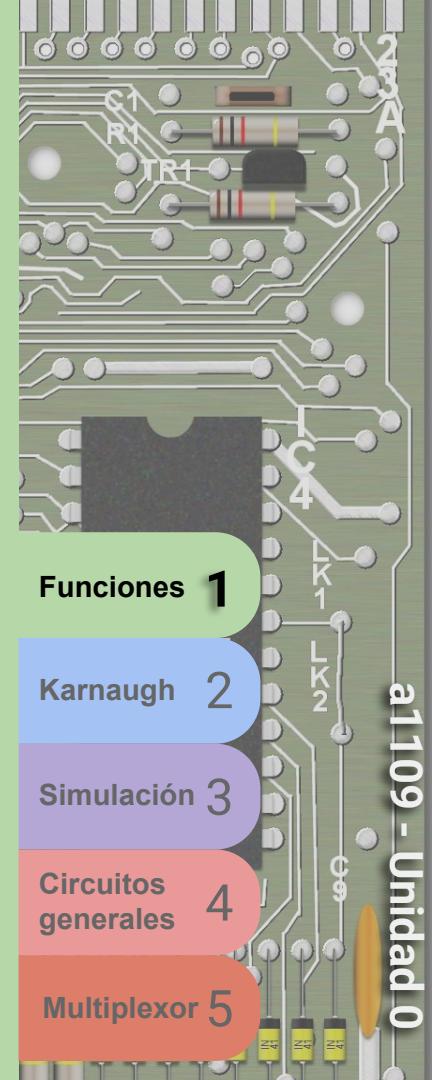
Las tres formas de representación de la función (Tabla de verdad, circuito con compuertas o por definición de operaciones booleanas) SON EQUIVALENTES! Es decir pasar de una forma a otra es completamente trivial. Las tres representan la misma función lógica.



Funciones más complejas

$$F(A, B, C) = \bar{A} \cdot B \cdot C + A \cdot \bar{B} \cdot C + A \cdot B \cdot \bar{C} + A \cdot B \cdot C$$

A esta forma de representar la función se la conoce como primera forma canónica. Esto quiere decir que se la representa como una suma (or) de productos (and) en donde cada producto (and) tiene a todas las variables de la función (en este caso A, B y C). Ya vimos que cada producto da como resultado uno para una combinación de entrada en particular. Cada producto que incluya a todas las variables de forma tal que haga que la función valga uno se lo conoce como término mínimo, o Minitermino. Existen tantos miniterminos como 2^N siendo N la cantidad de variables. Pero no todos estos miniterminos definen la función.



Funciones más complejas

$$F(A,B,C) = \overline{A} \cdot B \cdot C + A \cdot \overline{B} \cdot C + A \cdot B \cdot \overline{C} + A \cdot B \cdot C$$

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

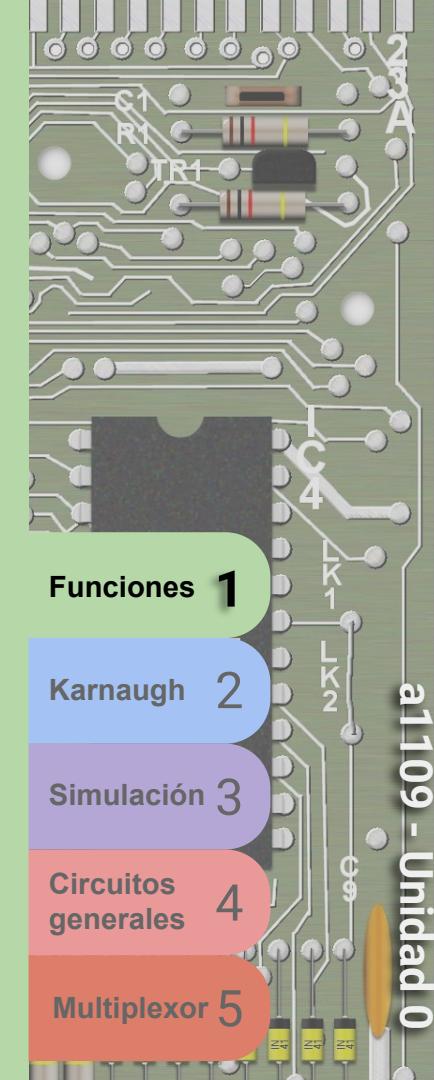
En el caso de la función mayoría, podemos expresarla como

$$F(A,B,C) = m_3 + m_5 + m_6 + m_7$$

Esto no es otra cosa más que una forma simplificada de representación. Podemos expresarlo también como:

$$F(A,B,C) = \sum m(3,5,6,7)$$

Los minitérminos m_0 , m_1 , m_2 y m_4 no hacen 1 a la función, por eso están excluidos.

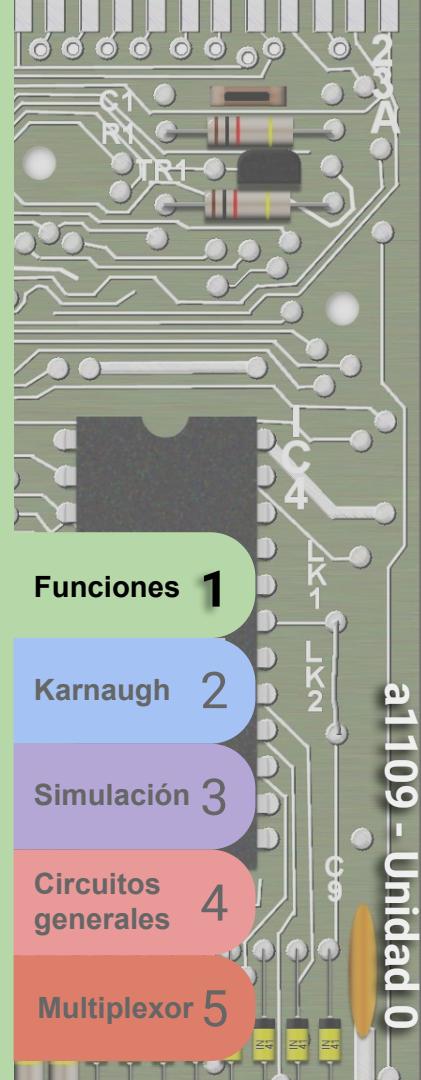


Funciones más complejas

$$F(A,B,C) = (A+B+C) \cdot (A+B+\bar{C}) \cdot (A+\bar{B}+C) \cdot (\bar{A}+B+C)$$

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

Podemos también pensar a la inversa. En vez de utilizar AND y detectar los unos, podemos utilizar OR y detectar los ceros. Para cada combinación de entrada donde F=0, tomamos los valores de las variables negando las que tengan uno. Por ejemplo para la entrada A=0,B=0,C=1 que hace F=0, tomamos la suma $A+B+\neg C$. Luego hacemos el producto de todas estas sumas generando así la representación de producto de sumas.



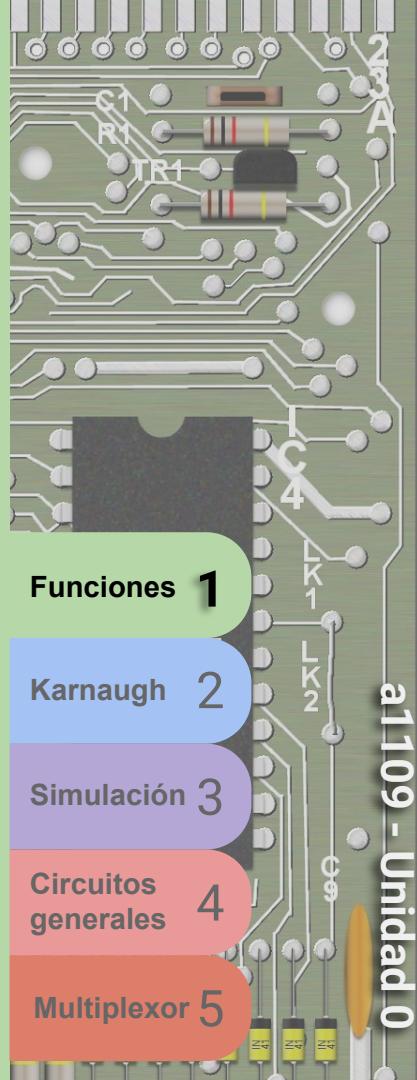
Funciones más complejas

$$F(A,B,C) = (A+B+C) \cdot (A+B+\bar{C}) \cdot (A+\bar{B}+C) \cdot (\bar{A}+B+C)$$

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

A cada uno de estos términos los llamamos maxítérminos. La función queda expresada en su segunda forma canónica.

$$F(A,B,C) = \prod M(0,1,2,4)$$



Postulados y teoremas

$$A \cdot 0 = 0$$

$$A \cdot 1 = A$$

$$A \cdot A = A$$

$$A \cdot \bar{A} = 0$$

$$A + 0 = A$$

$$A + 1 = 1$$

$$A + A = A$$

$$A + \bar{A} = 1$$

$$\bar{\bar{A}} = A$$

$$\overline{A \cdot B} = \bar{A} + \bar{B}$$

$$\overline{A + B} = \bar{A} \cdot \bar{B}$$

$$A \cdot B = B \cdot A$$

$$A + B = B + A$$

$$(A \cdot B) \cdot C = A \cdot (B \cdot C)$$

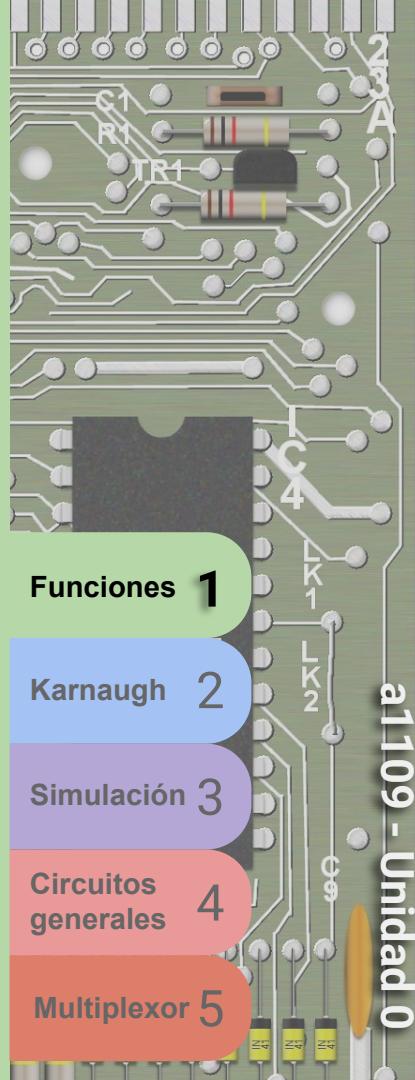
$$(A + B) + C = A + (B + C)$$

$$A + (A \cdot B) = A$$

$$A \cdot (A + B) = A$$

$$A \cdot (B + C) = A \cdot B + A \cdot C$$

$$A + (B \cdot C) = (A + B) \cdot (A + C)$$

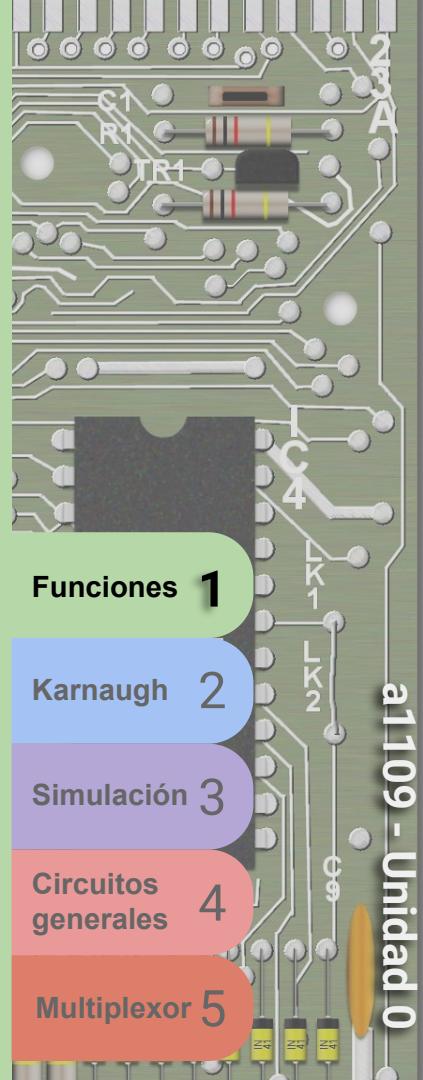


Funciones dadas

- No siempre encontramos funciones definidas en su primera o segunda forma normal. Una función puede estar definida de forma tal que todos sus términos no incluyan todas las variables. También podemos encontrar variables redundantes que no aportan nada a la función.

$$f = a \cdot (b + \bar{b}) + a \cdot a \cdot (b + b) + a \cdot (b + 1) \cdot c \cdot \bar{c}$$

Esta función no está en ninguna forma normal, o sea no se ve que esté claramente expresada como suma de productos o producto de sumas. Incluye valores constantes como $b+1$ los cuales deben ser simplificados usando los postulados de la lógica booleana.



Funciones dadas

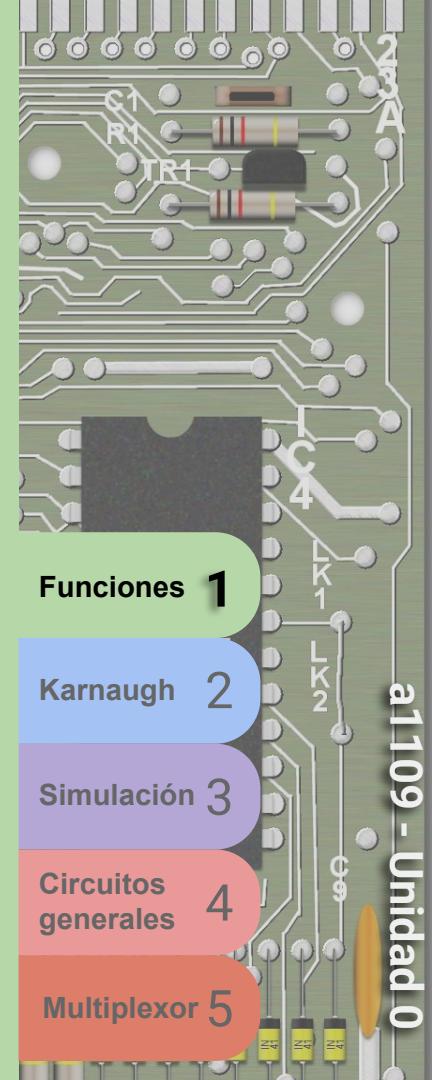
$$f = a \cdot (b + \bar{b}) + a \cdot a \cdot (b + b) + a \cdot (b + 1) \cdot c \cdot \bar{c}$$

$$f = a \cdot 1 + a \cdot b + a \cdot (b + 1) \cdot 0$$

$$f = a + a \cdot b$$

$$f = a$$

Utilizando los postulados podemos convertir la función en una versión simplificada y de ahí plantear alguna de las dos formas canónicas o quedarnos con la versión simplificada.

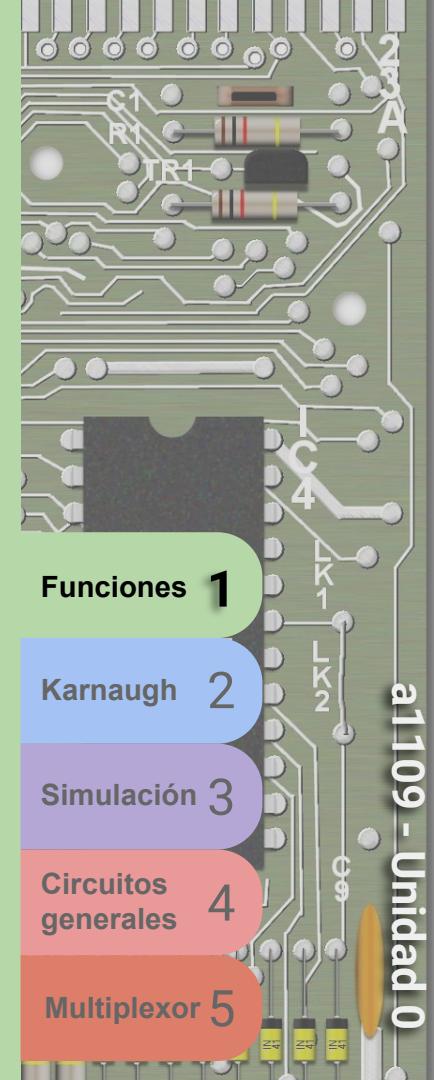


Funciones dadas

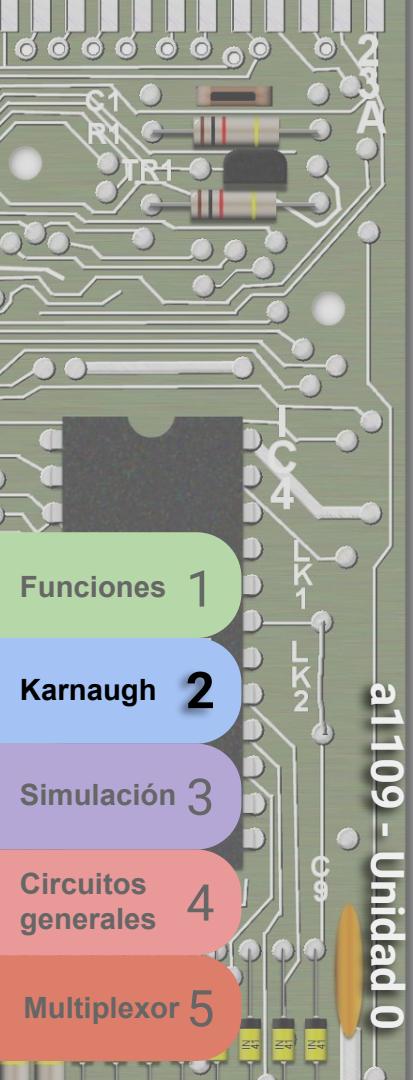
$$f = a \cdot (b + \bar{b}) + a \cdot a \cdot (b + b) + a \cdot (b + 1) \cdot c \cdot \bar{c}$$

Viendo la función, vemos que tiene 3 variables (a,b y c). Una alternativa a hacer la conversión de forma algebraica, es directamente plantear la tabla de verdad. Para ello damos todos los valores posibles a las variables (000, 001, 010,...,111) y vamos viendo qué valores obtiene la función. De aquí podemos plantear una forma canónica.

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



Simplificaciones

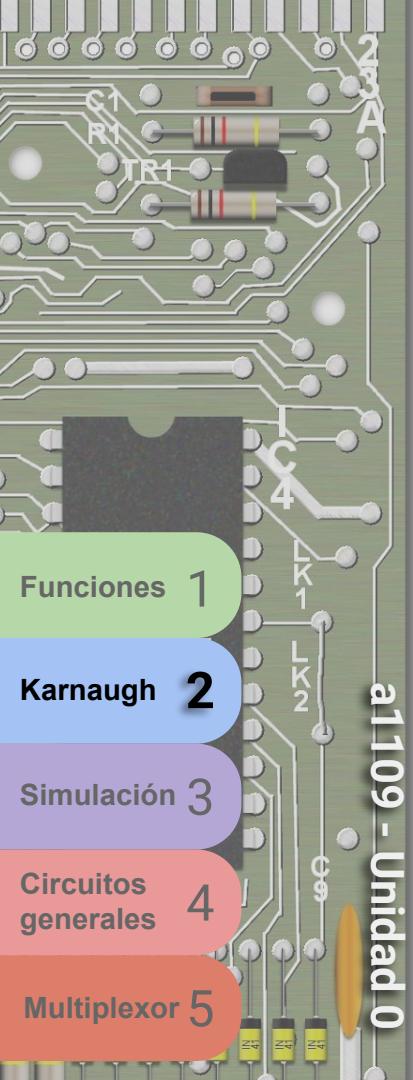


Simplificando

$$F(A,B,C) = \overline{A} \cdot B \cdot C + A \cdot \overline{B} \cdot C + \boxed{A \cdot B \cdot \overline{C} + A \cdot B \cdot C}$$

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

Vemos que en este caso, cuando $A=1$ y $B=1$ (remarcado en azul) existen dos valores para C ... 0 o 1 (remarcados en violeta). En cualquiera de los dos casos, la función toma el valor 1. Esto quiere decir que no importa el valor de C en los casos donde $A=1$ y $B=1$. Es por esto que podríamos simplificar ambos términos y escribir simplemente $A \cdot B$.

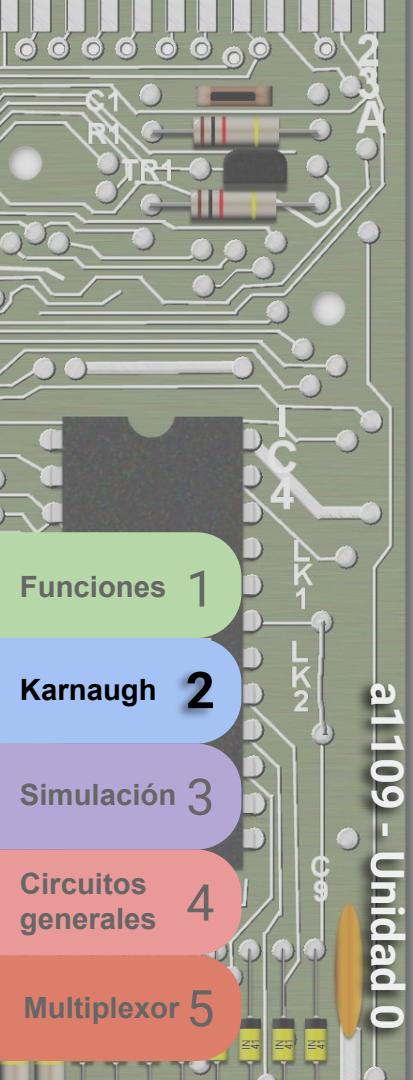


Simplificando

$$F(A, B, C) = \overline{A} \cdot B \cdot C + A \cdot \overline{B} \cdot C + A \cdot B \cdot \overline{C} + \overline{A} \cdot B \cdot \overline{C}$$

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

Algo similar ocurre en este caso, donde en azul vemos que los valores de B y C son 1 para ambos valores de A (remarcados en violeta) y en ambos casos la función F toma el valor 1. Esto quiere decir que no importa el valor de A siempre y cuando B=1 y C=1, la función toma 1. Por ende podemos simplificar a B.C.



Simplificando

$$F(A, B, C) = \overline{A} \cdot B \cdot C + A \cdot \overline{B} \cdot C + A \cdot B \cdot \overline{C} + A \cdot B \cdot C$$

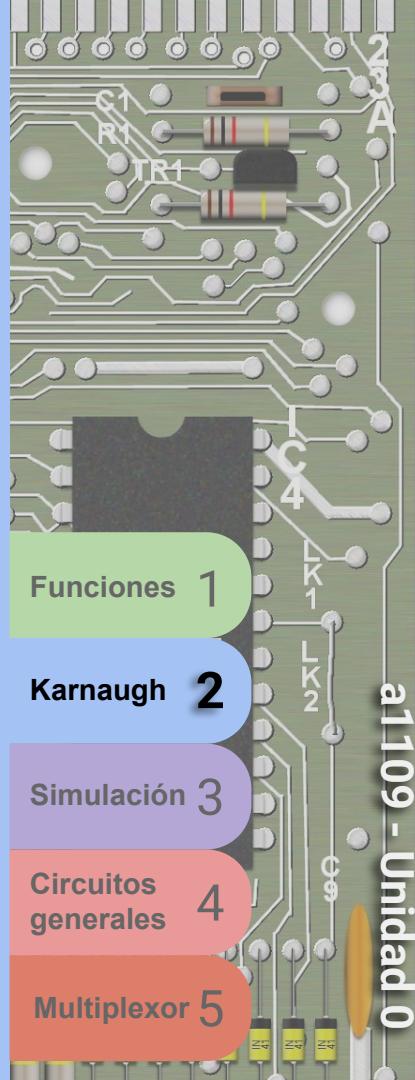
$$F(A, B, C) = \overline{A} \cdot B \cdot C + A \cdot \overline{B} \cdot C + A \cdot B \cdot \overline{C} + A \cdot B \cdot C + A \cdot B \cdot C + A \cdot B \cdot C$$

$$F(A,B,C) = \overline{A} \cdot B \cdot C + A \cdot B \cdot C + A \cdot \overline{B} \cdot C + A \cdot B \cdot \overline{C} + A \cdot B \cdot C$$

$$F(A,B,C) = (\overline{A} \cdot B \cdot C + A \cdot B \cdot C) + (A \cdot \overline{B} \cdot C + A \cdot B \cdot C) + (A \cdot B \cdot \overline{C} + A \cdot B \cdot C)$$

$$F(A, B, C) = B \cdot C + A \cdot C + A \cdot B$$

Utilizando algo de aritmética booleana sumamos términos ya existentes y los reacomodamos para que visualmente veamos todas las simplificaciones.

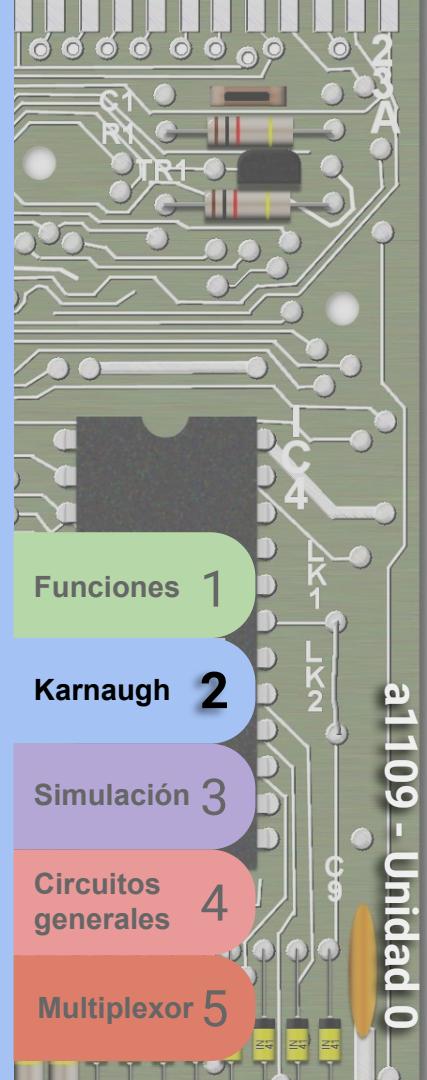


Método más simple

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

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

Reacomodamos la forma de ver la función utilizando un mapa de karnaugh. Esto es simplemente una forma distinta de acomodar los valores de la función. La forma en la que acomodamos las variables en el mapa de Karnaugh no es importante.

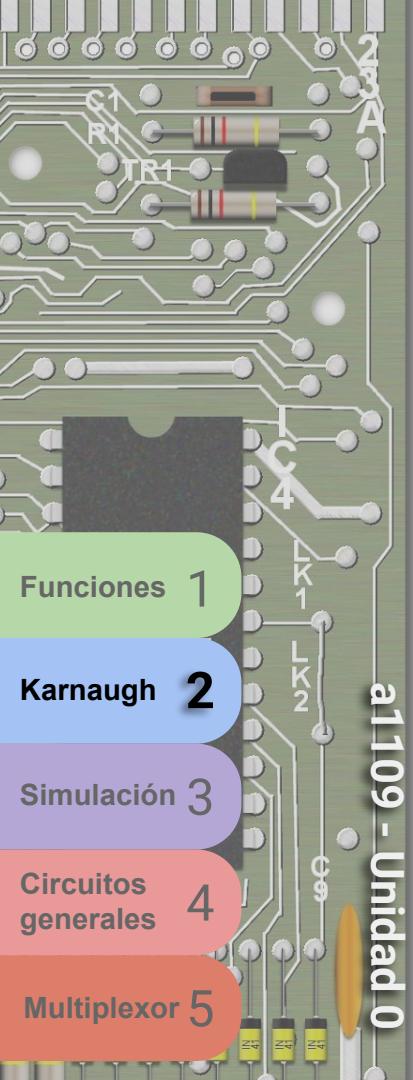


Método más simple

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

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

Lo que es importante es acomodar las filas y columnas siguiendo código de gray. Vemos que en rojo A=1,B=1, en violeta C=1 y entonces F=1, el cual marcamos en verde.

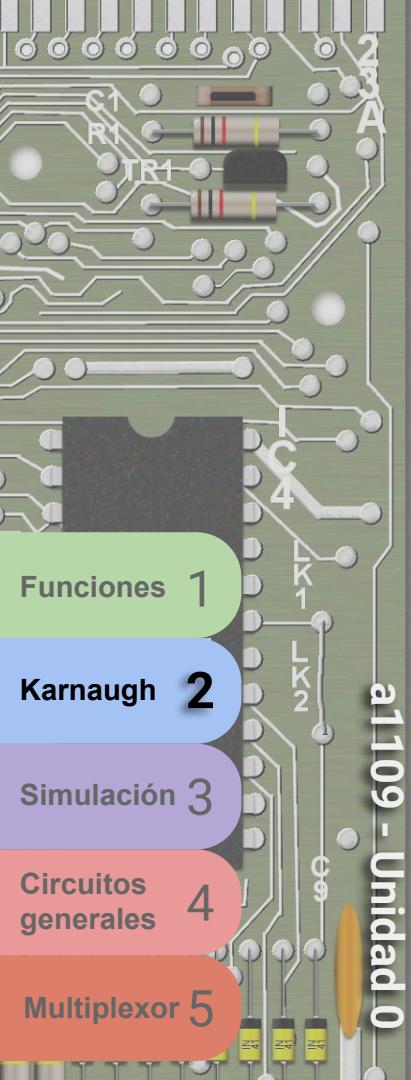


Método más simple

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

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

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



Funciones 1

Karnaugh 2

Simulación 3

Circuitos generales 4

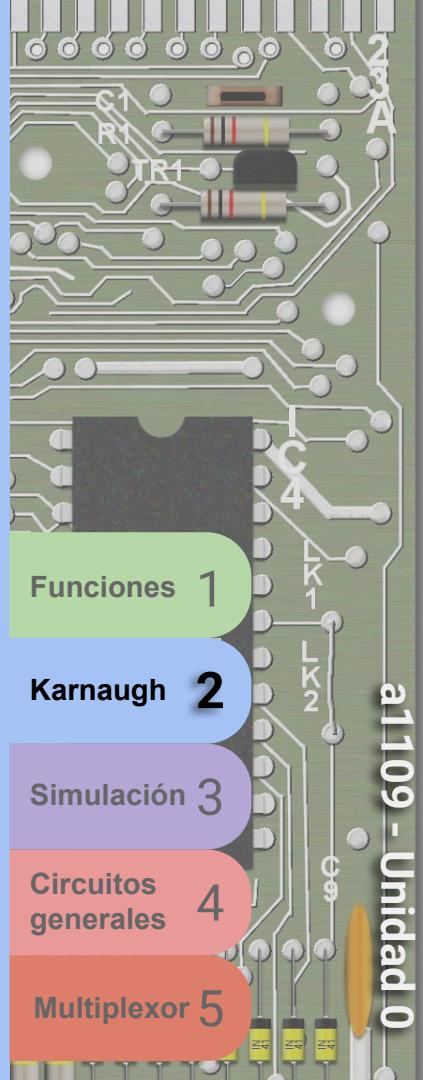
Multiplexor 5

Método más simple

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

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

Acomodamos los unos de la función en su posición correspondiente en el mapa. Esta forma de ubicarlos permite identificar términos candidatos a ser simplificados

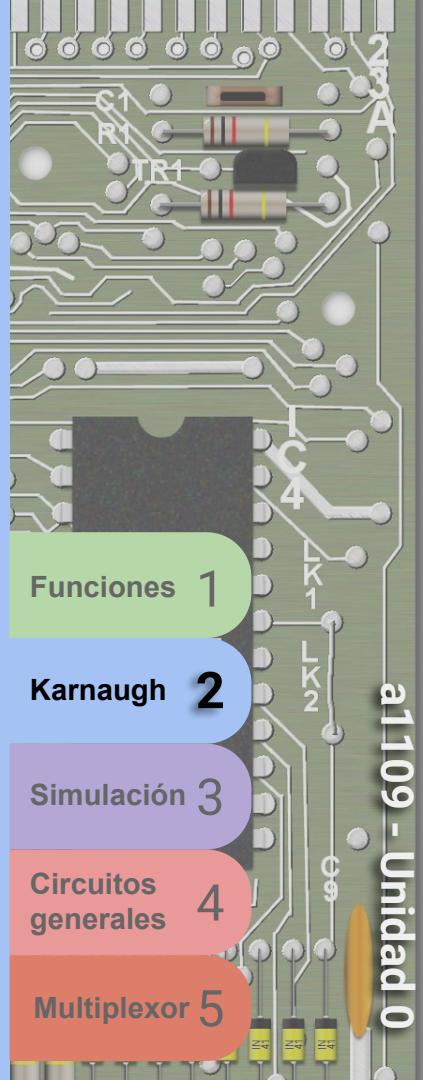


Método más simple

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

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

Resaltamos una de las posibles simplificaciones.
Notemos que A=1 y B=1 en ambos casos, mientras que C=0 y C=1 en los unos resaltados. Esto quiere decir que podemos simplificar C dejando solamente A.B.

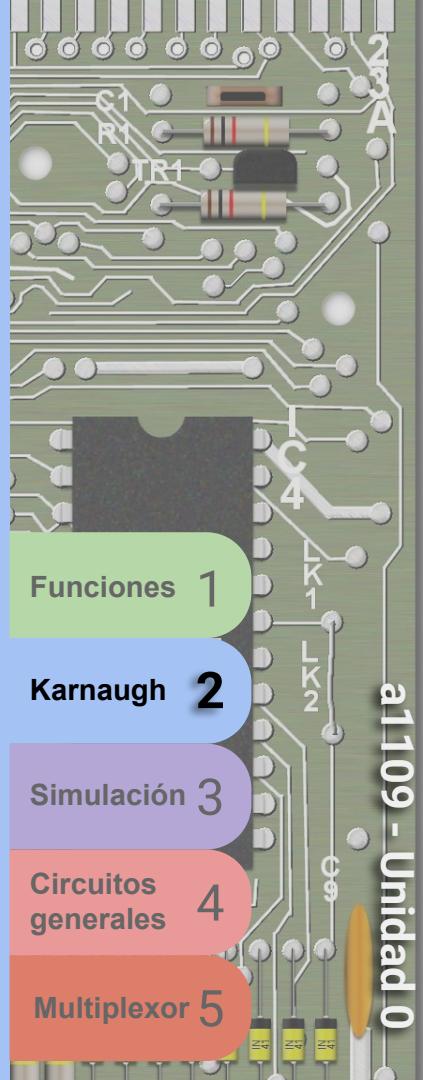


Método más simple

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

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

Vemos ahora que en este caso en la agrupación $B=1$ y $C=1$ tenemos los valores $A=0$ y $A=1$. Eso quiere decir que la variable A puede ser simplificada quedando B.C.

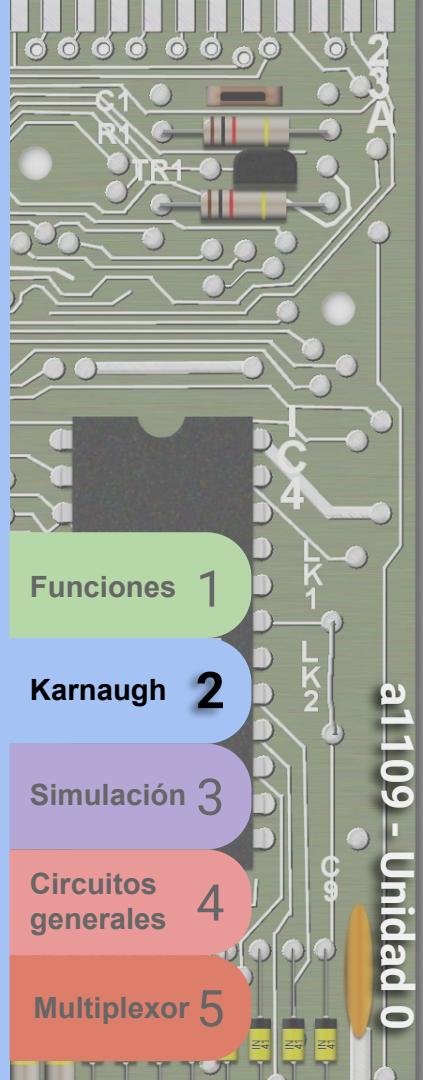


Método más simple

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

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

Por último nos queda el grupo A=1 C=1 para B=0,B=1.
Por ende simplificamos B quedando A,C.



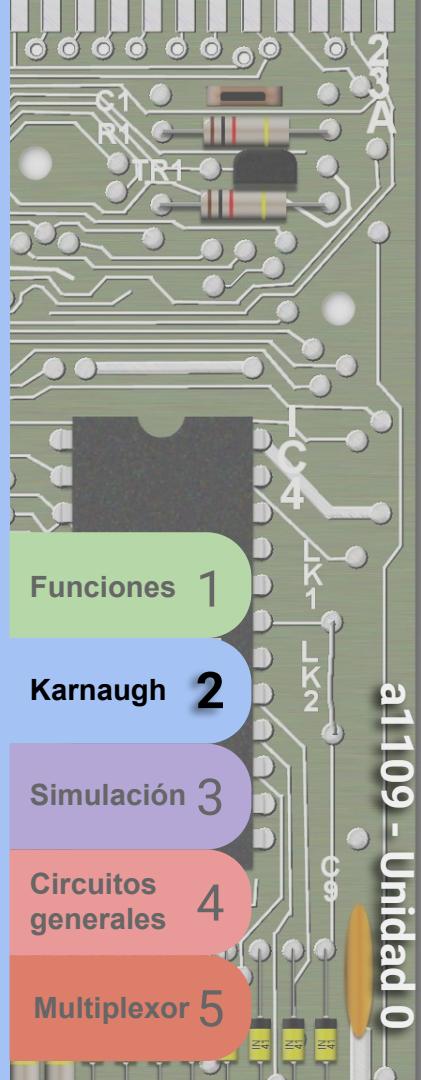
Método más simple

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

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

$$F(A, B, C) = B \cdot C + A \cdot C + A \cdot B$$

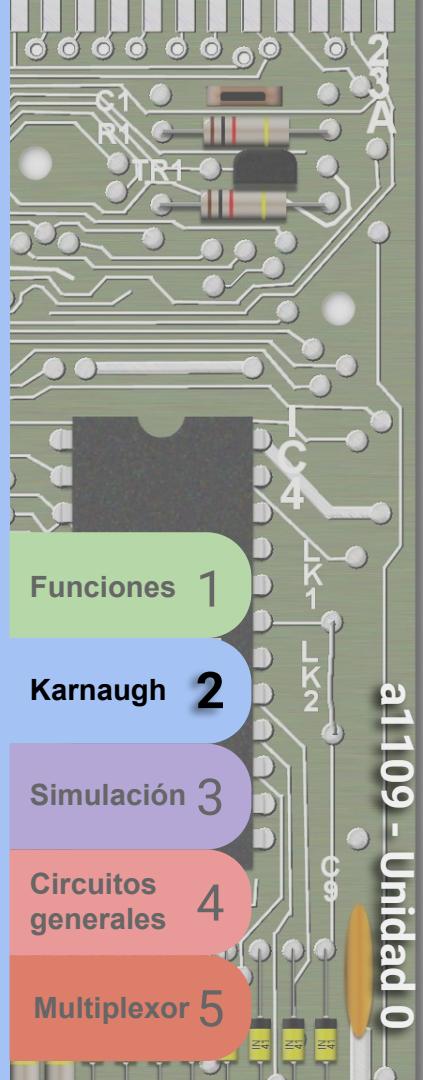
En este caso todos los unos de la función pudieron agruparse. Podríamos tener casos donde un uno no pueda agruparse. En ese caso, ese término no puede simplificarse.



Una función con valores no definidos

A	B	C	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	X
1	1	0	1
1	1	1	1

Notemos en esta función el valor X para 101. Este valor representa “don't care” (no importa). Eso quiere decir que puede tomar cualquier valor (0 o 1) ya que ese caso (101) no nos importa en la función.

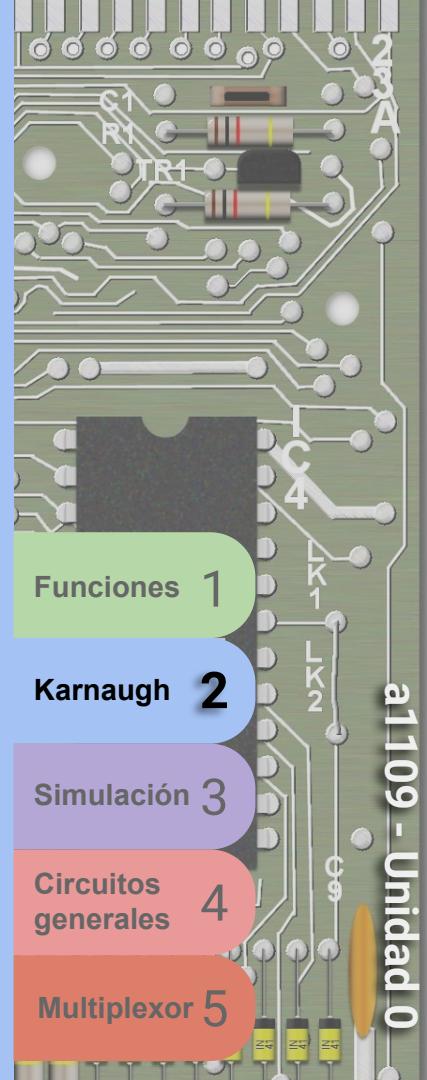


Una función con valores no definidos

A	B	C	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	X
1	1	0	1
1	1	1	1

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

En este caso, el valor X puede ser tomado como 1 o 0 dependiendo que tan conveniente sea.

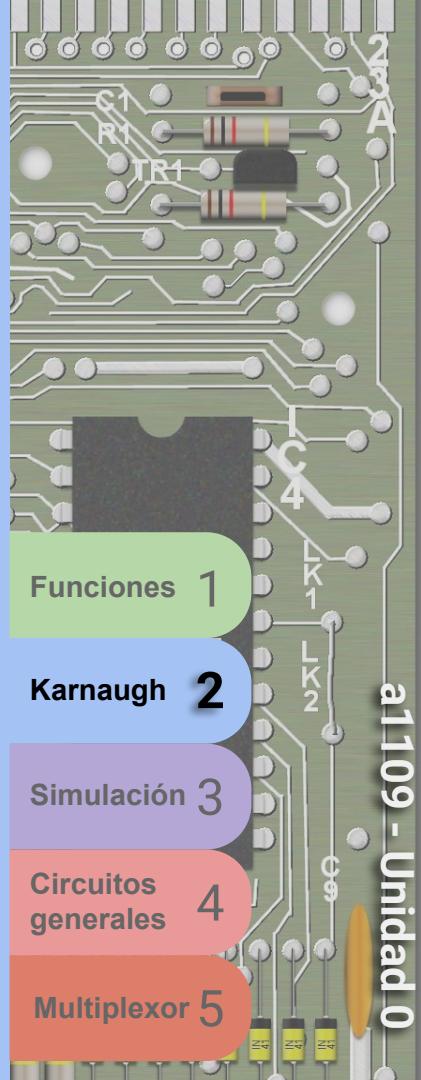


Una función con valores no definidos

A	B	C	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	X
1	1	0	1
1	1	1	1

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

Tenemos dos grupos. El primero en violeta agrupa 4 unos, vemos que agrupa 110, 111, 100, 101. El único que se mantiene constante es el primer 1 correspondiente a A. Por ende se simplifica como A.

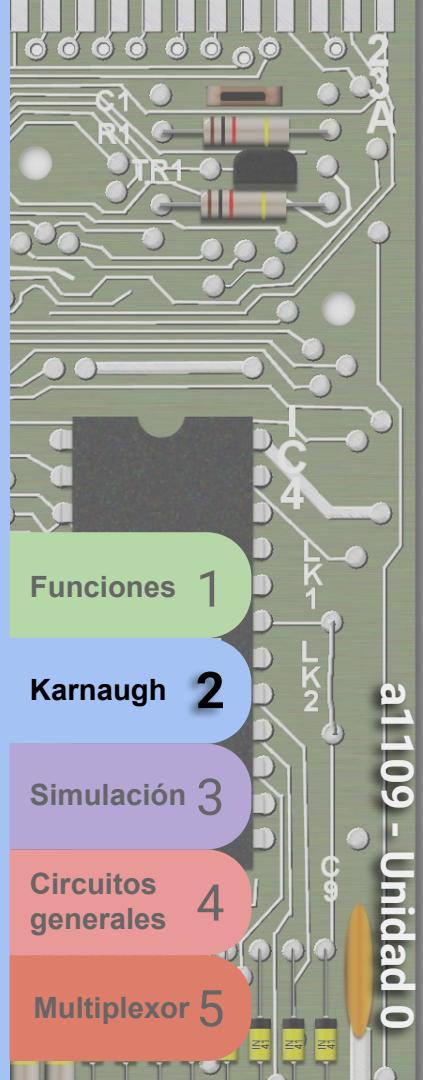


Una función con valores no definidos

A	B	C	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	X
1	1	0	1
1	1	1	1

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

El segundo grupo en rojo, vemos que “pega la vuelta” (adyacencia). Esto es válido como grupo. Vemos que agrupa 001 y 101. Vemos que A cambia (0 y 1) mientras que B=0 y C=1, por ende se simplifica como !B.C



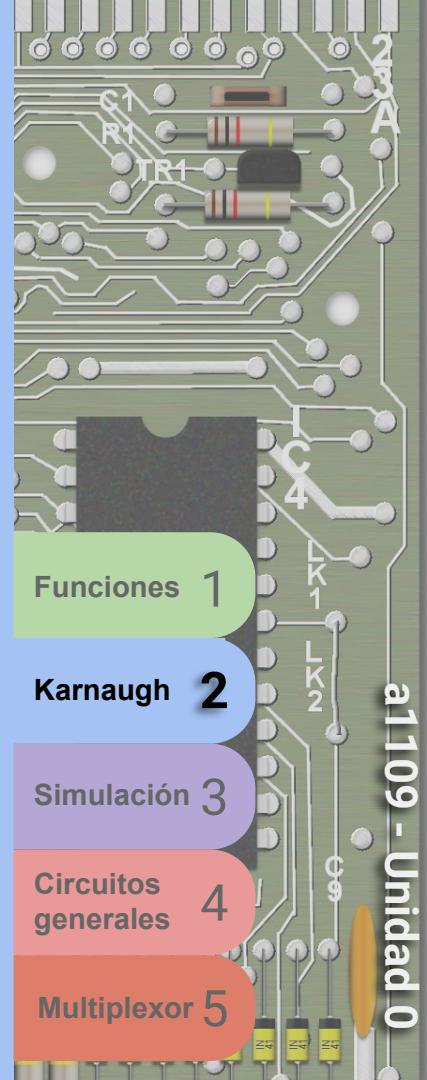
Una función con valores no definidos

A	B	C	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	X
1	1	0	1
1	1	1	1

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

$$F(A, B, C) = A + \overline{B} \cdot C$$

Vemos que el convertir X a 1 es muy conveniente ya que permite agrupar.



Funciones 1

Karnaugh 2

Simulación 3

Circuitos generales 4

Multiplexor 5

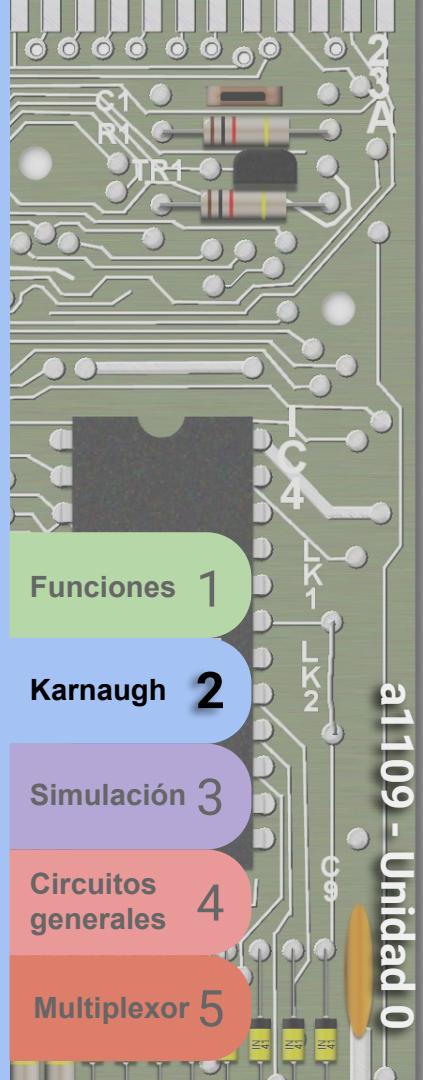
Una función con valores no definidos

A	B	C	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	X
1	1	0	1
1	1	1	1

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

Si hubiésemos tomado 0 como valor para 101, entonces solo podemos hacer dos simplificaciones.

$$F(A, B, C) = A \cdot B + A \cdot \bar{C} + \bar{A} \cdot \bar{B} \cdot C$$

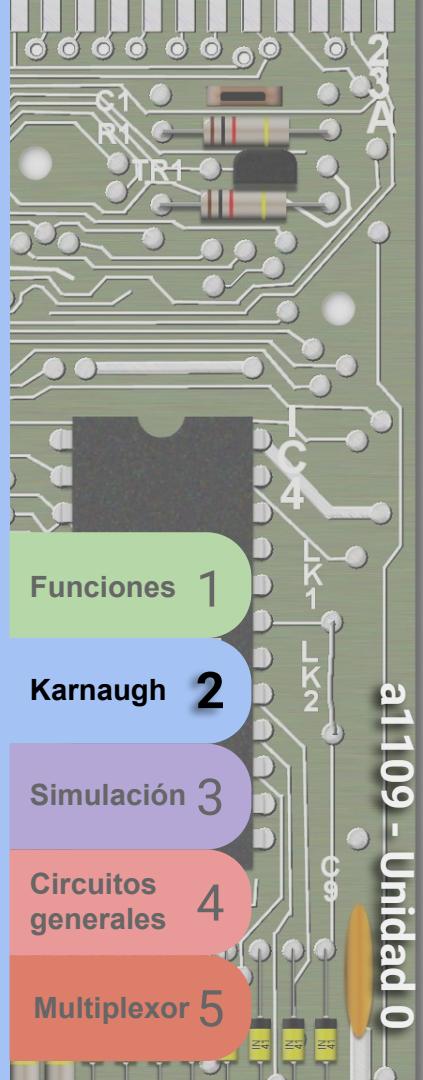


Producto de sumas

A	B	C	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	X
1	1	0	1
1	1	1	1

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

Podemos usar el mapa de karnaugh para simplificar por los ceros. Marcamos los ceros y agrupamos. Recordemos que los ceros generan productos de sumas, por ende las variables en 0 se complementan.



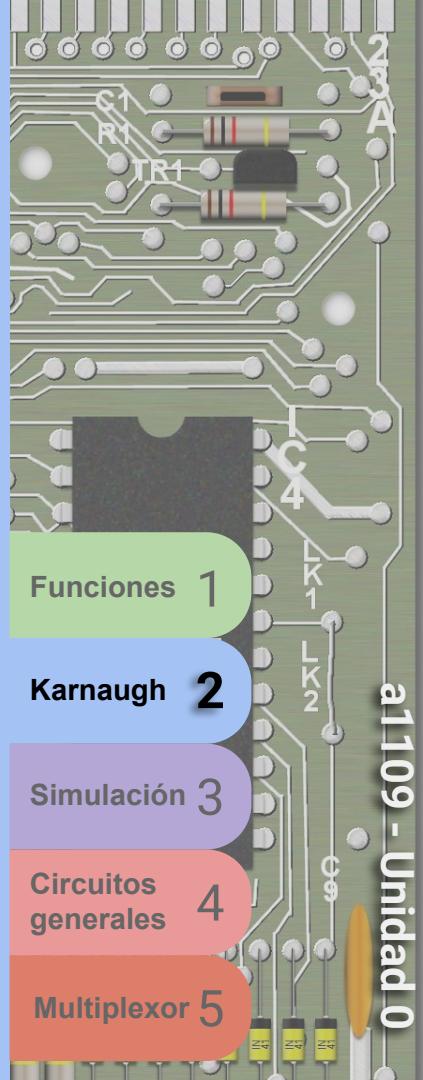
Producto de sumas

A	B	C	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	X
1	1	0	1
1	1	1	1

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

$$F(A, B, C) = (A+C) \cdot (A+\bar{B})$$

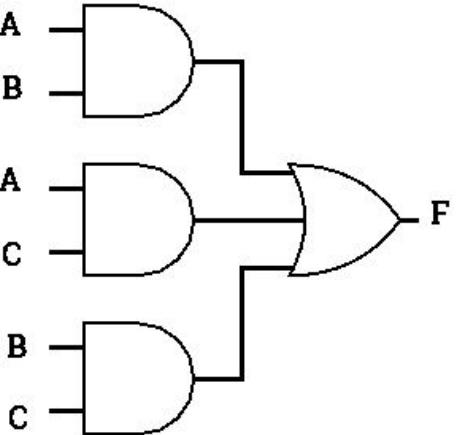
Vemos que 000 y 010 puede simplificarse B. En este caso los 0s se complementan, por ende queda A+C. Luego en 010 011 podemos simplificar C. Queda A+B. Vemos que el X conviene tomarlo como 1 e ignorarlo.



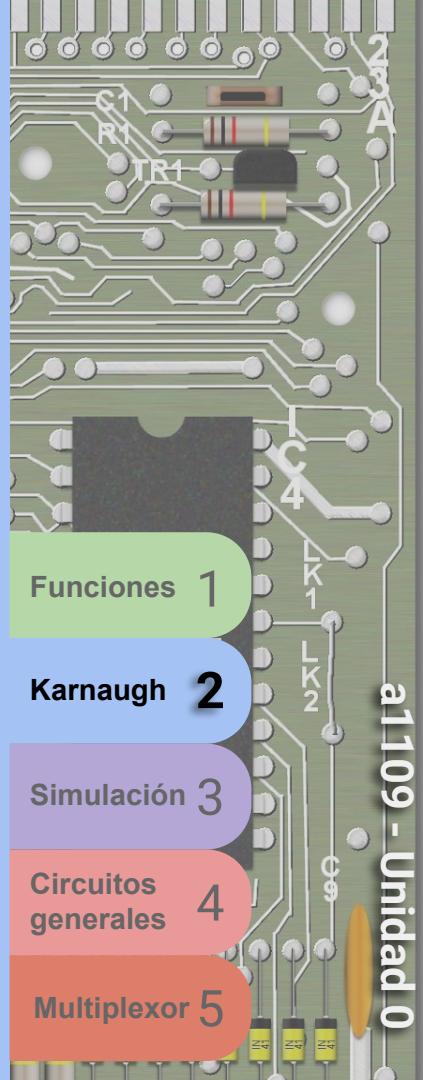
Función mayoría con compuertas

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

$$F(A, B, C) = B \cdot C + A \cdot C + A \cdot B$$

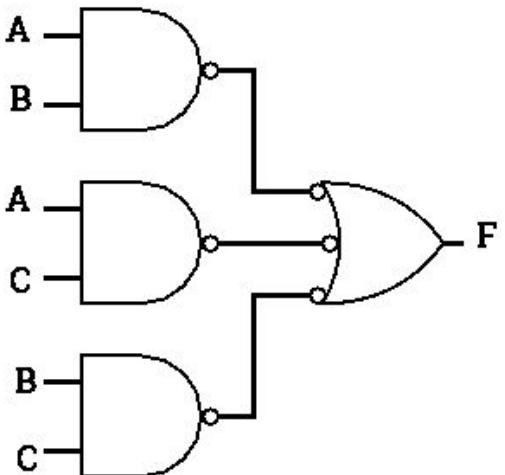
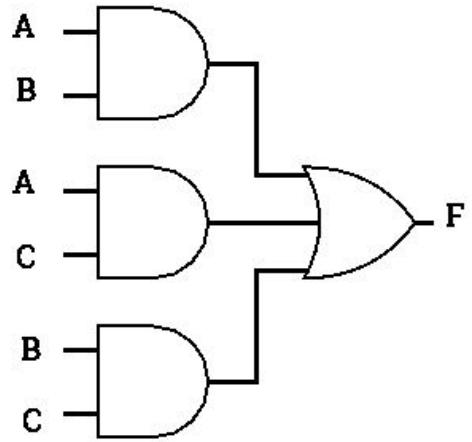


La función simplificada utiliza menos compuertas que la función original. Dado que ambas cumplen la misma función, son equivalentes, y nos conviene simplificar siempre y cuando sea posible.

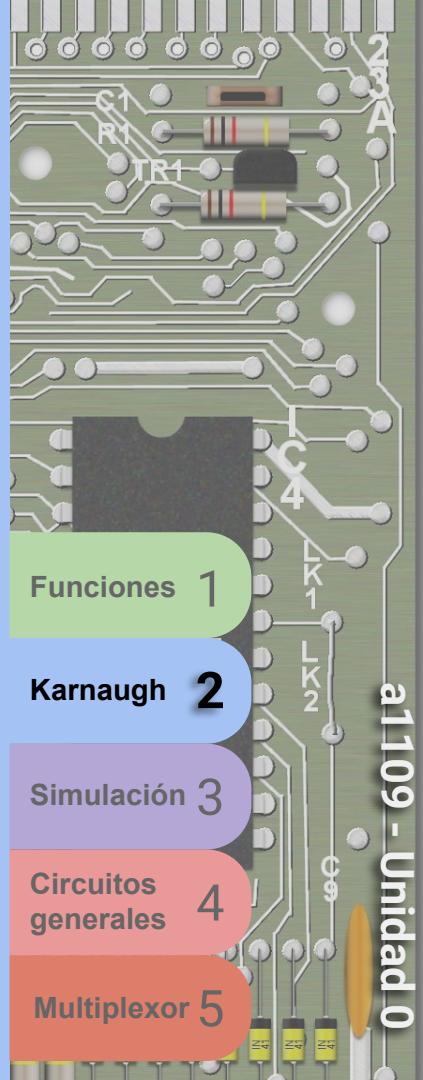


Implementar con un tipo de compuertas

$$F(A, B, C) = B \cdot C + A \cdot C + A \cdot B$$

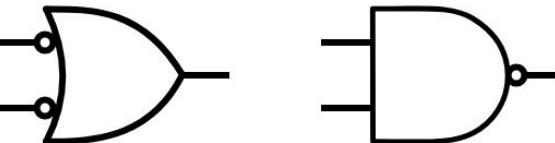
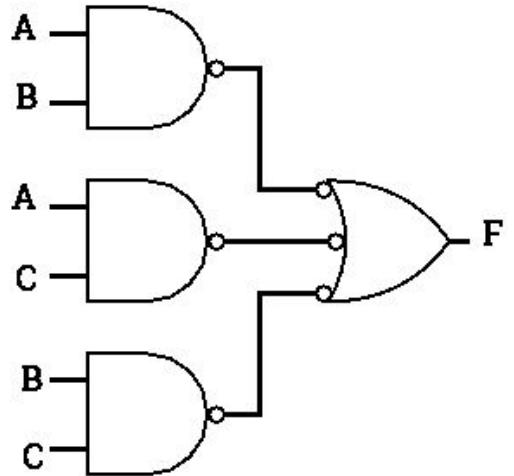


La función de la derecha cumple la misma función, nótese que la salida de cada AND está negada pero la entrada correspondiente en la OR también está negada, y dos negados se cancelan mutuamente.

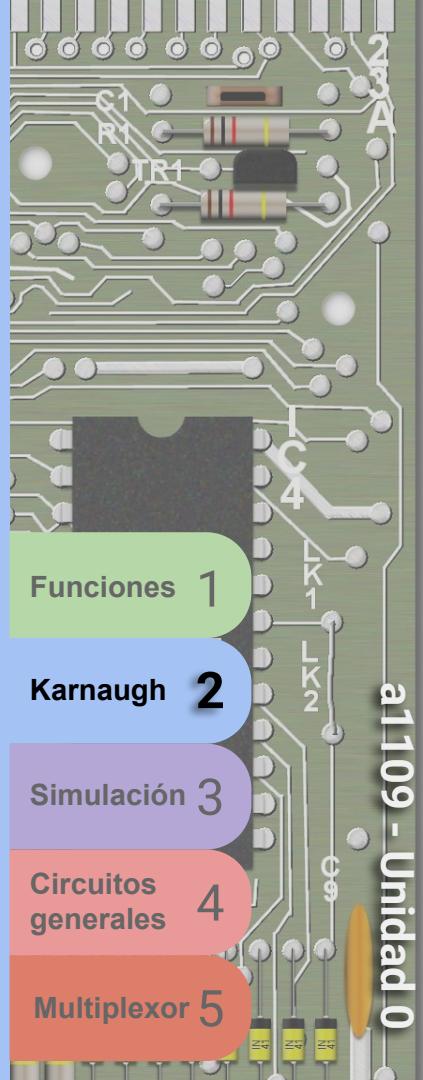


Implementar con un tipo de compuertas

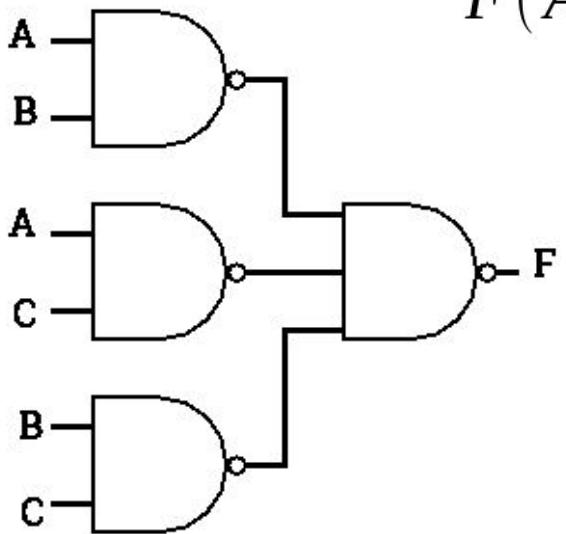
$$F(A, B, C) = B \cdot C + A \cdot C + A \cdot B$$



Utilizando DeMorgan sabemos que una compuerta OR con sus entradas negadas equivale a una compuerta NAND.

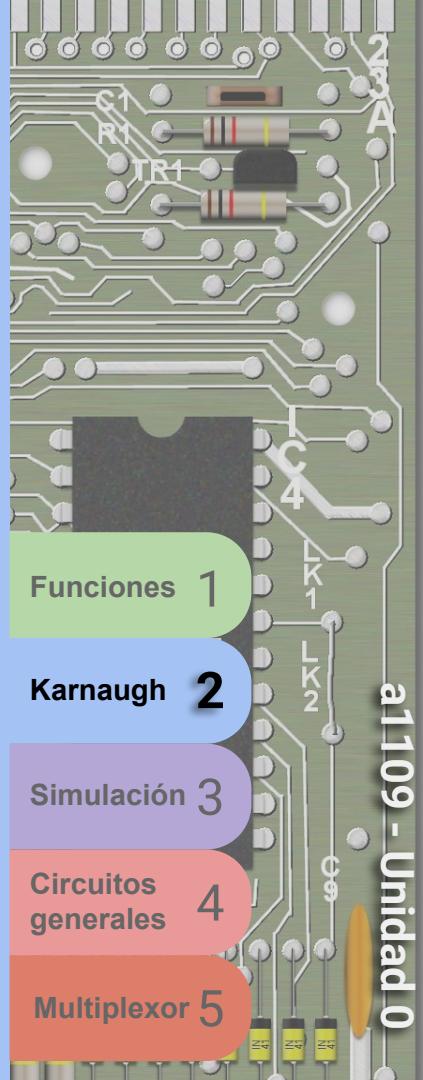


Implementar con un tipo de compuertas



$$F(A, B, C) = \overline{\overline{A} \cdot \overline{B} \cdot A \cdot C \cdot \overline{B} \cdot \overline{C}}$$

Veremos en la unidad de tecnología que las compuertas NAND tienen ventajas sobre las AND/OR en ciertas tecnologías, por ende preferimos la implementación con NAND.

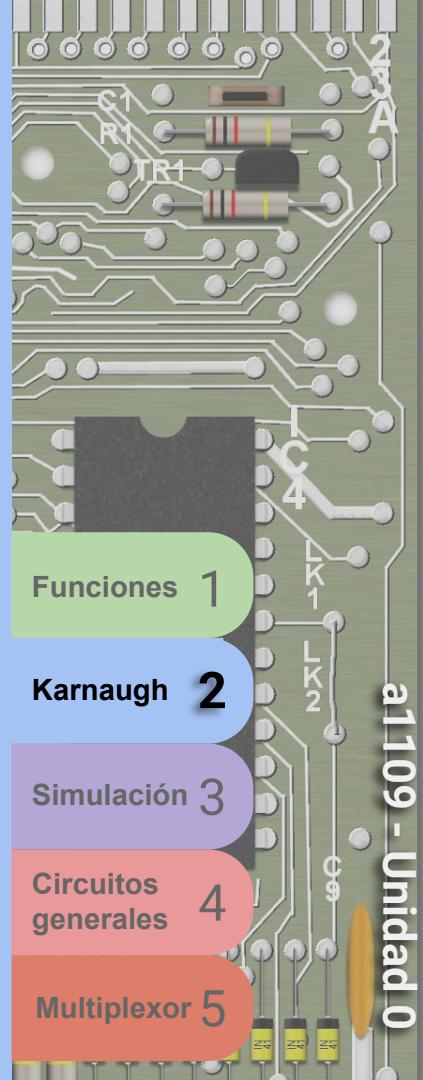


Implementar con un tipo de compuertas

$$F(A, B, C) = (A + B + C) \cdot (A + B + \bar{C}) \cdot (A + \bar{B} + C) \cdot (\bar{A} + B + C)$$

Si simplificamos la función en su segunda forma canónica obtendremos un producto de sumas.

Aplicando la misma estrategia de negar la salida y la entrada obtendremos compuertas NOR en vez de NAND. Dejamos esto como ejercicio.

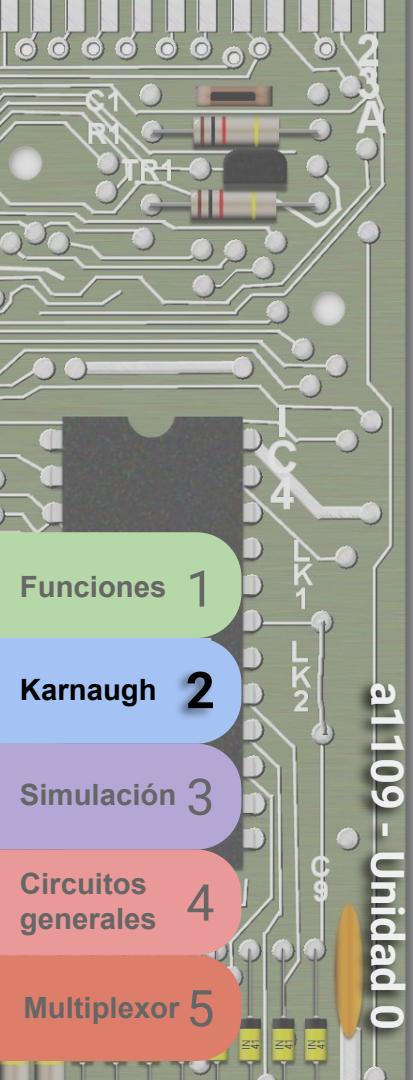
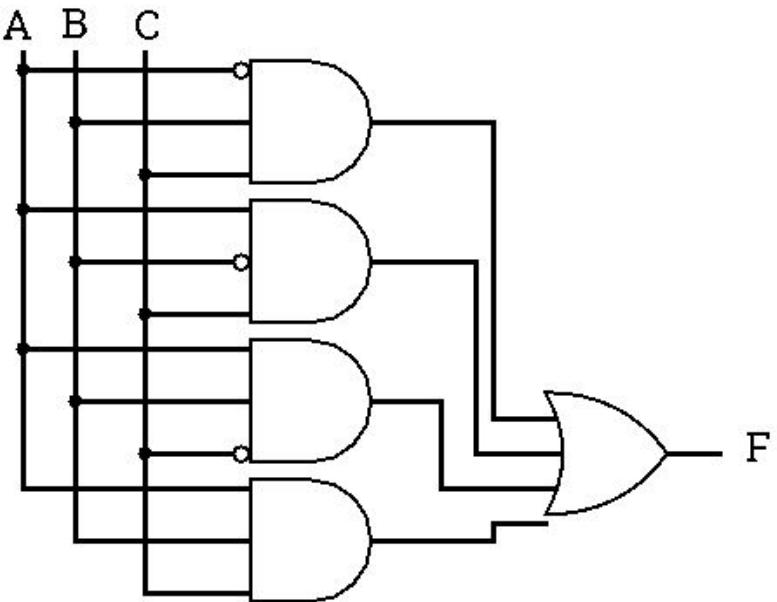


Lógica de dos niveles

Cualquier función de N variables puede resolverse en dos niveles (como suma de productos o producto de sumas).

$$F(A,B,C) = \overline{A} \cdot B \cdot C + A \cdot \overline{B} \cdot C + A \cdot B \cdot \overline{C} + A \cdot B \cdot C$$

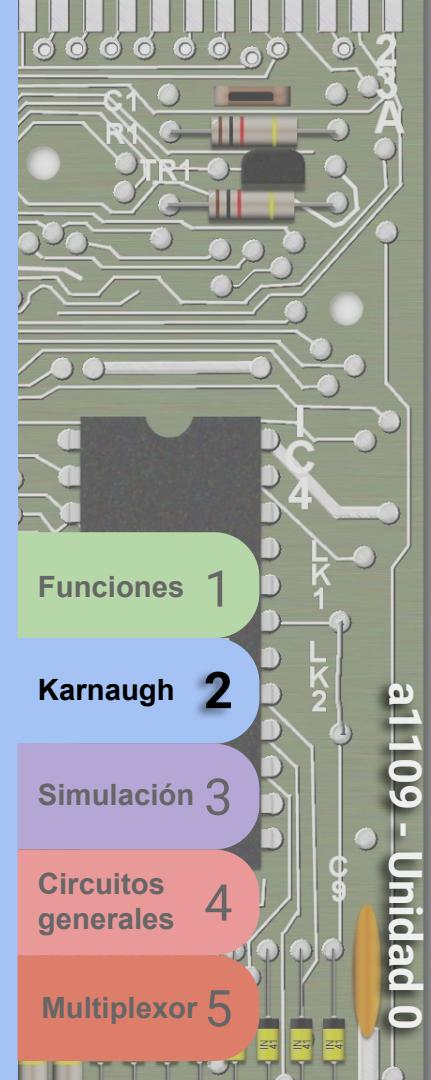
A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1



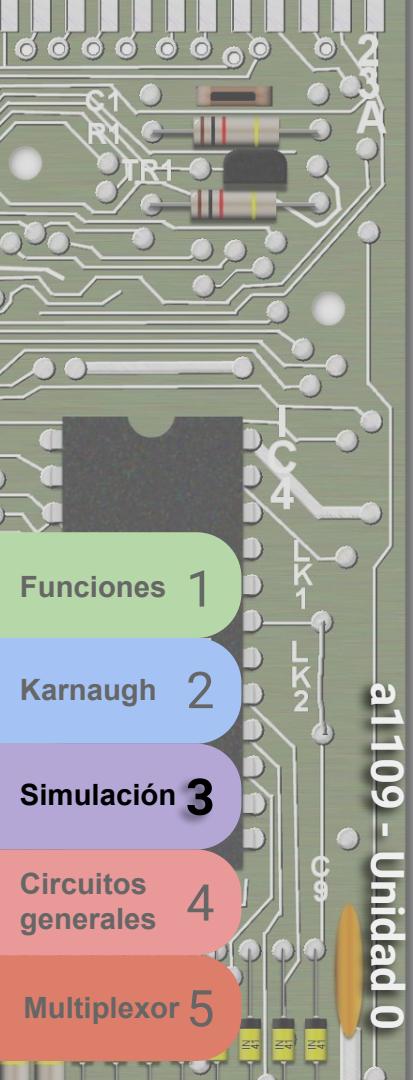
Lógica de dos niveles

$$f = a \cdot (b + \bar{b}) + a \cdot a \cdot (b + b) + a \cdot (b + 1) \cdot c \cdot \bar{c}$$

Aun cuando la función dada no se encuentre en forma canónica, podemos operar algebraicamente o encontrar su tabla de verdad y escribirla en una de las dos formas canónicas. Luego la implementación circuital podrá ser o no simplificada. En el peor de los casos, siempre podrá ser implementada como sumas de productos o productos de suma. En los casos triviales que se convierte a una función de dos variables conocidas se resuelve con una compuerta, y en el caso extremo con un cable o una constante.



Logisim

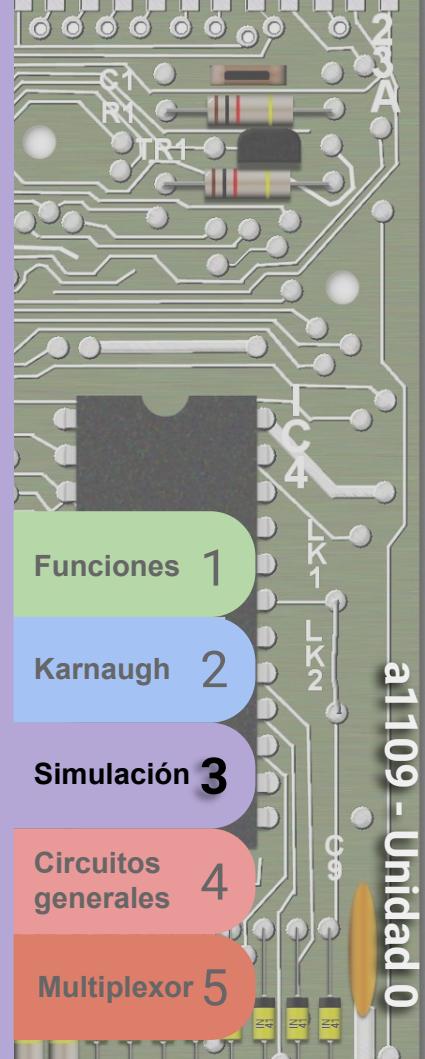
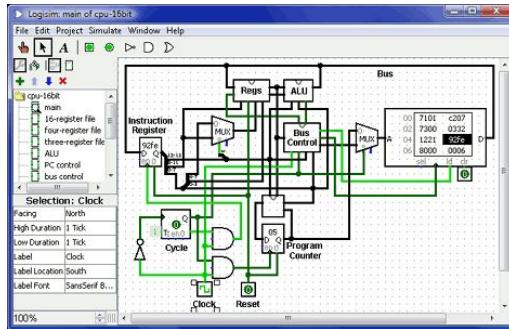


Logisim es un software de simulación de compuertas y circuitos lógicos.

<http://www.cburch.com/logisim/>

Está pensado para ser una herramienta didáctica, no un software de simulación de altas prestaciones. Está escrito en java siendo multiplataforma. Puede utilizarse en varios idiomas.

La idea es utilizarlo como herramienta para simular rápidamente circuitos. Si bien tiene algunas prestaciones avanzadas solo utilizaremos las básicas.



Funciones 1

Karnaugh 2

Simulación 3

Circuitos generales 4

Multiplexor 5

La manito permite cambiar estados de 0/1 en pines.



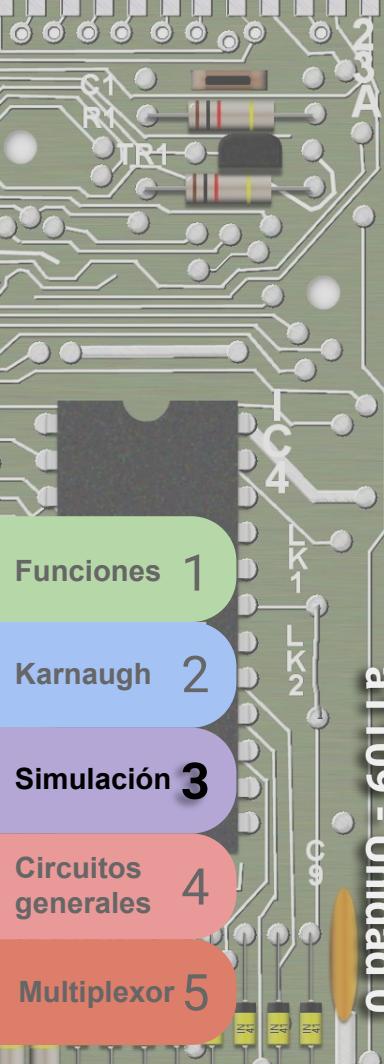
La flecha permite mover componentes o crear cables que unan terminales.

Los pines pueden ser de entrada o salida y pueden llevar nombres.

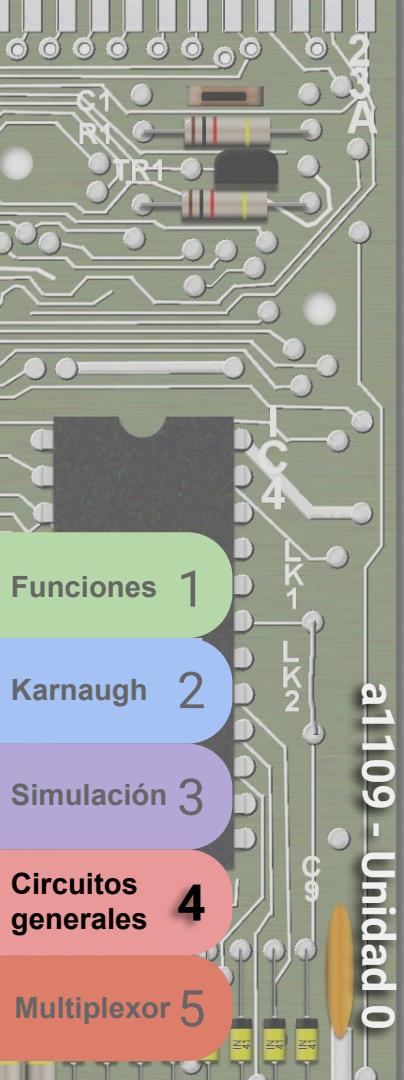
Las propiedades de las compuertas pueden editarse para soportar cantidad de entrada, negadores, etc.

The screenshot shows a logic simulation software window. The menu bar includes Archivo, Editar, Proyecto, Simular, Ventana, and Ayuda. The project tree on the left shows 'SinTítulo*' with a main component. The 'Wiring' section contains icons for Separador, Pin, Ver, Tunnel, Pull Resistor, Reloj, Constante, Power, Ground, Transistor, Transmission Gate, Bit Extender, and Puertas. The 'Puertas' section lists NOT, Buffer, AND, OR, NAND, NOR, XOR, XNOR, Detector Imparidad, Detector Paridad, Buffer Controlado, Inversor Controlado, Plexores, Aritmética, Memoria, Input/Output, and Base. A 'Pin' table is open, showing properties for pin A: Orientación (Este), ¿Salida? (No), Bits De Datos (1), ¿Tres-estados? (No), Comportamiento... (Invariant), Etiqueta (A), Posición De La E... (Norte), and Fuente Del Etiqu... (SansSerif Plano ...). The main workspace shows a NOT gate circuit with input A (labeled 0) and output 0.

Herramientas mínimas



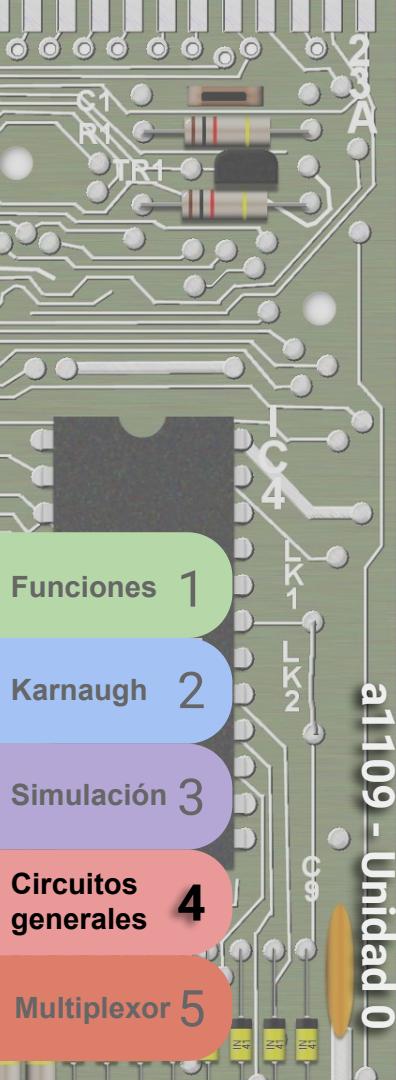
Circuitos combinatorios



Sumador de 2 números de 1 bit

Ahora que conocemos como implementar funciones (simplificadas, con lógica de 2 estados) podemos estudiar circuitos combinacionales comúnmente encontrados en diseños de circuitos. Uno de los más comunes es un sumador que permite sumar dos números de 1 bit. A este circuito lo conocemos como Half-Adder (semisumador) y posee como entrada dos números de 1 bit (A y B) y como salida la suma (S) y el carry (C).

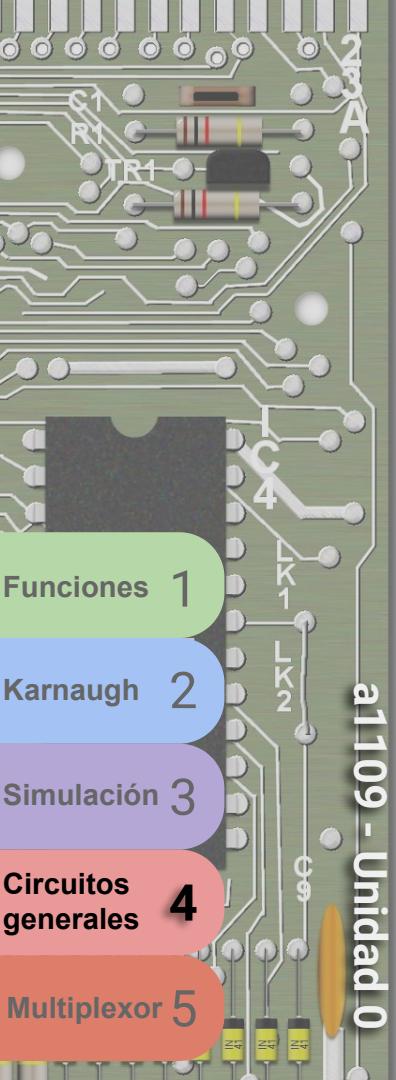
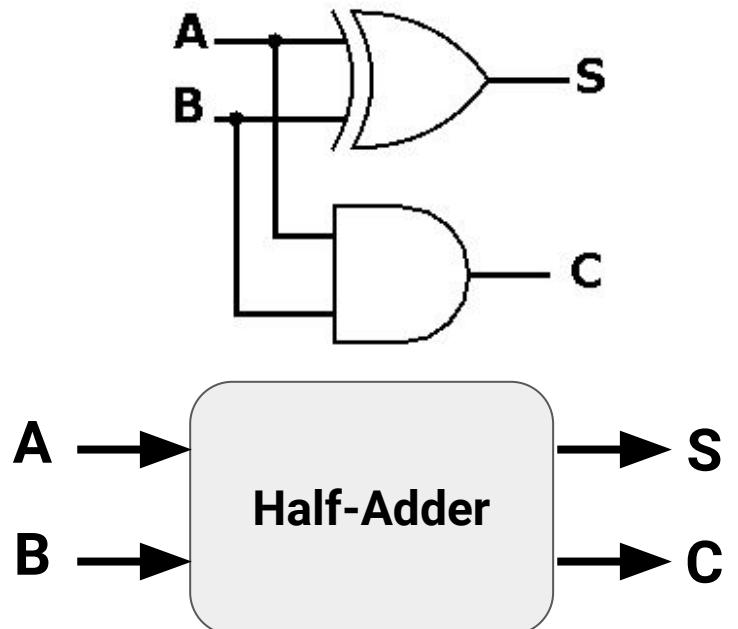
entrada	salida		
A	B	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0



Sumador de 2 números de 1 bit

Vemos la implementación circuital. Cada salida es una función. Vemos que S cumple con la tabla de verdad de la XOR y que C cumple con la AND.

entrada	salida		
A	B	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

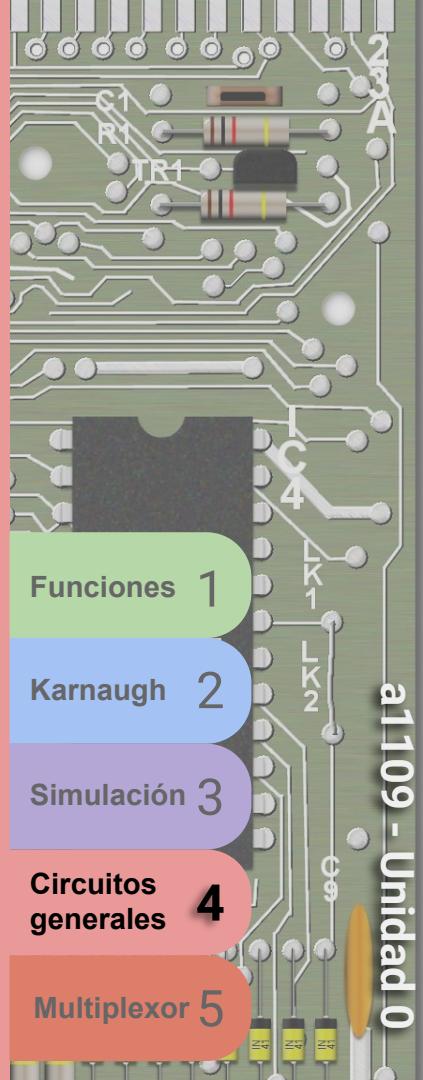
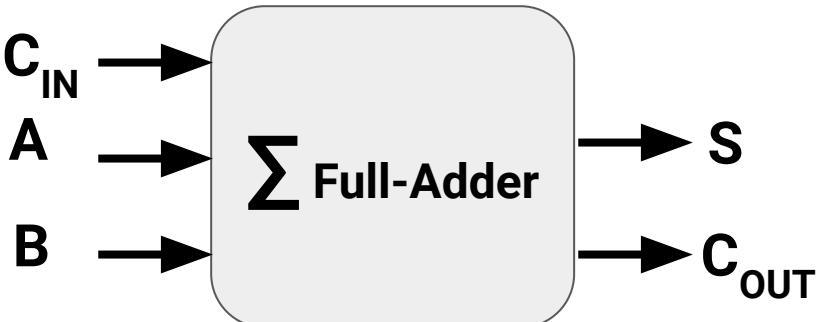


Sumador de 3 números de 1 bit

Si bien podemos juntar dos Half-Adder y armar un sumador que considere un carry entrante, también podemos directamente plantearlo desde la tabla de verdad.

Pero si miramos C_{OUT} tiene “pinta” de función mayoría. El caso de S es una XOR de 3 entradas. Podemos verlo planteando $C \text{ XOR } (A \text{ XOR } B)$. Resolver primero $A \text{ XOR } B$ y al resultado aplicar XOR C.

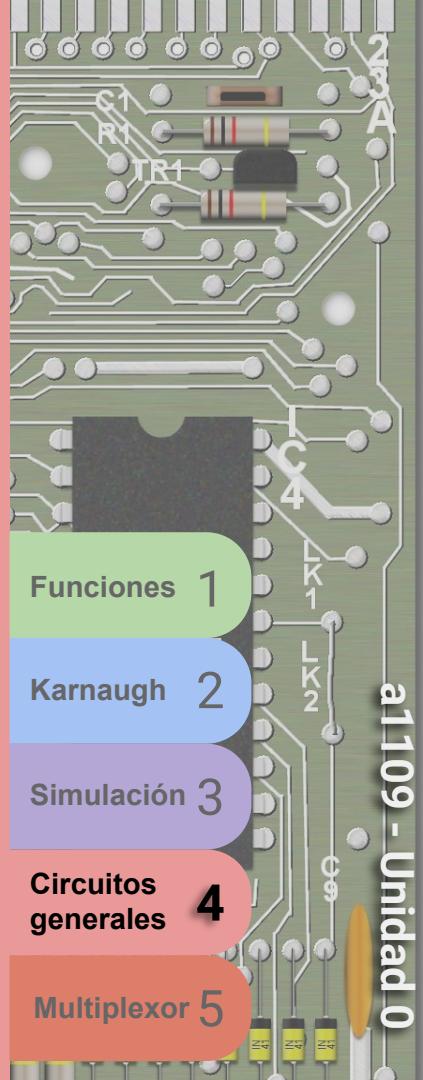
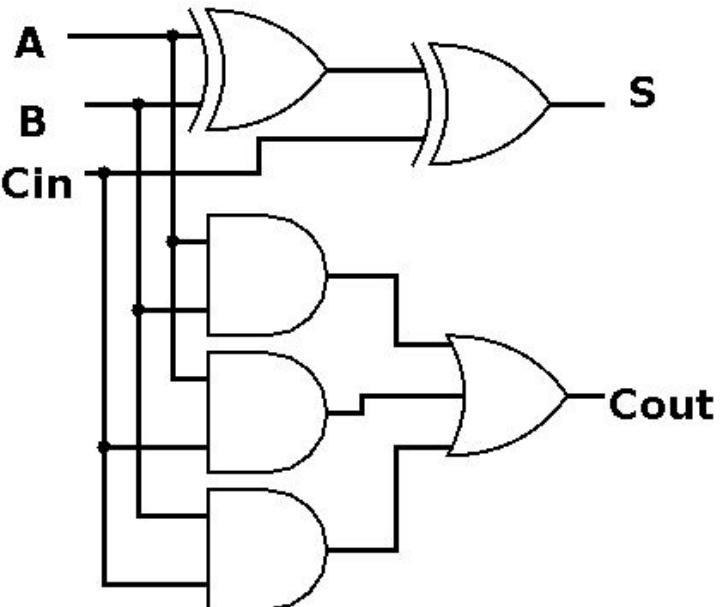
entrada		salida		
C_{IN}	A	B	C_{OUT}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



Sumador de 3 números de 1 bit

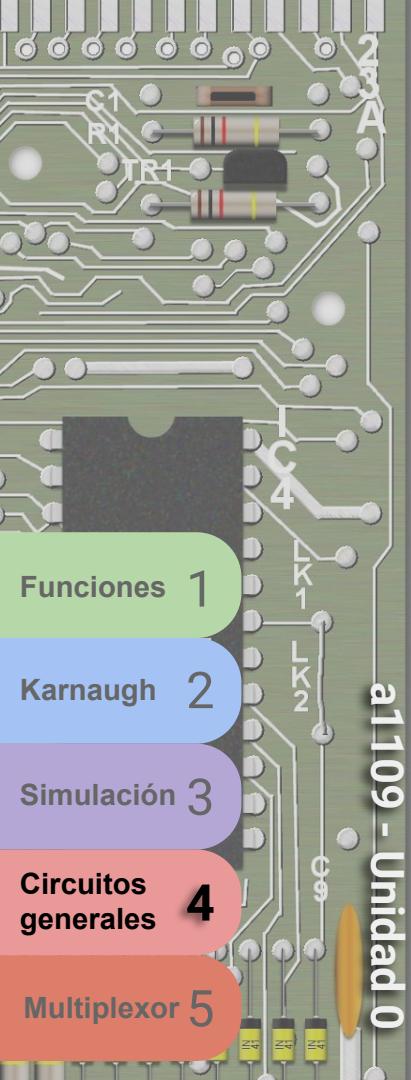
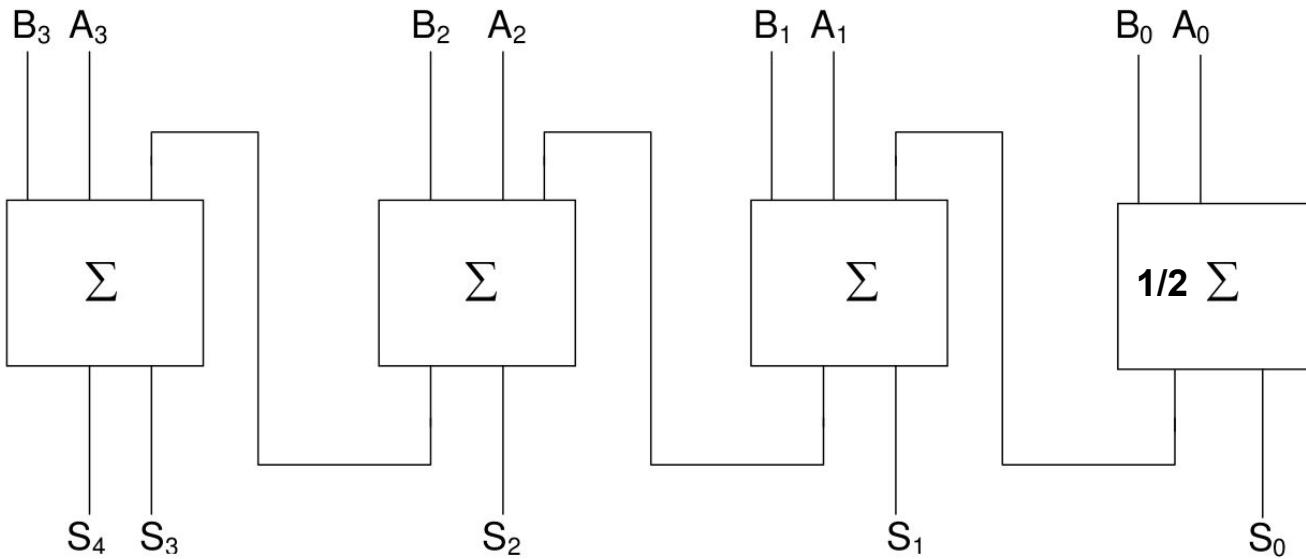
En esta implementación nos aseguramos que existan dos niveles de compuertas tanto para S como para C_{out} . Cuando veamos retardos veremos que es importante mantener los mismos retardos en todas las salidas.

entrada			salida	
C_{IN}	A	B	C_{OU}	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



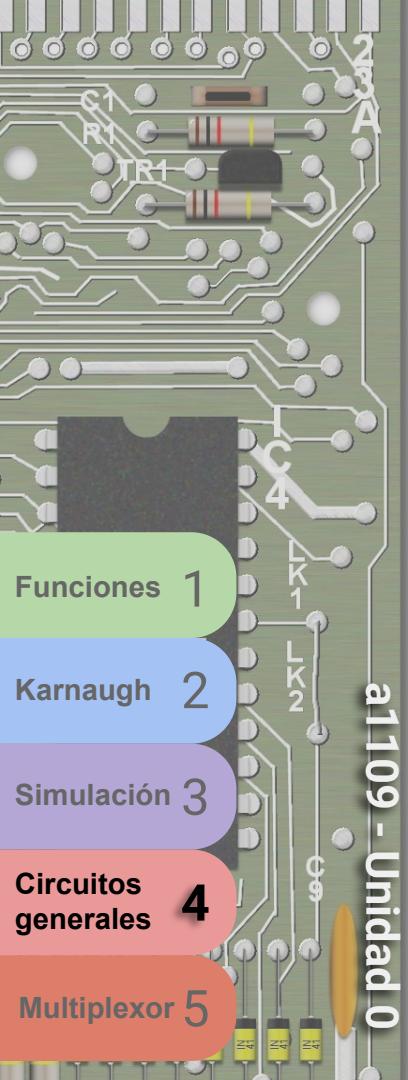
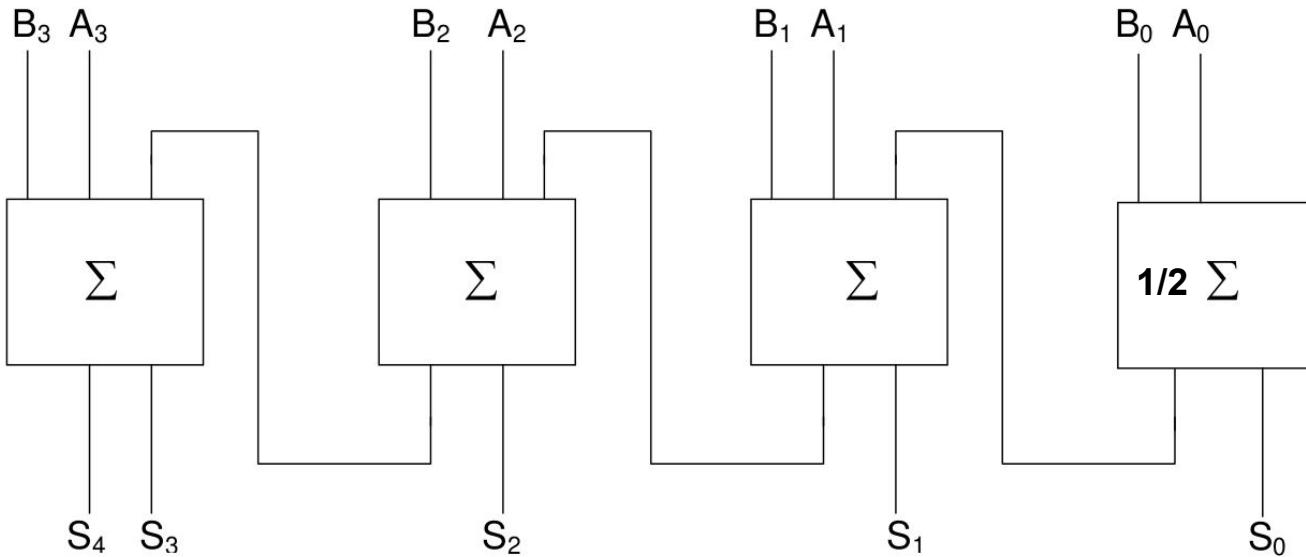
Encadenando sumadores

Si deseamos sumar 2 números de 4 bits, podemos tomar la salida de carry del primer sumador y conectarlo a la entrada de carry del siguiente, y así con el resto formando una cadena. S₄ en este caso es el carry de salida final.



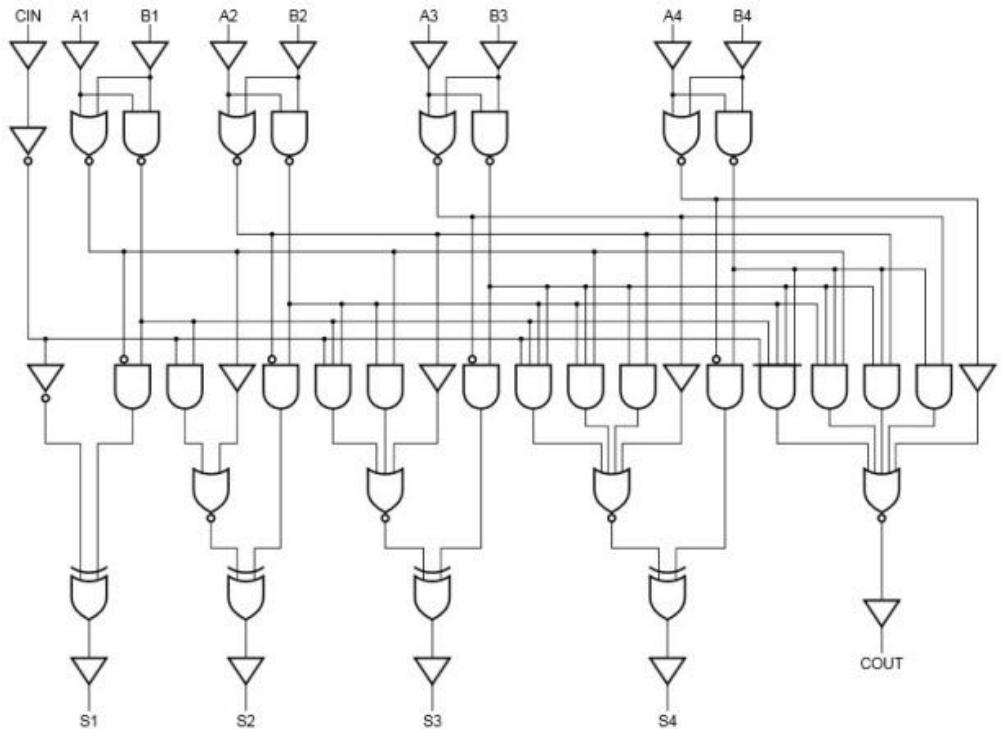
Encadenando sumadores

En la unidad de tecnología veremos que este encadenamiento implica tiempo. Debe resolverse la suma de los bits menos significativos antes de poder resolver los bits más significativos.

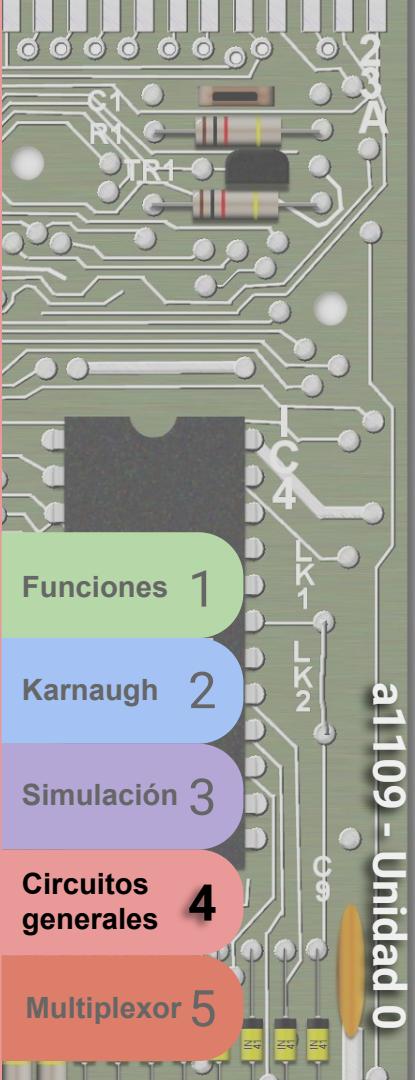


Encadenando sumadores

Existen sumadores con “carry look-ahead” los cuales anticipan el valor del carry en cada sumador.



<https://laptrinhx.com/carry-lookahead-adder-1766329680/>

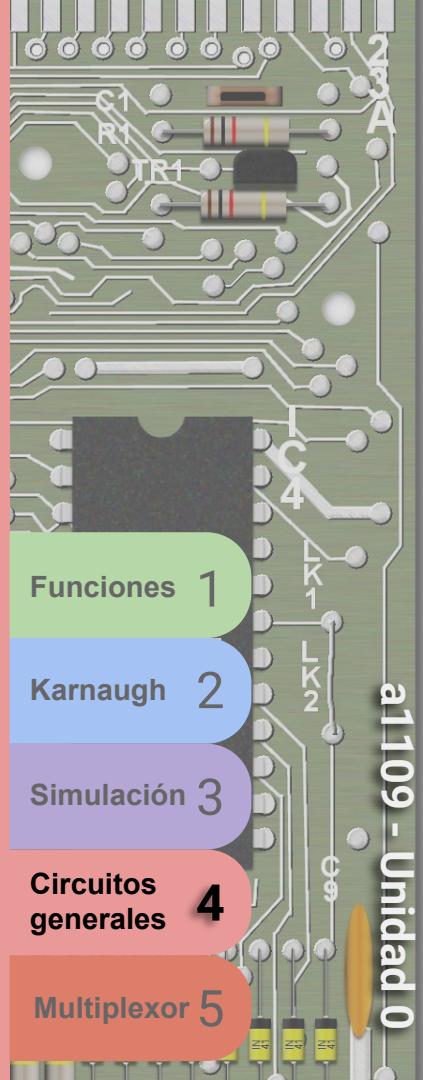


Restador de 3 números de 1 bit

Así como podemos armar un circuito que sume y considere un carry de entrada y uno de salida, podemos hacer el equivalente con la resta ($A - B$).

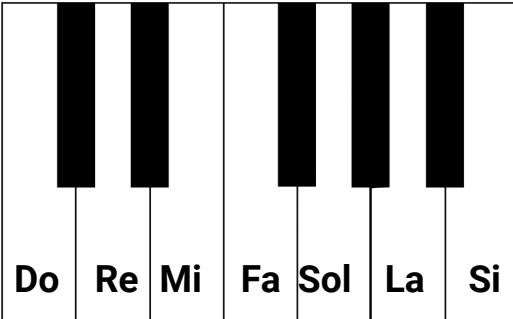
En este caso la señal de B_{IN} representa un préstamo (borrow) pedido por la etapa anterior, y B_{OUT} representa un pedido de préstamo a la etapa siguiente. Vemos que R es parecido a algo del Full-Adder.

entrada			salida	
A	B	B_{IN}	B_{OUT}	R
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	1	0
1	0	0	0	1
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

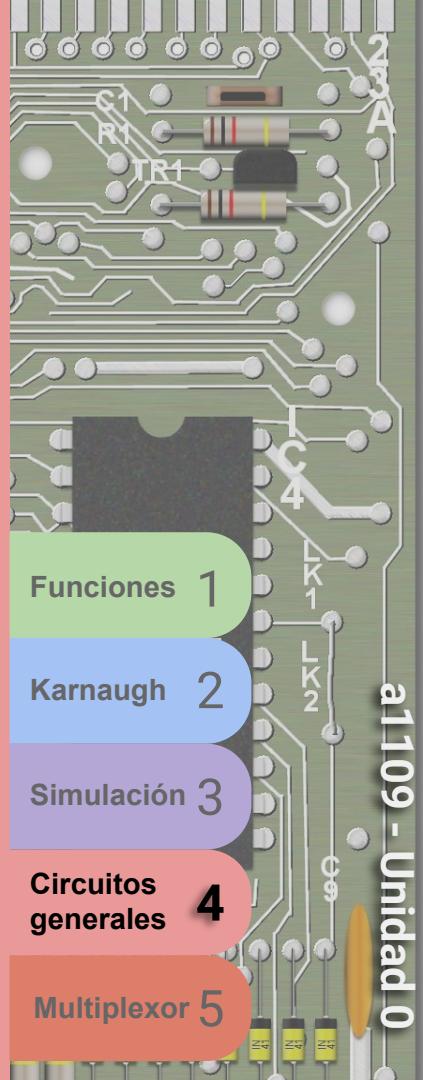


Codificador Simple

Un codificador es un circuito que toma una cantidad de entradas (por ejemplo las 7 teclas blancas del piano) y genera un número binario equivalente a la posición de la tecla apretada. No se permite presionar más de una tecla a la vez. Esto generaría una salida invalida.

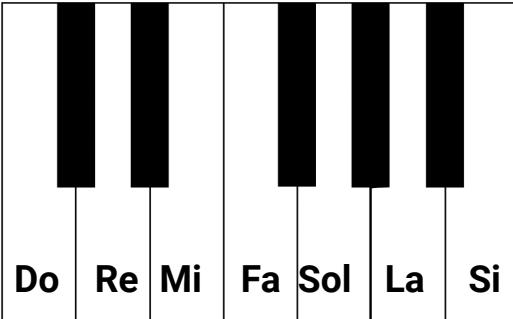


entrada							salida		
Do	Re	Mi	Fa	Sol	La	Si	S_2	S_1	S_0
0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	1
0	1	0	0	0	0	0	0	1	0
0	0	1	0	0	0	0	0	1	1
0	0	0	1	0	0	0	1	0	0
0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	1	1	1	1

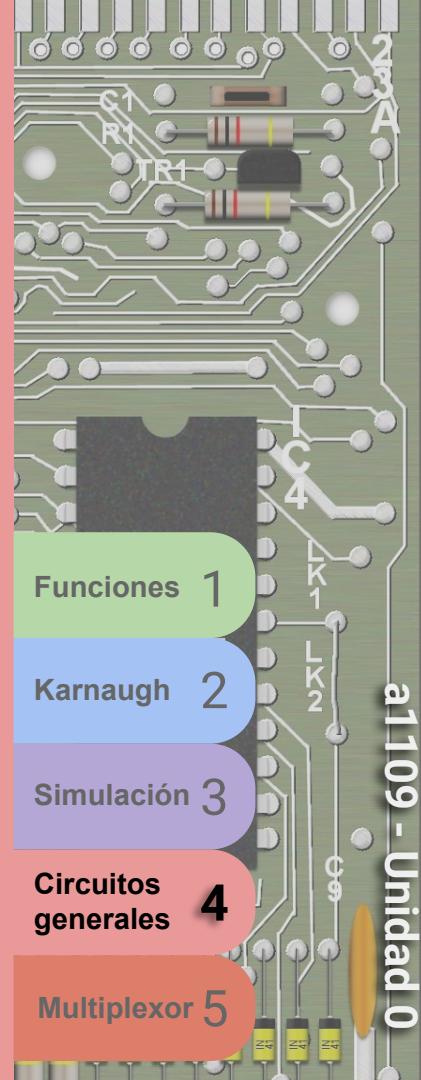


Codificador Simple

Vemos que cuando ninguna tecla se encuentra presionada (lo que generaría un 1 en la entrada) la salida es 000. Luego si Do se encuentra apretada la salida es 001, Re=010, Mi=011 Si=111. Vemos que este circuito posee 7 entradas, por ende tiene $2^7 = 128$ combinaciones en la tabla de verdad. Sin embargo solo algunas nos interesan.

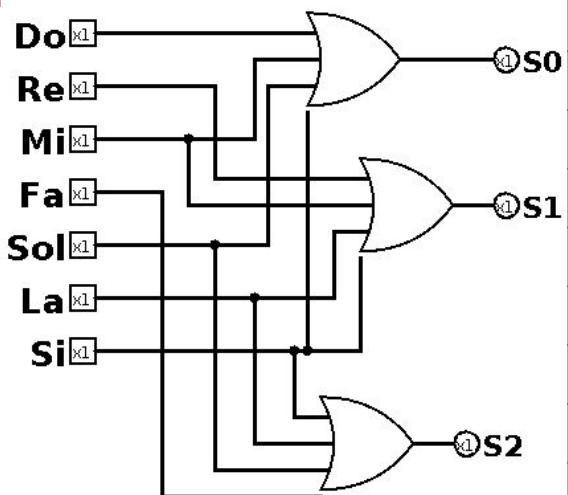


entrada							salida		
Do	Re	Mi	Fa	Sol	La	Si	S ₂	S ₁	S ₀
0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	1
0	1	0	0	0	0	0	0	1	0
0	0	1	0	0	0	0	0	1	1
0	0	0	1	0	0	0	1	0	0
0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	1	1	1	1

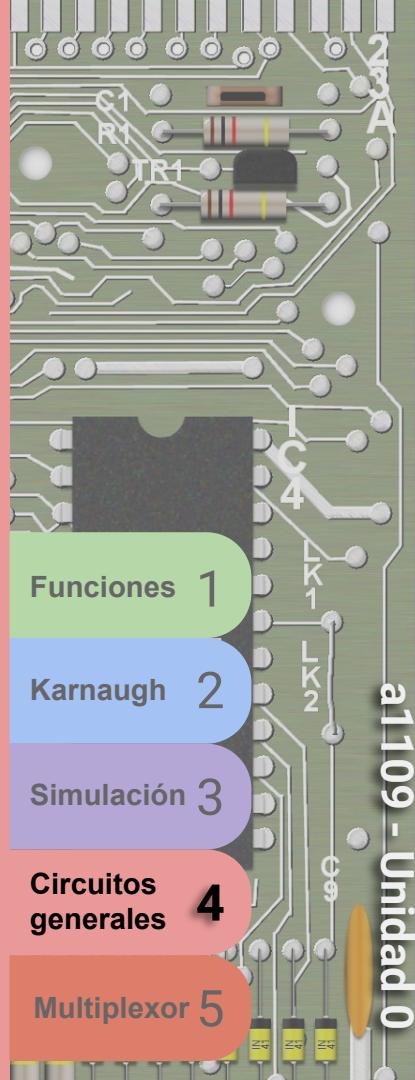


Codificador Simple

La implementación de este circuito consiste en compuertas OR que identifiquen los unos en las entradas que hacen que la salida sea 1. Ej: S_2 vale 1 siempre y cuando $Fa=1$ o $Sol=1$ o $La=1$ o $Si=1$. El problema ocurre cuando dos entradas se encuentran en uno a la vez.



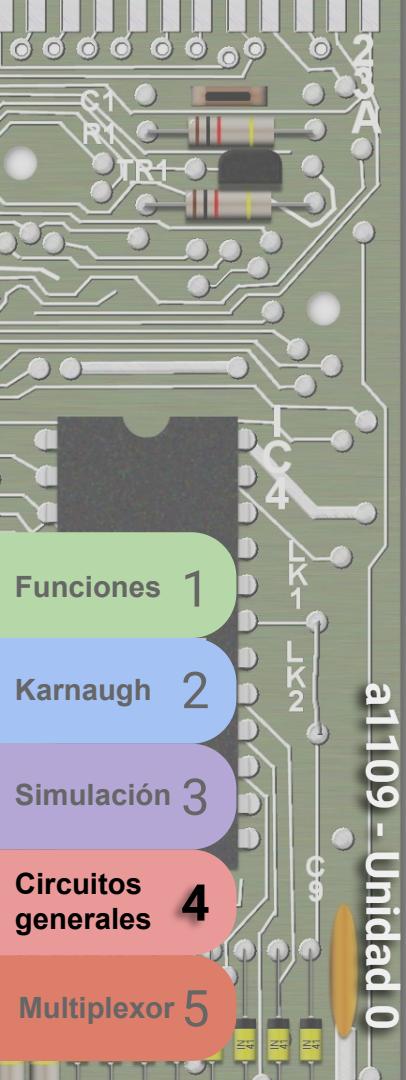
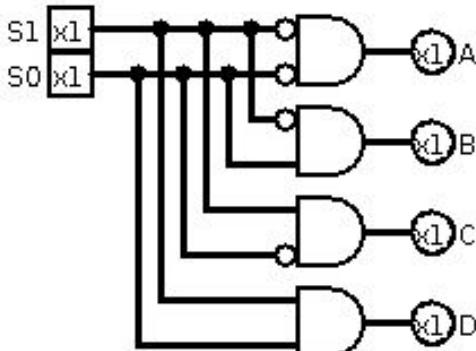
entrada							salida		
Do	Re	Mi	Fa	Sol	La	Si	S_2	S_1	S_0
0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	1
0	1	0	0	0	0	0	0	1	0
0	0	1	0	0	0	0	0	1	1
0	0	0	1	0	0	0	1	0	0
0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	1	1	1	1



Decodificador

El decodificador toma como entrada un número binario y genera una combinación de salida “one-hot”, o sea una combinación de salida donde solo una de las salidas tienen un uno. El decodificador es un circuito muy útil para transformar una dirección representada como un número binario de N bits en 2^N salidas donde solo una se encuentra en uno.

entrada		salida			
S_1	S_0	A	B	C	D
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1



Funciones 1

Karnaugh 2

Simulación 3

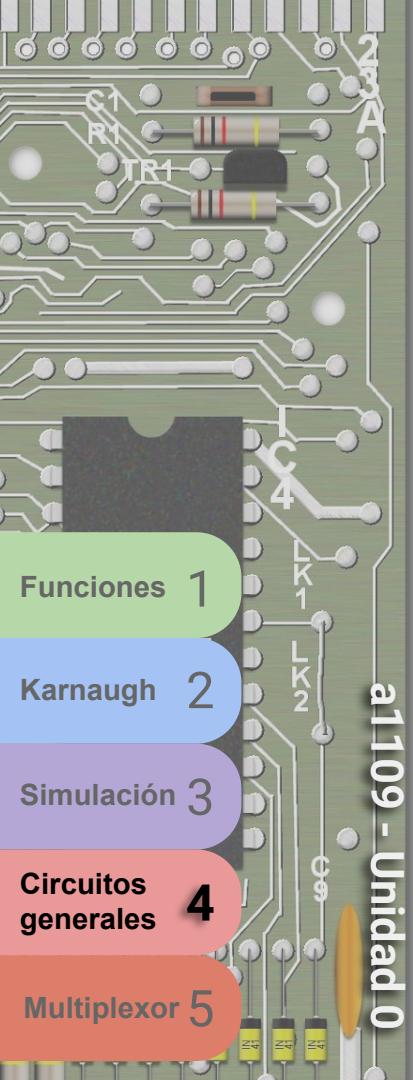
Circuitos generales 4

Multiplexor 5

Generador de Paridad

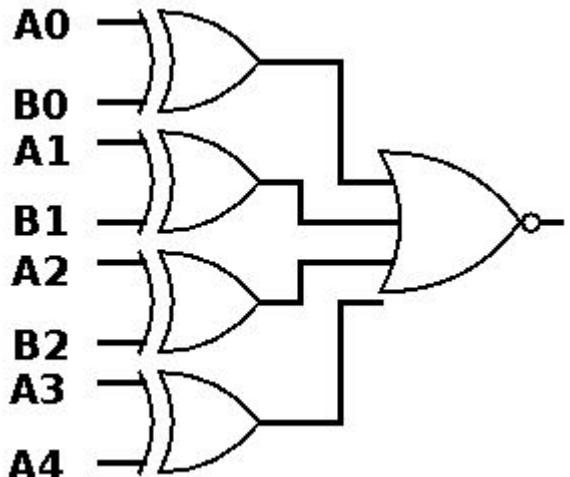
Vemos que la compuerta XOR calcula la paridad par en los unos para sus dos entradas. Cuando $A=B$ entonces la paridad es cero. Cuando $A \neq B$ entonces la paridad es 1. Nótese que si tenemos 3 entradas (o N) podemos ir encadenando XOR y tomando el resultado con el bit siguiente. Podemos transformar este generador de paridad en un validador de paridad tomando la salida P como entrada en un generador. En ese caso el resultado de P' del segundo generador deberá ser 0 si la paridad es válida.

A	B	XOR
0	0	0
0	1	1
1	0	1
1	1	0



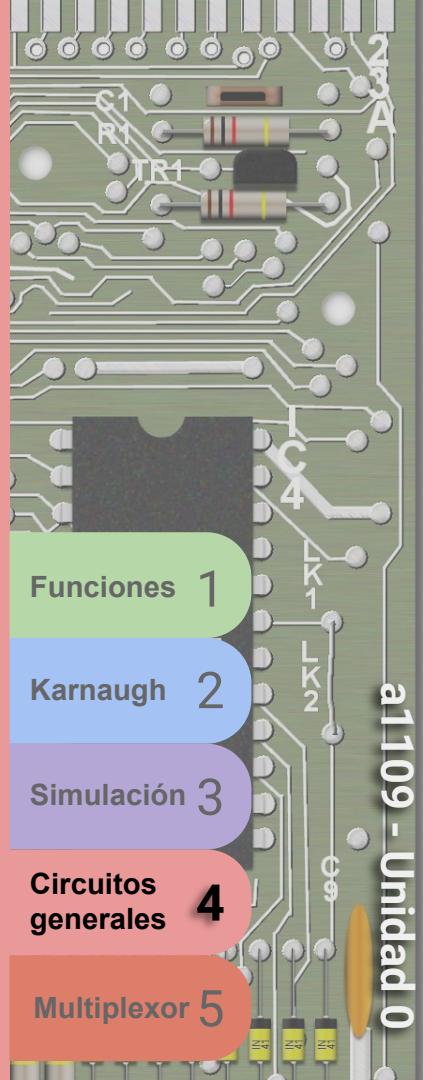
Comparador por Igual

Si disponemos de dos números (A y B) de 4 bits cada uno, de forma que $A=A_3A_2A_1A_0$ y $B=B_3B_2B_1B_0$, podemos restarlos y si el resultado es 0 saber si $A=B$. Pero viendo la tabla de verdad de la XOR, si $A=B$ entonces $A \text{ XOR } B = 0$. Luego con una NOR verificamos si alguna de las comparaciones bit a bit resultó en 1.

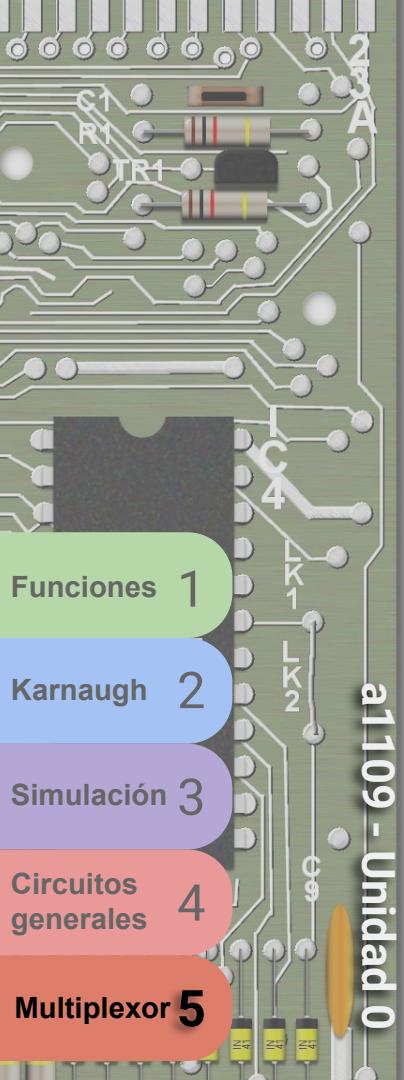


A	B	XOR
0	0	0
0	1	1
1	0	1
1	1	0

A	B	NOR
0	0	1
0	1	0
1	0	0
1	1	0

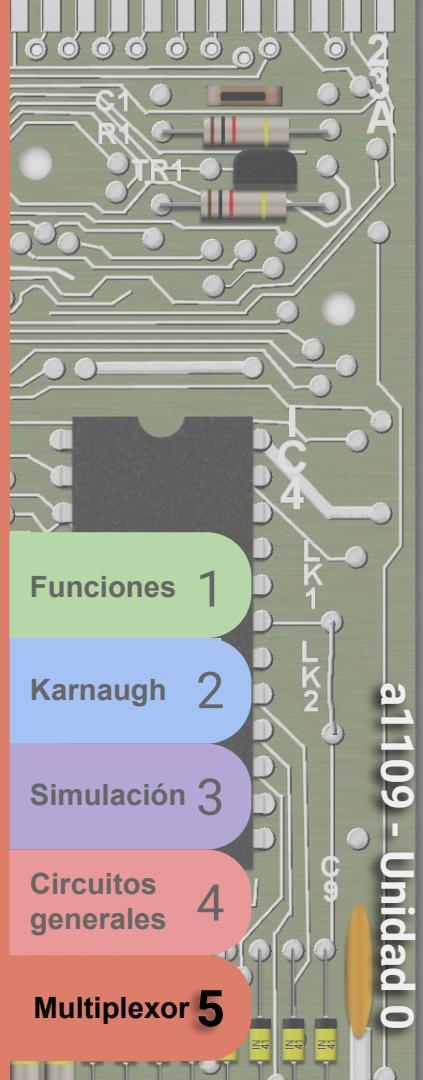
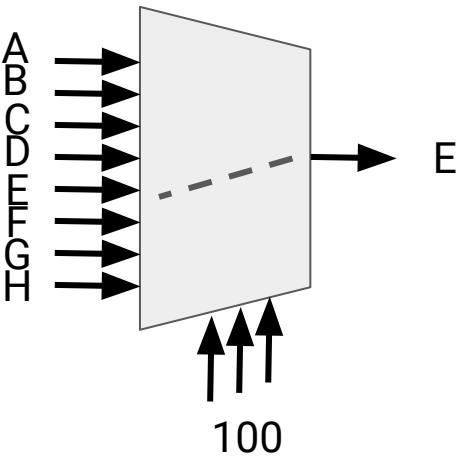
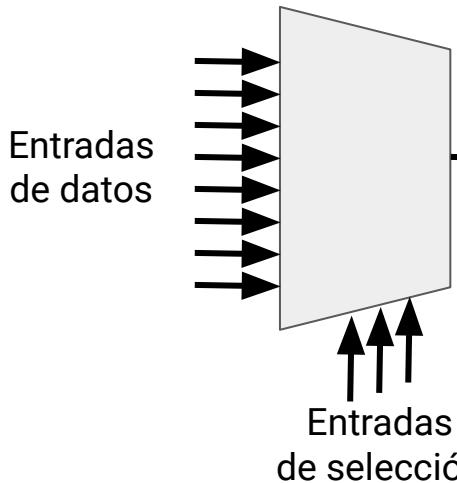


Multiplexores



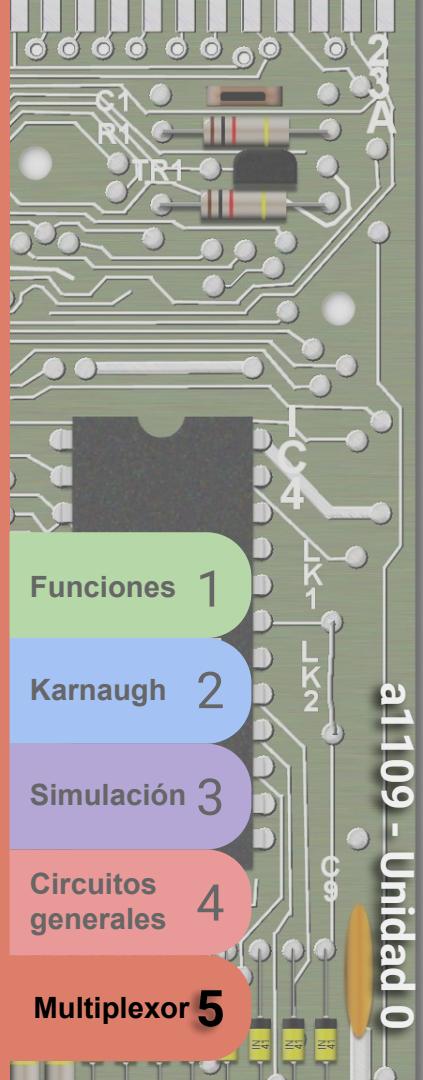
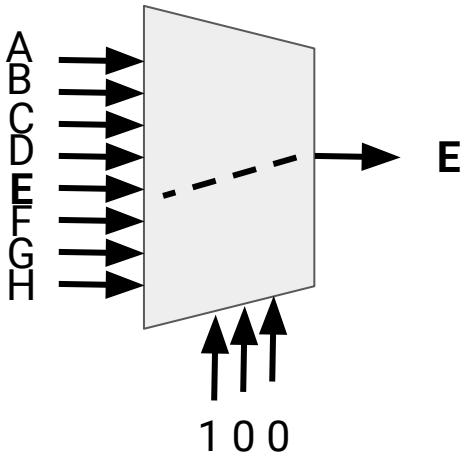
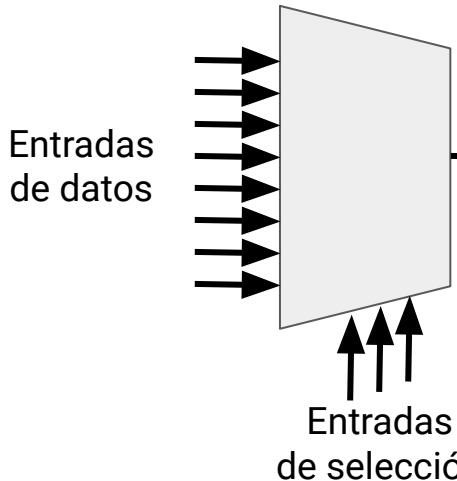
Multiplexores

El multiplexor es un circuito muy particular, ya que posee una salida pero dos tipos de entradas, las de selección y las de datos. Se lo representa de la siguiente forma, donde las entradas de datos se encuentran a su izquierda, la salida a la derecha y las entradas de selección por debajo.



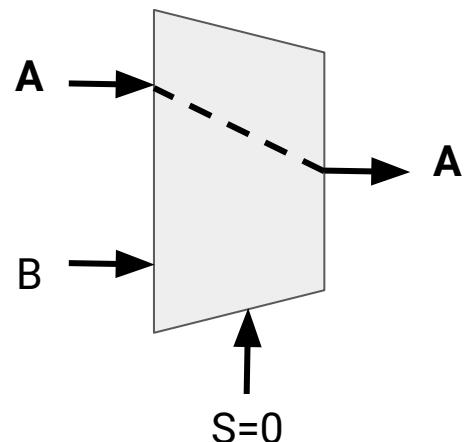
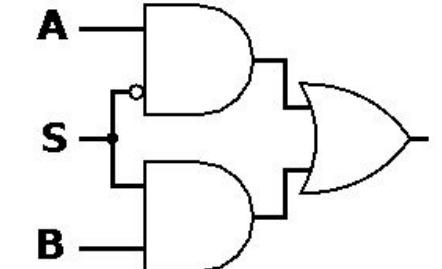
Multiplexores

El valor de la entrada de selección indica cuál de las entradas de datos está conectada a la salida. En este caso de MUX8 (8 entradas de datos) necesitamos $\log_2 8$ entradas de selección. Vemos en el ejemplo que si selección es 100 (4) se selecciona la entrada 4 (empezando de 0) y se la conecta a la salida.

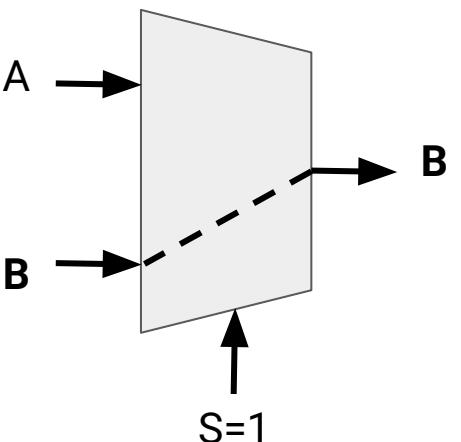


Multiplexores

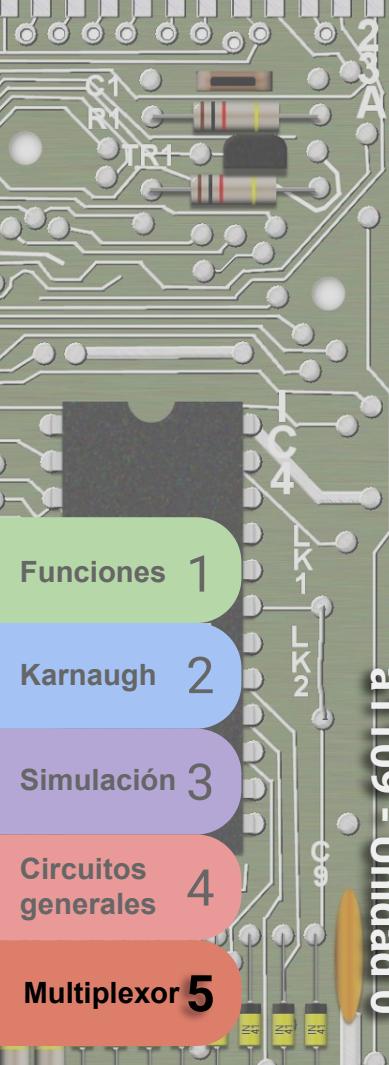
Vemos que el multiplexor más simple es aquel con dos entradas de datos y una de selección. Vemos que si $S=0$ entonces la salida queda conectada a la entrada A. Si $S=1$ entonces la salida queda conectada a la entrada B.



S	F
0	A
1	B

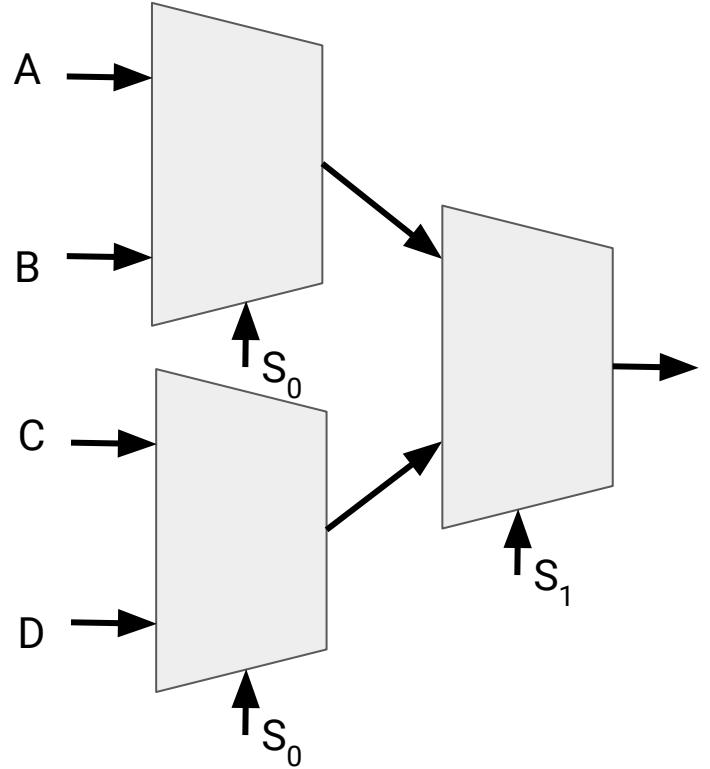


S	A	B	F
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

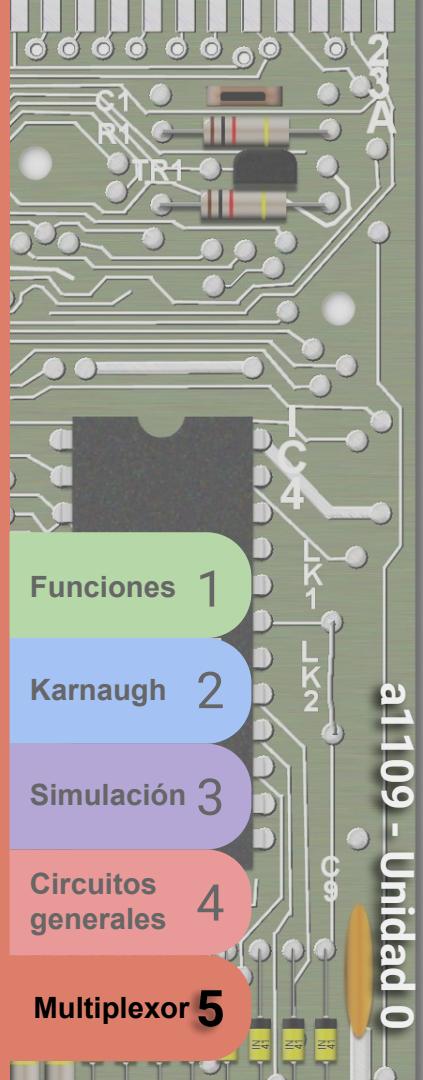
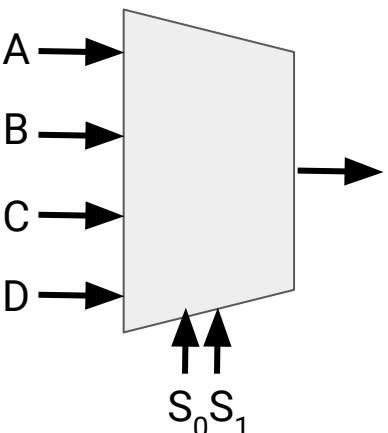


Multiplexores de órdenes mayores

Podemos construir multiplexores de órdenes mayores utilizando multiplexores más simples.

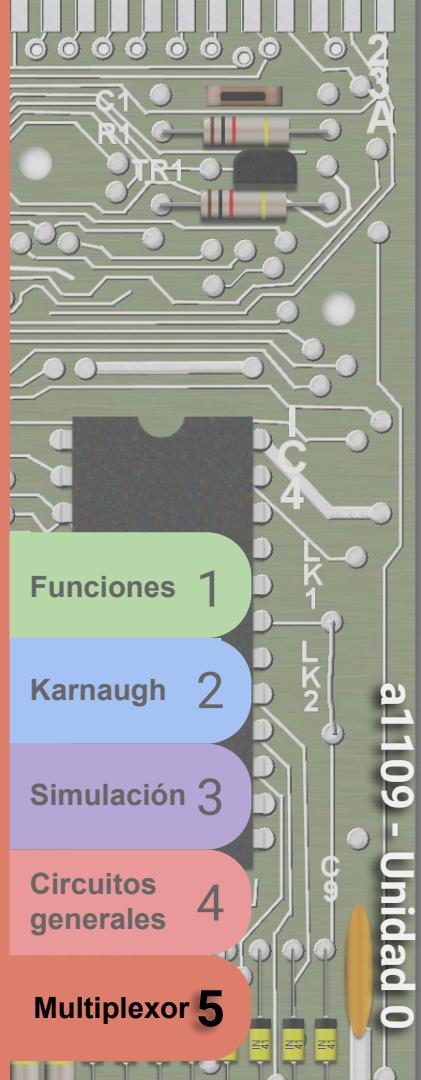
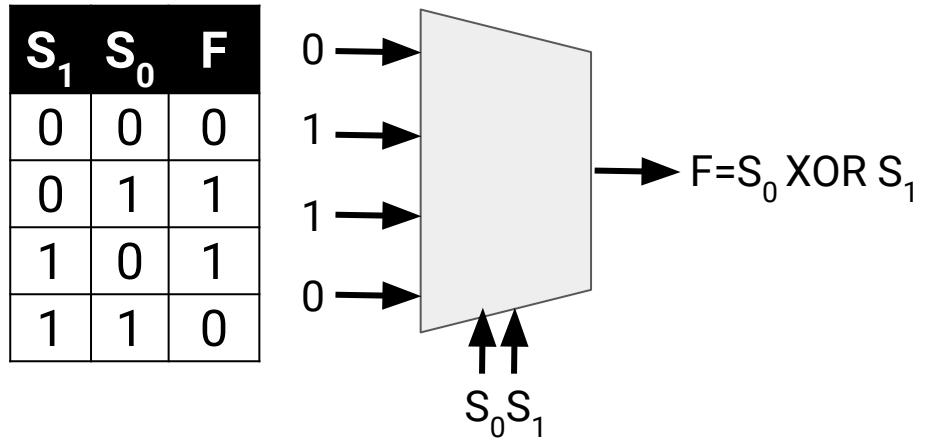


S_1	S_0	F
0	0	A
0	1	B
1	0	C
1	1	D



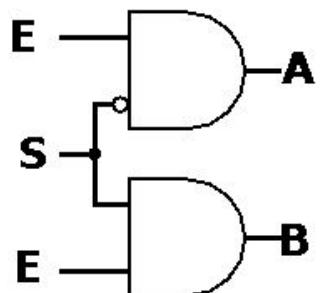
Funciones booleanas con multiplexores

Vemos que las entradas de datos pueden conectarse a variables, pero también pueden llevar valores constantes. Asignando valores fijos a A,B,C y D, podemos implementar cualquier función booleana de dos variables. Vemos por ejemplo cómo generar una compuerta XOR utilizando el MUX y entradas 0110.



Demultiplexor

Como contraparte del MUX, tenemos el demultiplexor. En este circuito tenemos una sola entrada de datos, N salidas y $\log_2 N$ entradas de selección. La idea es conectar la única entrada de datos a distintas salidas dependiendo de las líneas de selección. También puede implementarse en órdenes superiores.



S	E	A	B
0	0	0	0
0	1	1	0
1	0	0	0
1	1	0	1

S	A	B
0	E	0
1	0	E

