Organización del computador

Lógica digital

Organización

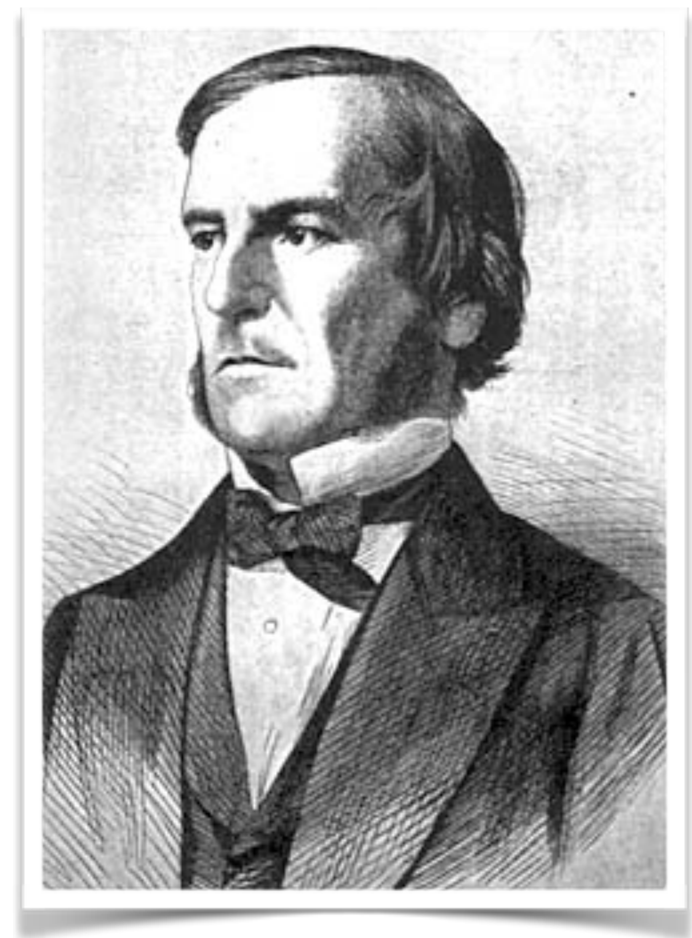
- Recordemos que la organización de un computador refiere al diseño específico de sus componentes, y como estas se coordinan para llevar a cabo una tarea
- Un factor decisivo en como se estructuran las componentes (y cómo estas están implementadas a través de circuitos) es cómo está representada la información
- Los sistemas modernos (a partir de The First Draft) utilizan el sistema binario (de donde proviene la palabra **bit** - **bi**nary digi**t**)

Lógica digital

- Los circuitos operan con dos valores eléctricos
- Pueden ser interpretados como 1 ó 0 y pensarlos como información
- Pueden ser interpretados como verdadero o falso y pensarlos como valores lógicos

Algebras de boole

George Boole



1815 — 1864

- Desarrolló una formalización algebraica en forma de cálculo que formaliza el razonamiento proposicional (Análisis matemático de la lógica)
- El sistema consiste es la declaración de operaciones algebraicas sobre valores booleanos y axiomas que formalizan su comportamiento

Lógica digital

Operadores booleanos

- Los operadores booleanos están descriptos por un conjunto de axiomas del estilo:

$$\frac{X}{X + Y} \qquad \frac{Y}{X + Y}$$

- También pueden ser interpretados como tablas de verdad
- Los operadores básicos del álgebra de boole son la conjunción o AND (conocido como producto), la disjunción u OR (conocido como coproducto) y la negación o NOT (conocido como complemento)

X AND Y

X	Y	XY
0	0	0
0	1	0
1	0	0
1	1	1

X OR Y

X	Y	X+Y
0	0	0
0	1	1
1	0	1
1	1	1

NOT X

X	$\overline{X}$
0	1
1	0

Lógica digital

Funciones booleanas

- Usando operadores booleanos se pueden definir funciones booleanas, por ejemplo:

  $f(X, Y, Z) = X.\overline{Z}+Y$

- Como es usual, el complemento tiene mayor precedencia seguido por el producto
- Las variables y variables negadas se denominan literales

X	Y	Z	X	.	$\overline{Z}$	+	Y
0	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0
0	1	0	0	0	1	1	1
0	1	1	0	0	0	1	1
1	0	0	1	1	1	1	0
1	0	1	1	0	0	0	0
1	1	0	1	1	1	1	1
1	1	1	1	0	0	1	1

- Las tablas de verdad se construyen: 1) se deben considerar todas las combinaciones posibles de valores para las variables, 2) se le asignan valores a los literales literales, y 3) se evalúan los operadores en orden de precedencia ($\overline{}$, ., + salvo que haya paréntesis indicando lo contrario) usando las tablas de cada uno

Lógica digital

Identidades booleanas

Identidad	$1.A=A$	$0+A=A$
Neutro	$0.A=0$	$1+A=1$
Idempotencia	$A.A=A$	$A+A=A$
Inversa	$A.\overline{A}=0$	$A+\overline{A}=1$
Conmutativa	$A.B=B.A$	$A+B=B+A$
Asociativa	$(A.B)C=A.(B.C)$	$(A+B)+C=A+(B+C)$
Distributiva	$A+B.C=(A+B).(A+C)$	$A.(B+C)=A.B+A.C$
Absorción	$A.(A+B)=A$	$A+A.B=A$
De Morgan	$\overline{A.B} = \overline{A}+\overline{B}$	$\overline{A+B} = \overline{A}.\overline{B}$

Lógica digital

Identidades booleanas

- Usando las identidades presentadas anteriormente es posible reducir una expresión booleana:

$$f(X, Y, Z) = (X+Y)(X+\overline{Y})\overline{X\overline{Z}}$$

$(X+Y)(X+\overline{Y})(\overline{X}+Z) =$	DeMorgan
$(XX + X\overline{Y}+YX+Y\overline{Y})(\overline{X}+Z) =$	Distributiva
$(X + X\overline{Y}+YX + 0)\ (\overline{X}+Z) =$	Indempotencia e Inversa
$(X + X(\overline{Y}+Y))\ (\overline{X}+Z) =$	Nula y Distributiva
$(X)\ (\overline{X}+Z) =$	Inversa, Identidad y Nula
$X\overline{X}+XZ =$	Distributiva
XZ	Inversa e Identidad

Lógica digital

Fórmulas equivalente

- El ejemplo anterior muestra que muchas fórmulas, a pesar de ser sintácticamente diferentes poseen la misma tabla de verdad revelando que son equivalentes
- En general son preferibles las formas normales, aun cuando pueden no ser la forma más compacta de denotar una cierta fórmula:
  - CNF (Conjuntive Normal Form) o producto de sumas:
    $f(X, Y, Z) = (X+Y)(X+Z)(X+Y+Z)$
  - DNF (Disjuntive Normal Form) o suma de productos:
    $f(X, Y, Z) = XY+XZ$

Lógica digital

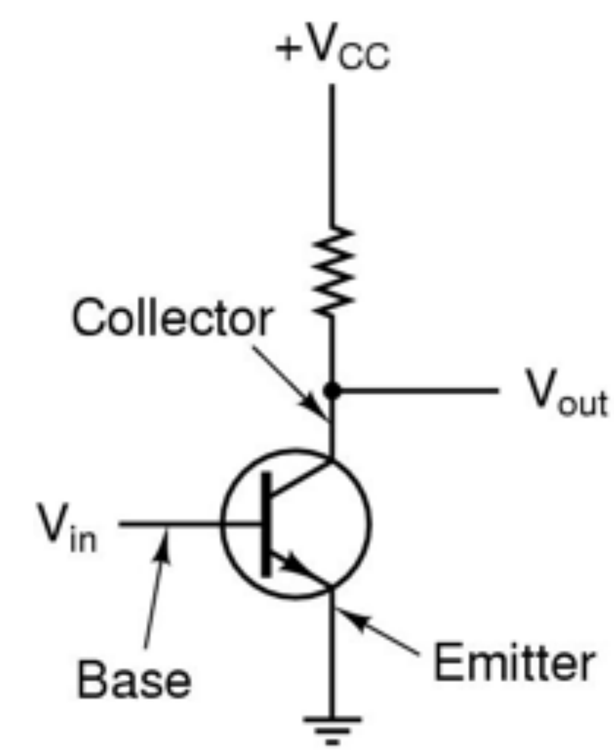
Fórmulas equivalente

X	Y	Z	X	.	$\overline{Z}$	+	Y
0	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0
0	1	0	0	0	1	1	1
0	1	1	0	0	0	1	1
1	0	0	1	1	1	1	0
1	0	1	1	0	0	0	0
1	1	0	1	1	1	1	1
1	1	1	1	0	0	1	1

- Convertir un fórmula cualquiera a **suma de productos** usando su tablada verdad es sencillo
- Se toman las filas que evalúan a 1 y luego se describe la fórmula a través de los valores de entrada de dichas filas:
  $f(X, Y, Z) = \overline{X}Y\overline{Z}+\overline{X}YZ+X\overline{Y}\overline{Z}+XY\overline{Z}+XYZ$
- Esta forma de codificar funciones booleanas muestra que solo basta $\overline{\ \ }$, ., + para escribir cualquier resultado en función de las variables de entrada

Circuitos booleanos

- Los **circuitos electrónicos** implementan funciones booleanas y mientras más simple la función, más pequeño será el circuito siendo a su vez más barato, con menor consumo y hasta más rápido. El álgebra de boole nos permitirá la reducción de circuitos
- Los circuitos electrónicos están formados por **compuertas** que son dispositivos electrónicos que producen un resultado en función de su entrada
- Una compuerta está formada por uno o más **transistores**



Circuitos booleanos

- Los **circuitos electrónicos** implementan funciones booleanas y mientras más simple la función, más pequeño será el circuito siendo a su vez más barato, consumo y hasta más rápido. El álgebra de permitirá la reducción de circuitos
- Los circuitos electrónicos están formados por **compuertas** que son dispositivos electrónicos que producen un resultado en función de su entrada
- Una compuerta está formada por uno o más **transistores**

Compuertas lógicas

- Las compuertas más simples se corresponden exactamente con los operadores booleanos elementales que vimos anteriormente:

X AND Y

X	Y	XY
0	0	0
0	1	0
1	0	0
1	1	1

X OR Y

X	Y	X+Y
0	0	0
0	1	1
1	0	1
1	1	1

NOT X

X	$\overline{X}$
0	1
1	0

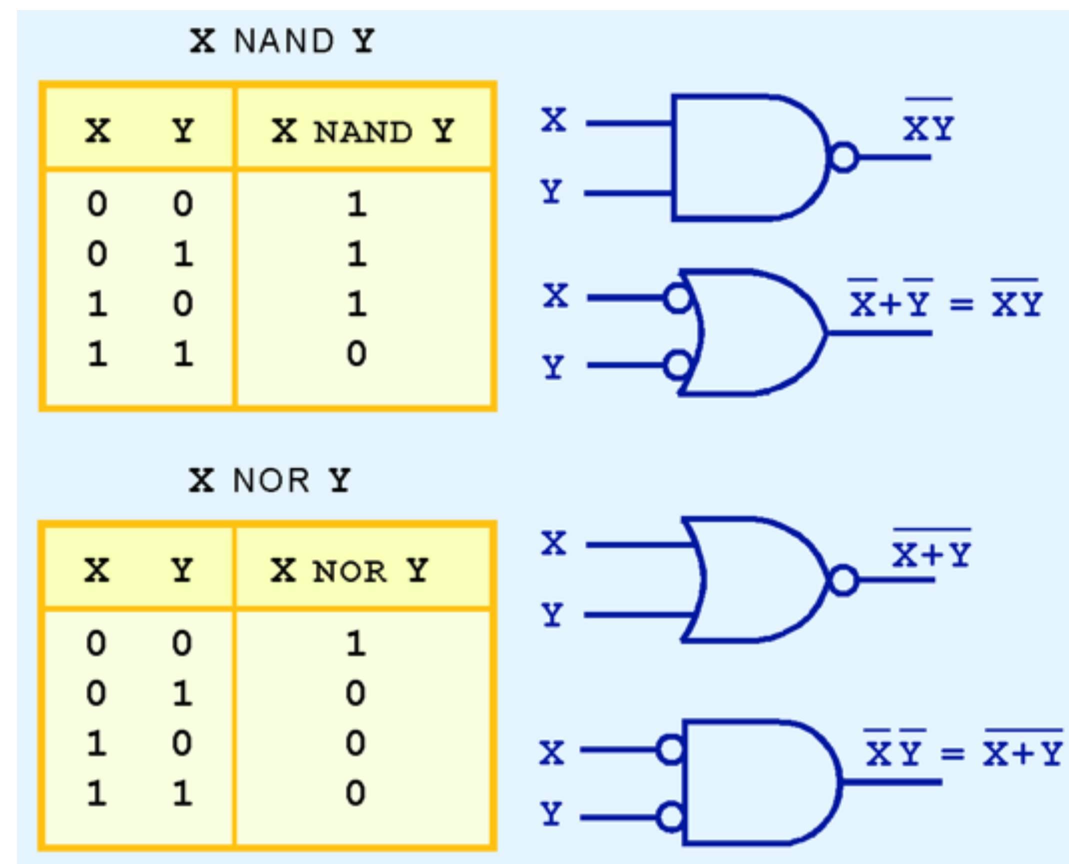
- Una compuerta de gran utilidad es el OR exclusivo XOR

X XOR Y

X	Y	$X \oplus Y$
0	0	0
0	1	1
1	0	1
1	1	0

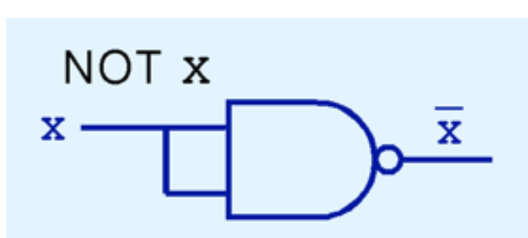
Compuertas lógicas

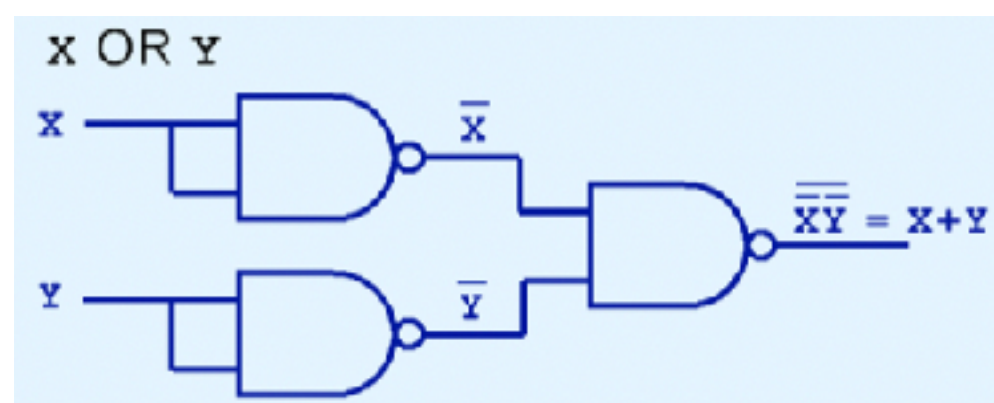
- Dos compuertas de mucha utilidad son las compuertas NAND y NOR
- Resultan sumamente baratas y por sí solas son un conjunto adecuado de operadores booleanos (ver que con ellas se puede implementar NOT y OR) y por lo tanto son suficiente para implementar todos los operadores restantes



Compuertas lógicas

- Dos compuertas de mucha utilidad son las compuertas NAND y NOR
- Resultan sumamente baratas y por sí solas son un conjunto adecuado de operadores booleanos (ver que con ellas se puede implementar NOT y OR) y por lo tanto son suficiente para implementar todos los operadores restantes

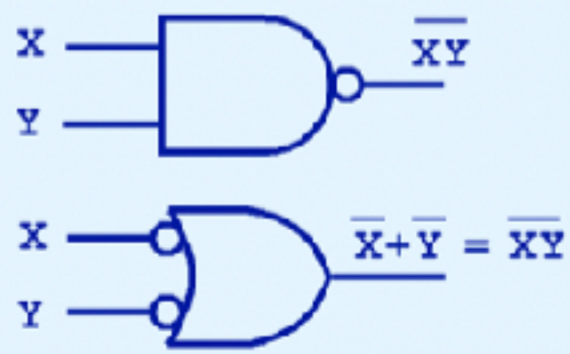




Compuertas lógicas

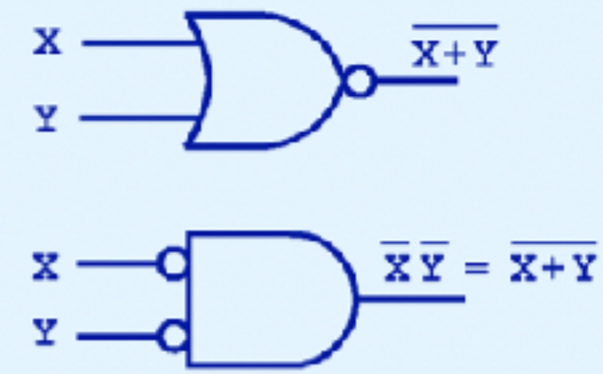
X NAND Y

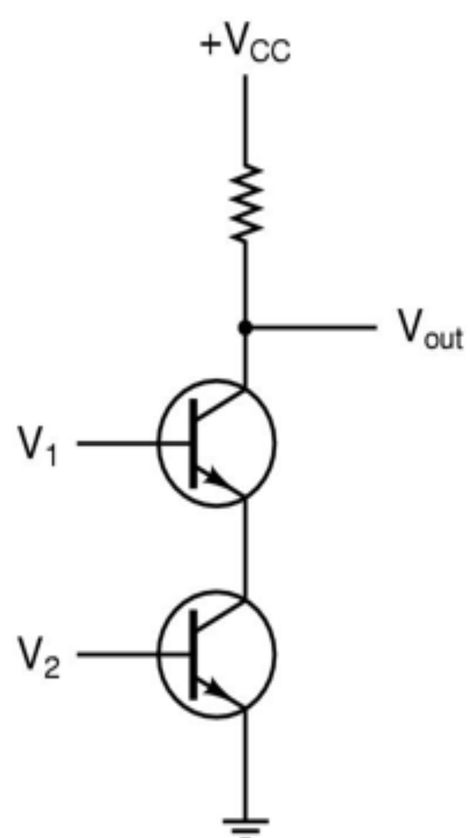
X	Y	X NAND Y
0	0	1
0	1	1
1	0	1
1	1	0

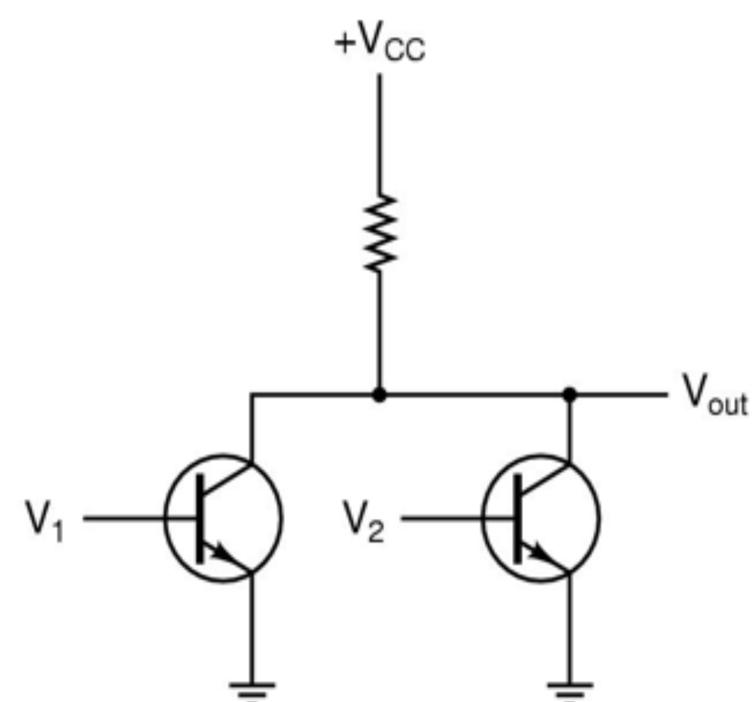


X NOR Y

X	Y	X NOR Y
0	0	1
0	1	0
1	0	0
1	1	0

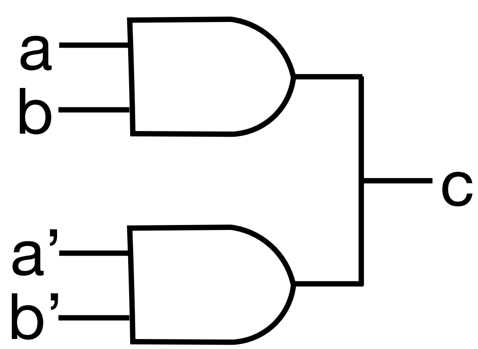






Three-state logic

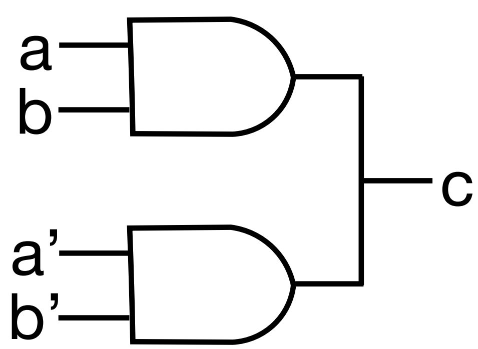
- Ahora bien ¿Cómo se conectan compuertas para hacer circuitos?



a	b	a'	b'	c	
0	0	0	0	0	
0	0	0	1	0	
0	0	1	0	0	
0	0	1	1	☠	corto circuito
0	1	0	0	0	
0	1	0	1	0	
0	1	1	0	0	
0	1	1	1	☠	corto circuito

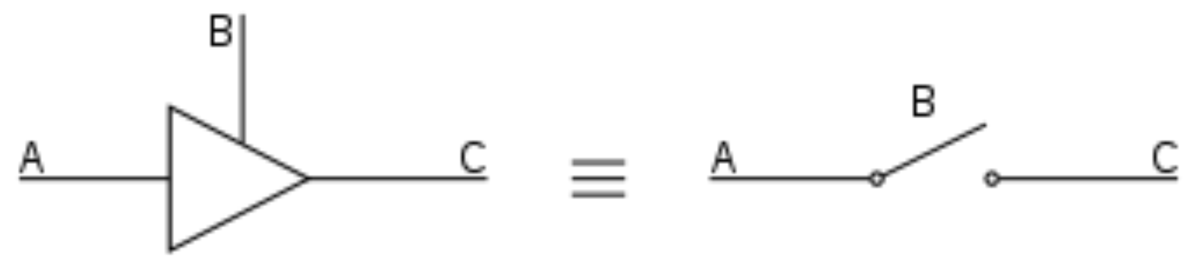
Three-state logic

- Ahora bien ¿Cómo se conectan compuertas para hacer circuitos?



a	b	a’	b’	c	
0	0	0	0	0	
0	0	0	1	0	
0	0	1	0	0	
0	0	1	1	☠	corto circuito
0	1	0	0	0	
0	1	0	1	0	
0	1	1	0	0	
0	1	1	1	☠	corto circuito

Three-state buffer



<table>
<tr><th>A</th><th>B</th><th>C</th></tr>
<tr><td>X</td><td>0</td><td>hi-Z</td></tr>
<tr><td>0</td><td rowspan="2">1</td><td>0</td></tr>
<tr><td>1</td><td>1</td></tr>
</table>

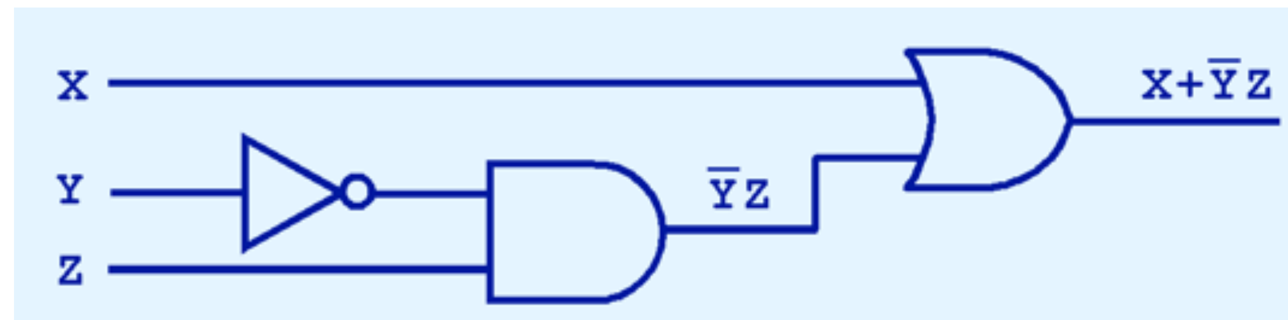
hi-Z distingue un estado de un circuito en el que no es posible observar ninguno de los estados lógicos 0 ó 1.

La utilización de three-state buffers permite que varios dispositivos compartan una misma salida si se tiene la precaución de que solo uno de estos tenga habilitada la salida.

Componentes digitales

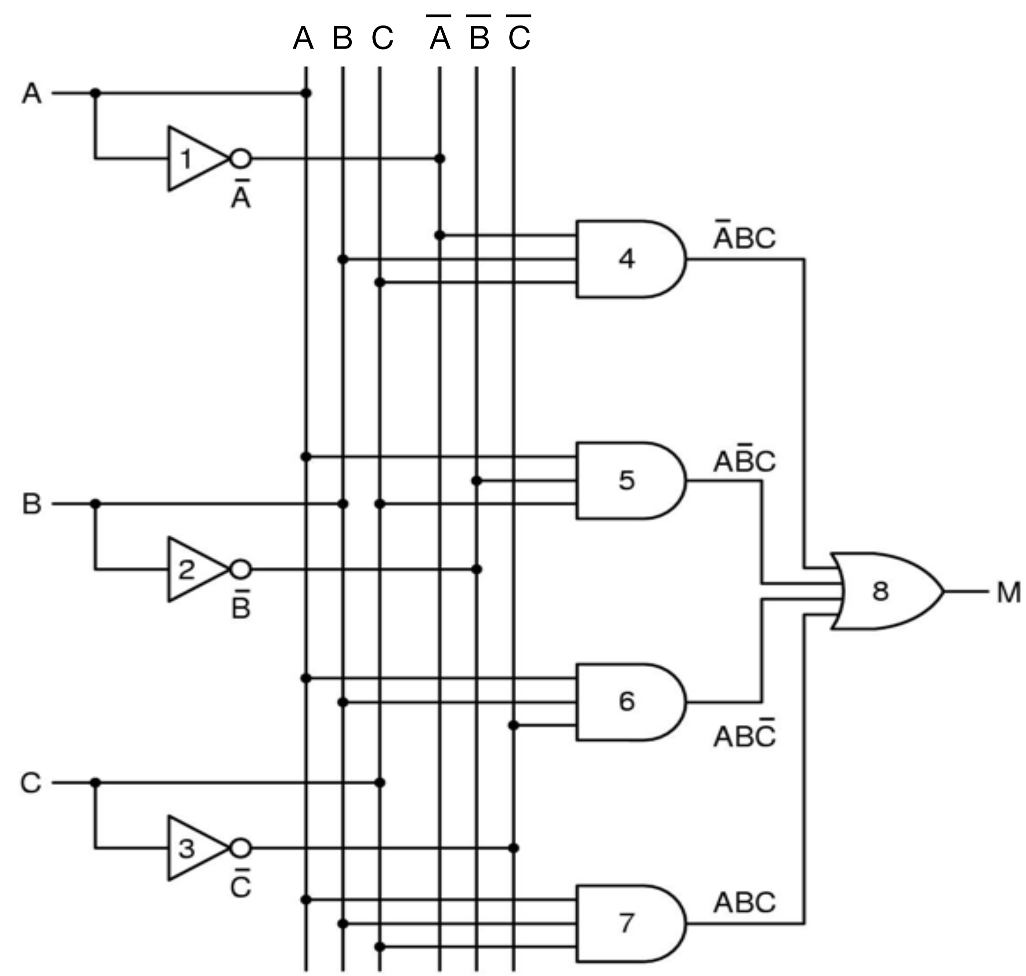
- Combinando compuertas se pueden implementar funciones booleanas a partir de interpretar los operadores como compuertas electrónicas:

$$F(X,Y,Z) = X+\overline{Y}Z$$



$M(A, B, C) = \overline{A}BC+A\overline{B}C+AB\overline{C}+ABC$

A	B	C	M
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



Circuitos combinatorios

- Implementan funciones booleanas pues su resultado está determinado por os valores presentes en las entradas del circuito
- El tiempo de respuesta es “instantáneo” (existe un tiempo de propagación de la corriente eléctrica a través de la compuerta pero suele ser despreciable en la enorme mayoría de los casos)
- La aritmética y la lógica de una CPU está implementada con circuitos combinatorios

Circuitos combinatorios

Half adder

- ¿Cómo se construye un circuito que sume dos bits?
- f(X, Y) = X + Y
- ¿Y si hay acarreo?

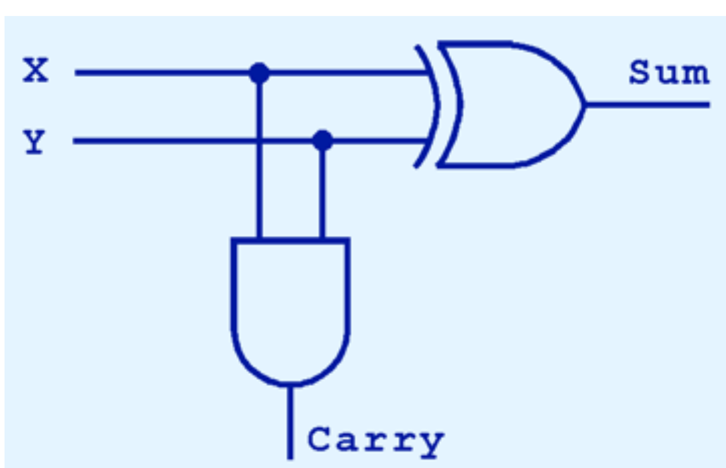
Inputs		Outputs	
X	Y	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

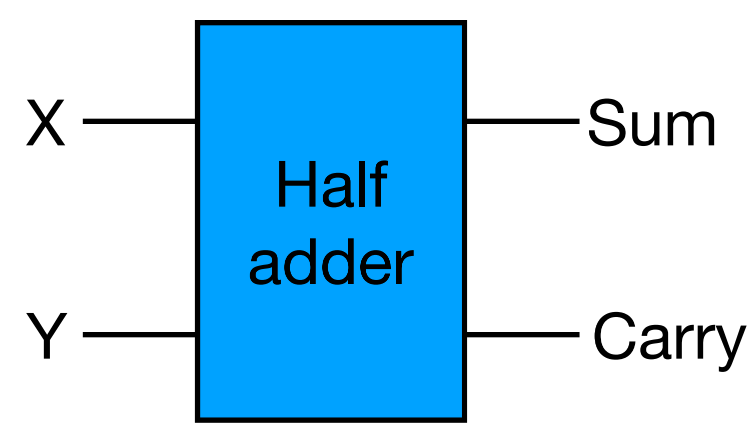
Circuitos combinatorios

Half adder

- ¿Cómo se construye un circuito que sume dos bits?
- f(X, Y) = X + Y
- ¿Y si hay acarreo?

Inputs		Outputs	
X	Y	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1





Circuitos combinatorios

Full adder

- Si debemos sumar números de más de 1 bit es necesario que el adder pueda aceptar el acarreo de los bits anteriores ¿Cómo se construye un circuito que sume dos bits?

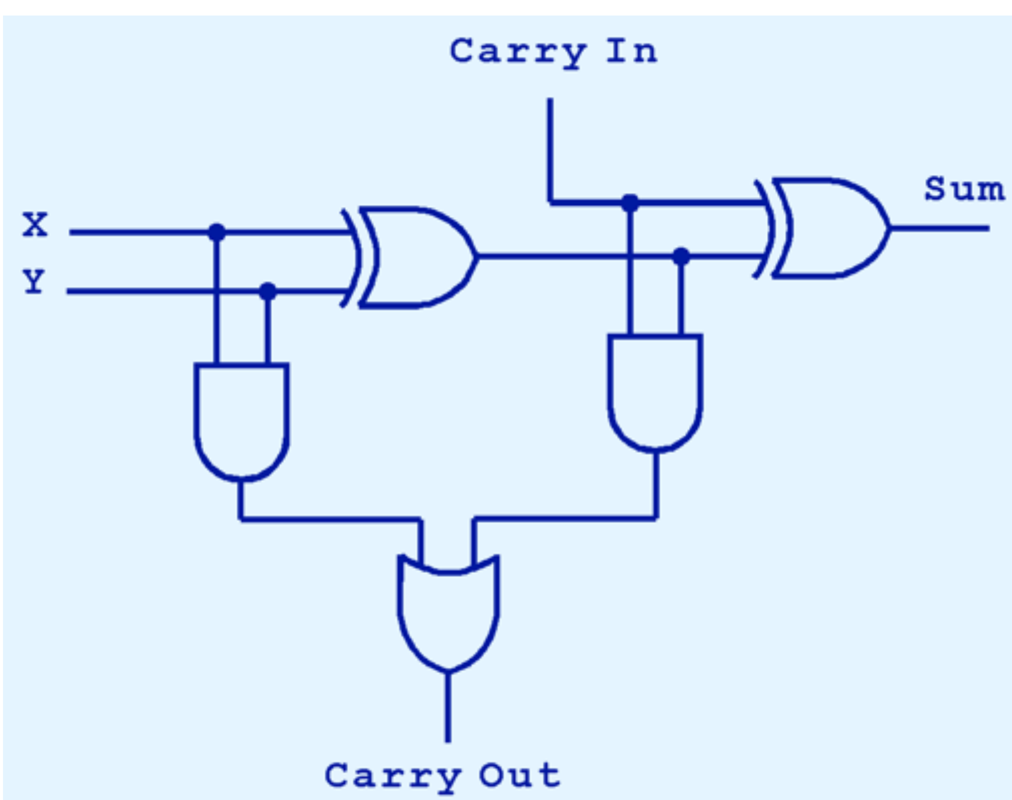
Inputs			Outputs	
X	Y	Carry In	Sum	Carry Out
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

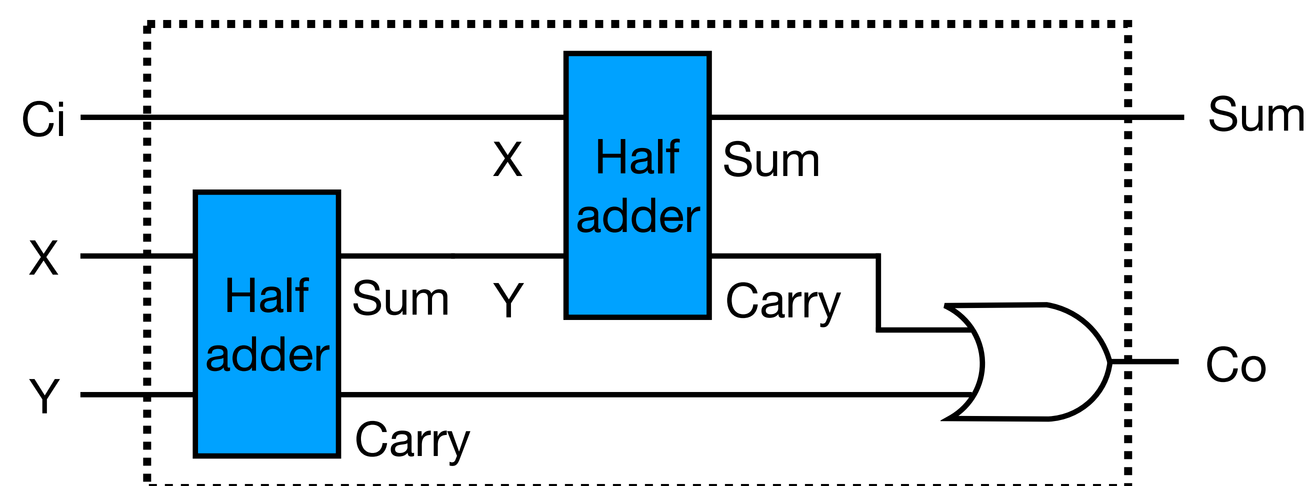
Circuitos combinatorios

Full adder

- ->Si debemos sumar números de más de 1 bit es necesario que el adder pueda aceptar el acarreo de los bits anteriores ¿Cómo se construye un circuito que sume dos bits?

Inputs			Outputs	
X	Y	Carry In	Sum	Carry Out
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



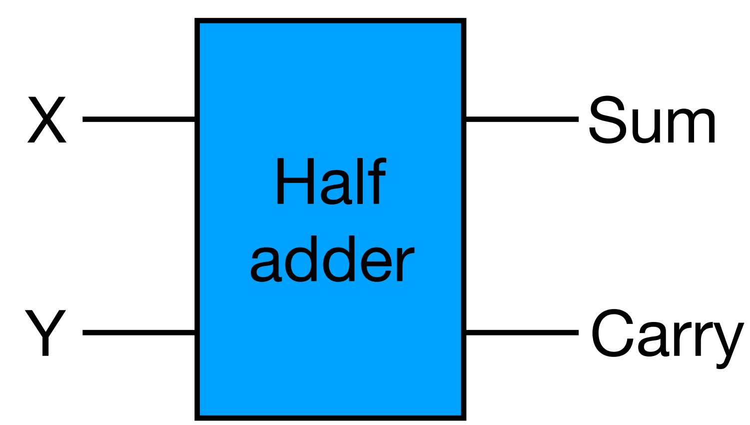


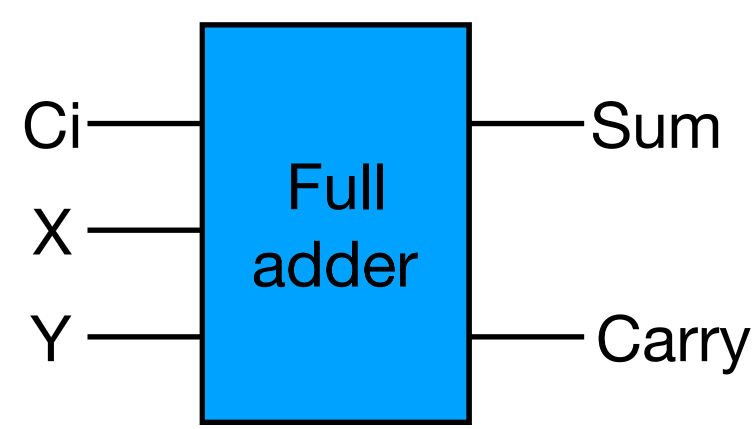
Circuitos combinatorios

Adder

- ¿Será posible diseñar un sumador de números binarios (Adder) con Full adders y Half adders?

$$\begin{array}{r} A_4A_3A_2A_1 \\ B_4B_3B_2B_1 \\ \hline C_5C_4C_3C_2C_1 \end{array}$$

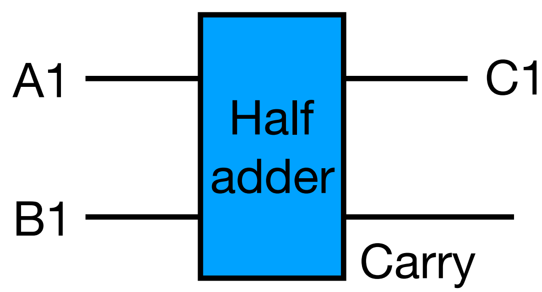




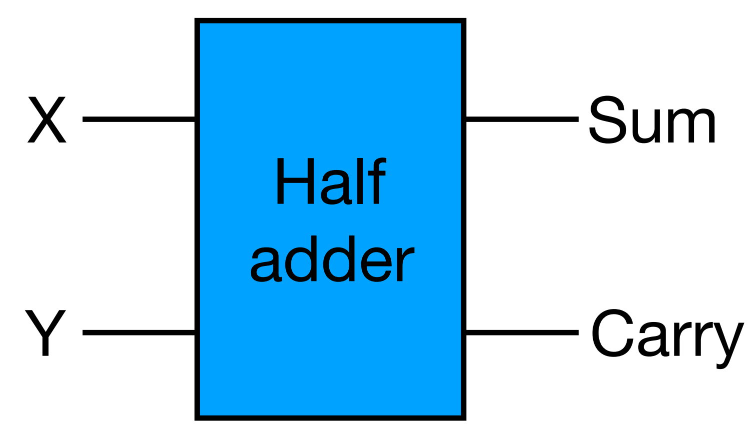
Circuitos combinatorios

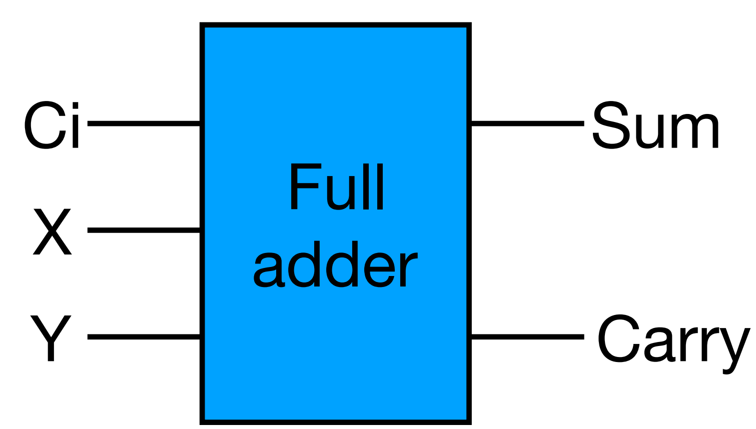
Adder

- ¿Será posible diseñar un sumador de números binarios (Adder) con Full adders y Half adders?



$$\begin{array}{r} A_4A_3A_2A_1 \\ B_4B_3B_2B_1 \\ \hline C_5C_4C_3C_2C_1 \end{array}$$

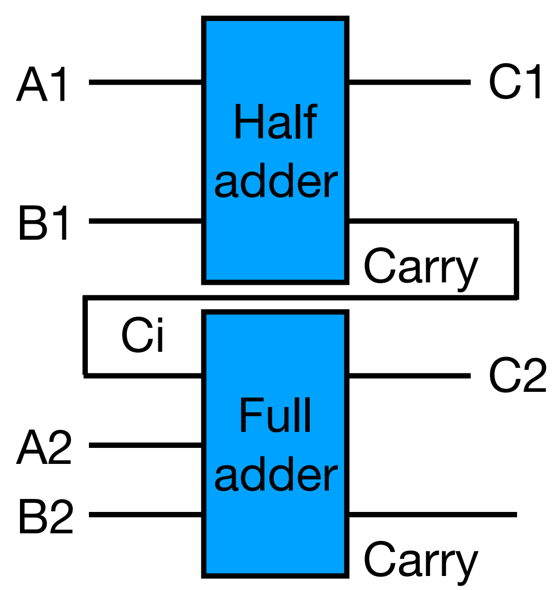




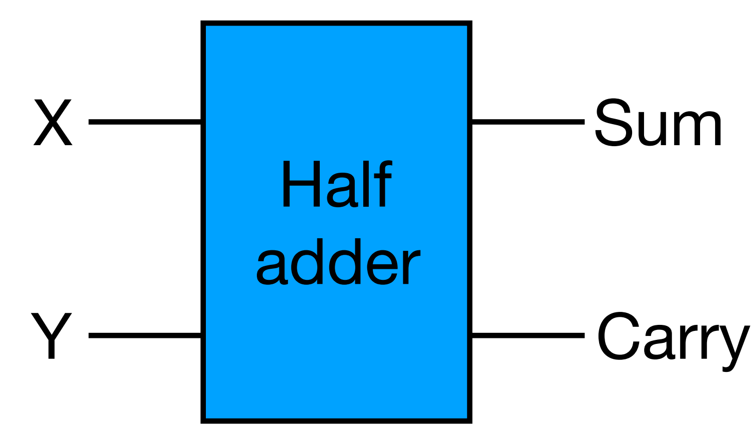
Circuitos combinatorios

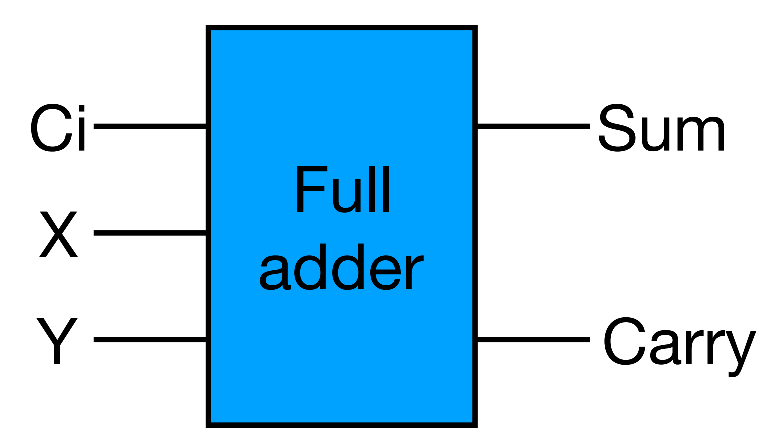
Adder

- ¿Será posible diseñar un sumador de números binarios (Adder) con Full adders y Half adders?



$$\begin{array}{r} A_4A_3A_2A_1 \\ B_4B_3B_2B_1 \\ \hline C_5C_4C_3C_2C_1 \end{array}$$

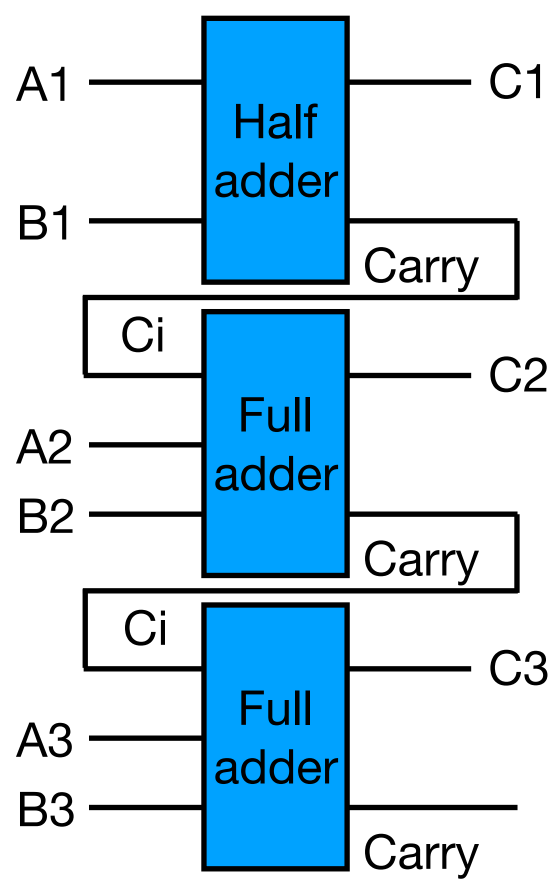




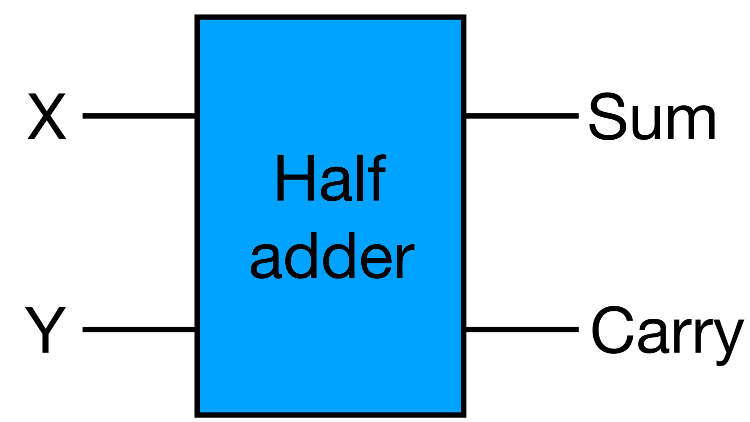
Circuitos combinatorios

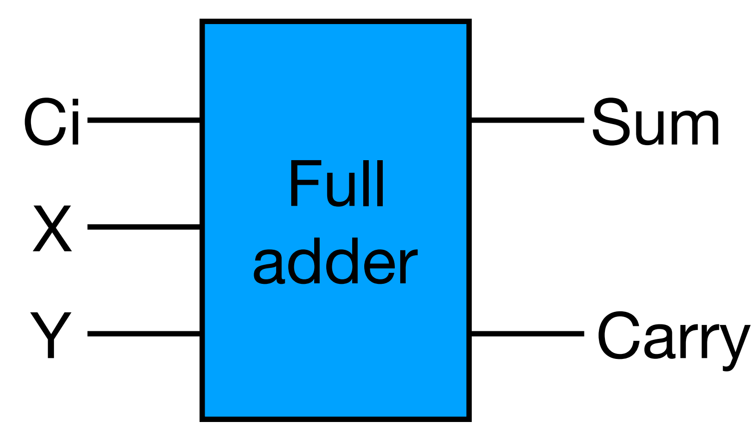
Adder

- ¿Será posible diseñar un sumador de números binarios (Adder) con Full adders y Half adders?



$$\begin{array}{r} A_4A_3A_2A_1 \\ B_4B_3B_2B_1 \\ \hline C_5C_4C_3C_2C_1 \end{array}$$

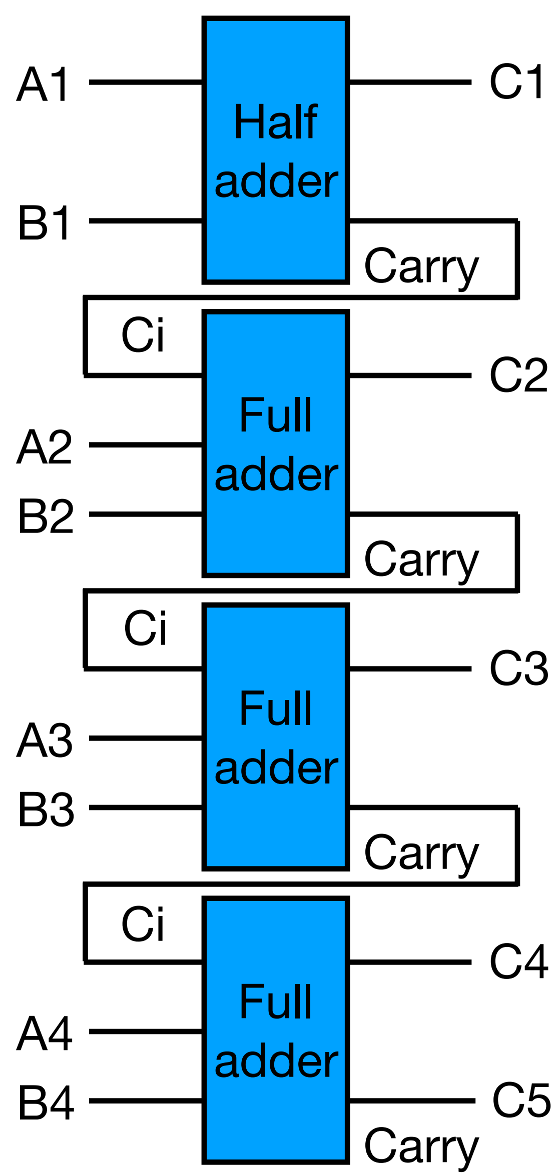




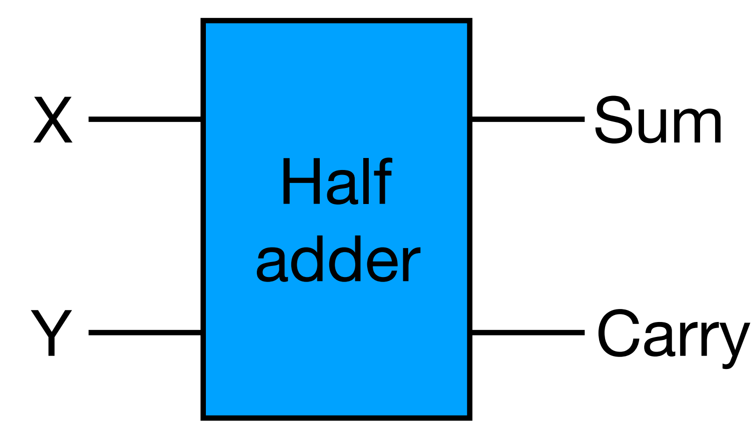
Circuitos combinatorios

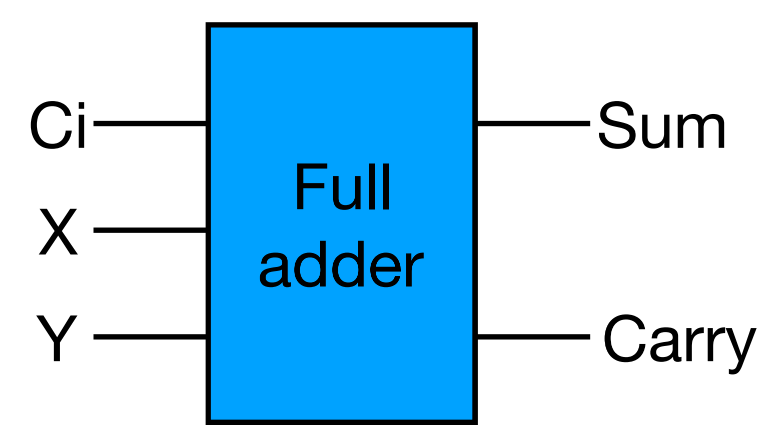
Adder

- ¿Será posible diseñar un sumador de números binarios (Adder) con Full adders y Half adders?



$$\begin{array}{r} A_4A_3A_2A_1 \\ B_4B_3B_2B_1 \\ \hline C_5C_4C_3C_2C_1 \end{array}$$

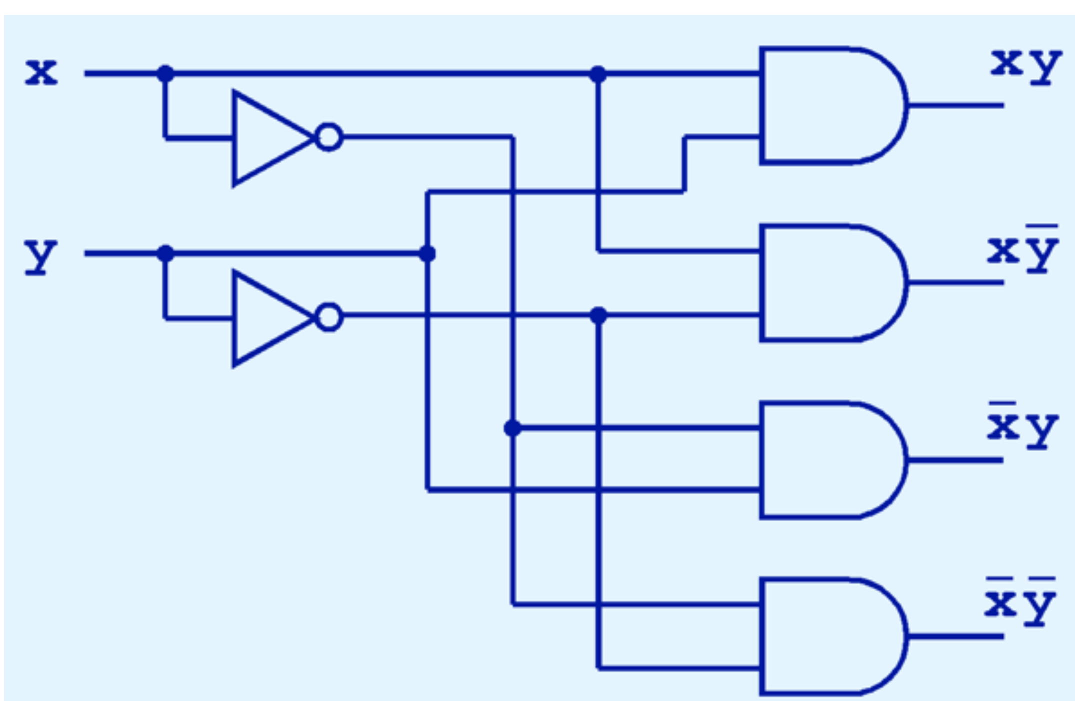


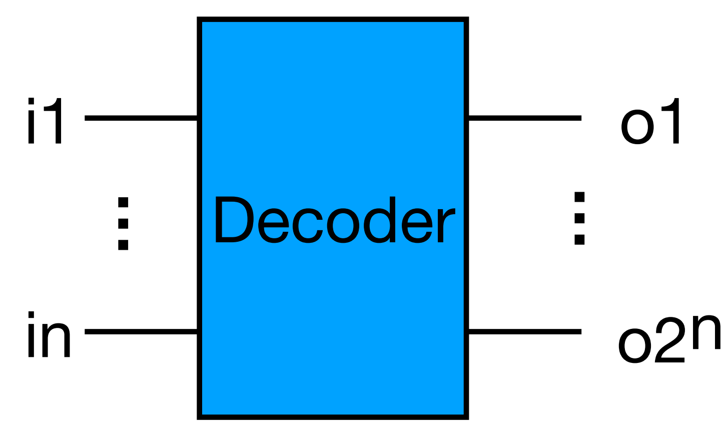


Circuitos combinatorios

Decoder

- Los **Decoders** poseen n entradas y 2^n salidas.
- Dada una combinación de valores de entrada se activa una única salida correspondiente a la combinación de entradas
- Son fundamentales para seccionar una posición de memoria a partir de una dirección

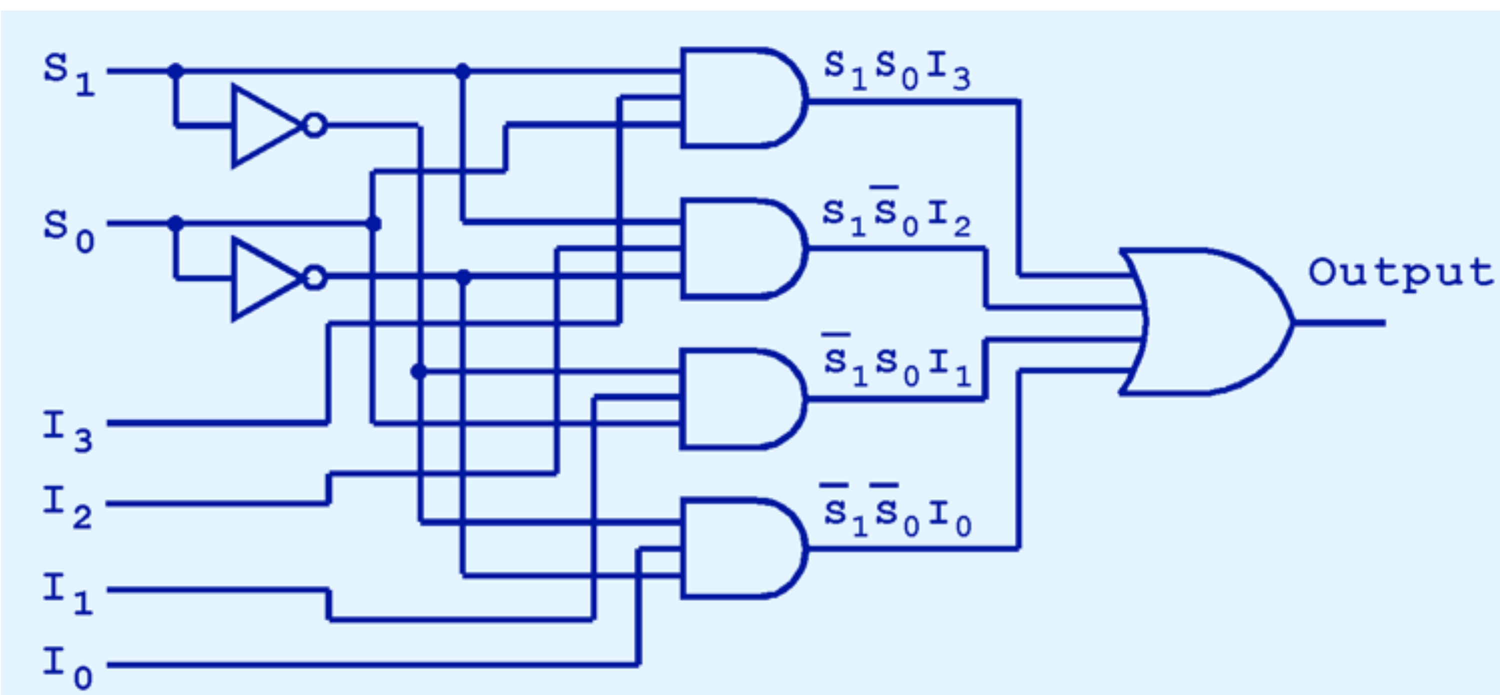


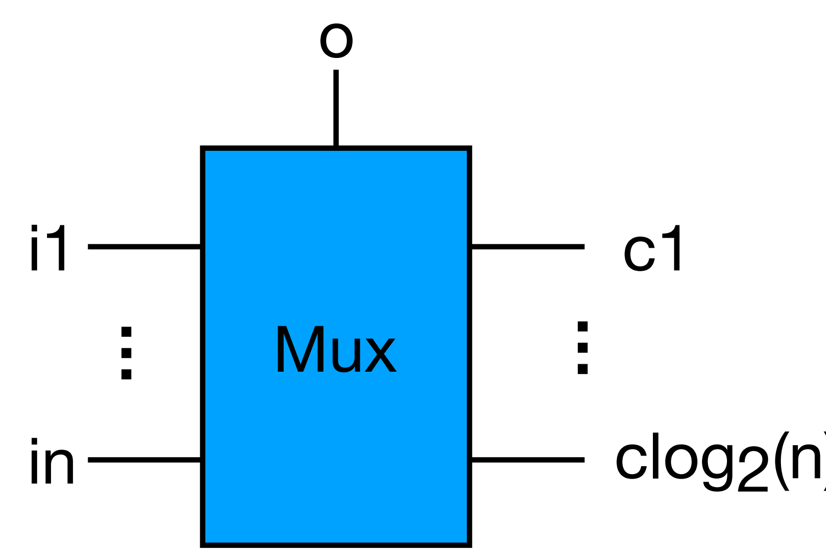


Circuitos combinatorios

Multiplexers

- Los **Multiplexers** (Mux) seleccionan 1 de n entradas a partir de $\log_2(n)$ líneas de control.





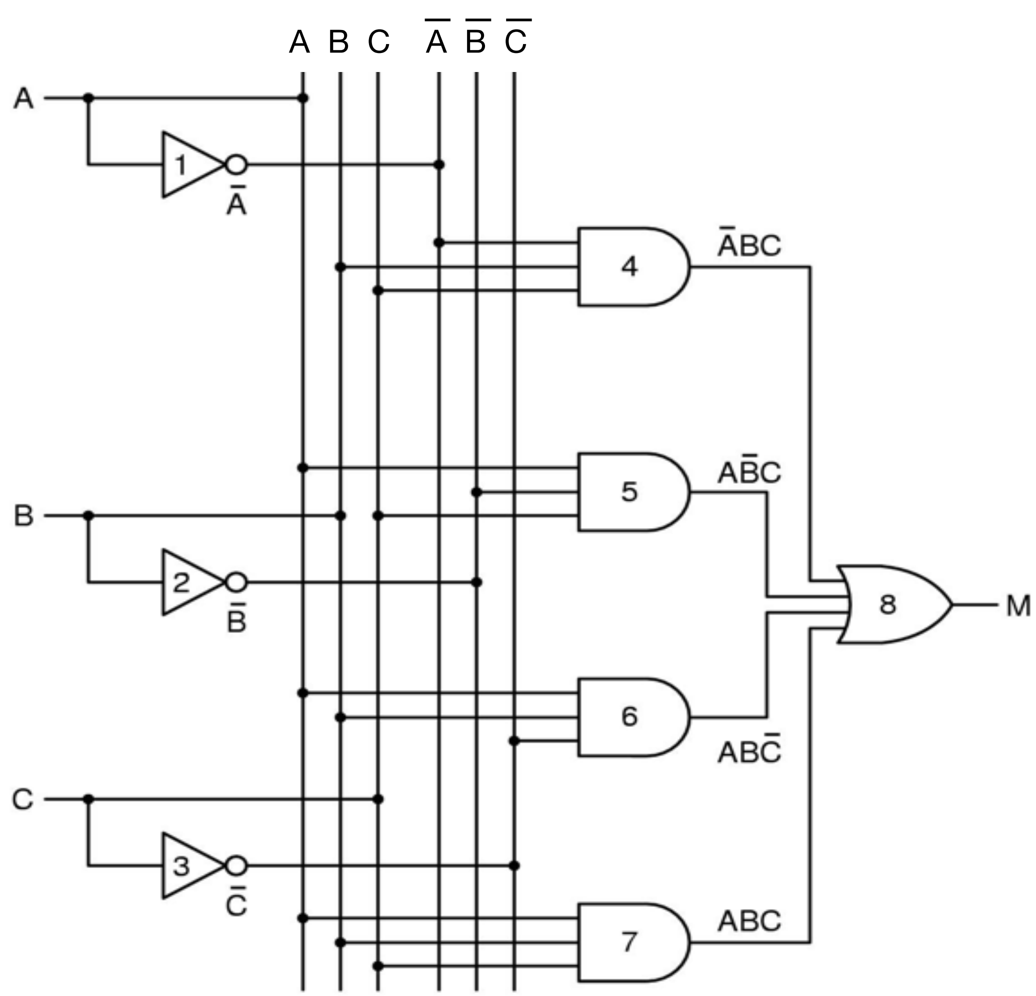
- Los **Demultiplexers** (Demux) hacen lo contrario, dada una entrada, seleccionan una de n salidas basándose en $\log_2(n)$ líneas de control

Circuitos combinatorios

Multiplexers (Ejemplo)

$M(A, B, C) = \overline{A}BC+A\overline{B}C+AB\overline{C}+ABC$

A	B	C	M
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

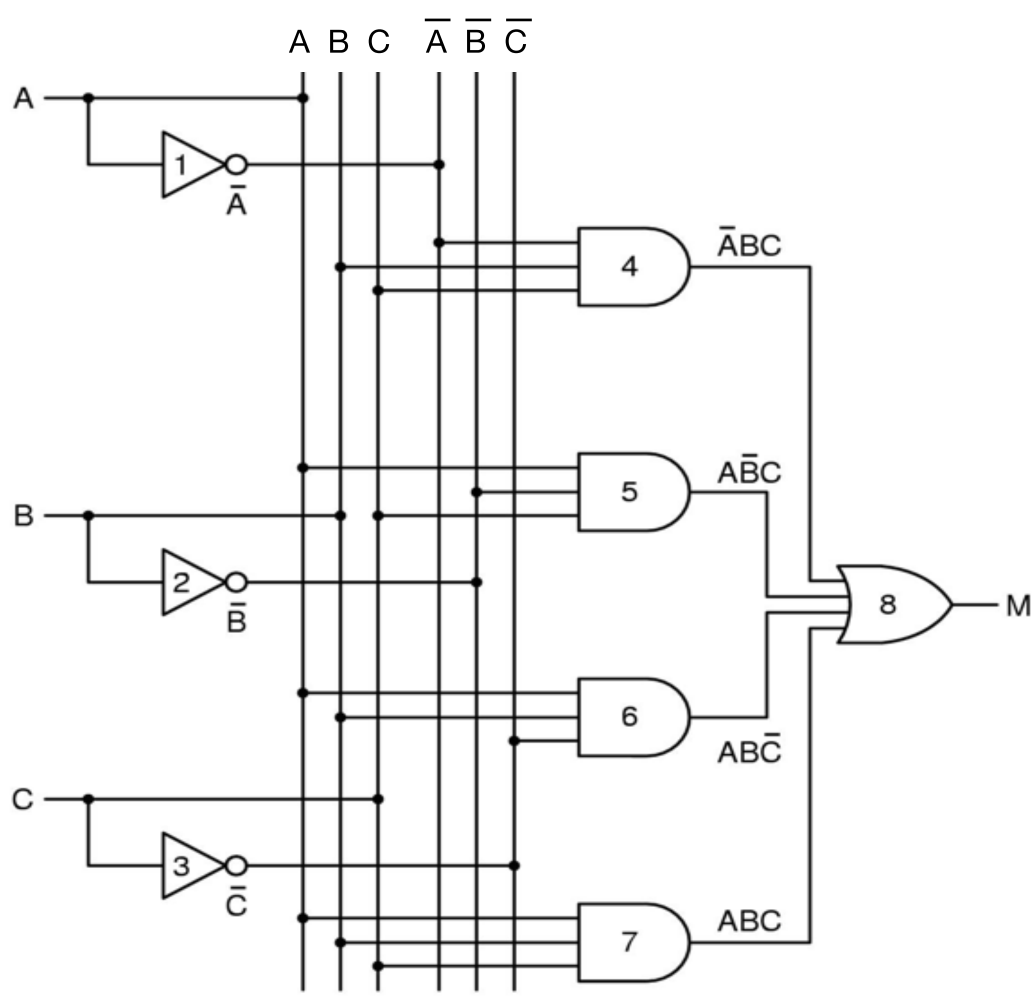


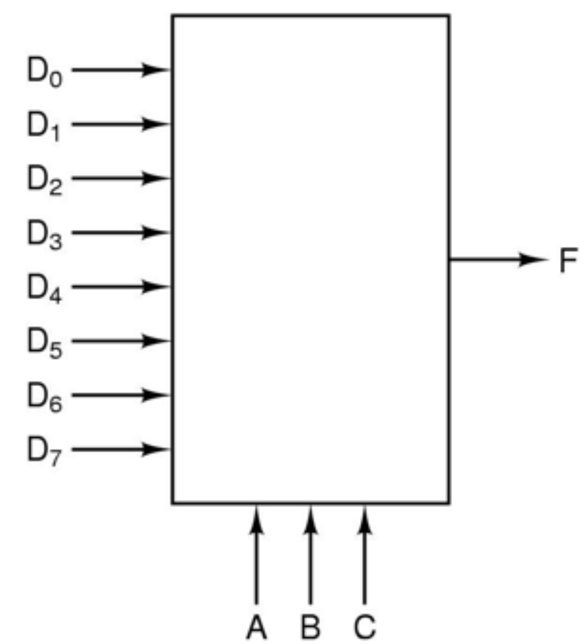
Circuitos combinatorios

Multiplexers (Ejemplo)

$M(A, B, C) = \overline{A}BC + A\overline{B}C + AB\overline{C} + ABC$

A	B	C	M
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



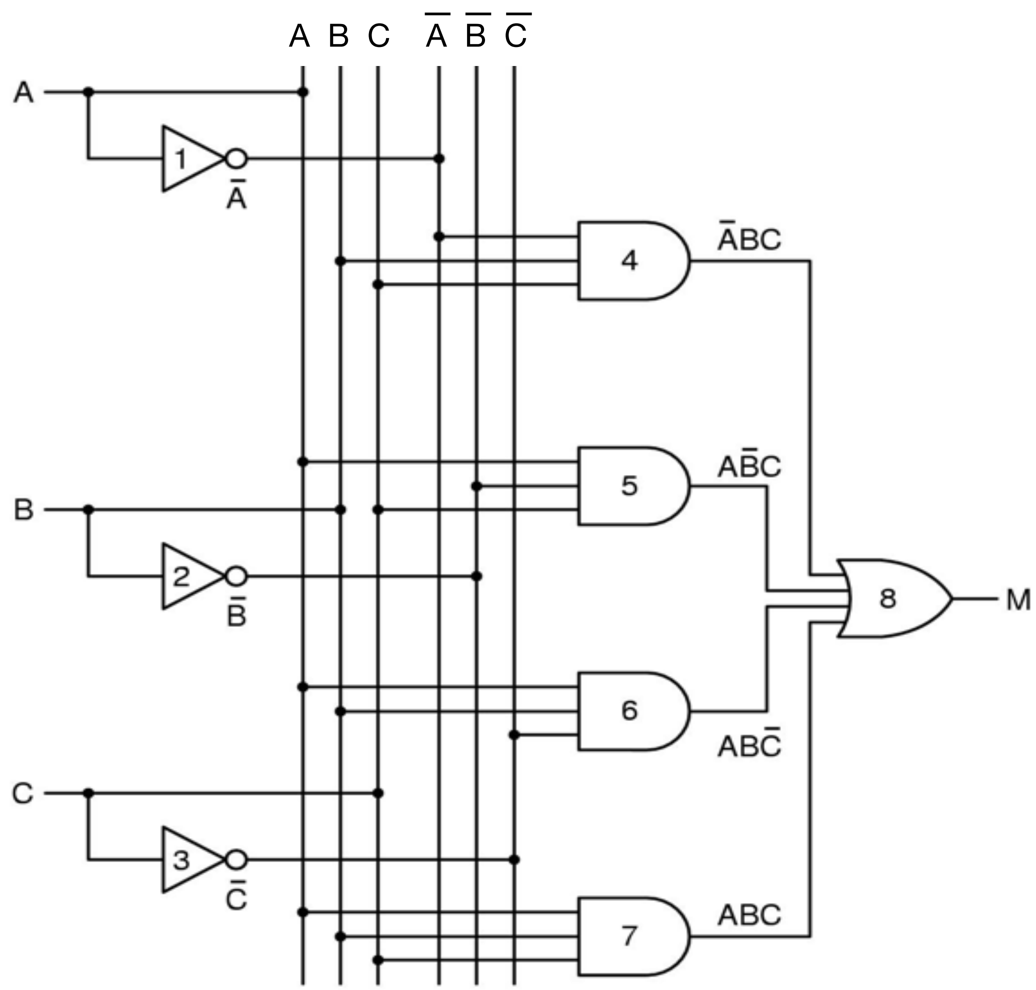


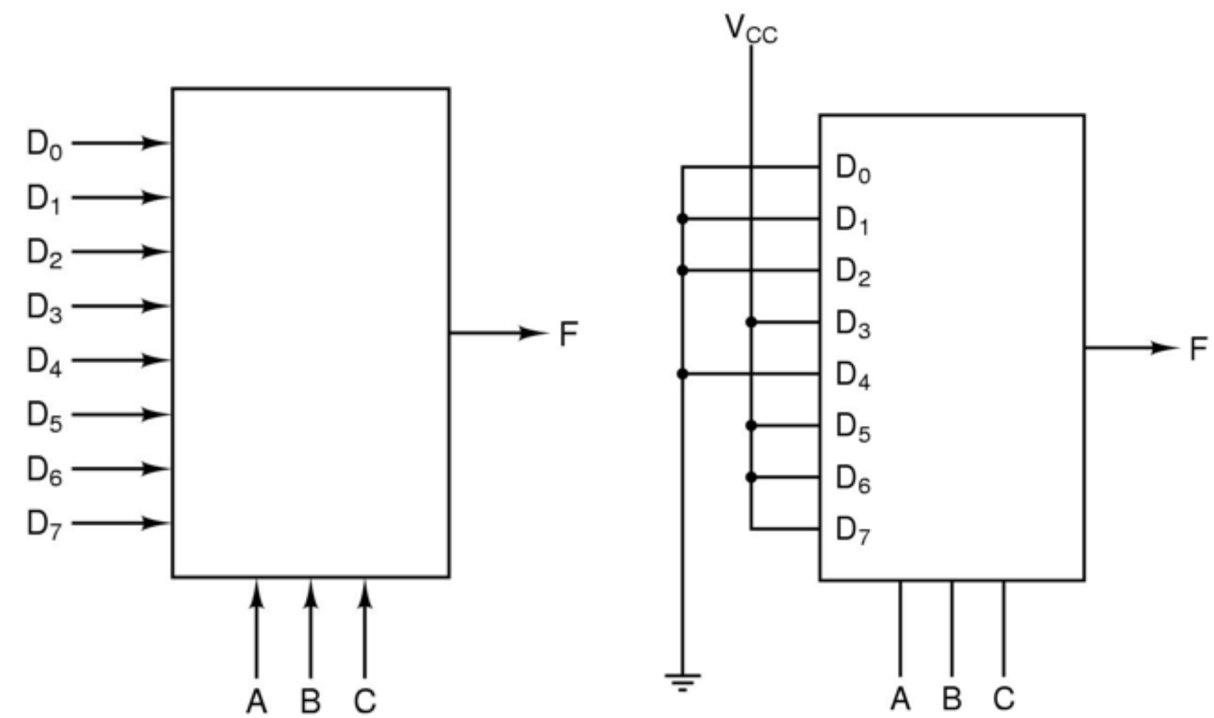
Circuitos combinatorios

Multiplexers (Ejemplo)

$$M(A, B, C) = \overline{A}BC + A\overline{B}C + AB\overline{C} + ABC$$

A	B	C	M
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





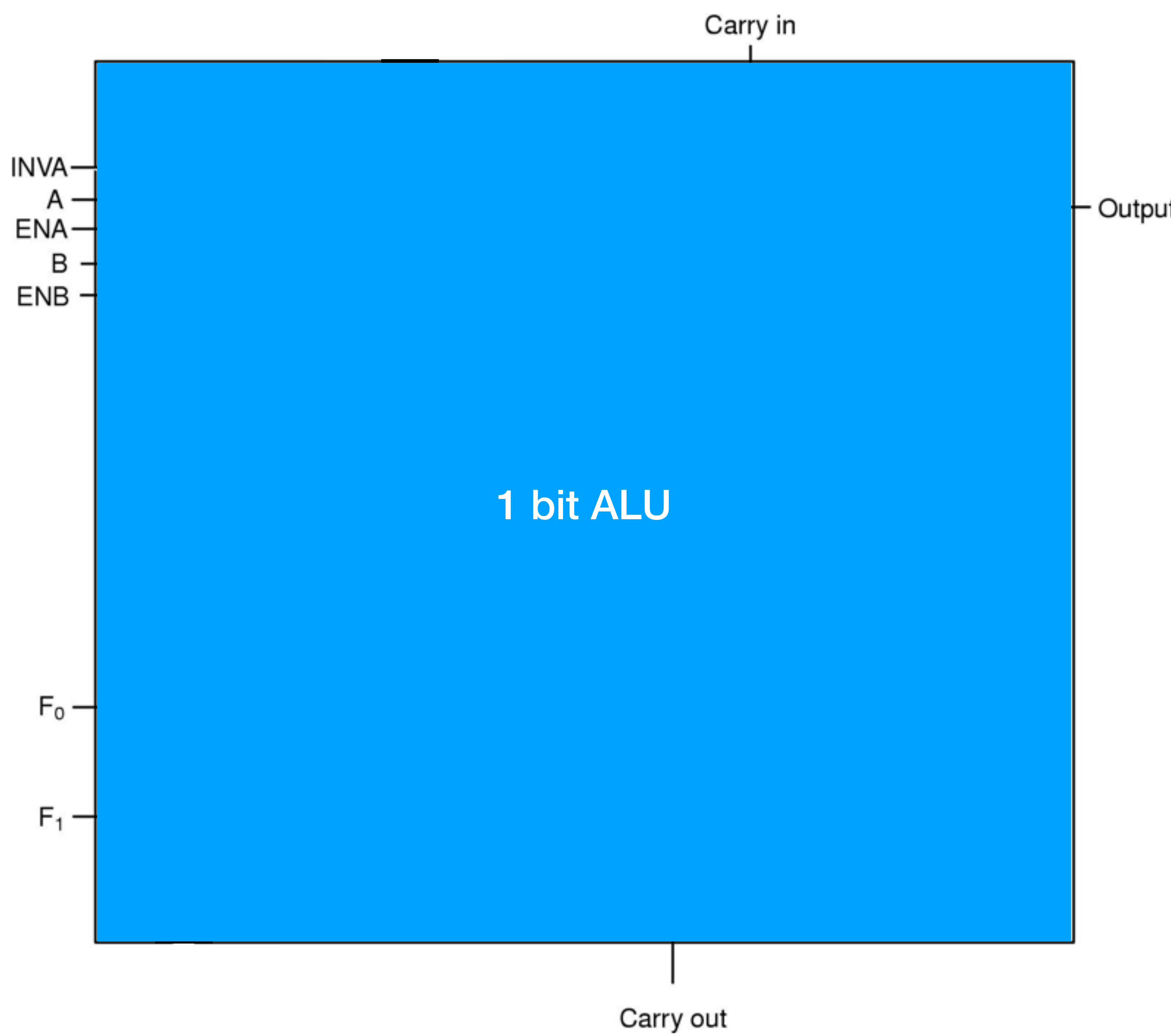
Circuitos combinatorios

Ejercicio

- Construir una **ALU** de 1 bit
- 3 entradas: A, B, Carry
- 4 operaciones: AND, OR, NOT, Suma(A, B, Carry)
- 2 Salidas: Result, $Carry_{out}$

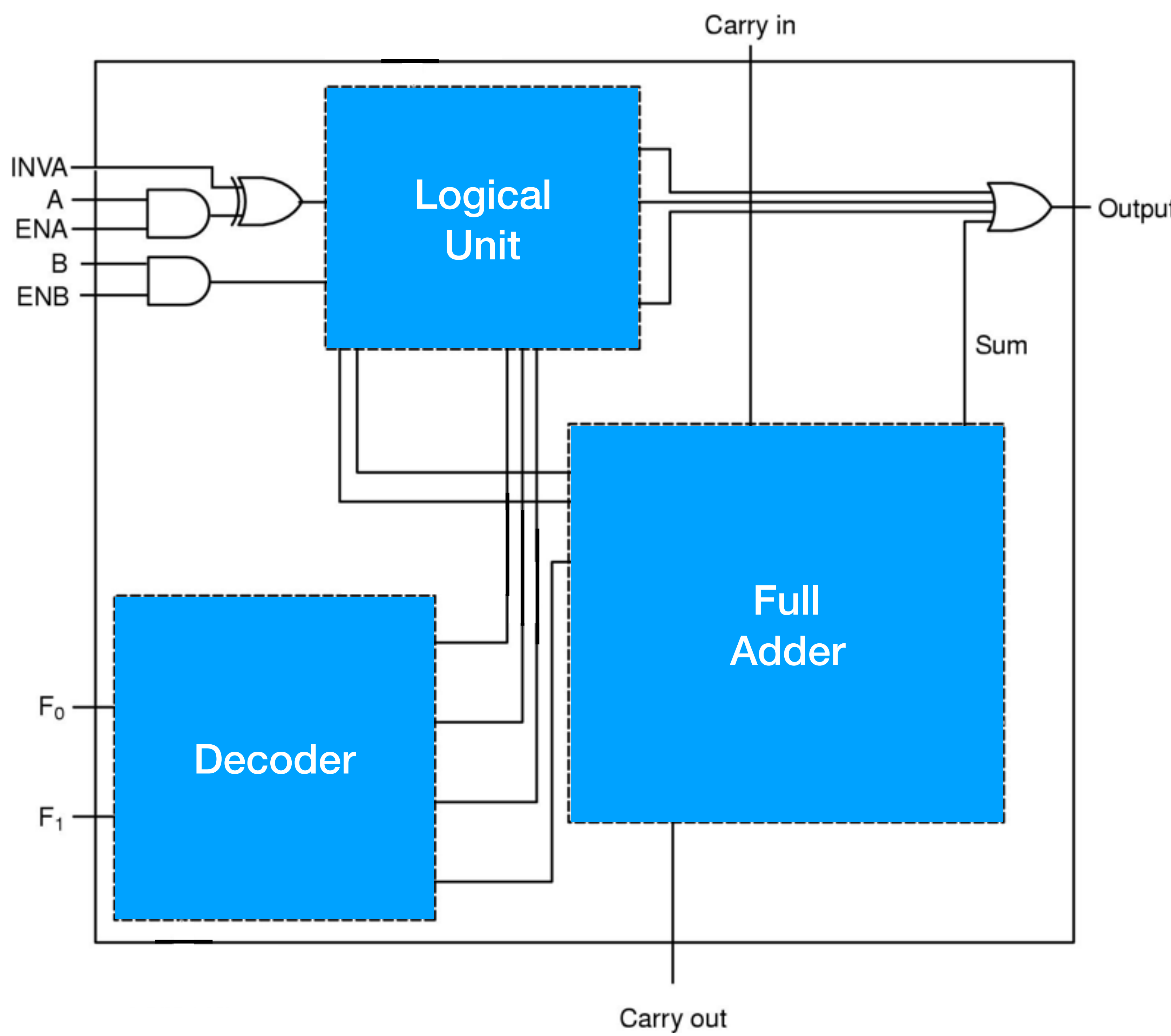
Circuitos combinatorios

Ejercicio



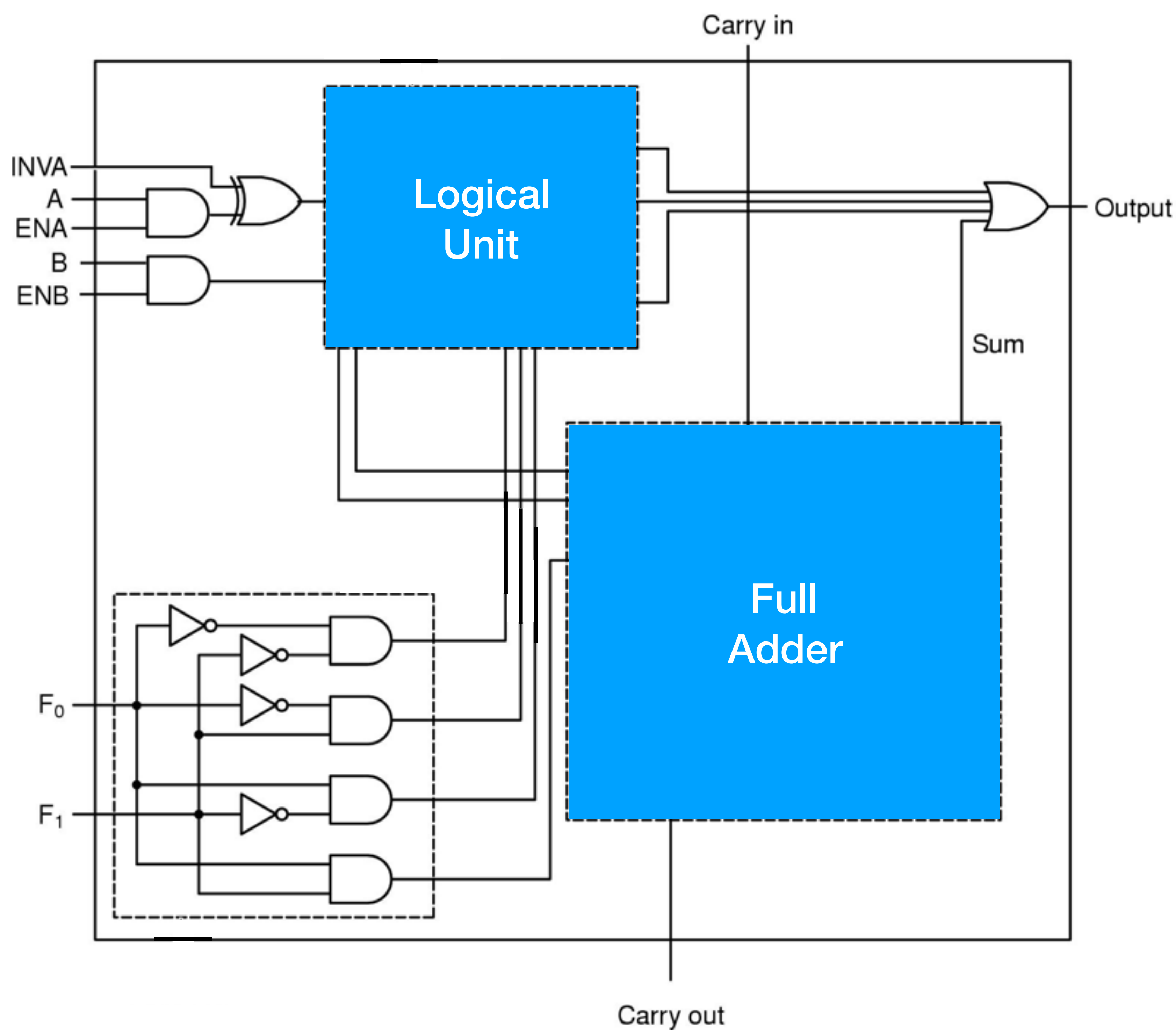
Circuitos combinatorios

Ejercicio



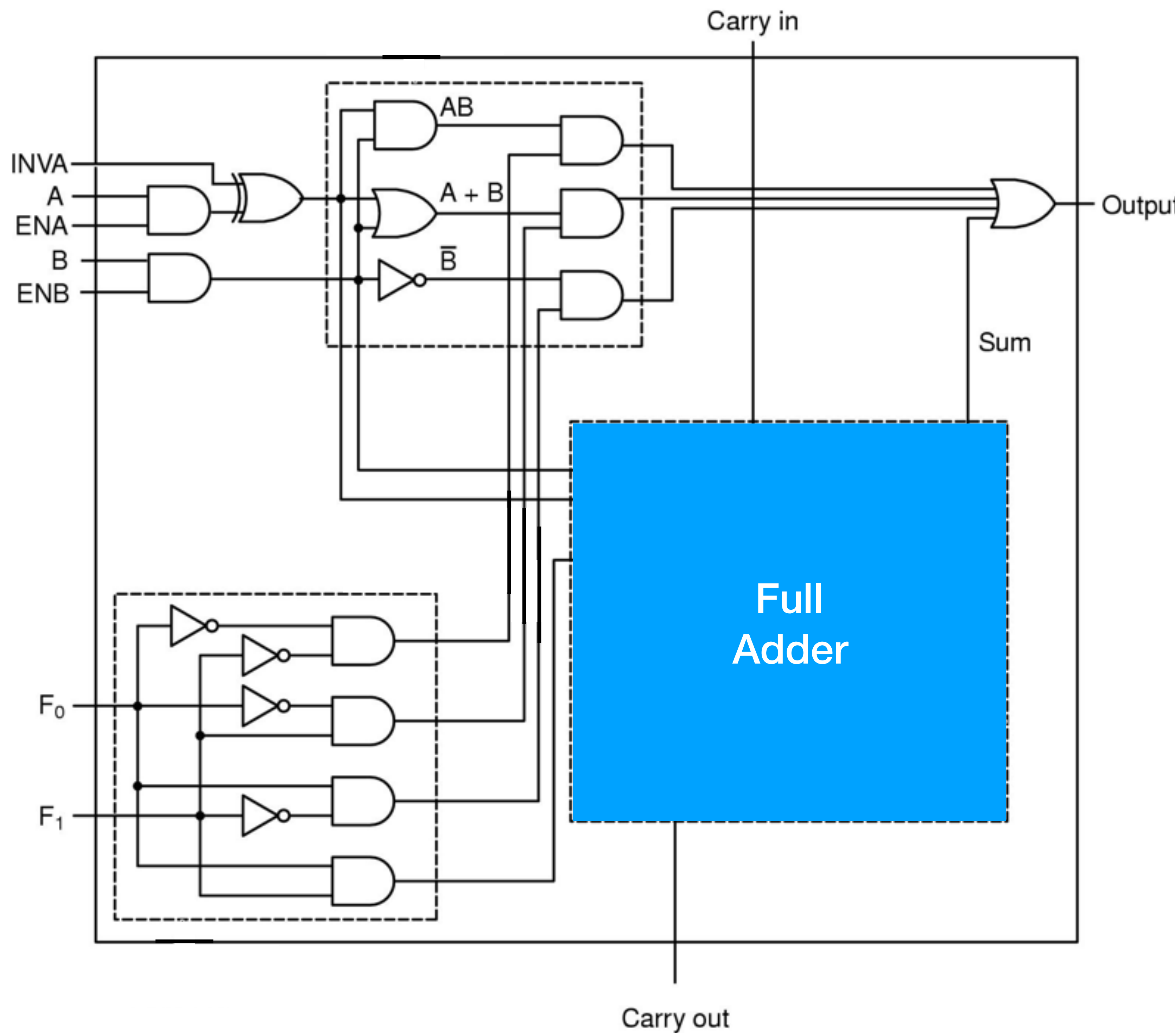
Circuitos combinatorios

Ejercicio



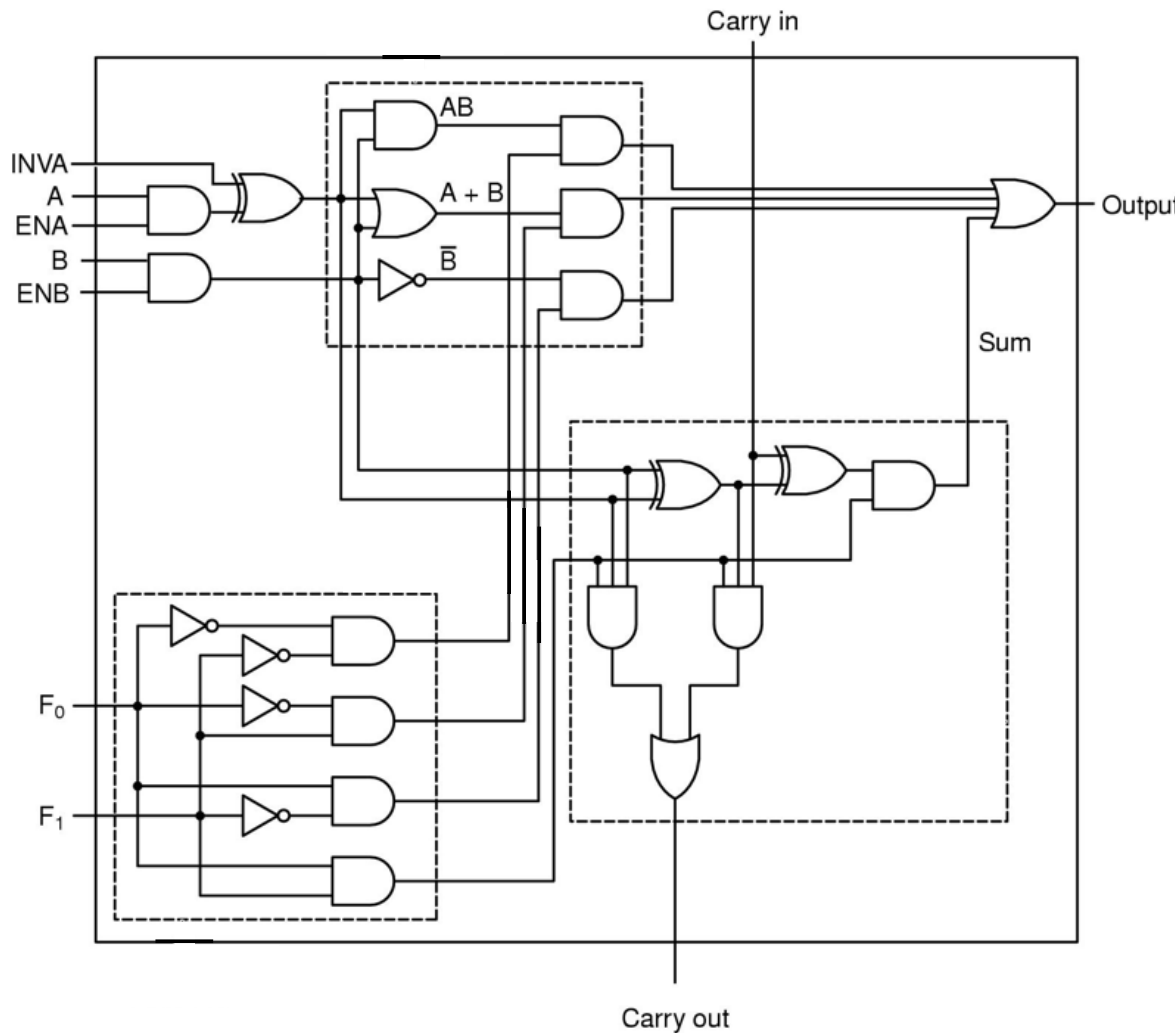
Circuitos combinatorios

Ejercicio



Circuitos combinatorios

Ejercicio



Circuitos combinatorios

Circuitos programables en ROM

Inputs					Outputs							
I_4	I_3	I_2	I_1	I_0	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
0	0	0	0	0	1	0	1	1	0	1	1	0
0	0	0	0	1	0	0	0	1	1	1	0	1
0	0	0	1	0	1	1	0	0	0	1	0	1
0	0	0	1	1	1	0	1	1	0	0	1	0
		.							.			
		.							.			
		.							.			
1	1	1	0	0	0	0	0	0	1	0	0	1
1	1	1	0	1	1	1	1	0	0	0	1	0
1	1	1	1	0	0	1	0	0	1	0	1	0
1	1	1	1	1	0	0	1	1	0	0	1	1

Circuitos combinatorios

Circuitos programables en ROM