Capítulo 1
Introducción

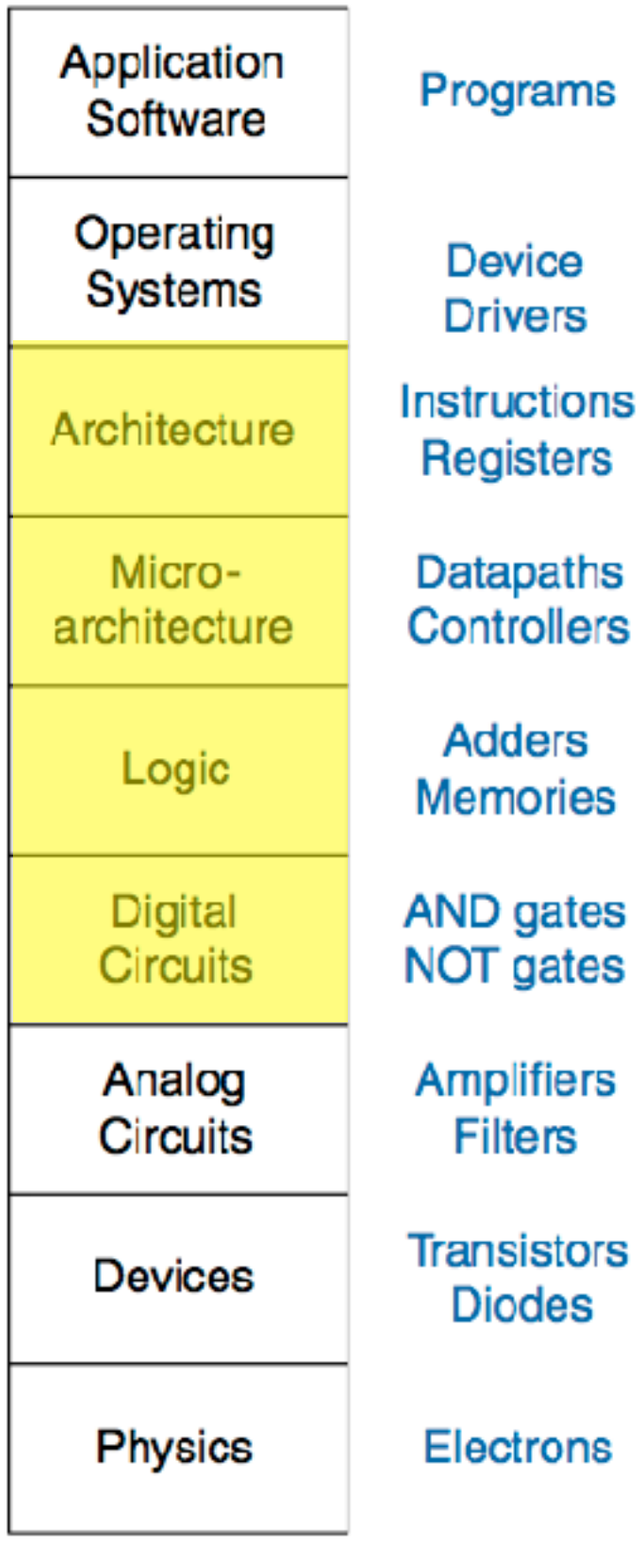
Cristián Tejos
Segundo Semestre 2017

Basado en apuntes de Marcelo Guarini
y el libro
Digital Design and Computer Architecture (1st edition),
D. M. Harris & S. L. Harris, Elsevier 2007

Administrando la complejidad

- Una característica que separa a un ingeniero de una persona normal es su enfoque sistemático para hacer frente a la complejidad
- Los sistemas digitales modernos se construyen con millones o billones de transistores.
- Ningún ser humano puede ser capaz de comprender estos sistemas escribiendo ecuaciones que describen el movimiento de los electrones en cada transistor y resolverlas simultáneamente.
- Es importante aprender a manejar la complejidad para construir un microprocesador sin quedar empantanados en miles y miles de detalles.

Abstracción



Utiliza las capacidades del sistema operativo para resolver problemas del usuario.

El sistema operativo comanda los detalles de bajo nivel como acceso al disco duro o la administración de memoria.

Describe un computador desde la perspectiva del programador. Ej. la arquitectura intel IA-32. Una arquitectura puede implementarse con distintas microarquitecturas, Ej Core 2 Duo, Intel 80486, AMD Athlon.

Une los niveles de abstracción lógico y de arquitectura

Conjuntos de compuertas lógicas interconectadas entre sí para formar estructuras más complejas

Compuertas lógicas con entradas y salidas restringidas a rangos discretos, que se utilizan para representar el 0 y el 1

Conjunto de dispositivos electrónicos interconectados entre sí para formar componentes (Ej. amplificadores). Entradas y salidas continuas.

Dispositivos modelados por sus relaciones entre voltaje y corriente, lo cual permite ignorar los electrones en forma individual.

Movimiento de los electrones (Mecánica cuántica, ecuaciones de Maxwell)

Disciplina

- Es el acto de restringir intencionalmente las alternativas de diseño con el objeto de trabajar en forma más productiva a un mayor nivel de abstracción.
- Una forma familiar de aplicar la disciplina es el uso de partes intercambiables

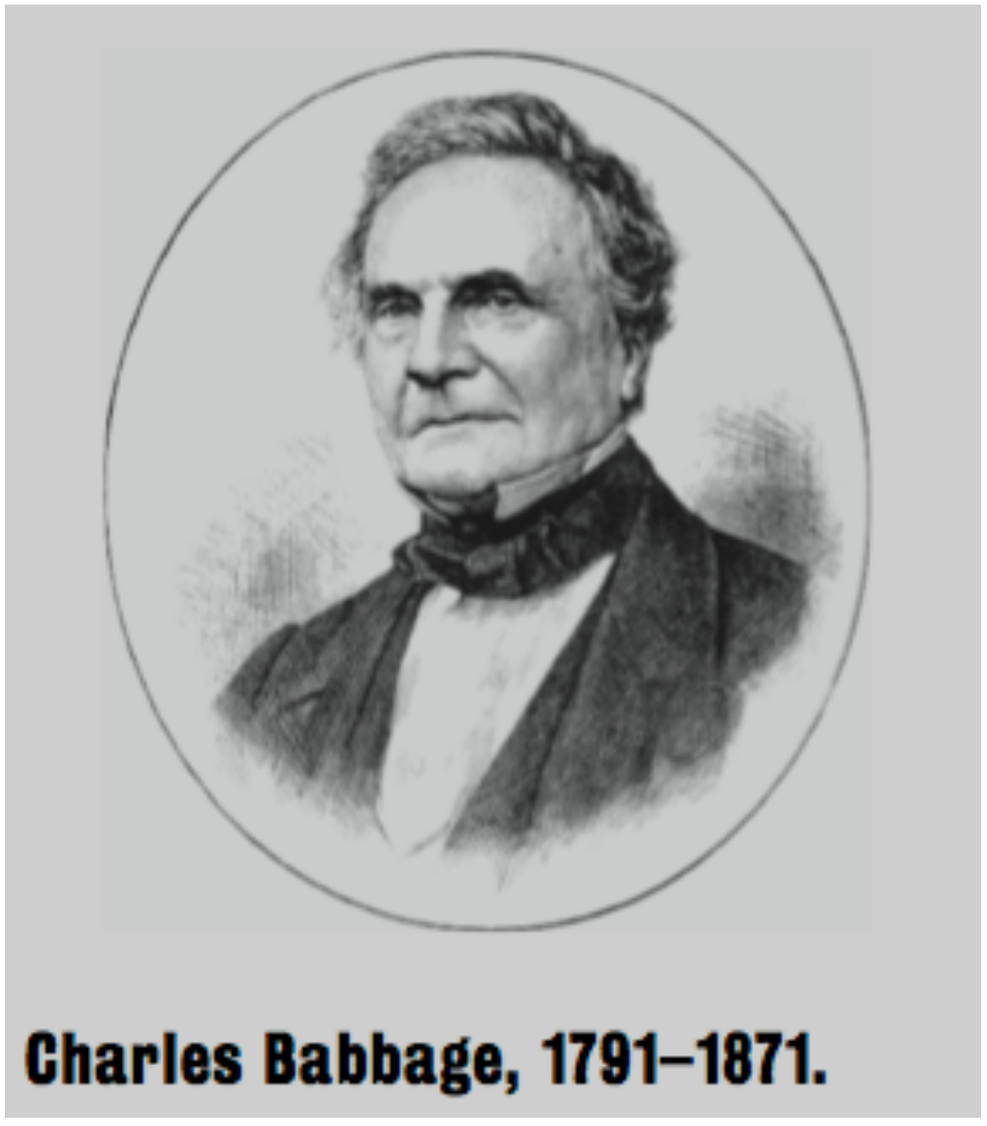
Otros principios, válidos tanto para software como para hardware

- *Jerarquía*: consiste en dividir un sistema en módulos, luego subdividir estos en submódulos hasta que las piezas resultan fáciles de entender.
- *Modularidad*: los módulos deben tener funciones e interfaces bien definidas, de tal forma de poder interconectarlos sin efectos adversos imprevistos.
- *Regularidad*: persigue uniformidad entre los módulos. Los módulos comunes son reutilizados repetidas veces, reduciendo de esta forma el número de módulos distintos que deben ser diseñados.

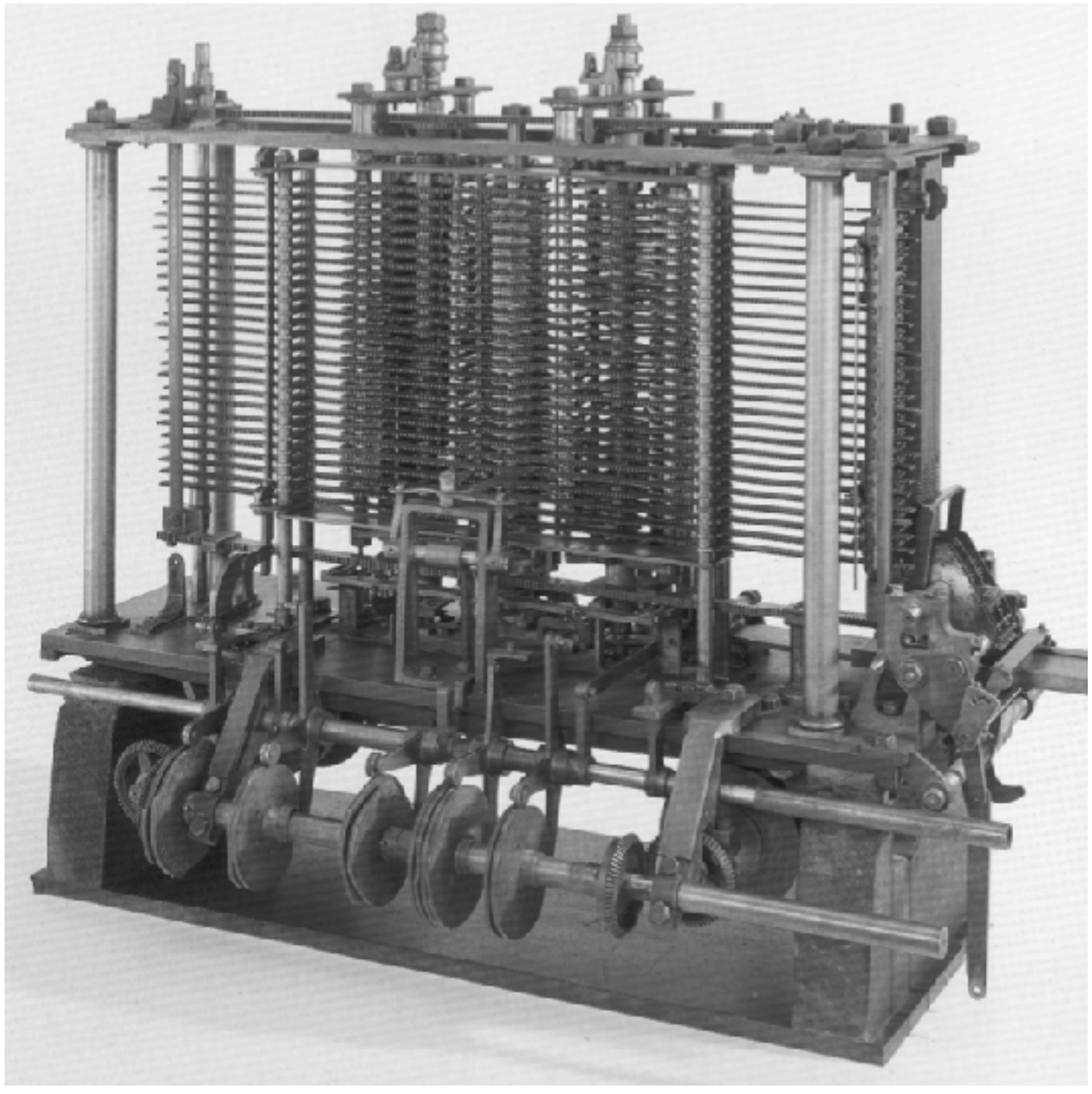
La abstracción digital

- La mayor parte de las variables físicas son continuas (Ej. el voltaje en un cable, la frecuencia de una oscilación, la posición de una masa)
- Los sistemas digitales representan información utilizando variables de valores discretos.

La abstracción digital



Charles Babbage, 1791–1871.



The Analytical Engine

La abstracción digital

- El motor analítico de Babbage utilizaba engranajes con diez posiciones marcadas desde 0 hasta 9. Escogió 25 filas de engranajes para tener una precisión de 25 dígitos.
- La mayor parte de los de los computadores electrónicos utilizan representación binaria en la cual un voltaje alto significa 1 y un voltaje bajo significa 0.
- La *cantidad de información D* en una variable discreta con *N* estados diferentes se mide en unidad de bits como

$$D = \log_2 N \text{ bits}$$

- Una variable binaria transmite $\log_2 2 = 1$ bit de información
- Una variable decimal $\log_2 10 = 3.322$ bits de información

La abstracción digital

- Una variable continua contiene teóricamente una cantidad de información infinita porque puede tomar un número infinito de valores.
- En la práctica, el ruido y los errores de medición limitan la información a 10 ó 16 bits para las señales continuas.
- Si las mediciones se realizan en forma rápida, el contenido es aún menor (unos 8 bits).
- En este curso nos enfocaremos en circuitos digitales, utilizando variables binarias: 1's y 0's

Sistemas numéricos

Sistema decimal

$$9742_{10} = 9 \times 10^3 + 7 \times 10^2 + 4 \times 10^1 + 2 \times 10^0$$

En el sistema decimal un número de *N* dígitos representa una de 10^N posibilidades: 0, 1, 2, 3, ..., 10^N-1

Sistema binario

$$10110_2 = 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = 22_{10}$$

En el sistema binario un número de *N* dígitos representa una de 2^N posibilidades: 0, 1, 2, 3, ..., 2^N-1.

Sistemas numéricos

1-Bit Binary Numbers	2-Bit Binnary Numbers	3-Bit Binary Numbers	4-Bit Binary Numbers	Decimal Equivalents
0	00	000	0000	0
1	01	001	0001	1
	10	010	0010	2
	11	011	0011	3
		100	0100	4
		101	0101	5
		110	0110	6
		111	0111	7
			1000	8
			1001	9
			1010	10
			1011	11
			1100	12
			1101	13
			1110	14
			1111	15

Sistemas numéricos

En general cualquier número de *N* dígitos en cualquier base se puede escribir como:

$$a_{N-1}\, b^{N-1} + a_{N-2}\, b^{N-2} + ... + a_2\, b^2 + a_1\, b^1 + a_0\, b^0$$

donde los a_i son los dígitos del número y b es la base del sistema numérico.

Por ejemplo $(324)_5$ se escribe:

$$3 \times 5^2 + 2 \times 5^1 + 4 \times 5^0$$

Sistemas numéricos

El polinomio es extensible para incluir números con parte entera y parte fraccionaria. Por ejemplo un número de *N* dígitos en su parte entera y *M* dígitos en su parte fraccionaria se escribe como:

$$a_{N-1}\, b^{N-1} + a_{N-2}\, b^{N-2} + \ldots\ldots + a_2\, b^2 + a_1\, b^1 + a_0\, b^0 +$$

$$a_{-1}\, b^{-1} + a_{-2}\, b^{-2} + \ldots\ldots + a_{-M}\, b^M$$

donde los a_i son los dígitos del número y b es la base del sistema numérico.

Sistemas numéricos

Conversión entre bases

De cualquier base a base decimal:

- Evaluar el polinomio utilizando aritmética decimal

De base decimal a cualquier base:

- Restas sucesivas
- Divisiones sucesivas (parte entera)
- Multiplicaciones sucesivas (parte fraccionaria)

Sistemas numéricos

Conversión entre bases

De cualquier base a decimal (resolver el polinomio utilizando aritmética decimal). Ejemplos:

$$10110_2 = 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = 22_{10}$$

$$324_5 = 3 \times 5^2 + 2 \times 5^1 + 4 \times 5^0 = 89_{10}$$

$$110{,}11_2 = 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 + 1 \times 2^{-1} + 1 \times 2^{-2} = 6{,}75_{10}$$

Sistemas numéricos

Conversión entre bases

De decimal a cualquier base (restas sucesivas). Ejemplo:

53.5_{10} a base 2

$$
\begin{aligned}
&53.5 - 2^5 = 21.5 \rightarrow 1 \\
&21.5 - 2^4 = 5.5 \rightarrow 1 \\
&5.5 - 2^3 < 0 \rightarrow 0 \\
&5.5 - 2^2 = 1.5 \rightarrow 1 \\
&1.5 - 2^1 < 0 \rightarrow 0 \\
&1.5 - 2^0 = 0.5 \rightarrow 1 \\
&0.5 - 2^{-1} = 0 \rightarrow 1 \\
&110101,1
\end{aligned}
$$

Sistemas numéricos

Conversión entre bases (parte entera)

De decimal a cualquier base (divisiones sucesivas). Ejemplo:

53_{10} a base 2

$$
\begin{array}{ll}
53 : 2 = 26 & \\
01 & \longrightarrow 1 \\
26 : 2 = 13 & \\
00 & \longrightarrow 0 \\
13 : 2 = 6 & \\
01 & \longrightarrow 1 \\
6 : 2 = 3 & \\
00 & \longrightarrow 0 \\
3 : 2 = 1 & \\
01 & \longrightarrow 1 \\
1 : 2 = 0 & \\
01 & \longrightarrow 1 \qquad 110101
\end{array}
$$

Sistemas numéricos

Conversión entre bases (parte fraccionaria)

De decimal a cualquier base (multiplicaciones sucesivas).
Ejemplo:

0.34375_{10} a base 2

$$
\begin{array}{lll}
0.34375 \times 2 & & \\
0.68750 & \longrightarrow 0 & \\
0.68750 \times 2 & & \\
1.37500 & \longrightarrow 1 & \\
0.37500 \times 2 & & \\
0.75000 & \longrightarrow 0 & \\
0.75000 \times 2 & & \\
1.50000 & \longrightarrow 1 & \\
0.50000 \times 2 & & \\
1.00000 & \longrightarrow 1 & 0.01011
\end{array}
$$

Sistemas numéricos

Conversión entre bases que son potencias una de la otra

1-Bit Binary Numbers	2-Bit Binnary Numbers	3-Bit Binary Numbers	4-Bit Binary Numbers	Decimal Equivalents
0	00	000	0000	0
1	01	001	0001	1
	10	010	0010	2
	11	011	0011	3
		100	0100	4
		101	0101	5
		110	0110	6
		111	0111	7
			1000	8
			1001	9
			1010	10
			1011	11
			1100	12
			1101	13
			1110	14
			1111	15

Sistemas numéricos

Números hexadecimales

Hexadecimal Digit	Decimal Equivalent	Binary Equivalent
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

Sistemas numéricos

Conversión entre números binarios, decimales y hexadecimales

Convertir $2ED_{16}$ a binario y a decimal

$2_{16} = 0010_2$, $E_{16} = 1110_2$, $D_{16} = 1101_2$, entonces

$2ED_{16} = 1011101101_2$

$$2ED_{16} = 2 \times 16^2 + E \times 16^1 + D \times 16^0 = 749_{10}$$

Representación de dígitos decimales en códigos binarios

Código BCD

Código 8421

Binario		Decimal
0000	$\longrightarrow$	0
0001	$\longrightarrow$	1
0010	$\longrightarrow$	2
0011	$\longrightarrow$	3
0100	$\longrightarrow$	4
0101	$\longrightarrow$	5
0110	$\longrightarrow$	6
0111	$\longrightarrow$	7
1000	$\longrightarrow$	8
1001	$\longrightarrow$	9

$529 \longrightarrow 0101\ 0010\ 1001$

$4673,51 \longrightarrow 0100\ 0110\ 0111\ 0011\ ,\ 0101\ 0001$

Representación de dígitos decimales en códigos binarios

6421	→	Decimal	4321	→	Decimal	5211	→	Decimal
0000	→	0	0000	→	0	0000	→	0
0001	→	1	0001	→	1	0001	→	1
0010	→	2	0010	→	2	0010	→	1
0011	→	3	0011	→	3	0011	→	2
0100	→	4	0100	→	3	0100	→	2
0101	→	5	0101	→	4	0101	→	3
0110	→	6	1000	→	4	0110	→	3
1000	→	6	0110	→	5	0111	→	4
0111	→	7	1001	→	5	1000	→	5
1001	→	7	0111	→	6	1001	→	6
1010	→	8	1010	→	6	1010	→	6
1011	→	9	1011	→	7	1011	→	7
			1100	→	7	1100	→	7
			1101	→	8	1101	→	8
			1110	→	9	1110	→	8
						1111	→	9

Representación de dígitos decimales en códigos binarios

Decimal	Binario		Ex. de Tres
0	0000	$\longrightarrow$	0011
1	0001	$\longrightarrow$	0100
2	0010	$\longrightarrow$	0101
3	0011	$\longrightarrow$	0110
4	0100	$\longrightarrow$	0111
5	0101	$\longrightarrow$	1000
6	0110	$\longrightarrow$	1001
7	0111	$\longrightarrow$	1010
8	1000	$\longrightarrow$	1011
9	1001	$\longrightarrow$	1100

Códigos Cíclicos

Código GRAY

Sea $g_n, \cdots, g_1, g_0$ una palabra en código Gray de $n+1$ bits
y sea $b_n, \cdots, b_1, b_0$, su correspondiente número binario . Entonces

$$g_i = b_i \oplus b_{i+1} \qquad 0 \leq i \leq n-1$$

$$g_n = b_n$$

Donde

x	y	$x \oplus y$
0	0	0
0	1	1
1	0	1
1	1	0

De Gray a binario es:

$$b_n = g_n$$

$$b_i = g_n \oplus g_{n-1} \oplus \cdots \oplus g_i$$

Códigos cíclicos

Código GRAY

Generación de código Gray por reflexión

00	0	00	0	000
01	0	01	0	001
11	0	11	0	011
10	0	10	0	010
	1	10	0	110
	1	11	0	111
	1	01	0	101
	1	00	0	100
			1	100
			1	101
			1	111
			1	110
			1	010
			1	011
			1	001
			1	000

Representación de números con signo

Signo y magnitud

La magnitud del número se expresa en el sistema binario posicional y el signo corresponde a un bit adicional que se agrega a la izquierda del bit más significativo

$$+ a_{n-1} \ldots a_0 , a_{-1} \ldots a_{-m} = 0\, a_{n-1} \ldots a_0 , a_{-1} \ldots a_{-m}$$

$$- a_{n-1} \ldots a_0 , a_{-1} \ldots a_{-m} = 1\, a_{n-1} \ldots a_0 , a_{-1} \ldots a_{-m}$$

Ejemplos:

+ 6 = 00000110	- 6 = 10000110
+ 43 = 00101011	- 43 = 10101011

Representación de números con signo

Representación complementaria

- Simplifica significativamente las operaciones aritméticas binarias
- Se utilizan dos tipos de representaciones complementarias
- Complemento a la base (complemento de 2)
- Complemento a la base disminuida (complemento de 1)

Representación de números con signo

Número binario en complemento de 2

$$2^n - N_2$$

donde n es el número de bits del número binario N_2

Ejemplo: determinar el complemento de 2 del número binario 0110100010_2

$$\begin{aligned} 2^{10} - 0110100010_2 &= 10000000000_2 - 0110100010_2 \\ &= 1001011110_{Comp_2} \end{aligned}$$

Representación de números con signo

Número binario en complemento de 2

Una forma más fácil de realizar la conversión es

$$
\begin{array}{rcl}
(N)_2 & = & (0110100010)_2 \\
(\bar{N})_2 & = & (1001011101)_2 \\
 & + & (0000000001)_2 \\
\hline
 & & (1001011110)_{Comp-2}
\end{array}
$$

Otra forma simple

$$
\text{se complementa} \rightarrow \quad \underbrace{\underbrace{01101000}_{10010111}\ \underbrace{10}_{10}} \quad \leftarrow \text{se deja igual}
$$

Representación de números con signo

Número binario en complemento de 1

$$2^n - N_2 - 1$$

donde n es el número de bits del número binario N_2

Ejemplo: determinar el complemento de 1 del número binario 0110100010_2

$$\bar{N}_2 = 1001011101_2$$

Representación de números con signo

Decimal	Signo y Magnitud	Complemento de Dos	Complemento de Uno
+15	01111	01111	01111
+14	01110	01110	01110
+13	01101	01101	01101
+12	01100	01100	01100
+11	01011	01011	01011
+10	01010	01010	01010
+9	01001	01001	01001
+8	01000	01000	01000
+7	00111	00111	00111
+6	00110	00110	00110
+5	00101	00101	00101
+4	00100	00100	00100
+3	00011	00011	00011
+2	00010	00010	00010
+1	00001	00001	00001
0	00000	00000	00000
	10000	—	11111

Representación de números con signo

Decimal	Signo y Magnitud	Complemento de Dos	Complemento de Uno
-1	10001	11111	11110
-2	10010	11110	11101
-3	10011	11101	11100
-4	10100	11100	11011
-5	10101	11011	11010
-6	10110	11010	11001
-7	10111	11001	11000
-8	11000	11000	10111
-9	11001	10111	10110
-10	11010	10110	10101
-11	11011	10101	10100
-12	11100	10100	10011
-13	11101	10011	10010
-14	11110	10010	10001
-15	11111	10001	10000
-16	—	10000	—

Complemento de 1 y complemento de 2

Ejemplo para cuatro dígitos binarios

Decimal	Comp. de 1
7	0111
6	0110
5	0101
4	0100
3	0011
2	0010
1	0001
0	0000
- 0	1111
- 1	1110
- 2	1101
- 3	1100
- 4	1011
- 5	1010
- 6	1001
- 7	1000

Decimal	Comp. de 2
7	0111
6	0110
5	0101
4	0100
3	0011
2	0010
1	0001
0	0000
- 1	1111
- 2	1110
- 3	1101
- 4	1100
- 5	1011
- 6	1010
- 7	1001
- 8	1000

Aritmética binaria: complemento de 1

Ejemplo:

Sumar en complemento de uno los números $(19)_{10}$ y $(7)_{10}$.

$$\begin{aligned} (19)_{10} &= (010011)_{Comp-1} \\ (7)_{10} &= (0111)_{Comp-1} \end{aligned}$$

$\downarrow$

$$\begin{array}{rcl} (19)_{10} & & (010011)_{Comp-1} \\ + \quad (7)_{10} & + & (000111)_{Comp-1} \\ \hline (26)_{10} & & (011010)_{Comp-1} \end{array}$$

Aritmética binaria: complemento de 1

Ejemplo:

$$\begin{array}{rrcrll} & (19)_{10} & = & & (010011)_{Comp-1} & \\ + & (17)_{10} & = & + & (010001)_{Comp-1} & \\ \hline & (36)_{10} & \neq & & (100100)_{Comp-1} & \leftarrow \text{Error} \end{array}$$

El error se genera porque 6 dígitos binarios sólo permiten representar hasta el número +31. El problema se resuelve aumentando el número de dígitos.

$$\begin{array}{rrcrll} & (19)_{10} & = & & (0010011)_{Comp-1} & \\ + & (17)_{10} & = & + & (0010001)_{Comp-1} & \\ \hline & (36)_{10} & = & & (0100100)_{Comp-1} & \leftarrow \text{Correcto} \end{array}$$

Aritmética binaria: complemento de 1

Ejemplo:

$$
\begin{array}{rcl}
(12)_{10} & = & (01100)_{Comp-1} \\
(-12)_{10} & = & (10011)_{Comp-1}
\end{array}
$$

$$
\begin{array}{rcll}
(23)_{10} & = & (010111)_{Comp-1} & \\
(-12)_{10} & = + & (110011)_{Comp-1} & \\
\hline
(11)_{10} & \neq & (1001010)_{Comp-1} & \leftarrow \text{Error}
\end{array}
$$

Solución: cada vez que se produce un *carry* del dígito más significativo hacia afuera, es necesario sumar uno al resultado para corregir el error

$$
\begin{array}{r}
(001010)_{Comp-1} \\
+(000001)_{Comp-1} \\
\hline
(001011)_{Comp-1}
\end{array}
$$

Aritmética binaria: complemento de 2

Ejemplo:

Sumar en complemento de dos los números $(19)_{10}$ y $(7)_{10}$.

$$\begin{array}{rcl} (19)_{10} & = & (010011)_{Comp-2} \\ (7)_{10} & = + & (000111)_{Comp-2} \\ \hline (26)_{10} & = & (011010)_{Comp-2} \end{array}$$

Aritmética binaria: complemento de 2

Ejemplo: sumar los números -19 y -17,

$$\begin{array}{rrcrl} & (-19)_{10} & = & (101101)_{Comp-2} & \\ + & (-17)_{10} & = & + \quad (101111)_{Comp-2} & \\ \hline & (-36)_{10} & \neq & (1011100)_{Comp-2} & \leftarrow \text{Error} \end{array}$$

Desechando el bit de *carry* vemos que el resultado es positivo.
El problema se genera porque el número de dígitos utilizados no es suficiente, extendiendo el bit de signo un lugar hacia la izquierda
Se resuelve el problema

$$\begin{array}{rrcrl} & (-19)_{10} & = & (1101101)_{Comp-2} & \\ + & (-17)_{10} & = & + \quad (1101111)_{Comp-2} & \\ \hline & (-36)_{10} & = & (11011100)_{Comp-2} & \leftarrow \text{Correcto} \end{array}$$

Aritmética binaria: complemento de 2

Resumen para adición en complemento de 2

1. Suma de números positivos
2. Suma de números negativos
3. Resta con resultado positivo
4. Resta con resultado negativo

Aritmética binaria: complemento de 2

1. Suma de números positivos

$$
\begin{array}{r}
(7)_{10} = 00000111 \\
+(4)_{10} = 00000100 \\
\hline
(11)_{10} = 00001011
\end{array}
$$

Aritmética binaria: complemento de 2

2. Suma de números negativos

$$(-5)_{10} = \quad 11111011$$

$$+(-9)_{10} = \quad 11110111$$

$$(-14)_{10} = 1\,11110010$$

Se desecha

Aritmética binaria: complemento de 2

3. Resta con resultado positivo

$$(15)_{10} = \ \ 00001111$$
$$+(\text{-}6)_{10} = \ \ 11111010$$

$$(9)_{10} = 100001001$$

↑

Se desecha

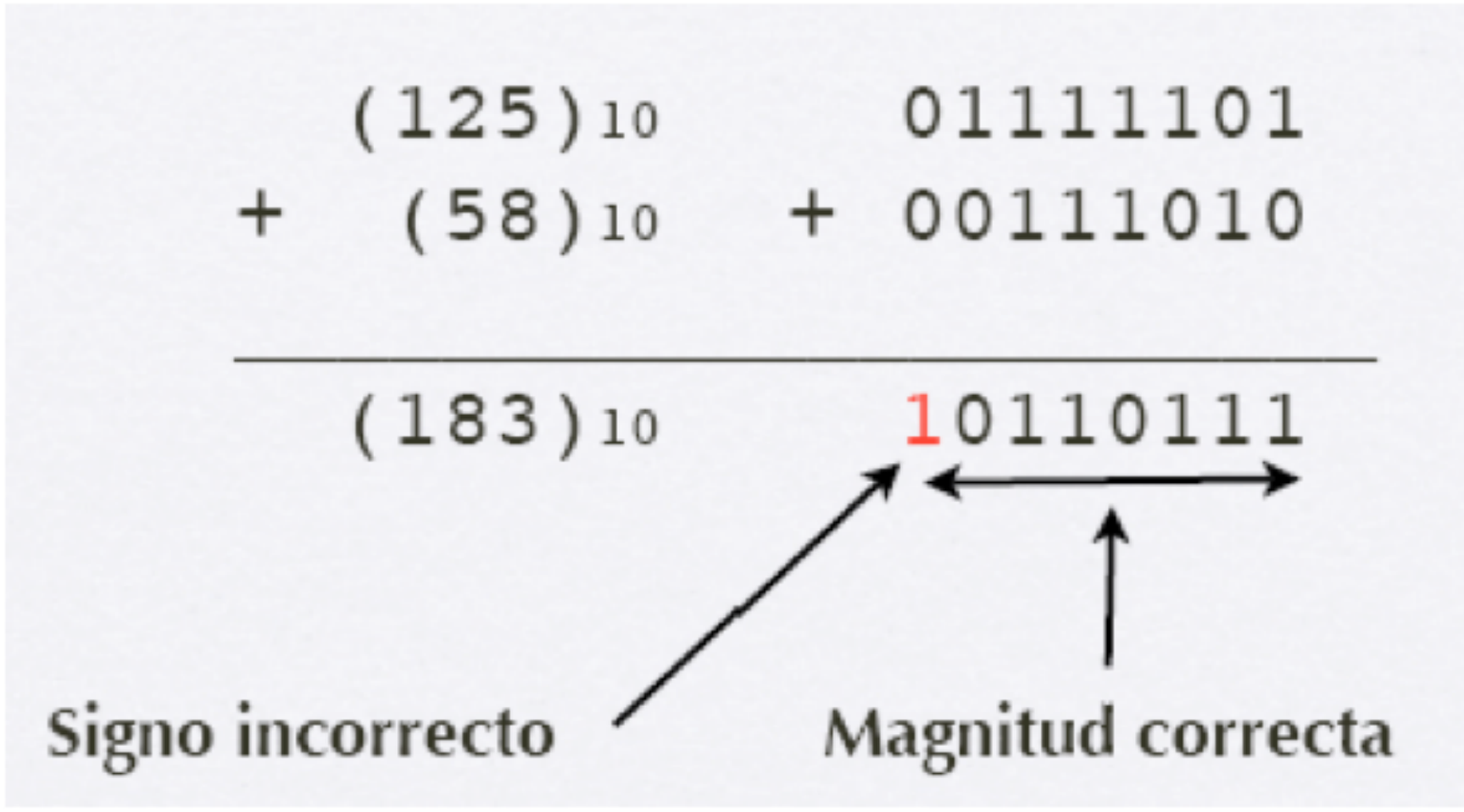
Aritmética binaria: complemento de 2

4. Resta con resultado negativo

$$
\begin{array}{rr}
(16)_{10} = & 00010000 \\
+(\text{-}24)_{10} = & 11101000 \\
\hline
(\text{-}8)_{10} = & 11111000
\end{array}
$$

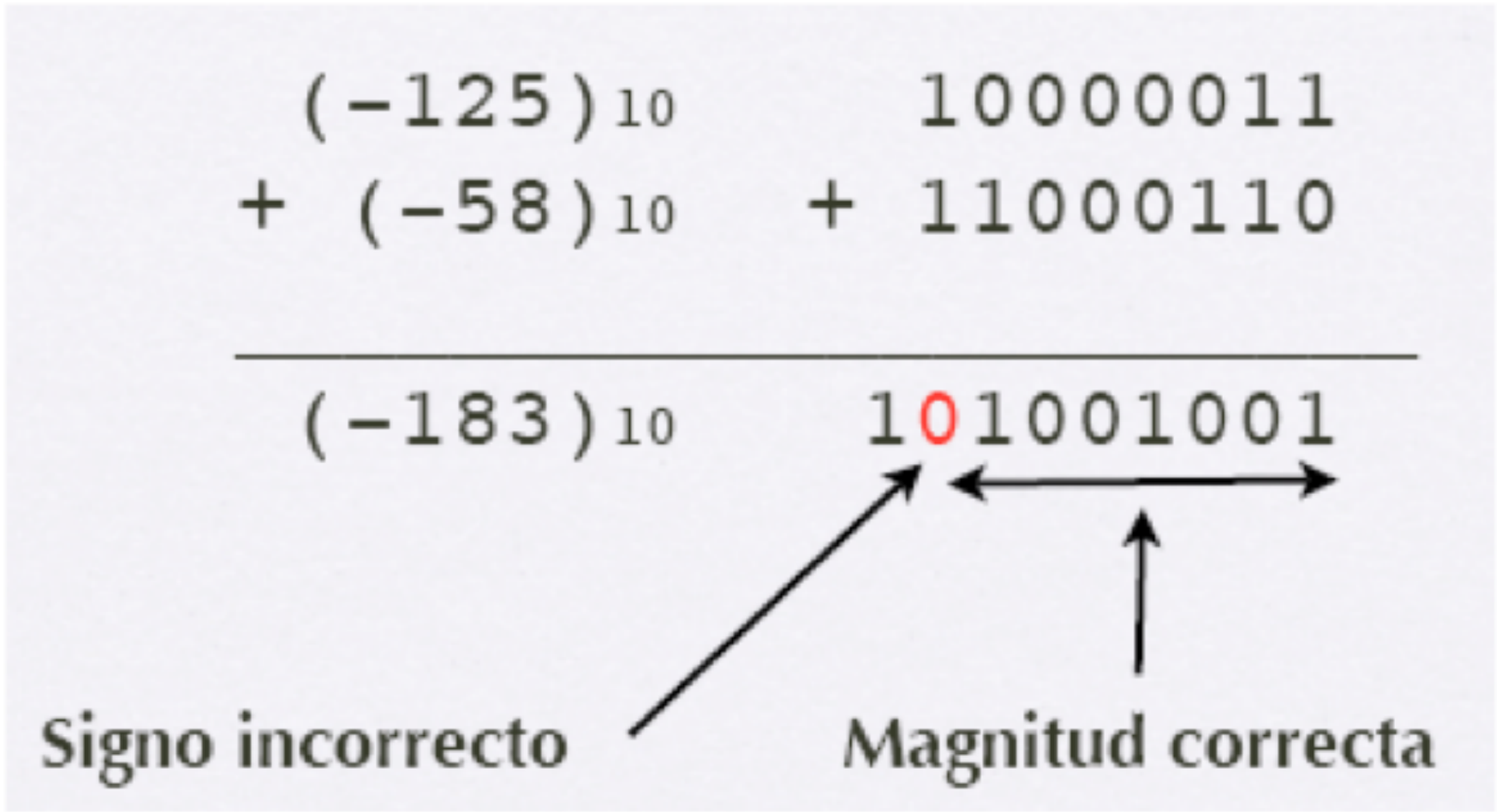
Aritmética binaria: complemento de 2

Overflow positivo



Aritmética binaria: complemento de 2

Overflow negativo



Aritmética binaria: BCD

Resultado es directamente BCD

$(13)_{10}$	0001	0011
+ $(24)_{10}$	+ 0010	0100
$(37)_{10}$	0011	0111

Aritmética binaria: BCD

Resultado no es BCD válido

$(58)_{10}$		0101	1000
+ $(74)_{10}$	+	0111	0100
		no BCD → 1100	1100 ← no BCD
+ $(66)_{10}$	+	0110	0110
$(132)_{10}$		0001	0011	0010

En la primera suma se generan dos dígito que no son BCD. Esta situación se corrige sumando seis (0110) a cada uno de los dígitos erróneos.

REGLA: sumar seis a cada dígito que resulta no BCD.

Aritmética binaria: BCD

Resultado es BCD incorrecto

		1	1	
$(78)_{10}$			0111	1000
+ $(89)_{10}$	+		1000	1001
		0001	0000	0001
+ $(66)_{10}$	+	0000	0110	0110
$(167)_{10}$		0001	0110	0111

En la primera suma se generan dos dígitos BCD incorrectos. Esta situación se detecta por la ocurrencia de los dos "*carry*" intermedios, en color azul. A cada dígito se le sumó seis para alcanzar el resultado correcto

REGLA: sumar seis a dígitos BCD que generan *"carry"* intermedio"

Aritmética binaria: BCD

Resultado es BCD incorrecto

(58)10		0101	1000	
+ (45)10	+	0100	0101	
BCD →		1001	1101	← no BCD
+ (06)10	+	0000	0110	
no BCD →		1010	0011	← BCD
+ (60)10	+	0110	0000	
(103)10	0001	0000	0011	

La primera suma genera un dígito BCD y uno no BCD, al cual se le suma seis para corregirlo. La suma hace que el próximo dígito deje de ser BCD. Por lo tanto se debe sumar seis en una segunda etapa para llegar al resultado correcto.

Códigos detectores de errores

- **Principio**: En un código binario, la ocurrencia de un error simple en uno de los dígitos de cualquier palabra del código puede producir otra palabra, incorrecta pero válida.
- **Ejemplo**: Un error en el dígito menos significativo de la palabra 0110 (número 6) del código BCD, resulta en la palabra 0111 (número 7). Como 0111 pertenece al código BCD, será interpretada en forma incorrecta por el receptor.
- **Código detector de errores simples**: Posee la propiedad de que la ocurrencia de un error simple, transforma una palabra válida en una inválida. Esta propiedad se obtiene, haciendo que la distancia entre todas las palabras del código sea al menos dos, es decir, que la diferencia de bits sea al menos dos.

Códigos detectores de errores

Ejemplos de códigos para los 10 dígitos decimales:

Dígito decimal	BCD + paridad par 8 4 2 1 *p*	Código 2 de 5 0 1 2 4 7
0	0 0 0 0 0	0 0 0 1 1
1	0 0 0 1 1	1 1 0 0 0
2	0 0 1 0 1	1 0 1 0 0
3	0 0 1 1 0	0 1 1 0 0
4	0 1 0 0 1	1 0 0 1 0
5	0 1 0 1 0	0 1 0 1 0
6	0 1 1 0 0	0 0 1 1 0
7	0 1 1 1 1	1 0 0 0 1
8	1 0 0 0 1	0 1 0 0 1
9	1 0 0 1 0	0 0 1 0 1

Códigos autocorrectores de errores simples

- **Principio**: En general, se dice que un código es autocorrector, si la palabra del código puede deducirse a partir de la palabra errónea.
- **Ejemplo**: supongamos un código que tiene sólo dos palabras, 000 y 111. Si un solo error ocurre en la primera palabra, esta puede cambiar a 001, 010 o 100. Si un solo error ocurre en la segunda, esta puede cambiar a 110, 101 o 011. Como los dos conjuntos de errores son diferentes, entonces, asumiendo sólo un error simple, es posible determinar la palabra correcta.
- **Código autocorrector de errores simples**: Debe poseer la cualidad corregir errores simples, por lo que la distancia entre las palabras del código debe ser de 3.

Código Hamming

Principios básicos para construir un código Hamming

• A cada palabra de ***m*** dígitos del código original, se agregan ***k*** dígitos de "chequeo" de paridad, p_1, p_2, ... p_k , formando una nueva palabra de ***(m+k)*** dígitos.

• A la posición de cada uno de los ***(m+k)*** dígitos se le asigna un valor decimal, ***1*** al más significativo hasta ***m+k*** al menos significativo.

• Se efectúan ***k*** "chequeos" de paridad en algunos dígitos de la palabra del código, registrando su valor (***1*** o ***0***), dependiendo, si se ha detectado o no un error.

• Los "chequeos" de paridad dan origen a un número binario c_1, c_2, ... c_k, cuyo valor corresponde a la posición del bit erróneo, o es cero si no hay error.

• El número ***k*** debe ser suficientemente grande para describir la posición de cualquiera de los errores simples posibles. Consecuentemente, ***k*** debe satisfacer la desigualdad :

$$2^k \geq m + k + 1$$

Ejemplo: código Hamming para BCD

BCD		Decimal
0000	→	0
0001	→	1
0010	→	2
0011	→	3
0100	→	4
0101	→	5
0110	→	6
0111	→	7
1000	→	8
1001	→	9

Las palabras contienen 4 bits, por lo tanto ***m=4***

Ejemplo: código Hamming para BCD

Con ***m* = 4** y utilizando $2^k \geq m+k+1$ se tiene ***k* = 3**

Posición del Error	Número de posición
	c_3 c_2 c_1
0 (no hay error)	0 0 0
1	0 0 1
2	0 1 0
3	0 1 1
4	1 0 0
5	1 0 1
6	1 1 0
7	1 1 1

$c_1 \longrightarrow p_1$ se selecciona para establecer **paridad par** en las posiciones **1, 3, 5, 7**

$c_2 \longrightarrow p_2$ se selecciona para establecer **paridad par** en las posiciones **2, 3, 6, 7**

$c_3 \longrightarrow p_3$ se selecciona para establecer **paridad par** en las posiciones **4, 5, 6, 7**

Ejemplo: código Hamming para BCD

Ejemplo para el número 4 (0100 en BCD):

	Posición						
	1	2	3	4	5	6	7
	p_1	p_2	m_1	p_3	m_2	m_3	m_4
Mensaje original en BCD →			0		1	0	0
Paridad par en 1, 3, 5, 7 implica $p_1 = 1$ →	1		0		1	0	0
Paridad par en 2, 3, 6, 7 implica $p_2 = 0$ →	1	0	0		1	0	0
Paridad par en 4, 5, 6, 7 implica $p_3 = 1$ →	1	0	0	1	1	0	0

Ejemplo: código Hamming para BCD

Código completo:

Dígito Decimal	Posición	1 p_1	2 p_2	3 m_1	4 p_3	5 m_2	6 m_3	7 m_4
0		0	0	0	0	0	0	0
1		1	1	0	1	0	0	1
2		0	1	0	1	0	1	0
3		1	0	0	0	0	1	1
4		1	0	0	1	1	0	0
5		0	1	0	0	1	0	1
6		1	1	0	0	1	1	0
7		0	0	0	1	1	1	1
8		1	1	1	0	0	0	0
9		0	0	1	1	0	0	1

Código Hamming para largo arbitrario

Algoritmo para construir códigos Hamming de largo arbitrario:

1. Los bits de paridad (p1, p2, p3, etc.) se ubican en las posiciones que son potencias de dos (i.e. 1, 2, 4, 8, etc.).

2. Los bits de datos se ubican en las posiciones que no son potencias de dos (i.e. 3, 5, 6, 7, etc.).

3. Los bits de paridad verifican la paridad de sólo algunos bits, siguiendo la siguiente regla: El bit de paridad "*n*", partiendo de la posición 2^{n-1}, comprueba 2^{n-1} bits, salta 2^{n-1} bits, comprueba 2^{n-1} bits, salta 2^{n-1} bits, etc. (e.g. p1 comprueba los bits: 1, 3, 5, 7,.....; p2 comprueba los bits: 2, 3, 6, 7, 10, 11, ...; p3 comprueba los bits: 4, 5, 6, 7, 12, 13, 14, 15, 20, ...).

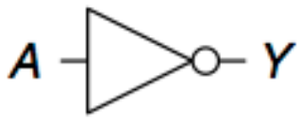
Recuerden que el número de bits de un código corrector de errores simples debe cumplir la siguiente la regla:

$$2^k \geq m + k + 1$$

donde *m* es el número de bits de datos, y *k* es el número de bits de paridad.

Compuertas lógicas

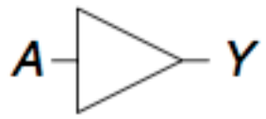
NOT



$Y = \overline{A}$

A	Y
0	1
1	0

BUF



$Y = A$

A	Y
0	0
1	1

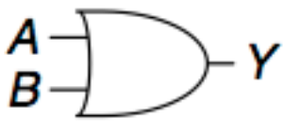
AND



$Y = AB$

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

OR



$Y = A + B$

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

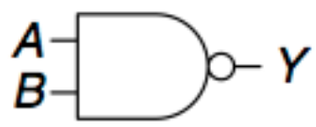
XOR



$Y = A \oplus B$

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

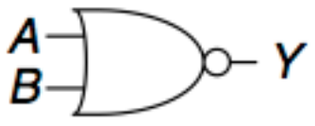
NAND



$Y = \overline{AB}$

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

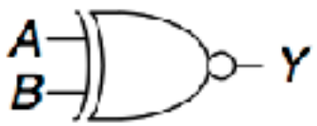
NOR



$Y = \overline{A + B}$

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

XNOR



$Y = \overline{A \oplus B}$

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

Compuertas lógicas

NOR3



$$Y = \overline{A + B + C}$$

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

AND4



$$Y = ABCD$$

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

Compuertas lógicas reales

- Las variables discretas 1 y 0 se representan utilizando cantidades físicas continuas (voltaje, posición, nivel, etc.)
- Es necesario definir una forma de relacionar un valor continuo con un valor discreto.
- Ej.: consideremos representar una señal binaria *A* con un voltaje, 0 volts indica *A* = 0 y 5 volts indica *A* = 1.
- ¿Qué pasaría entonces con 4,3 volts o con 2.8

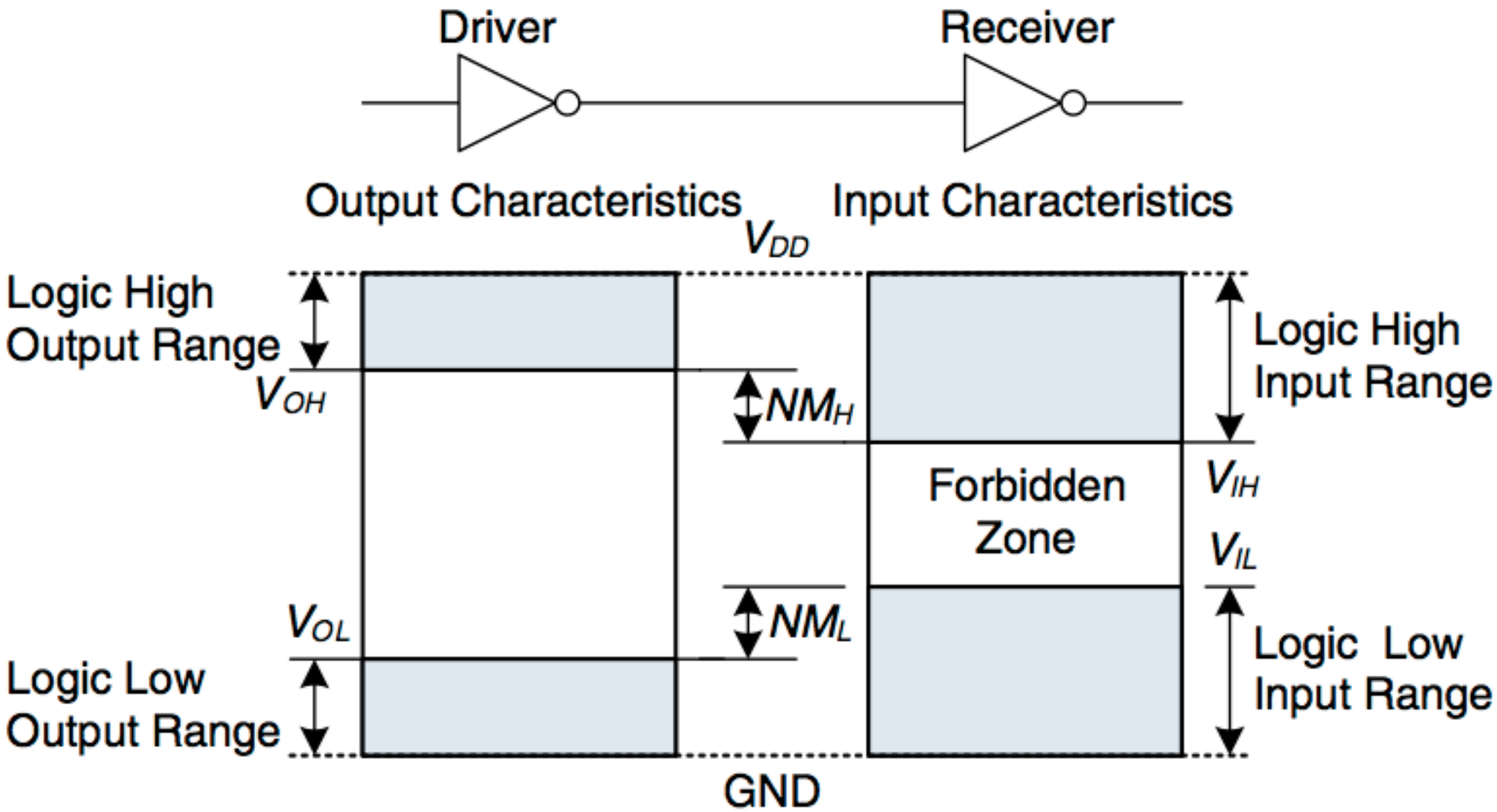
Compuertas lógicas reales

Voltaje de alimentación

- Supongamos que el valor mínimo en el sistema es 0 volts.
- El voltaje más alto viene de la fuente de alimentación y normalmente se le llama V_{DD}.
- V_{DD} fue generalmente 5 V, pero a medida que los transistores progresaron a tamaños más pequeños, V_{DD} se ha reducido a 3.3 V, 2.5 V, 1.8 V, 1.5V, 1.2 V, o aún menos, con el objeto de ahorrar energía y evitar la sobrecarga de los transistores.

Compuertas lógicas reales

Niveles lógicos

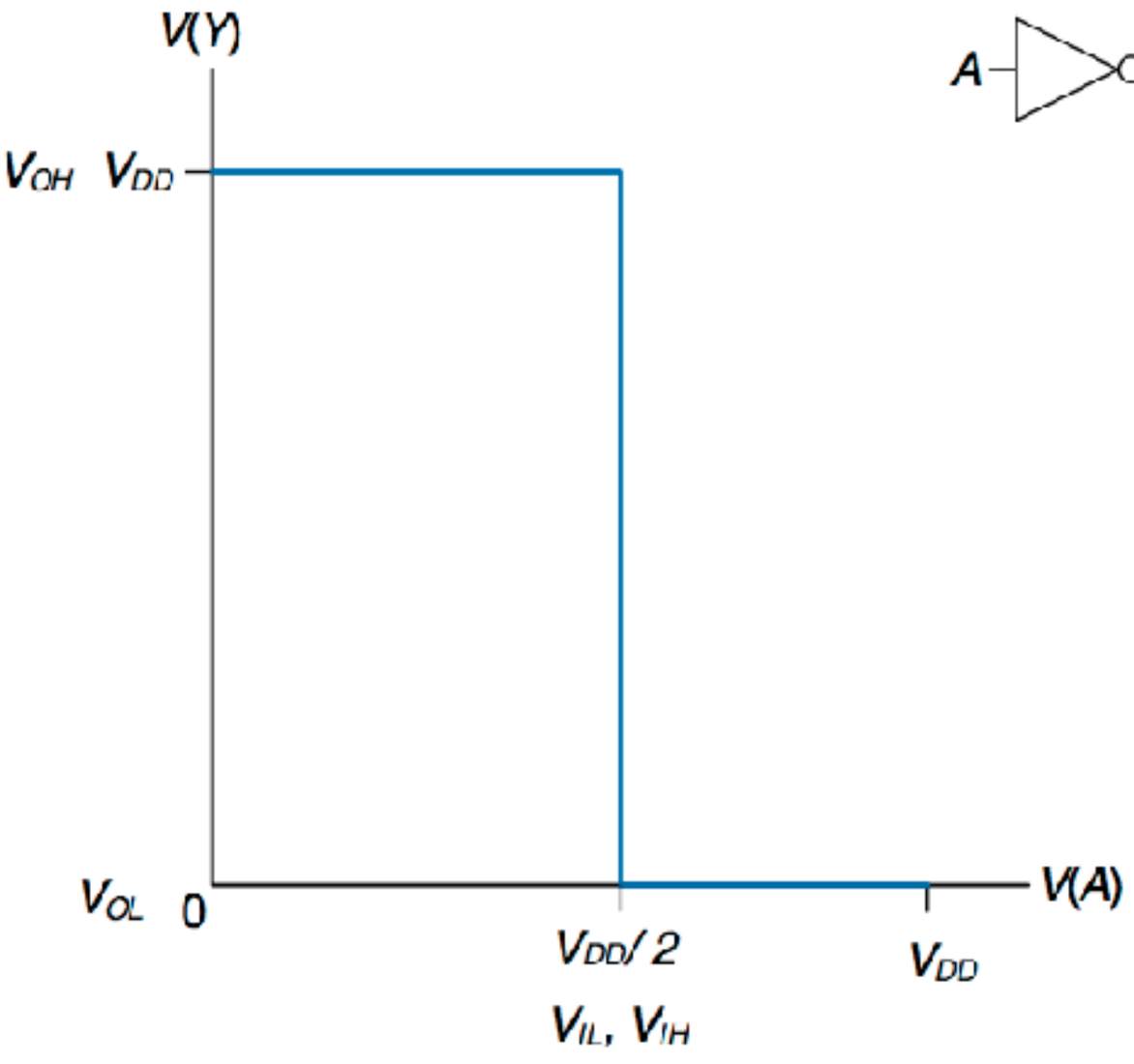


NM_H : Noise Marging High, margen de ruido para el 1
NM_L : Noise Marging Low, margen de ruido para el 0

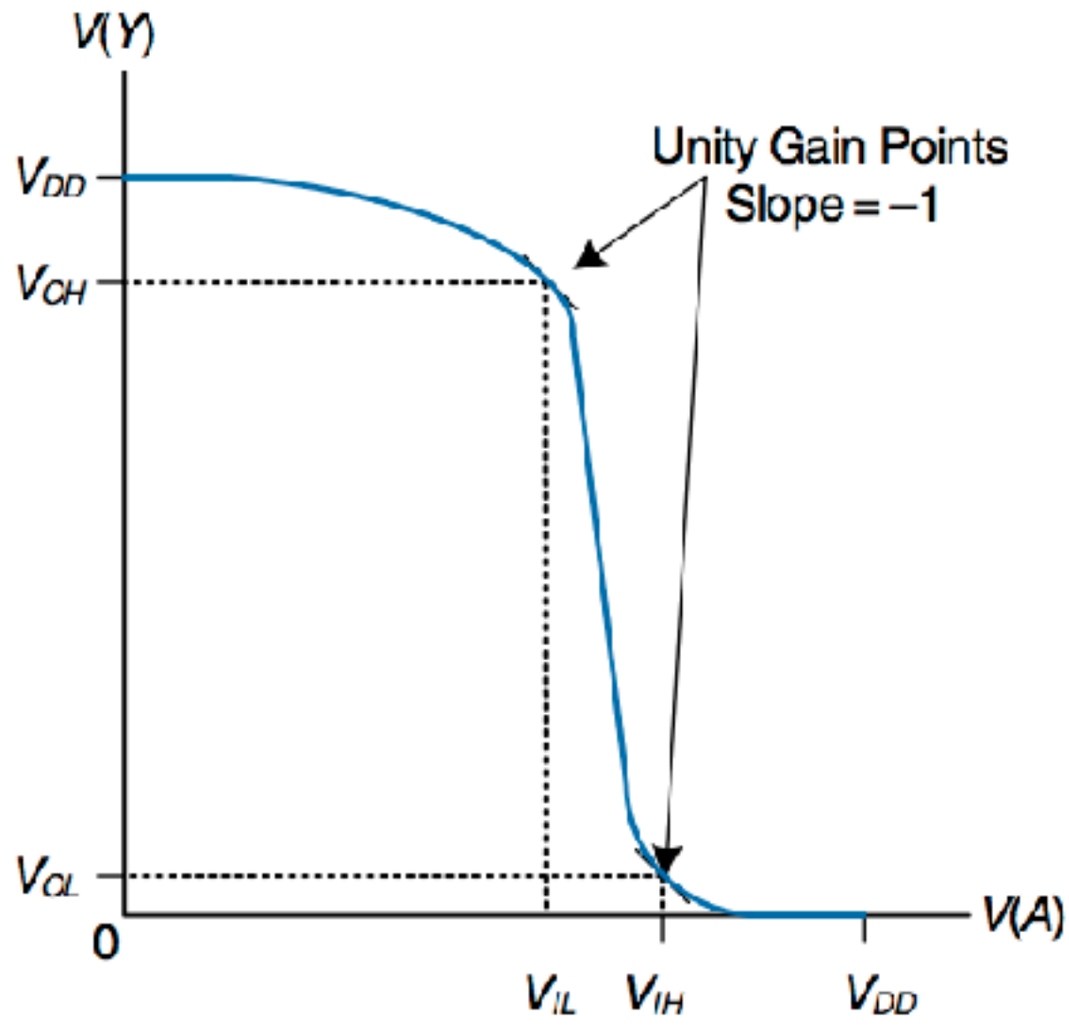
Compuertas lógicas reales

Características de transferencia de CC

Características ideales



Características reales



Compuertas lógicas reales

Niveles lógicos de las familias de 5.0 y de 3.3 V

Logic Family	V_{DD}	V_{IL}	V_{IH}	V_{OL}	V_{OH}
TTL	5 (4.75–5.25)	0.8	2.0	0.4	2.4
CMOS	5 (4.5–6)	1.35	3.15	0.33	3.84
LVTTL	3.3 (3–3.6)	0.8	2.0	0.4	2.4
LVCMOS	3.3 (3–3.6)	0.9	1.8	0.36	2.7

Compuertas lógicas reales

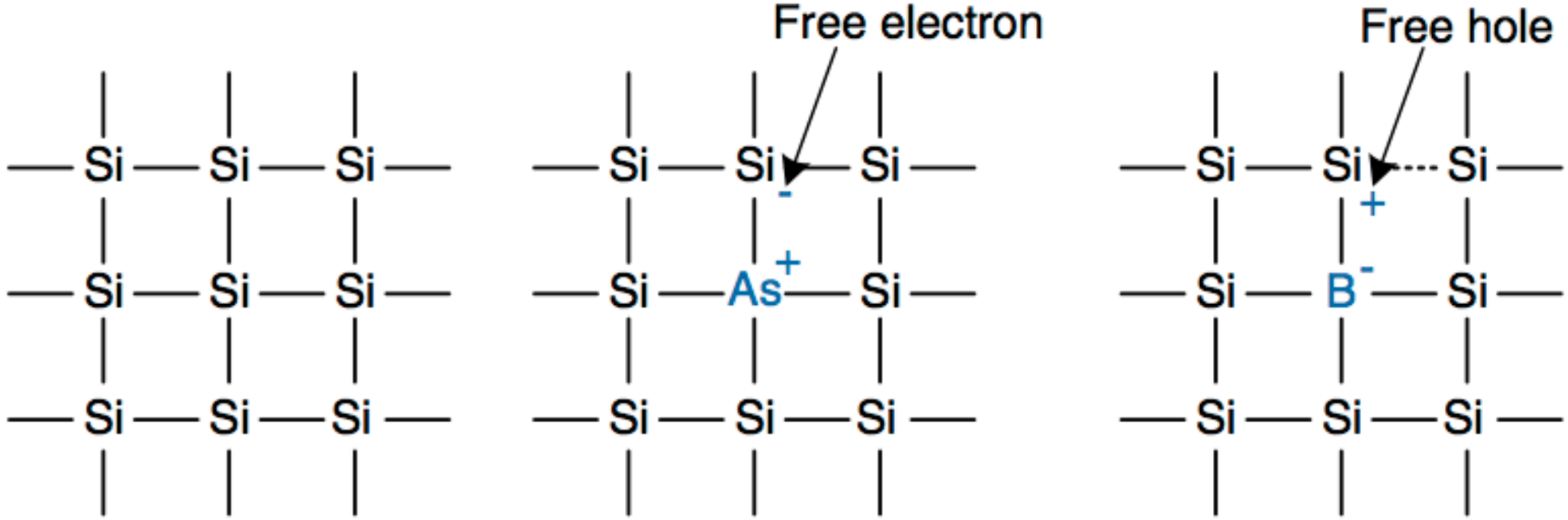
Compatibilidad entre las familias lógicas

		Receiver			
		TTL	CMOS	LVTTL	LVCMOS
Driver	TTL	OK	NO: $V_{OH} < V_{IH}$	MAYBE[a]	MAYBE[a]
	CMOS	OK	OK	MAYBE[a]	MAYBE[a]
	LVTTL	OK	NO: $V_{OH} < V_{IH}$	OK	OK
	LVCMOS	OK	NO: $V_{OH} < V_{IH}$	OK	OK

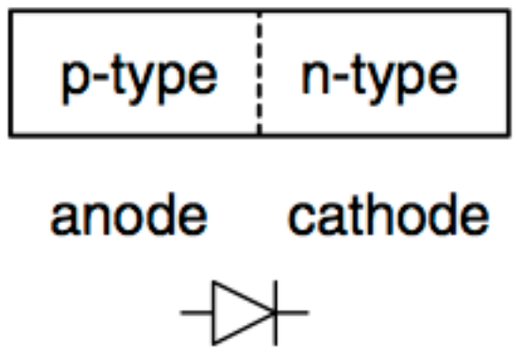
Compuertas lógicas reales

Transistores MOS

Materiales utilizados

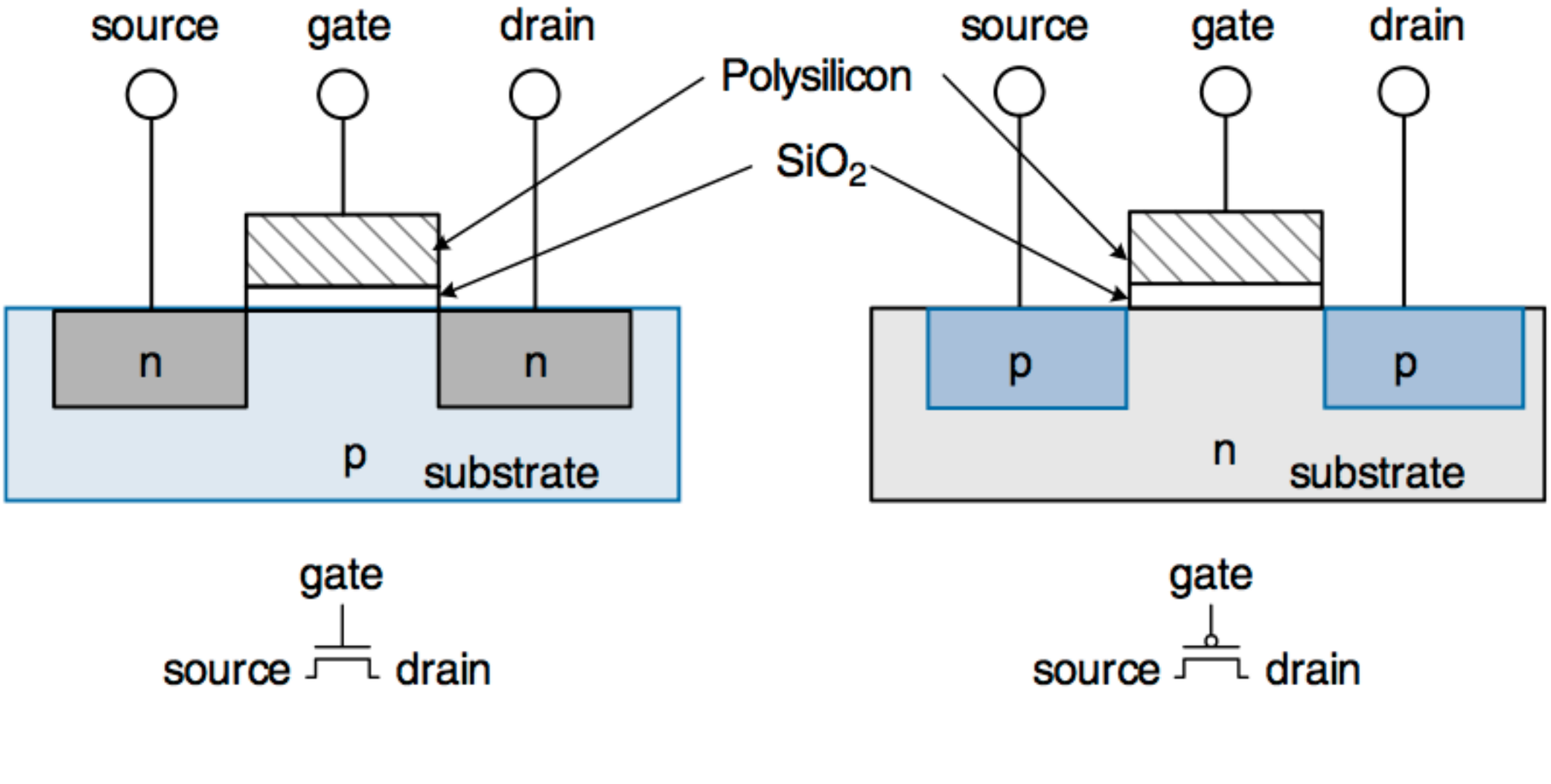


Diodo: estructura y símbolo



Compuertas lógicas reales

Transistores MOS



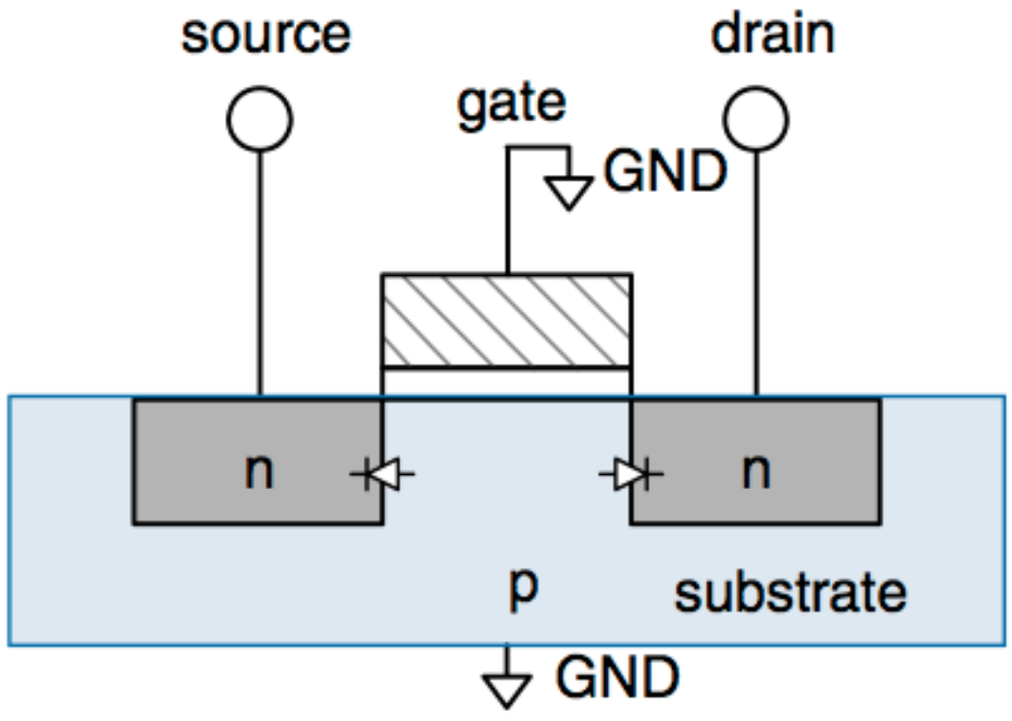
(a) nMOS

(b) pMOS

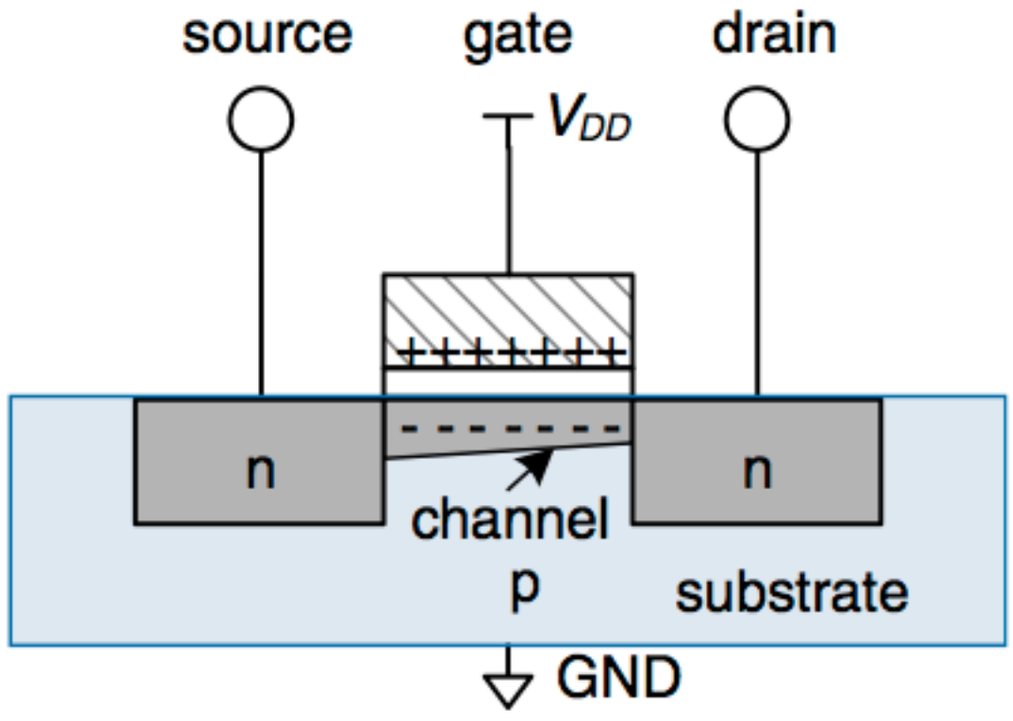
Compuertas lógicas reales

Transistores MOS

Operación del transistor MOS de canal n



(a)

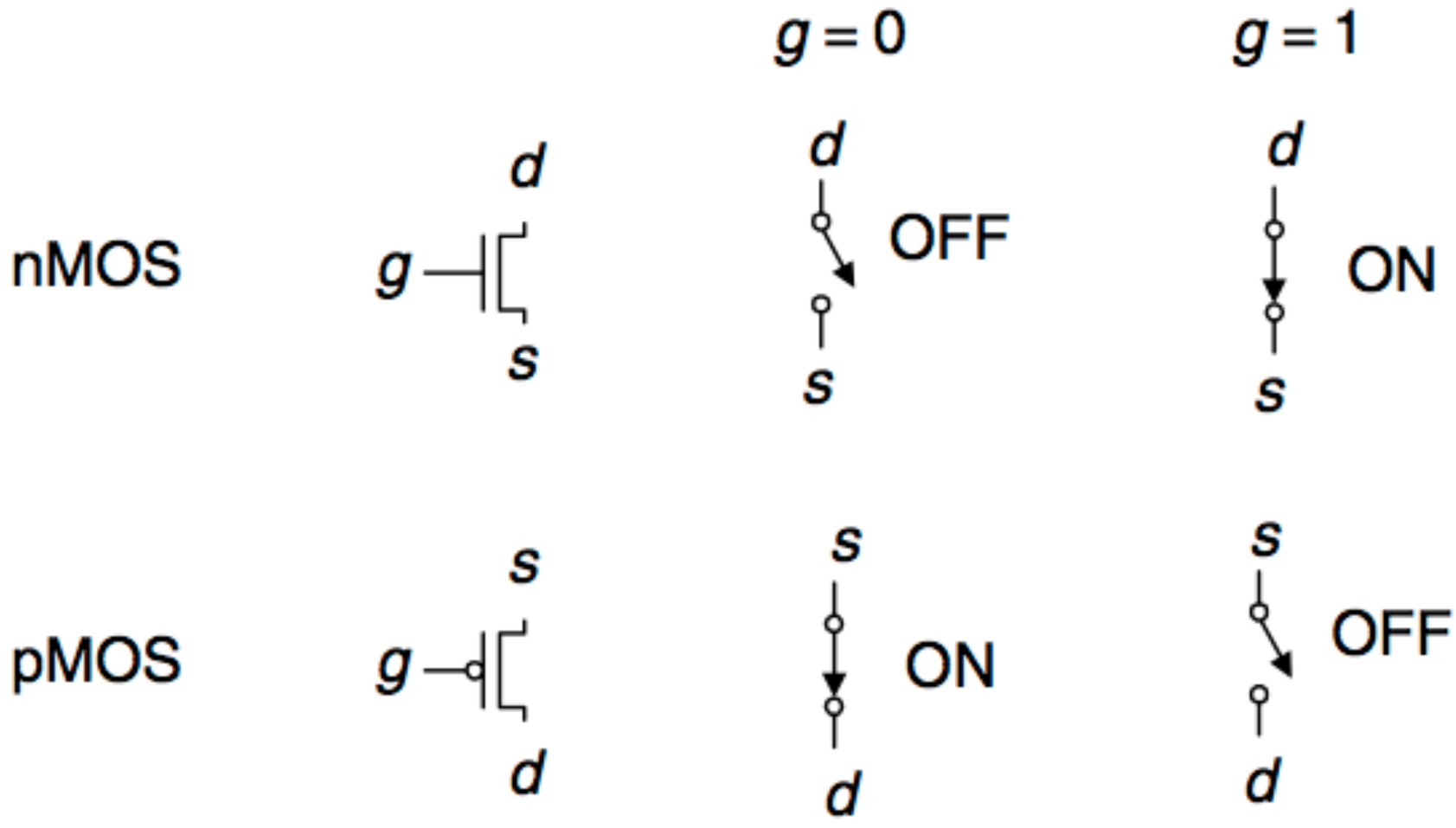


(b)

Compuertas lógicas reales

Transistores MOS

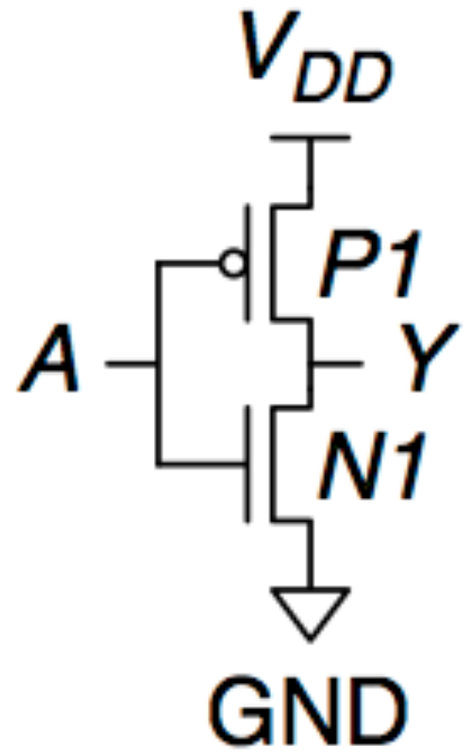
Operación de los transistores MOS como interruptores



Compuertas lógicas reales

Compuertas CMOS

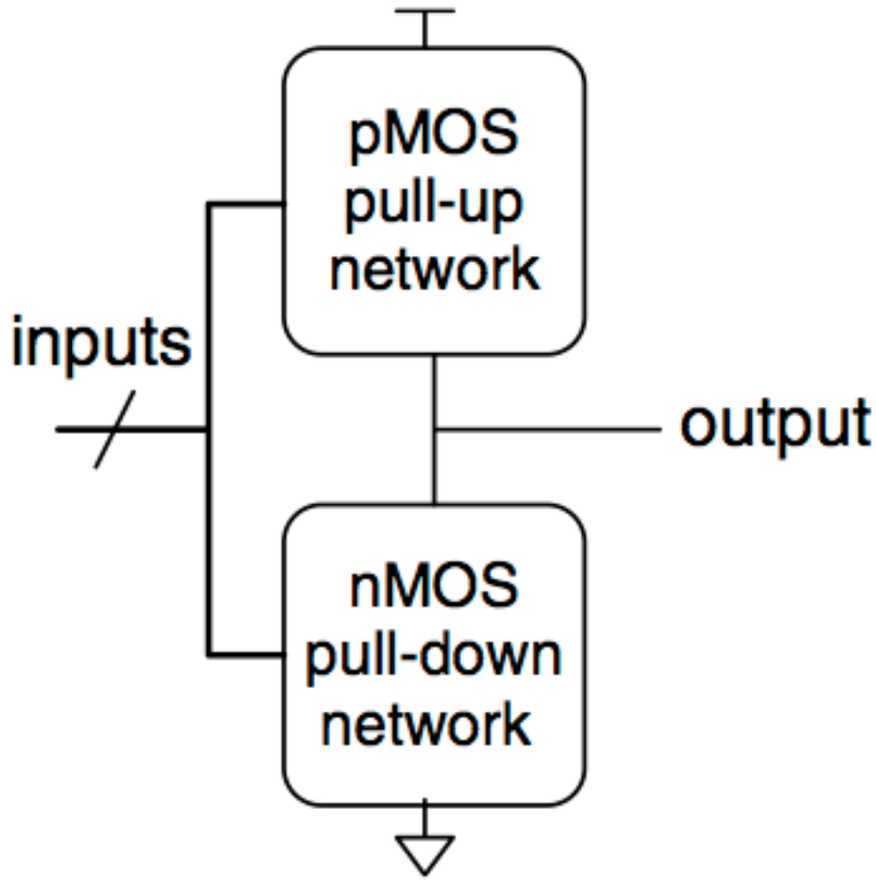
Compuerta NOT



Compuertas lógicas reales

Compuertas CMOS

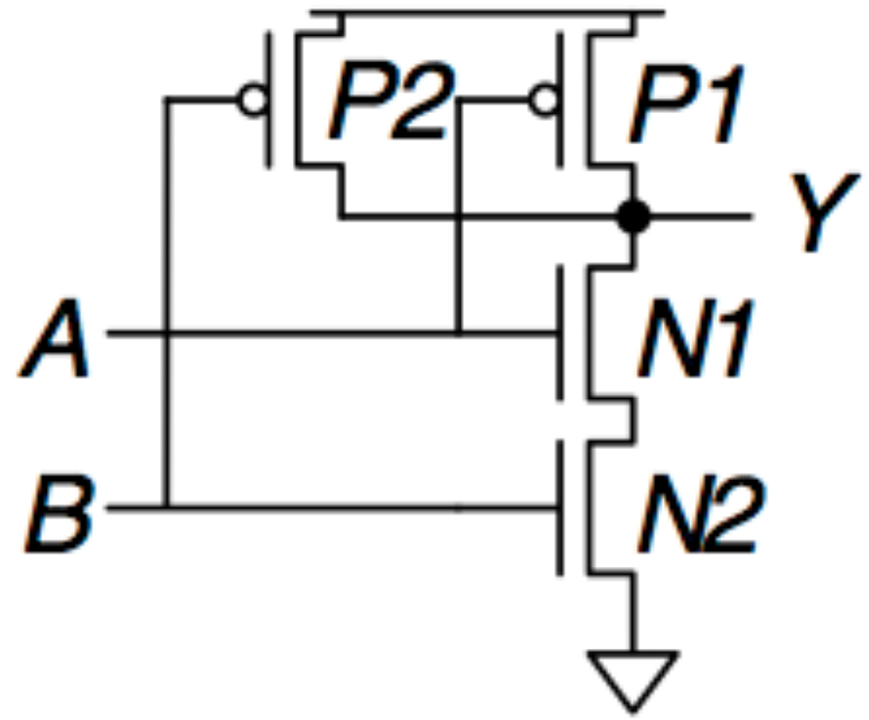
Forma general de una compuerta inversora



Compuertas lógicas reales

Compuertas CMOS

Compuerta NAND de dos entradas

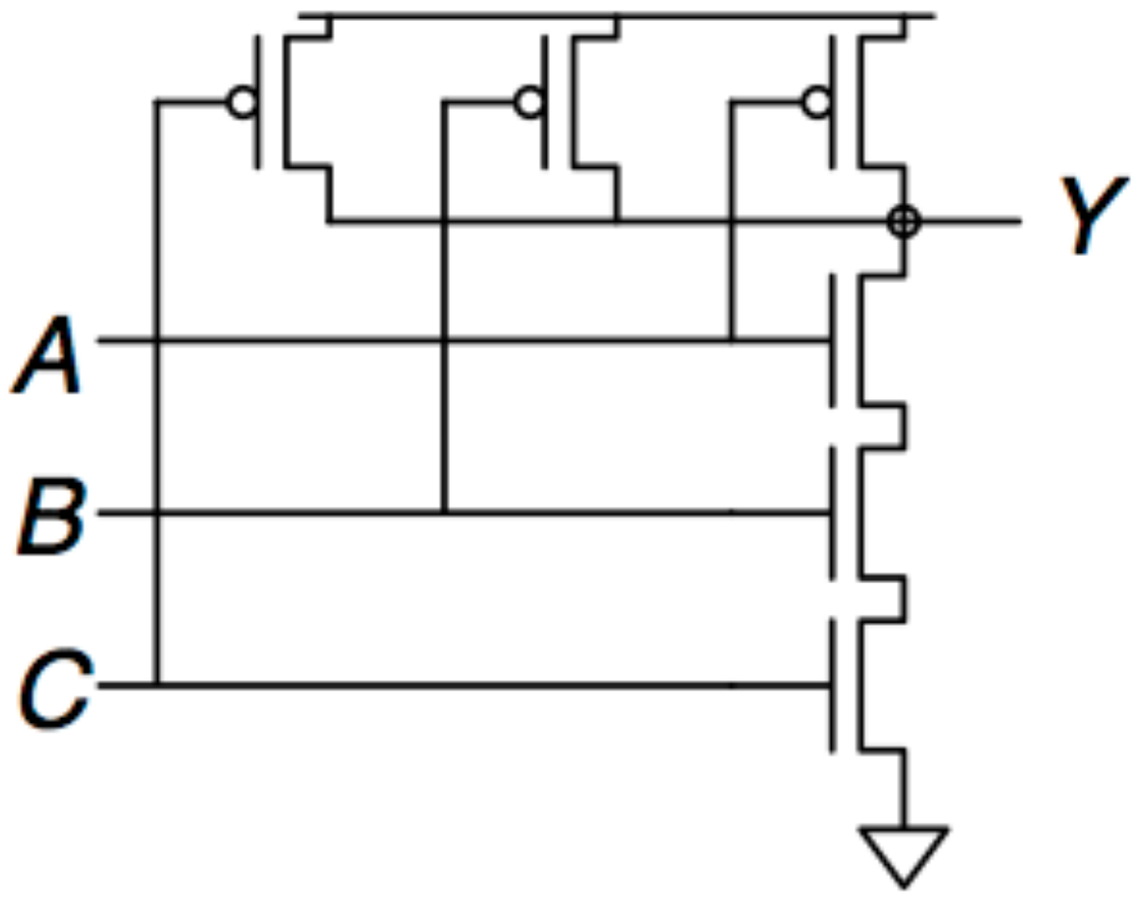


A	*B*	Pull-Down Network	Pull-Up Network	*Y*
0	0	OFF	ON	1
0	1	OFF	ON	1
1	0	OFF	ON	1
1	1	ON	OFF	0

Compuertas lógicas reales

Compuertas CMOS

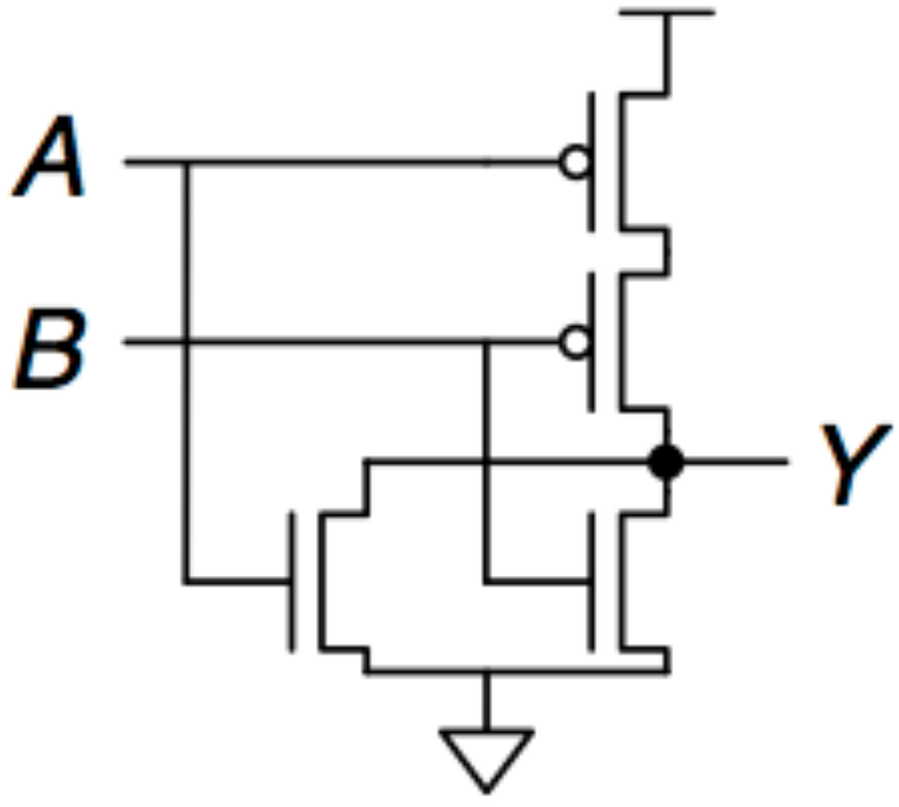
Compuerta NAND de tres entradas



Compuertas lógicas reales

Compuertas CMOS

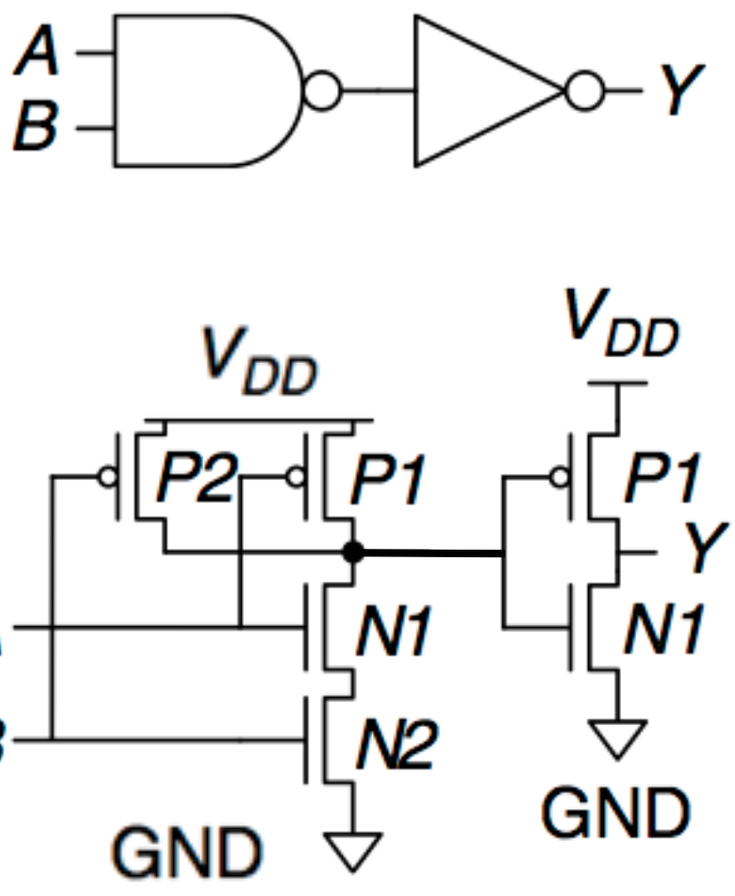
Compuerta NOR de dos entradas



Compuertas lógicas reales

Compuertas CMOS

Compuerta AND de dos entradas



Compuertas lógicas reales

Compuertas CMOS

Compuertas de transmisión

