

Grado en Ingeniería Informática
2018-2019

Apuntes
Tecnología de Computadores

Jorge Rodríguez Fraile¹



Esta obra se encuentra sujeta a la licencia Creative Commons
Reconocimiento - No Comercial - Sin Obra Derivada

¹Universidad: 100405951@alumnos.uc3m.es | Personal: jrf1616@gmail.com

ÍNDICE GENERAL

I Teoría	3
-----------------	----------

Parte I

Teoría



Tecnología de Computadores

Ingeniería Informática

1º Curso, 2º Cuatrimestre

Universidad Carlos III de Madrid



Programa

- Unidad Didáctica I: Fundamentos
 - 1. Representación de la información en los sistemas digitales
 - 2. Algebra de Boole y puertas lógicas
- Unidad Didáctica II: Circuitos combinacionales
 - 3. Circuitos combinacionales
 - 4. Circuitos combinacionales aritméticos. Aritmética Binaria



Programa

- Unidad Didáctica III: Circuitos secuenciales
 - 5. Biestables
 - 6. Circuitos secuenciales síncronos
 - 7. Registros y contadores
- Unidad Didáctica IV: Componentes funcionales
 - 8. Memorias
 - 9. Dispositivos lógicos programables
 - 10. Introducción a los sistemas digitales y los microprocesadores



Bibliografía

- Existe una amplia bibliografía sobre el tema
 - Pueden consultarse numerosas obras disponibles en biblioteca
- Algunos libros recomendados
 - FLOYD, T.L.: Fundamentos de Sistemas Digitales. Prentice-Hall
 - HAYES, J.P.: Introducción al Diseño Lógico Digital. Addison-Wesley
 - MANDADO, E.: Sistemas Electrónicos Digitales, Ed. Marcombo Boixareu Editores
- Se indicará bibliografía específicamente recomendada para cada tema



Material

- Disponible a través de Aula Global
- Incluye:
 - Programa de la asignatura
 - Cronograma
 - Trasparencias de clases
 - Ejercicios propuestos
 - Ejercicios de examen resueltos
 - Manual de prácticas



Prácticas

- **4 prácticas cuya realización es obligatoria en convocatoria ordinaria**
- Quartus® II Web Edition Software
 - Permite diseñar circuitos digitales y simularlos
 - Licencia gratuita
 - Disponible en la página web del distribuidor de la herramienta (<https://www.altera.com/download/sw/dnl-sw-index.jsp>)
 - Tamaño de la descarga: >1 Gb
 - Se utilizará a lo largo del curso para trabajos y ejemplos prácticos



Evaluación

Evaluación continua

- Presentarse a prácticas y a los controles I y II

Actividad	Unidades Didácticas	Valoración
Control I: elimina materia para examen final si nota > 5 y prácticas aprobadas	I y II	14%
Control II: elimina materia para examen final si nota > 5 y prácticas aprobadas	III y IV	21%
Prácticas (Obligatorio)	Cuatro prácticas	25%
Examen final (Obligatorio salvo materia eliminada) nota mínima 3,5/10	Parte 1: I,II Parte 2: III, IV	40%
TOTAL		100%



Evaluación

- **Controles**

- Si se obtiene una nota mayor o igual a 5 y se aprueban las prácticas, la materia correspondiente se elimina de cara al examen final (**solo ordinario**)

- **Convocatoria ordinaria:**

- **Examen final comprende todo el temario**
 - **Nota mínima: 3,5**
 - Se deben realizar ambos controles y las prácticas para puntuar por evaluación continua
 - Si no se cumple con evaluación continua, el examen final puntuará sobre 6.

- **Convocatoria extraordinaria:**

- **Examen final comprende todo el temario**
 - **Nota mínima: 3,5**
 - Se escogerá la mejor nota de evaluación continua/examen sobre 10

- No se guardan notas de un curso académico para otro



Preguntas?



Representación de la Información en los Sistemas Digitales

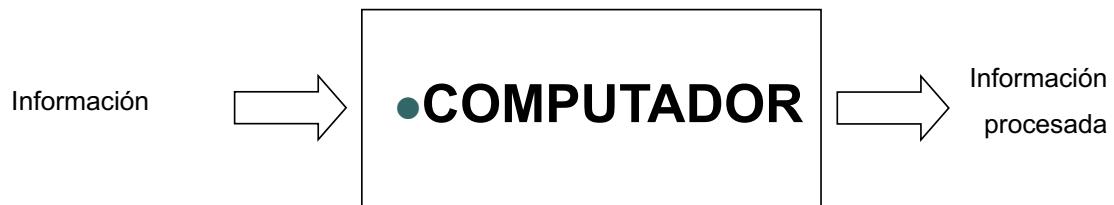
© Luis Entrena, Celia López,
Mario García, Enrique San Millán

Universidad Carlos III de Madrid



Introducción a los computadores

- Computador: Máquina que procesa información





Sistemas analógicos y digitales

- Sistemas analógicos: aquellos cuyas variables toman valores continuos en el tiempo
 - Las magnitudes físicas son en su mayoría analógicas
- Sistemas digitales: aquellos cuyas variables toman valores discretos en el tiempo
 - Se utilizan valores discretos llamados dígitos
 - Precisión limitada
 - Las cantidades digitales son más fáciles de manejar
 - Las magnitudes analógicas se pueden convertir a magnitudes digitales mediante muestreo



Sistemas analógicos y digitales

- Sistema Analógico
- Sistema digital





Sistemas binarios

- Sistemas binarios: sistemas digitales que sólo utilizan dos posibles valores
 - Los dígitos binarios se denominan bits (Binary digiT)
 - Se representan mediante los símbolos 0 y 1, ó L y H
 - Los sistemas binarios son casi los únicos utilizados. Por extensión, se utiliza el término digital como sinónimo de binario
- ¿Por qué binario?
 - Más fiable: mayor inmunidad frente al ruido
 - Más sencillo de construir: sólo hay que distinguir entre dos valores



Índice

- Sistemas de Numeración
- Conversiones entre sistemas de numeración
- Códigos Binarios:
 - Códigos BCD
 - Códigos progresivos y cíclicos
 - Códigos alfanuméricos
 - Códigos detectores y códigos correctores de errores
 - Representación de números enteros y reales



Sistemas de Numeración

- Permiten representar los números mediante dígitos
- El sistema que utilizamos habitualmente es el sistema decimal:
 - $N = a_n 10^n + a_{n-1} 10^{n-1} + \dots + a_1 10 + a_0$
 - Ejemplo: $272_{10} = 2*10^2 + 7*10 + 2$
- Se puede hacer lo mismo pero utilizando bases diferentes a 10:

$$N = a_n b^n + a_{n-1} b^{n-1} + \dots + a_1 b + a_0$$

Dígito Peso
 ↓
 a_{n-1} ↓
 Base



Sistemas de Numeración

- En un sistema con base b los dígitos posibles son:
 - 0, 1, ..., $b-1$
- Con n dígitos se pueden representar b^n números posibles, desde el 0 hasta el b^n-1
- Esta representación sirve también para números que no sean naturales:
 - Ejemplo: $727,23_{10} = 7*10^2 + 2*10 + 7 + 2*10^{-1} + 2*10^{-2}$
- Los sistemas que se utilizan en los sistemas digitales son: binario ($b=2$), octal ($b=8$) y hexadecimal($b=16$)



Sistema Binario

- En este sistema la base es 2. Permiten representar perfectamente la información en los sistemas digitales.
 - Los dígitos posibles son 0 y 1. Un dígito en sistema binario se denomina “bit”.
 - Con n bits se pueden representar 2^n números
- El bit de mayor peso se denomina **bit más significativo** o **MSB** (“Most Significant Bit”), y el bit de menor peso se denomina **bit menos significativo** o **LSB** (“Least Significant Bit”)

MSB LSB

Habitualmente el MSB se escribe a la izquierda
y el LSB a la derecha

- Ejemplo: $(1001010)_2 = 1 \cdot 2^6 + 1 \cdot 2^3 + 1 \cdot 2^1 = 74_{10}$



Sistema Octal

- En este sistema la base es 8.
 - Los dígitos son 0,1,2,3,4,5,6,7
 - Con n dígitos se pueden representar 8^n números
- Está muy relacionado con el sistema binario (8 es una potencia de 2, en concreto $2^3=8$)
 - Esto permite convertir fácilmente de octal a binario y de binario a octal
- Ejemplo:

$$137_8 = 1*8^2 + 3*8^1 + 7*8^0 = 95_{10}$$



Sistema Hexadecimal

- En este sistema la base es 16.
 - Los dígitos son 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F.
 - Está relacionado con el sistema binario ($2^4=16$)
 - Un dígito hexadecimal permite representar lo mismo que 4 bits (ya que $2^4=16$). Un dígito hexadecimal se denomina también “**nibble**”.
 - Dos dígitos hexadecimales equivalen por tanto a 8 bits. El conjunto de 8 bits o dos dígitos hexadecimales, se denomina “**byte**”.
- Notaciones: $23AF_{16} = 23AF_{hex} = 23AFh = 0x23AF = 0x23\ 0xAF$.
- Ejemplo: $23AFh = 2*16^3 + 3*16^2 + 10*16 + 15 = 9135_{10}$



Conversiones entre sistemas de numeración

- Pasar de cualquier sistema a sistema decimal:
 - $N = a_n b^n + a_{n-1} b^{n-1} + \dots + a_1 b + a_0$
 - Ejemplos:
 - $1001010_2 = 1*2^6 + 1*2^3 + 1*2^1 = 74_{10}$
 - $137_8 = 1*8^2 + 3*8^1 + 7*8^0 = 95_{10}$
 - $23AFh = 2*16^3 + 3*16^2 + 10*16 + 15 = 9135_{10}$
- Para pasar de decimal a otro sistema:
 - Método de descomposición en pesos
 - Método de divisiones sucesivas por la base



Método de descomposición en pesos

- Consiste en descomponer el número en potencias de la base.
 - Se busca la potencia de la base (menor) más cercana al número.
 - Se van buscando potencias sucesivamente para que la suma de todas ellas sea el número decimal que se quiere convertir.
 - Finalmente los pesos de las potencias utilizadas se utilizan para representar el número en la base buscada.
- Éste método es útil sólo para sistemas donde las potencias de la base son conocidas. Por ejemplo para sistema binario: 1, 2, 8, 16, 32, 64, 128, 256, ...
- Ejemplo:
 - $25_{10} = 16 + 8 + 1 = 2^4 + 2^3 + 2^0 = 11001_2$



Método de divisiones sucesivas por la base

- Consiste en dividir el número decimal a convertir sucesivamente por la base y los cocientes obtenidos en las divisiones anteriores
 - El último cociente obtenido es el MSB del resultado
 - Los restos obtenidos son el resto de dígitos, siendo el primero de los restos obtenidos el LSB
 - Ejemplo: $25 \underline{) 2 __}$

A diagram showing the conversion of the decimal number 25 to binary. It uses successive division by 2, with remainders circled in red. The remainders are 1, 0, 0, 1, and 1, from top to bottom. An arrow labeled "LSB" points to the first remainder (1), and another arrow labeled "MSB" points to the last remainder (1). A large red arrow on the right points to the final binary representation: $25_{10} = 11001_2$.

- Este método es más general que el anterior. Sirve para convertir de decimal a cualquier otra base.



Conversión de números reales

- La conversión de binario a decimal se hace igual que para números enteros (utilizando pesos negativos para la parte decimal):

$$101,011_2 = 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^1 + 0 \cdot 2^{-1} + 1 \cdot 2^{-2} + 1 \cdot 2^{-3} = \\ = 4 + 1 + 0,25 + 0,125 = 5,375_{10}$$

- La conversión de decimal a binario, se hace en dos partes:

- Se convierte primero la parte entera por el método de divisiones sucesivas por la base (o por descomposición de pesos)
- Luego se convierte la parte decimal por un método análogo, multiplicaciones sucesivas por la base.



Método de multiplicaciones sucesivas por la base (parte decimal)

- Consiste en multiplicar la parte decimal del número por la base sucesivamente.
 - Se multiplica la parte decimal del número por 2. La parte entera del resultado es el primer dígito (MSB de la parte decimal) de la conversión
 - Se vuelve a tomar la parte decimal, y se multiplica por 2 otra vez, y nuevamente la parte entera es el siguiente dígito.
 - Se itera tantas veces como se quiera, según la precisión que se quiera obtener en la conversión.
- Ejemplos:

$$0,3125_{10} = 0,0\ 0101_2$$

$$0,3125 \times 2 = 0,625 \Rightarrow 0$$

$$0,625 \times 2 = 1,25 \Rightarrow 1$$

$$0,25 \times 2 = 0,5 \Rightarrow 0$$

$$0,5 \times 2 = 1 \Rightarrow 1$$

$$0,1_{10} = 0,0\ 0011\ 0011\dots_2$$

$$0,1 \times 2 = 0,2 \Rightarrow 0$$

$$0,2 \times 2 = 0,4 \Rightarrow 0$$

$$0,4 \times 2 = 0,8 \Rightarrow 0$$

$$0,8 \times 2 = 1,6 \Rightarrow 1$$

$$0,6 \times 2 = 1,2 \Rightarrow 1$$

0,2 × 2 = 0,4 => 0 <- se repiten las cuatro cifras, periódicamente

$$0,4 \times 2 = 0,8 \Rightarrow 0$$

$$0,8 \times 2 = 1,6 \Rightarrow 1$$

...



Otros métodos de conversión

- Los sistemas octal y hexadecimal están relacionados con el binario, ya que sus bases son potencias exactas de 2 (la base binaria). Esto permite convertir entre estos sistemas de forma muy sencilla:
 - **OCTAL a BINARIO:** Convertir cada dígito en 3 bits
 - Ejemplo: $735_8 = \underbrace{111}_3 \underbrace{011}_3 \underbrace{101}_3_2$
 - **BINARIO a OCTAL:** Agrupar en grupos de 3 bits y convertirlos de forma independiente a octal.
 - Ejemplo: $\underbrace{1}_3 \underbrace{011}_3 \underbrace{100}_3 \underbrace{011}_3_2 = 1343_8$
 - **HEXADECIMAL a BINARIO:** Convertir cada dígito en 4 bits
 - Ejemplo: $3B2h = \underbrace{0011}_4 \underbrace{1011}_4 \underbrace{0010}_4_2$
 - **BINARIO a HEXADECIMAL:** Agrupar en grupos de 4 bits y convertirlos de forma independiente a octal
 - Ejemplo: $\underbrace{10}_4 \underbrace{1110}_4 \underbrace{0011}_4_2 = 2E3h$



Códigos Binarios

- Los códigos binarios son códigos que utilizan únicamente 0s y 1s para representar la información
- La información que se puede representar con códigos binarios puede ser de múltiples tipos:
 - Números naturales
 - Números enteros
 - Números reales
 - Caracteres alfabéticos y otros símbolos
- Una misma información (por ejemplo un número natural) se puede representar utilizando diferentes códigos.
 - Es importante especificar siempre qué código que se está utilizando cuando se representa una información en un código binario.



Código Binario Natural

- Es un código binario en el que se representa un número natural mediante su representación en sistema binario
 - Es el código binario más simple
 - Aprovecha que la representación en sistema binario de un número natural utiliza únicamente 0s y 1s
- Notación: Utilizaremos el indicador “BIN” para indicar que un código binario es el código binario natural:
 - $1001_{\text{BIN}} = 1001_2$



Códigos BCD (“Binary-Coded Decimal”)

- Permiten representar números naturales de una forma alternativa al binario natural.
- Se asigna un código de 4 bits a cada dígito decimal. Un número decimal se codifica en BCD dígito a dígito.
- El código BCD más habitual es el BCD natural → (existen otros códigos BCD).
- Ejemplo:
 - $78_{10} = 0111\ 1000_{BCD}$
- La codificación BCD de un número no tiene por qué coincidir con el código binario natural:
 - $78_{10} = 1001110_{BIN}$
- INCONVENIENTE: No todas las combinaciones corresponden a un código BCD. Por ejemplo, 1110_{BCD} no existe.
- VENTAJA: Facilidad de conversión decimal-binario.

dígito decimal	Código BCD
0	0 0 0 0
1	0 0 0 1
2	0 0 1 0
3	0 0 1 1
4	0 1 0 0
5	0 1 0 1
6	0 1 1 0
7	0 1 1 1
8	1 0 0 0
9	1 0 0 1



Códigos progresivos y cíclicos

- Dos codificaciones binarias se dice que son **adyacentes** si sólo hay bit diferente entre ambas.
 - **0000** y **0001** son adyacentes, ya que sólo difieren en el último bit
 - **0001** y **0010** no son adyacentes, ya que los dos últimos bits son diferentes
- Un código se dice que es **progresivo** si todas las codificaciones consecutivas son adyacentes.
 - El código binario natural no es progresivo, ya que 0001 y 0010 no son adyacentes.
- Un código se dice que es **cíclico** si además la primera y la última codificación son adyacentes.
- Los códigos progresivos y cíclicos más utilizados son:
 - Código Gray
 - Código Johnson



Código Gray

- El Código Gray es un código progresivo y cíclico
- Ejemplo de Código Gray de 3 bits:

Decimal	Código Gray	
0	0 0 0	
1	0 0 1	
2	0 1 1	
3	0 1 0	 Todas las codificaciones consecutivas son adyacentes
4	1 1 0	
5	1 1 1	
6	1 0 1	
7	1 0 0	



Código Gray

- Construcción del código Gray de n bits:
 - Primero se copian los códigos de n-1 bits y se añaden otros n-1 copiando los anteriores en orden inverso
 - Luego se añade un cero a la izquierda en los de arriba y un uno en los de abajo
- Código de 1 bit:

	0
	1
- Código de 2 bits:

0	0
1	1
1	0
0	1

$$\begin{array}{r} 0 \\ 1 \\ \hline 1 \\ 0 \end{array} \quad \rightarrow \quad \begin{array}{r} 0 \ 0 \\ 0 \ 1 \\ \hline 1 \ 1 \\ 1 \ 0 \end{array}$$



Código Gray

- Código de 3 bits:

0 0		0 0 0
0 1		0 0 1
1 1		0 1 1
1 0		0 1 0
1 0	→	1 1 0
1 1		1 1 1
0 1		1 0 1
0 0		1 0 0

- Iterando se pueden construir los códigos Gray de n bits



Conversión entre los códigos Gray y Binario Natural

Se puede convertir directamente de Gray a Binario y de Binario a Gray, sin necesidad de construir toda la tabla:

BINARIO A GRAY:

$$(A_0 A_1 A_2 \dots A_n)_{\text{BIN}} \rightarrow (B_0 B_1 B_2 \dots B_n)_{\text{GRAY}}$$

- $B_0 = A_0$
- $B_1 = A_0 + A_1$
- $B_2 = A_1 + A_2$
- ...
- $B_n = A_{n-1} + A_{n-2}$



Ejemplo:

$$1011_{\text{BIN}} \rightarrow 1110_{\text{GRAY}}$$

GRAY A BINARIO:

$$(A_0 A_1 A_2 \dots A_n)_{\text{GRAY}} \rightarrow (B_0 B_1 B_2 \dots B_n)_{\text{BIN}}$$

- $B_0 = A_0$
- $B_1 = A_1 + B_0$
- $B_2 = A_2 + B_1$
- ...
- $B_n = A_n + B_{n-1}$



Ejemplo:

$$1011_{\text{GRAY}} \rightarrow 1101_{\text{BIN}}$$

BIN	GRAY
0 0 0 0	0 0 0 0
0 0 0 1	0 0 0 1
0 0 1 0	0 0 1 1
0 0 1 1	0 0 1 0
0 1 0 0	0 1 1 0
0 1 0 1	0 1 1 1
0 1 1 0	0 1 0 1
0 1 1 1	0 1 0 0
1 0 0 0	1 1 0 0
1 0 0 1	1 1 0 1
1 0 1 0	1 1 1 1
1 0 1 1	1 1 1 0
1 1 0 0	1 0 1 0
1 1 0 1	1 0 1 1
1 1 1 0	1 0 0 1
1 1 1 1	1 0 0 0



Código Johnson

- Es otro código progresivo y cíclico
- En cada codificación aparecen agrupados los ceros a la izquierda y los unos a la derecha, o viceversa.
- Ejemplo código Johnson de 3 bits:

Decimal	Johnson		
0	0	0	0
1	0	0	1
2	0	1	1
3	1	1	1
4	1	1	0
5	1	0	0



Códigos alfanuméricos

- Representan símbolos, que pueden ser:
 - Dígitos
 - Letras mayúsculas y minúsculas
 - Signos de puntuación
 - Caracteres de control (espacio, salto de línea, retorno de carro, etc.)
 - Otros símbolos gráficos (operadores matemáticos, etc.)
- Un código alfanumérico mínimo que contenga los 10 dígitos, las 26 letras del alfabeto inglés, mayúsculas y minúsculas (52), necesita al menos 6 bits.
- Los códigos más utilizados en la actualidad son:
 - Código ASCII (7 bits)
 - Códigos ASCII extendidos (8 bits)
 - Códigos unicode (8-32 bits)



Códigos ASCII y ASCII extendidos

- El código ASCII (“American Standard Code for Information Interchange”) fue publicado por primera vez en 1963.
- Es un código de 7 bits (128 códigos) estándar que contiene:
 - Dígitos
 - Letras mayúsculas y minúsculas del alfabeto inglés internacional
 - Signos de puntuación
 - Caracteres básicos de control
- Los códigos ASCII extendidos se utilizan para añadir caracteres adicionales:
 - No son estándar, difieren de una región a otra
 - Los 128 primeros códigos coinciden con el ASCII estándar por compatibilidad



Código ASCII Estándar

	0	1	2	3	4	5	6	7
0	NUL	DLE	space	Ø	@	P	~	p
1	SOH	DC1 XON	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3 XOFF	#	3	C	S	c	s
4	EOT	DC4	\$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	'	7	G	W	g	w
8	BS	CAN	(8	H	X	h	x
9	HT	EM)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	:	K	[k	{
C	FF	FS	,	<	L	\	l	
D	CR	DS	-	=	M]	m	}
E	SO	RS	.	>	N	^	n	~
F	SI	US	/	?	O	_	o	del



Códigos ASCII Extendidos

EJEMPLO:

ACII extendido LATIN-1
(ISO 8859-1)

	-0	-1	-2	-3	-4	-5	-6	-7	-8	-9	-A	-B	-C	-D	-E	-F
0-	0000	0008	0010	0018	0020	0028	0030	0038	0040	0048	0050	0058	0060	0068	0070	0078
1-	0080	0091	0093	0095	0094	0096	0090	0097	0098	0099	00914	00916	0091C	00919	0091E	00917
2-	!	"	#	\$	%	&	'	()	*	+	,	.	/		
3-	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4-	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5-	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	-
6-	~	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7-	p	q	r	s	t	u	v	w	x	y	z	{	}	~	~	~
8-																
9-																
A-	í	é	£	¤	¥	₩	₪	₪	₪	₪	₪	₪	₪	₪	₪	₪
B-	º	±	²	³	⁴	µ	¶	·	·	·	·	·	·	·	·	·
C-	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	È	Ì	Í	Í	Í
D-	Ð	Ñ	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ý	Þ	Þ	Þ	Þ
E-	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	í	í
F-	ð	ñ	ó	ô	õ	ö	÷	ø	ù	ú	û	ý	þ	þ	þ	þ



Códigos ASCII Extendidos

EJEMPLO:
ASCII extendido Cirílico
ISO 8859-5

	-0	-1	-2	-3	-4	-5	-6	-7	-8	-9	-A	-B	-C	-D	-E	-F
0-		0001	0002	0003	0004	0005	0006	0007	0008	0009	000A	000B	000C	000D	000E	000F
1-	0010	0011	0012	0013	0014	0015	0016	0017	0018	0019	001A	001B	001C	001D	001E	001F
2-	!	#	#	\$	%	&	'	()	*	,	-	.	/		
3-	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4-	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5-	P	Q	R	S	T	U	V	W	X	Y	Z	[]	\	^	-
6-	~	а	б	с	д	е	ф	г	х	и	ј	к	л	м	п	о
7-	р	q	г	с	т	у	в	w	х	у	з	{	}		~	
8-																
9-																
A-	Ё	Ђ	Ѓ	Є	Ѕ	І	Ї	Ј	Љ	Њ	Ћ	Ќ	-	Ў	Џ	
B-	Ӑ	Ӗ	Ҫ	Ӱ	Ӯ	Ӳ	ӱ	ӳ	Ӵ	ӵ	Ӷ	ӷ	Ӹ	ӹ	ӻ	Ӽ
C-	Ӑ	Ӗ	Ҫ	Ӱ	Ӯ	Ӳ	ӱ	ӳ	Ӵ	ӵ	Ӷ	ӷ	Ӹ	ӹ	Ӽ	ӽ
D-	Ӑ	Ӗ	Ҫ	Ӱ	Ӯ	Ӳ	ӱ	ӳ	Ӵ	ӵ	Ӷ	ӷ	Ӹ	ӹ	Ӽ	ӽ
E-	Ӑ	Ӗ	Ҫ	Ӱ	Ӯ	Ӳ	ӱ	ӳ	Ӵ	ӵ	Ӷ	ӷ	Ӹ	ӹ	Ӽ	ӽ
F-	Ӑ	Ӗ	Ҫ	Ӱ	Ӯ	Ӳ	ӱ	ӳ	Ӵ	ӵ	Ӷ	ӷ	Ӹ	ӹ	Ӽ	ӽ



Códigos Unicode

- Los códigos Unicode (“Universal Code”) fueron creados en 1991 para tener códigos alfanuméricos estándar, comunes en todas las regiones
 - Se utiliza el mismo código unicode para idiomas Chino, Árabe, etc.
- Como máximo necesitan 32 bits
 - Los primeros 7 bits permiten la compatibilidad con ASCII
 - Con 1 byte se puede representar el código US-ASCII
 - Con 2 bytes: caracteres latinos y alfabetos árabes, griego, cirílico, armenio, hebreo, sirio y thaana.
 - Con 3 bytes: resto de caracteres utilizados en todos los lenguajes
 - Con 4 bytes: caracteres gráficos y poco comunes
- Diferentes versiones de representación. Las más comunes:
 - **UTF-8:** Códigos de 1 byte, pero son de longitud variable (se pueden utilizar 4 grupos de 1 byte para representar un símbolo)
 - **UCS-2:** Códigos de 2 bytes de longitud fija
 - **UTF-16:** Códigos de 2 bytes, de longitud variable (se pueden utilizar 2 grupos de 2 bytes para representar un símbolo)
 - **UTF-32:** Códigos de 4 bytes

Códigos Unicode



EJEMPLO:

- Parte del Unicode correspondiente al alfabeto cirílico
 - Se necesita el segundo byte para la representación
 - Las codificaciones completas se pueden encontrar en:

<http://www.unicode.org/charts>



Códigos detectores y correctores de errores

- En los sistemas digitales pueden aparecer errores
 - Errores físicos de los circuitos
 - Interferencias electromagnéticas (EMI)
 - Fallos de alimentación eléctrica
 - Etc.
- Códigos detectores de error:
 - Pueden permitir detectar un error en la codificación
- Códigos correctores de error:
 - Permiten detectar un error y además corregirlo
- Los códigos detectores de error y los códigos correctores de error no utilizan las 2^n posibles codificaciones con n bits



Códigos detectores de error

- Códigos de paridad:

- Añaden un bit adicional (paridad del número) que permite detectar errores simples en la codificación (error en 1 bit)
- La paridad que se considera es la de la **suma** de los n bits de la codificación
 - **NOTA:** la paridad no tiene nada que ver con si la codificación binaria es par o impar (un número binario es par si acaba en 0 e impar si acaba en 1).
- Dos posibles convenios:
 - Añadir un 0 cuando la paridad sea par y 1 cuando sea impar: Se denomina código de paridad par (ya que considerando la suma de los n bits + el bit de paridad la paridad siempre es par)
 - Añadir un 1 cuando la paridad sea par y 0 cuando sea impar: Se denomina código de paridad impar (ya que la suma de los n bits + bit paridad es siempre impar)



Códigos detectores de error

- Ejemplo de paridad:

Código detector de errores (código de paridad impar) a partir del código binario natural de dos bits:

0 0	1
0 1	0
1 0	0
1 1	1

- Ejemplo de utilidad del código detector:

Si utilizamos este código en una comunicación entre dos sistemas binarios, el sistema receptor podría detectar si hay un error comprobando la paridad.

Ejemplo: Se transmite la codificación 001, y el receptor recibe la codificación 000 (error en el último bit).

Paridad de 001: impar
Paridad de 000: par



No coinciden:
Error detectado



Códigos detectores de error

- Existen más códigos detectores de error:
 - Número de unos:
 - Se añade a la codificación la suma de unos de la codificación (no sólo la paridad de la suma, sino la suma completa)
 - Número de transiciones:
 - Se añade a la codificación el número de transiciones de 0 a 1 y de 1 a 0 en la codificación
 - Códigos CRC (Cyclic Redundancy Checking):
 - Buscan añadir el menor número posible de bits que permitan detectar el mayor número posible de fallos
 - Estos códigos también permiten corregir algunos errores
- Los códigos más utilizados son los de **paridad** (por su sencillez) y **CRC** (por su eficacia)



Códigos correctores de error

- Los códigos correctores permiten no sólo detectar sino también pueden corregir un error.
- Para que un código permita corregir errores, la distancia mínima (número mínimo de bits diferentes entre dos codificaciones) debe ser mayor de 2.
 - Se puede corregir la codificación buscando la codificación más cercana perteneciente al código
- Hamming describió un método general para construir códigos con distancia mínima de 3, conocidos como **códigos de Hamming**
- Estos códigos son importantes, a partir de ellos se obtienen muchos de los utilizados en sistemas de comunicaciones (por ejemplo los códigos de bloque Reed-Solomon)



Codificación de números enteros y reales

- Además de los códigos binarios vistos hasta ahora, hay otros códigos importantes que se utilizan para representar números enteros y números reales:
 - Números enteros: Códigos de signo y magnitud, Complemento a Uno, Complemento a Dos
 - Números reales: Códigos de Punto fijo y Coma Flotante
- Estos códigos se estudiarán en detalle en el Tema 4: Aritmética Binaria



Referencias

- Fundamentos de Sistemas Digitales. Thomas L. Floyd. Pearson Prentice Hall
- Introducción al Diseño Lógico Digital. John P. Hayes. Addison-Wesley
- Diseño Digital. John F. Wakerly. Pearson Prentice Hall



Algebra de Boole y puertas lógicas

© Luis Entrena, Celia López,
Mario García, Enrique San Millán

Universidad Carlos III de Madrid



Índice

- Postulados y propiedades fundamentales del Álgebra de Boole
- Funciones y expresiones booleanas
- Puertas lógicas. Tecnologías digitales. Implementación de funciones lógicas
- Minimización de funciones lógicas



Álgebra de Boole

- Fundamentos matemáticos de los circuitos digitales
- Denominada Álgebra de Boole en honor de su inventor, George Boole
 - “*An Investigation of the Laws of Thought*” (1854)
- Un álgebra se define por un conjunto de elementos con unas operaciones. En nuestro caso:
 - $B = \{0, 1\}$
 - $\Phi = \{+, \bullet\}$



Postulados del Álgebra de Boole

- Ley de composición interna
 - $\forall a, b \in B \Rightarrow a + b \in B, a \bullet b \in B$
- Elementos neutros
 - $\forall a \in B \Rightarrow \exists$ elementos neutros (0 y 1 respectivamente)
 $a + 0 = a$
 $a \bullet 1 = a$
- Propiedad commutativa
 - $\forall a, b \in B \Rightarrow a + b = b + a$
 $a \bullet b = b \bullet a$
- Propiedad distributiva
 - $\forall a, b, c \in B \Rightarrow a + b \bullet c = (a + b) \bullet (a + c)$
 $a \bullet (b + c) = a \bullet b + a \bullet c$



Postulados del Álgebra de Boole

- Elemento inverso o complementario
 - $\forall a \in B \Rightarrow \exists \bar{a} \in B$

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

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



Propiedades fundamentales del Álgebra de Boole

- Dualidad: Toda ley válida tiene una dual, que se obtiene cambiando $0 \leftrightarrow 1$ y $+ \leftrightarrow \bullet$
- Idempotencia
 - $\forall a \in B \Rightarrow a + a = a$
 $a \bullet a = a$
 - Demostración:
$$a = a + 0 = a + a\bar{a} = (a + a)(a + \bar{a}) = (a + a) \bullet 1 = a + a$$
- $\forall a \in B \Rightarrow a + 1 = 1$
 $a \bullet 0 = 0$



Propiedades fundamentales del Álgebra de Boole

- De las propiedades anteriores se pueden definir las operaciones básicas

a	b	a+b
0	0	0
0	1	1
1	0	1
1	1	1

a	b	a•b
0	0	0
0	1	0
1	0	0
1	1	1

a	\bar{a}
0	1
1	0

- Tabla de verdad: proporciona el valor de una función para todas las posibles combinaciones de valores de las entradas



Propiedades fundamentales del Álgebra de Boole

- Involución
 - $\forall a \in B \Rightarrow \bar{\bar{a}} = a$
- Absorción
 - $\forall a, b \in B \Rightarrow a + ab = a$
 $a(a+b) = a$
 - Demostración:
$$a + ab = a \bullet 1 + ab = a(1 + b) = a \bullet 1 = a$$
- Propiedad asociativa
 - $\forall a, b, c \in B \Rightarrow (a + b) + c = a + (b + c)$
$$(a \bullet b) \bullet c = a \bullet (b \bullet c)$$



Propiedades fundamentales del Álgebra de Boole

- Leyes de De Morgan:

- $\forall a, b \in B \Rightarrow \overline{a + b} = \overline{a} \overline{b}$
 $\overline{a \bullet b} = \overline{a} + \overline{b}$

- Demostración:

$$(a + b) + \overline{a} \overline{b} = (a + b + \overline{a})(a + b + \overline{b}) = 1 \bullet 1$$
$$(a + b) \bullet \overline{a} \overline{b} = (a\overline{a}\overline{b}) + (\overline{b}\overline{a}\overline{b}) = 0 + 0$$

luego $(a+b)$ es el inverso de $\overline{a} \overline{b}$



Funciones y expresiones booleanas

● Definiciones:

- Una variable lógica o booleana es cualquier elemento $x \in B = \{0, 1\}$
- Un literal es una variable negada o sin negar
- Función lógica o booleana:

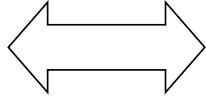
$$f : B^n \rightarrow B$$
$$(x_1, x_2, \dots, x_n) \rightarrow y$$



Representación de funciones lógicas

- Expresión
- Tabla de verdad

$$f(a, b) = a + b$$



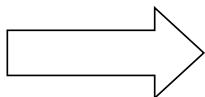
a	b	f(a,b)
0	0	0
0	1	1
1	0	1
1	1	1



Obtención de la tabla de verdad a partir de una expresión

- Basta evaluar la expresión para cada una de las combinaciones de valores de las entradas

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



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	1
1	1	0	1
1	1	1	1



Función mintérmino

- Expresión: un producto en el que aparecen todas las variables, negadas o no
- Tabla de verdad: tiene un 1 en una posición y 0 en todas las demás
- Ejemplo:

$$f(a,b,c) = \bar{a}b\bar{c} = m_2 \quad \longleftrightarrow$$

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

- Regla para obtener la expresión:

- 0 → variable negada
- 1 → variable sin negar



Función maxtérmico

- Expresión: una suma en la que aparecen todas las variables, negadas o no
- Tabla de verdad: tiene un 0 en una posición y 1 en todas las demás
- Ejemplo:

$$f(a,b,c) = (a + \bar{b} + c) = M_2 \quad \longleftrightarrow \quad \begin{array}{c|ccc} a & b & c & f \\ \hline 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{array}$$

- Regla para obtener la expresión:

- 0 → variable sin negar
- 1 → variable negada

CUIDADO: al contrario que los mintérminos!



Teorema de Expansión de Shannon

- Toda función booleana se puede descomponer de las siguientes formas

$$f(x_1, x_2, \dots, x_n) = \bar{x}_i f(x_1, \dots, x_{i-1}, 0, x_{i+1}, \dots, x_n) + x_i f(x_1, \dots, x_{i-1}, 1, x_{i+1}, \dots, x_n)$$

$$f(x_1, x_2, \dots, x_n) = [\bar{x}_i + f(x_1, \dots, x_{i-1}, 1, x_{i+1}, \dots, x_n)][x_i + f(x_1, \dots, x_{i-1}, 0, x_{i+1}, \dots, x_n)]$$

- Demostración

$$\begin{aligned} x_i = 0 \Rightarrow f(x_1, x_2, \dots, x_n) &= 1 \bullet f(x_1, \dots, 0, \dots, x_n) + 0 \bullet f(x_1, \dots, 1, \dots, x_n) = \\ &= f(x_1, \dots, 0, \dots, x_n) \end{aligned}$$

$$\begin{aligned} x_i = 1 \Rightarrow f(x_1, x_2, \dots, x_n) &= 0 \bullet f(x_1, \dots, 0, \dots, x_n) + 1 \bullet f(x_1, \dots, 1, \dots, x_n) = \\ &= f(x_1, \dots, 1, \dots, x_n) \end{aligned}$$

- La otra forma se demuestra por dualidad



Corolario del Teorema de Expansión de Shannon

- Aplicando recursivamente el Teorema:

$$\begin{aligned}f(a,b,c) &= \bar{a}f(0,b,c) + af(1,b,c) = \\&= \bar{a}(\bar{b}f(0,0,c) + bf(0,1,c)) + a(\bar{b}f(1,0,c) + bf(0,1,c)) = \\&= \bar{a}\bar{b}f(0,0,c) + \bar{a}bf(0,1,c) + a\bar{b}f(1,0,c) + abf(0,1,c) = \\&= \bar{a}\bar{b}\bar{c}f(0,0,0) + \bar{a}\bar{b}cf(0,0,1) + \bar{a}\bar{b}c\bar{f}(0,1,0) + \bar{a}\bar{b}c\bar{f}(0,1,1) + \\&\quad + a\bar{b}\bar{c}f(1,0,0) + a\bar{b}cf(1,0,1) + a\bar{b}c\bar{f}(1,1,0) + abc\bar{f}(1,1,1) = \\&= \sum_{i=1}^3 m_i k_i\end{aligned}$$

- Una función es igual a la suma de todos los mintérminos (m_i) afectados por un coeficiente (k_i) igual al valor que toma la función al sustituir cada variable por un 0 o un 1 según que en el mintérmino aparezca la variable negada o sin negar, respectivamente



Primera forma canónica

- Una función se puede expresar como la suma de los mintérminos para los que la función vale 1

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

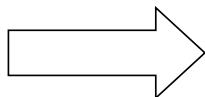
$$f(a,b,c) = \sum_{3} (0,2,5) = \sum_{3} m(0,2,5) = \\ = \bar{a}\bar{b}\bar{c} + \bar{a}b\bar{c} + a\bar{b}c$$



Segunda forma canónica

- Una función se puede expresar como el producto de los maxtérminos para los que la función vale 0

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



$$\begin{aligned}f(a,b,c) &= \prod_{3} (1,3,4,6,7) = \prod_{3} M(1,3,4,6,7) = \\&= (a + b + \bar{c})(a + \bar{b} + \bar{c})(\bar{a} + b + c) \\&\quad (\bar{a} + \bar{b} + c)(\bar{a} + \bar{b} + \bar{c})\end{aligned}$$

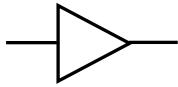
CUIDADO: al contrario que los mintérminos!



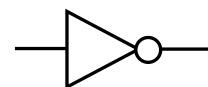
Puertas lógicas

- Las puertas lógicas son circuitos electrónicos que realizan las funciones básicas del Álgebra de Boole
- Para cada puerta utilizaremos un símbolo
- Identidad
$$z = a$$
- Puerta NOT o inversor
$$z = \bar{a}$$

a	a
0	0
1	1



a	\bar{a}
0	1
1	0



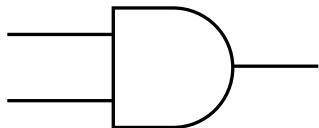


Puertas AND y OR

- Puerta AND

$$z = a \bullet b$$

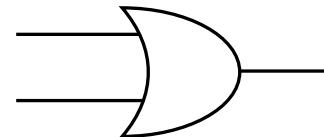
a	b	$a \bullet b$
0	0	0
0	1	0
1	0	0
1	1	1



- Puerta OR

$$z = a + b$$

a	b	$a+b$
0	0	0
0	1	1
1	0	1
1	1	1



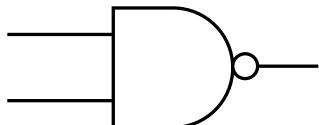


Puertas NAND y NOR

- Puerta NAND

$$z = \overline{a \bullet b} = \overline{a} + \overline{b}$$

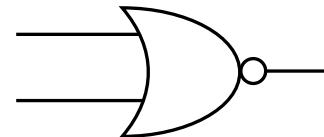
a	b	$\overline{a \bullet b}$
0	0	1
0	1	1
1	0	1
1	1	0



- Puerta NOR

$$z = \overline{a + b} = \overline{a} \overline{b}$$

a	b	$\overline{a + b}$
0	0	1
0	1	0
1	0	0
1	1	0





Puertas XOR y XNOR

- Puerta XOR (OR-Exclusiva)

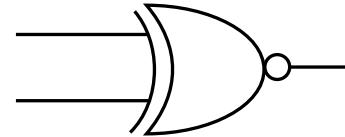
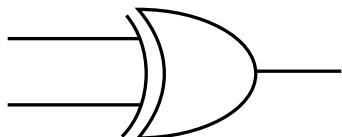
$$z = a \oplus b = \bar{a}b + a\bar{b} = (\bar{a} + \bar{b})(a + b)$$

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

- Puerta XNOR (NOR-Exclusiva)

$$z = \overline{a \oplus b} = ab + \bar{a}\bar{b} = (\bar{a} + b)(a + \bar{b})$$

a	b	$\overline{a \oplus b}$
0	0	1
0	1	0
1	0	0
1	1	1





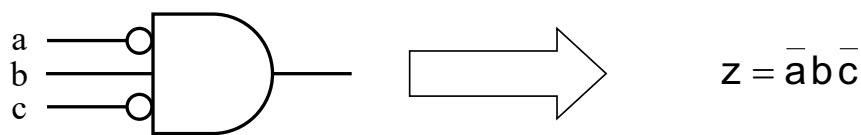
Generalización a n entradas

	Valor de la salida	
Puerta	0	1
AND	Alguna entrada = 0	Todas las entradas = 1
OR	Todas las entradas = 0	Alguna entrada = 1
NAND	Todas las entradas = 1	Alguna entrada = 0
NOR	Alguna entrada = 1	Todas las entradas = 0
XOR	Hay un nº par de entradas = 1	Hay un nº impar de entradas = 1
XNOR	Hay un nº impar de entradas = 1	Hay un nº par de entradas = 1



Otros símbolos

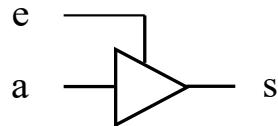
- Un círculo en una entrada o una salida indica negación





Buffer triestado

- Un tipo especial de puerta lógica que puede poner su salida en alta impedancia

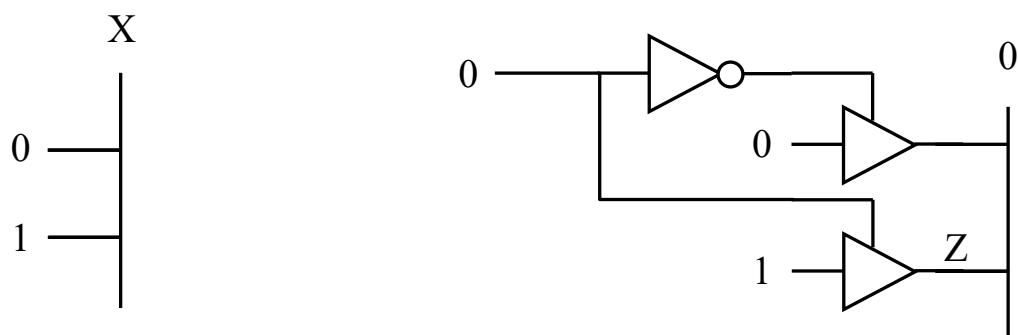


e	a	s
0	0	Z
0	1	Z
1	0	0
1	1	1



Buffer triestado

- Los buffers triestado son útiles para permitir múltiples conexiones a un mismo punto evitando cortocircuitos



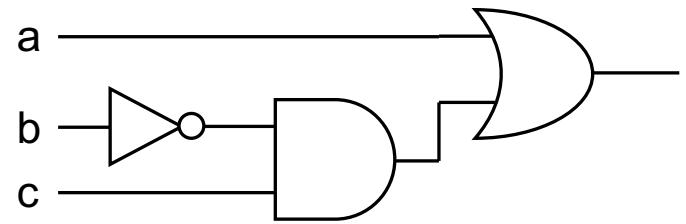
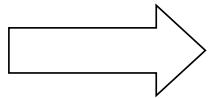
Cortocircuito!



Realización de una función lógica con puertas lógicas

- A partir de la expresión de la función, sustituimos las operaciones lógicas por puertas lógicas
- Ejemplo:

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





Conjuntos completos

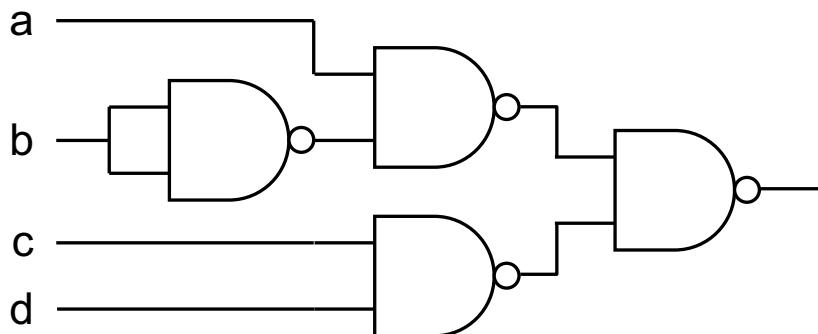
- Un conjunto de funciones es funcionalmente completo si cualquier función lógica puede realizarse con las funciones del conjunto solamente
 - {AND} **no** es un conjunto completo
 - {AND, NOT} es un conjunto completo
 - {OR, NOT} es un conjunto completo
 - {NAND} es un conjunto completo
 - {NOR} es un conjunto completo
- Los conjuntos {NAND} y {NOR} tienen la ventaja de que permiten realizar cualquier función lógica con un sólo tipo de puerta lógica



Realización de circuitos con puertas NAND

- Aplicación directa de las leyes de De Morgan
- Ejemplo: $f(a,b,c) = a\bar{b} + c\bar{d} =$

$$= \overline{\overline{a}\bar{b} + c\bar{d}} = \overline{\overline{a}\bar{b}} \bullet \overline{c\bar{d}}$$

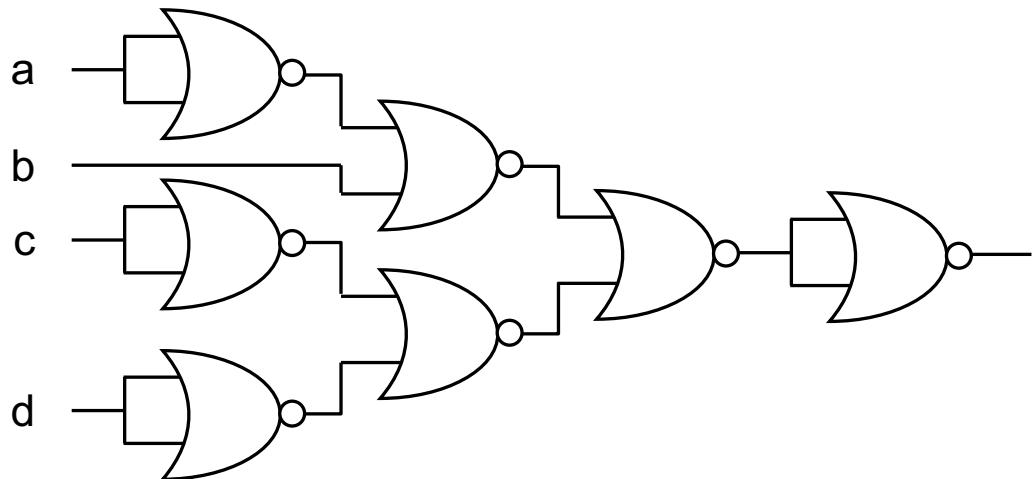




Realización de circuitos con puertas NOR

- Aplicación directa de las leyes de De Morgan
- Ejemplo: $f(a,b,c) = a\bar{b} + cd =$

$$= \overline{\overline{ab}} + \overline{\overline{cd}} = \overline{\overline{a} + b} + \overline{\overline{c} + d}$$





Minimización de funciones lógicas

- Una función lógica tiene múltiples expresiones equivalentes
 - La forma más sencilla dará lugar a una implementación mejor
- Criterios de optimización:
 - En tamaño o área:
 - Menor número de puertas lógicas
 - Puertas lógicas con el menor número de entradas
 - En velocidad o retardo:
 - Menor número de puertas lógicas desde una entrada hasta la salida
- Nos centraremos en la optimización en área



Minimización de funciones lógicas

• Métodos de optimización

- Manual: aplicación directa de las leyes del Álgebra de Boole
 - Muy difícil, no sistemático
- En dos niveles: el objetivo es obtener una expresión óptima en forma de suma de productos o productos de sumas
 - Existen soluciones sistemáticas y óptimas
 - Aplicable manualmente (para pocas variables) o con ayuda de un computador
- Multinivel
 - Mejor solución, aunque mucho más difícil
 - Sólo posible con ayuda de un computador



Métodos de los mapas de Karnaugh

- Método de optimización en dos niveles
- Se puede realizar manualmente hasta 6 variables
- Se basa en la Propiedad de adyacencia
 - $\forall E, x \in B \Rightarrow E x + E \bar{x} = E(x + \bar{x}) = E$
 $(E + x)(E + \bar{x}) = E + (x \bullet \bar{x}) = E \quad (\text{dual})$
 - Dos términos son adyacentes si son idénticos excepto por un literal, que aparece negado en un término y no negado en el otro
 - Los dos términos se simplifican en uno sólo con eliminación del literal que los diferencia



Aplicación de la propiedad de adyacencia

- Ejemplo:

$$\begin{aligned}f(a,b,c) &= \sum_3(0,1,2,3,7) = \overbrace{\bar{a}\bar{b}\bar{c}} + \overbrace{\bar{a}\bar{b}c} + \overbrace{\bar{a}b\bar{c}} + \overbrace{\bar{a}bc} + abc = \\&= \overbrace{\bar{a}\bar{b}} + \overbrace{\bar{a}b} + \overbrace{bc} \\&= \overbrace{\bar{a}} + \overbrace{bc}\end{aligned}$$

- La observación de las adyacencias puede ser difícil en la práctica



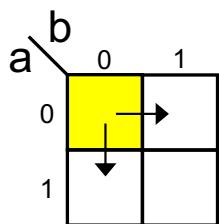
Mapas de Karnaugh

- Mapa que presenta la tabla de verdad de una función de manera que los términos adyacentes son contiguos:
 - Una casilla para cada combinación o término
 - Las casillas se numeran en código Gray
 - En un mapa de n variables, cada casilla tiene n casillas adyacentes que se corresponden con las combinaciones que resultan de invertir el valor de cada una de las n variables

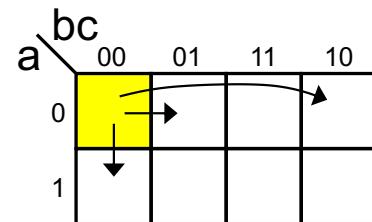
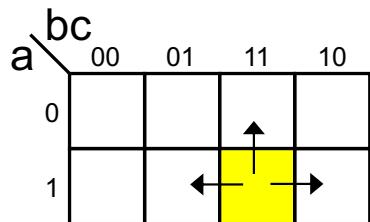


Mapas de Karnaugh: adyacencias

- Dos variables



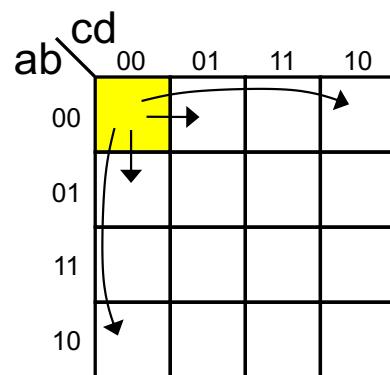
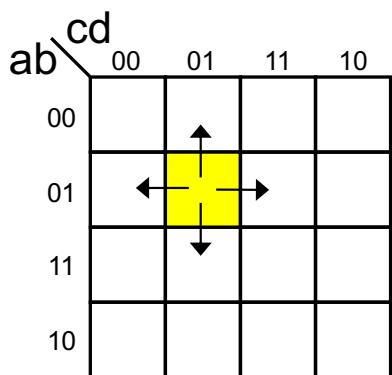
- Tres variables





Mapas de Karnaugh: adyacencias

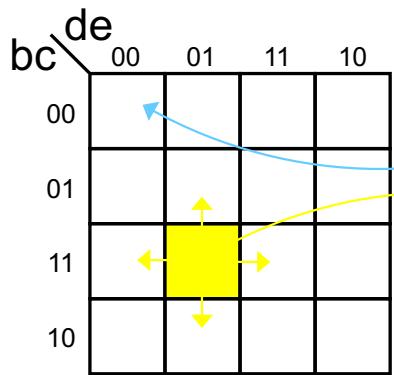
- Cuatro variables



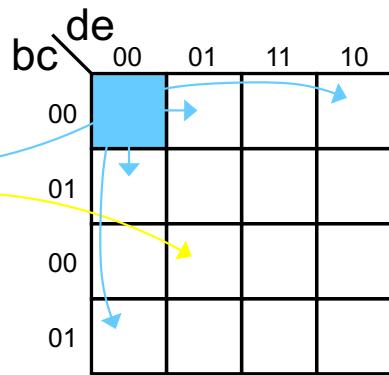


Mapas de Karnaugh: adyacencias

- Cinco variables



$a = 0$



$a = 1$



Mapas de Karnaugh: numeración de las casillas

- Dos variables

a\b	0	1
0	0	1
1	2	3

- Cuatro variables

ab\cd	00	01	11	10
00	0	1	3	2
01	4	5	7	6
11	12	13	15	14
10	8	9	11	10

- Tres variables

a\bc	00	01	11	10
0	0	1	3	2
1	4	5	7	6



Mapas de Karnaugh: numeración de las casillas

- Cinco variables

bc \ de	00	01	11	10
00	0	1	3	2
01	4	5	7	6
11	12	13	15	14
10	8	9	11	10

$a = 0$

bc \ de	00	01	11	10
00	16	17	19	18
01	20	21	23	22
11	28	29	31	30
10	24	25	27	26

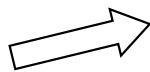
$a = 1$



Representación de una función en el Mapa de Karnaugh

- Se marcan las casillas que corresponden a los mintérminos o los maxtérminos de la función
- Ejemplo:

$$\begin{aligned} f(a,b,c) &= \sum_3 (0,1,2,3,7) = \\ &= \prod_3 (4,5,6) \end{aligned}$$



		bc	00	01	11	10
a	c	0	1	1	1	1
		1			1	



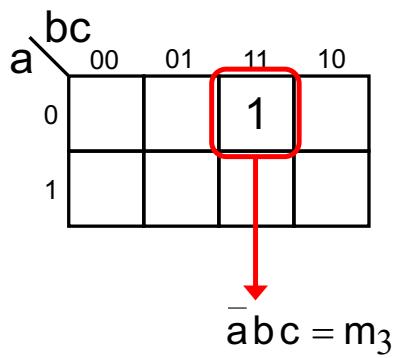
		bc	00	01	11	10
a	c	0				
		1	0	0		0



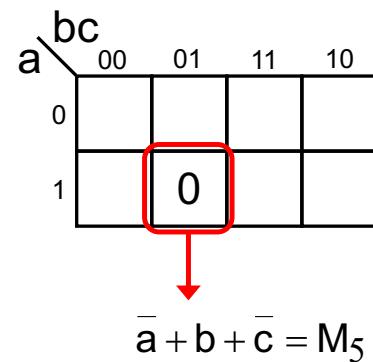
Obtención de una expresión a partir del Mapa de Karnaugh

- Se siguen las reglas para mintérminos y maxtérminos

- Regla para mintérminos
 - 0 → variable negada
 - 1 → variable sin negar



- Regla para maxtérminos
 - 0 → variable sin negar
 - 1 → variable negada



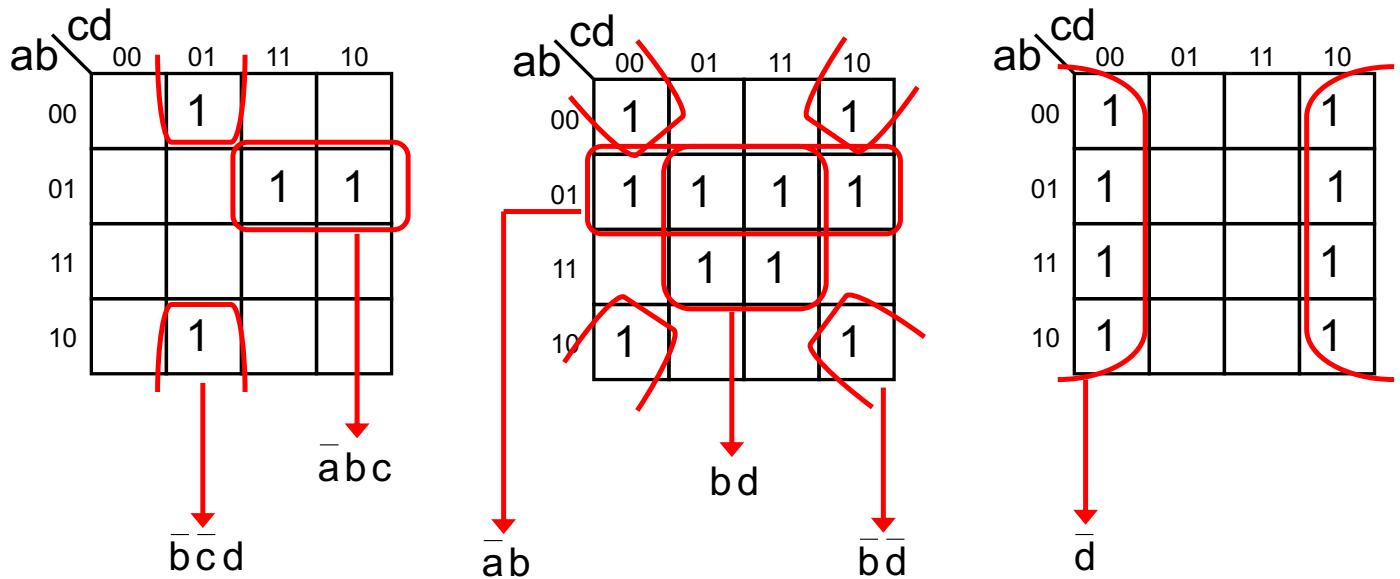


Simplificación mediante Mapas de Karnaugh

- Dos opciones
 - Por mintérminos (unos): se obtiene una suma de productos
 - Por maxtérminos (ceros): se obtiene un producto de sumas
- Buscar grupos de casillas adyacentes
 - Un grupo de 2 casillas adyacentes elimina 1 variable
 - Un grupo de 4 casillas adyacentes elimina 2 variables
 - Un grupo de 8 casillas adyacentes elimina 3 variables
 - Un grupo de 16 casillas adyacentes elimina 4 variables
 -
- Objetivo: cubrir todos los mintérminos (maxtérminos) con los grupos más grandes posibles y con el menor número de grupos
 - Se pueden repetir términos, si es necesario (propiedad de absorción)



Simplificación: formación de grupos





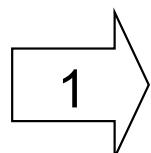
Simplificación mediante Mapas de Karnaugh: Algoritmo

- Algoritmo sistemático
 1. Cubrir las casillas que no pueden formar grupos de 2
 2. Cubrir las casillas que pueden formar grupos de 2, pero no de 4
 3. Cubrir las casillas que pueden formar grupos de 4, pero no de 8
 4. Cubrir las casillas que pueden formar grupos de 8, pero no de 16
 5. ...
- Si en algún paso hay más de una opción:
 - Comenzar siempre cubriendo las casillas que tienen menos opciones



Simplificación mediante Mapas de Karnaugh: Ejemplo

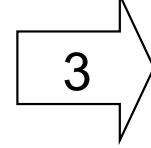
ab \ cd	00	01	11	10
00		1		
01			1	1
11			1	1
10			1	



ab \ cd	00	01	11	10
00		1		
01			1	1
11			1	1
10			1	



ab \ cd	00	01	11	10
00		1		
01			1	1
11			1	1
10			1	



ab \ cd	00	01	11	10
00		1		
01			1	1
11			1	1
10			1	



Funciones incompletas

- Una función incompletamente especificada (o simplemente incompleta) es aquella que no está especificada para alguna combinación de valores de sus entradas
- Las funciones incompletas se dan en la práctica:
 - Cuando las entradas provienen de otro circuito que no puede producir determinadas combinaciones por construcción
 - Cuando existen casos en que el valor de la función no tiene sentido o es indiferente
- Notación:
 - Un valor indiferente se representa con 'X' ó '-'
 - El conjunto de términos indiferentes ("don't cares") se denota con la letra Δ



Funciones incompletas

- Ejemplo: Función que determina si un número BCD es impar
 - Los números del 10 al 15 no tienen sentido en BCD

$$\begin{aligned}f(b_3, b_2, b_1, b_0) &= \sum_4 (1, 3, 5, 7, 9) + \Delta(10, 11, 12, 13, 14, 15) = \\&= \prod_4 (0, 2, 4, 6, 8) + \Delta(10, 11, 12, 13, 14, 15)\end{aligned}$$

Combinaciones indiferentes

b3	b2	b1	b0	f
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	1
1	0	1	0	X
1	0	1	1	X
1	1	0	0	X
1	1	0	1	X
1	1	1	0	X
1	1	1	1	X



Minimización de funciones incompletas

- Los términos indiferentes son “comodines”: se pueden cubrir o no, según convenga para formar grupos más grandes

		$b_1 b_0$	$b_3 b_2$			
		00	01	11	10	
		00	1	1		
		01	1	1		
		11	X	X	X	X
		10	1	X	X	

$$f(b_3, b_2, b_1, b_0) = \overline{b_3} b_0 + \overline{b_2} \overline{b_1} b_0$$

		$b_1 b_0$	$b_3 b_2$			
		00	01	11	10	
		00	1	1		
		01	1	1		
		11	X	X	X	X
		10	1	X	X	

$$f(b_3, b_2, b_1, b_0) = b_0$$

Correcto



Funciones múltiples

- En los circuitos digitales se implementan generalmente funciones múltiples: varias funciones a la vez o una función de múltiples salidas
- Las funciones múltiples se pueden implementar de forma óptima al considerarlas conjuntamente
 - Se pueden compartir términos o partes comunes para ahorrar lógica
- La descomposición de funciones múltiples de manera que se maximicen los términos comunes es difícil
 - Los algoritmos son difíciles de aplicar manualmente
 - Generalmente lo haremos por inspección

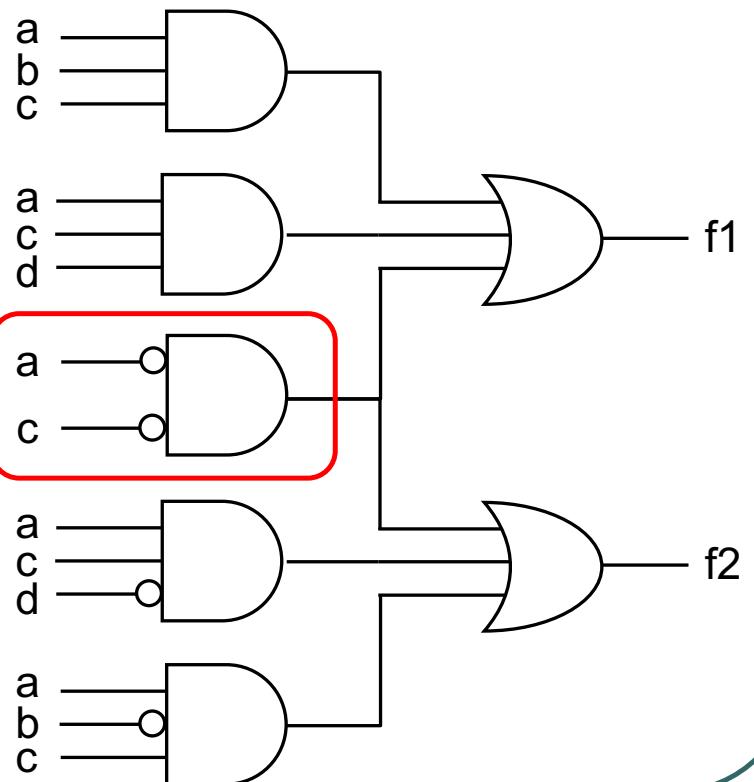


Funciones múltiples: Ejemplo

$$f_1(a,b,c,d) = \overline{a}\overline{c} + abc + acd$$

$$f_2(a,b,c,d) = \overline{a}\overline{c} + \overline{a}\overline{b}c + ac\overline{d}$$

Términos comunes





Funciones múltiples: Ejemplo

- Es posible encontrar más términos comunes

$$f_1(a,b,c,d) = \overline{a}\overline{c} + abc + acd = \overline{a}\overline{c} + \overline{a}\overline{c}\overline{d} + acd$$

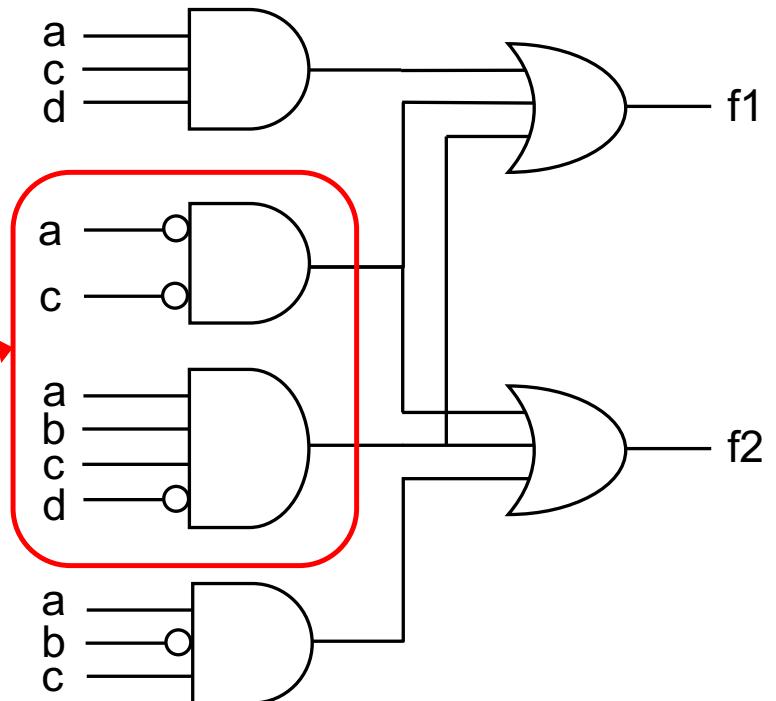
$$f_2(a,b,c,d) = \overline{a}\overline{c} + a\overline{b}c + a\overline{c}\overline{d} = \overline{a}\overline{c} + \overline{a}\overline{c}\overline{d} + a\overline{b}c$$

- Las expresiones de las funciones no son óptimas por separado, pero sí son óptimas en conjunto!
- Las herramientas de diseño incluyen algoritmos para minimizar funciones múltiples



Funciones múltiples: Ejemplo

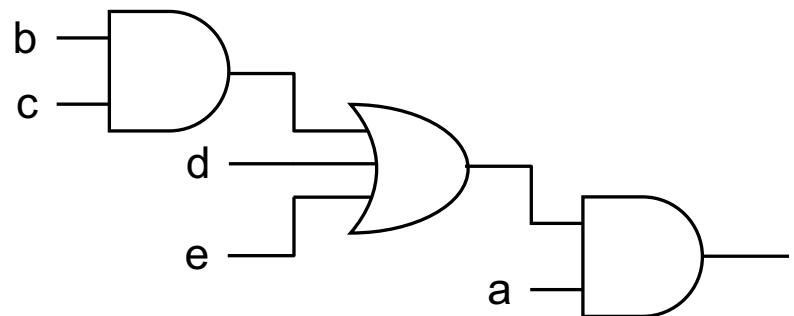
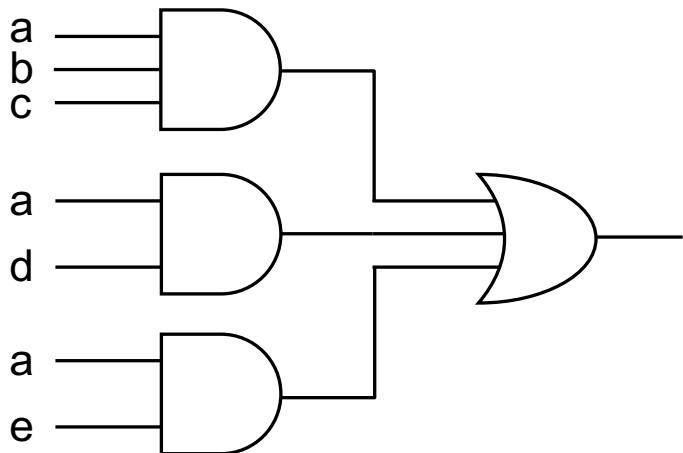
Términos comunes





Síntesis multinivel

- Si eliminamos la restricción a dos niveles, se pueden encontrar mejores soluciones
 - Se utilizan algoritmos heurísticos, con ayuda de un ordenador
- Ejemplo: $f(a,b,c,d,e) = abc + ad + ae = a(bc + d + e)$



Multinivel



Herramientas de optimización

- Métodos manuales:
 - Sólo en 2 niveles, pocas variables
- Herramientas software
 - Multinivel, múltiples funciones, muchas variables
 - Optimización en área o en retardo
 - Generalmente incorporadas en herramientas de síntesis lógica
- Herramientas de síntesis lógica
 - Funcionan como un compilador, a partir de la descripción del diseño en forma esquemática o mediante un Lenguaje de Descripción de Hardware
 - Optimizan el diseño y generan las puertas lógicas en una tecnología determinada



Referencias

- “Introducción al diseño lógico digital”. J. P. Hayes. Ed. Addison-Wesley
- “Circuitos y sistemas digitales”. J. E. García Sánchez, D. G. Tomás, M. Martínez Iniesta. Ed. Tebar-Flores



Algebra de Boole y puertas lógicas

© Luis Entrena, Celia López,
Mario García, Enrique San Millán

Universidad Carlos III de Madrid



Índice

- Postulados y propiedades fundamentales del Álgebra de Boole
- Funciones y expresiones booleanas
- Puertas lógicas. Tecnologías digitales. Implementación de funciones lógicas
- Minimización de funciones lógicas



Álgebra de Boole

- Fundamentos matemáticos de los circuitos digitales
- Denominada Álgebra de Boole en honor de su inventor, George Boole
 - “*An Investigation of the Laws of Thought*” (1854)
- Un álgebra se define por un conjunto de elementos con unas operaciones. En nuestro caso:
 - $B = \{0, 1\}$
 - $\Phi = \{+, \bullet\}$



Postulados del Álgebra de Boole

- Ley de composición interna
 - $\forall a, b \in B \Rightarrow a + b \in B, a \bullet b \in B$
- Elementos neutros
 - $\forall a \in B \Rightarrow \exists$ elementos neutros (0 y 1 respectivamente)
 $a + 0 = a$
 $a \bullet 1 = a$
- Propiedad commutativa
 - $\forall a, b \in B \Rightarrow a + b = b + a$
 $a \bullet b = b \bullet a$
- Propiedad distributiva
 - $\forall a, b, c \in B \Rightarrow a + b \bullet c = (a + b) \bullet (a + c)$
 $a \bullet (b + c) = a \bullet b + a \bullet c$



Postulados del Álgebra de Boole

- Elemento inverso o complementario
 - $\forall a \in B \Rightarrow \exists \bar{a} \in B$

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

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



Propiedades fundamentales del Álgebra de Boole

- Dualidad: Toda ley válida tiene una dual, que se obtiene cambiando $0 \leftrightarrow 1$ y $+ \leftrightarrow \bullet$
- Idempotencia
 - $\forall a \in B \Rightarrow a + a = a$
 $a \bullet a = a$
 - Demostración:
$$a = a + 0 = a + a\bar{a} = (a + a)(a + \bar{a}) = (a + a) \bullet 1 = a + a$$
- $\forall a \in B \Rightarrow a + 1 = 1$
 $a \bullet 0 = 0$



Propiedades fundamentales del Álgebra de Boole

- De las propiedades anteriores se pueden definir las operaciones básicas

a	b	a+b
0	0	0
0	1	1
1	0	1
1	1	1

a	b	a•b
0	0	0
0	1	0
1	0	0
1	1	1

a	\bar{a}
0	1
1	0

- Tabla de verdad: proporciona el valor de una función para todas las posibles combinaciones de valores de las entradas



Propiedades fundamentales del Álgebra de Boole

- Involución
 - $\forall a \in B \Rightarrow \bar{\bar{a}} = a$
- Absorción
 - $\forall a, b \in B \Rightarrow a + ab = a$
 $a(a+b) = a$
 - Demostración:
$$a + ab = a \bullet 1 + ab = a(1 + b) = a \bullet 1 = a$$
- Propiedad asociativa
 - $\forall a, b, c \in B \Rightarrow (a + b) + c = a + (b + c)$
$$(a \bullet b) \bullet c = a \bullet (b \bullet c)$$



Propiedades fundamentales del Álgebra de Boole

- Leyes de De Morgan:

- $\forall a, b \in B \Rightarrow \overline{a + b} = \overline{a} \overline{b}$
 $\overline{a \bullet b} = \overline{a} + \overline{b}$

- Demostración:

$$(a + b) + \overline{a} \overline{b} = (a + b + \overline{a})(a + b + \overline{b}) = 1 \bullet 1$$
$$(a + b) \bullet \overline{a} \overline{b} = (a\overline{a}\overline{b}) + (\overline{b}\overline{a}\overline{b}) = 0 + 0$$

luego $(a+b)$ es el inverso de $\overline{a} \overline{b}$



Funciones y expresiones booleanas

● Definiciones:

- Una variable lógica o booleana es cualquier elemento $x \in B = \{0, 1\}$
- Un literal es una variable negada o sin negar
- Función lógica o booleana:

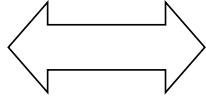
$$f : B^n \rightarrow B$$
$$(x_1, x_2, \dots, x_n) \rightarrow y$$



Representación de funciones lógicas

- Expresión
- Tabla de verdad

$$f(a, b) = a + b$$



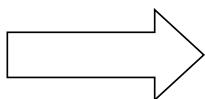
a	b	f(a,b)
0	0	0
0	1	1
1	0	1
1	1	1



Obtención de la tabla de verdad a partir de una expresión

- Basta evaluar la expresión para cada una de las combinaciones de valores de las entradas

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



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	1
1	1	0	1
1	1	1	1



Función mintérmino

- Expresión: un producto en el que aparecen todas las variables, negadas o no
- Tabla de verdad: tiene un 1 en una posición y 0 en todas las demás
- Ejemplo:

$$f(a,b,c) = \bar{a}b\bar{c} = m_2 \quad \longleftrightarrow$$

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

- Regla para obtener la expresión:

- 0 → variable negada
- 1 → variable sin negar



Función maxtérmico

- Expresión: una suma en la que aparecen todas las variables, negadas o no
- Tabla de verdad: tiene un 0 en una posición y 1 en todas las demás
- Ejemplo:

$$f(a,b,c) = (a + \bar{b} + c) = M_2 \quad \longleftrightarrow$$

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

- Regla para obtener la expresión:

- 0 → variable sin negar
- 1 → variable negada

CUIDADO: al contrario que los mintérminos!



Teorema de Expansión de Shannon

- Toda función booleana se puede descomponer de las siguientes formas

$$f(x_1, x_2, \dots, x_n) = \bar{x}_i f(x_1, \dots, x_{i-1}, 0, x_{i+1}, \dots, x_n) + x_i f(x_1, \dots, x_{i-1}, 1, x_{i+1}, \dots, x_n)$$

$$f(x_1, x_2, \dots, x_n) = [\bar{x}_i + f(x_1, \dots, x_{i-1}, 1, x_{i+1}, \dots, x_n)][x_i + f(x_1, \dots, x_{i-1}, 0, x_{i+1}, \dots, x_n)]$$

- Demostración

$$\begin{aligned} x_i = 0 \Rightarrow f(x_1, x_2, \dots, x_n) &= 1 \bullet f(x_1, \dots, 0, \dots, x_n) + 0 \bullet f(x_1, \dots, 1, \dots, x_n) = \\ &= f(x_1, \dots, 0, \dots, x_n) \end{aligned}$$

$$\begin{aligned} x_i = 1 \Rightarrow f(x_1, x_2, \dots, x_n) &= 0 \bullet f(x_1, \dots, 0, \dots, x_n) + 1 \bullet f(x_1, \dots, 1, \dots, x_n) = \\ &= f(x_1, \dots, 1, \dots, x_n) \end{aligned}$$

- La otra forma se demuestra por dualidad



Corolario del Teorema de Expansión de Shannon

- Aplicando recursivamente el Teorema:

$$\begin{aligned}f(a,b,c) &= \bar{a}f(0,b,c) + af(1,b,c) = \\&= \bar{a}(\bar{b}f(0,0,c) + bf(0,1,c)) + a(\bar{b}f(1,0,c) + bf(0,1,c)) = \\&= \bar{a}\bar{b}f(0,0,c) + \bar{a}bf(0,1,c) + a\bar{b}f(1,0,c) + abf(0,1,c) = \\&= \bar{a}\bar{b}\bar{c}f(0,0,0) + \bar{a}\bar{b}cf(0,0,1) + \bar{a}\bar{b}c\bar{f}(0,1,0) + \bar{a}\bar{b}c\bar{f}(0,1,1) + \\&\quad + a\bar{b}\bar{c}f(1,0,0) + a\bar{b}cf(1,0,1) + a\bar{b}c\bar{f}(1,1,0) + abc\bar{f}(1,1,1) = \\&= \sum_{i=1}^3 m_i k_i\end{aligned}$$

- Una función es igual a la suma de todos los mintérminos (m_i) afectados por un coeficiente (k_i) igual al valor que toma la función al sustituir cada variable por un 0 o un 1 según que en el mintérmino aparezca la variable negada o sin negar, respectivamente



Primera forma canónica

- Una función se puede expresar como la suma de los mintérminos para los que la función vale 1

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

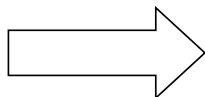
$$f(a,b,c) = \sum_{3} (0,2,5) = \sum_{3} m(0,2,5) = \\ = \bar{a}\bar{b}\bar{c} + \bar{a}b\bar{c} + a\bar{b}c$$



Segunda forma canónica

- Una función se puede expresar como el producto de los maxtérminos para los que la función vale 0

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



$$\begin{aligned}f(a,b,c) &= \prod_{3} (1,3,4,6,7) = \prod_{3} M(1,3,4,6,7) = \\&= (a + b + \bar{c})(a + \bar{b} + \bar{c})(\bar{a} + b + c) \\&\quad (\bar{a} + \bar{b} + c)(\bar{a} + \bar{b} + \bar{c})\end{aligned}$$

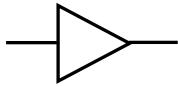
CUIDADO: al contrario que los mintérminos!



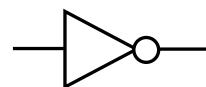
Puertas lógicas

- Las puertas lógicas son circuitos electrónicos que realizan las funciones básicas del Álgebra de Boole
- Para cada puerta utilizaremos un símbolo
- Identidad
$$z = a$$
- Puerta NOT o inversor
$$z = \bar{a}$$

a	a
0	0
1	1



a	\bar{a}
0	1
1	0



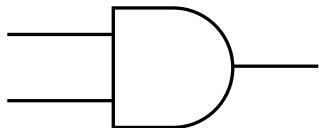


Puertas AND y OR

- Puerta AND

$$z = a \bullet b$$

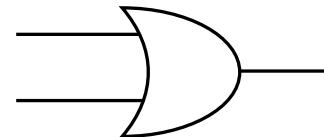
a	b	$a \bullet b$
0	0	0
0	1	0
1	0	0
1	1	1



- Puerta OR

$$z = a + b$$

a	b	$a+b$
0	0	0
0	1	1
1	0	1
1	1	1



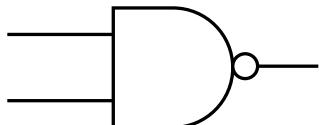


Puertas NAND y NOR

- Puerta NAND

$$z = \overline{a \bullet b} = \overline{a} + \overline{b}$$

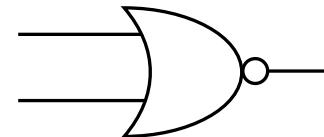
a	b	$\overline{a \bullet b}$
0	0	1
0	1	1
1	0	1
1	1	0



- Puerta NOR

$$z = \overline{a + b} = \overline{a} \overline{b}$$

a	b	$\overline{a + b}$
0	0	1
0	1	0
1	0	0
1	1	0





Puertas XOR y XNOR

- Puerta XOR (OR-Exclusiva)

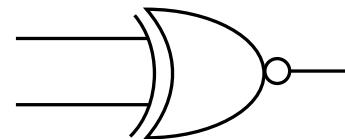
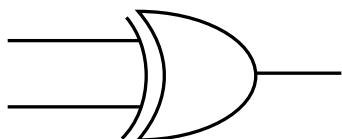
$$z = a \oplus b = \bar{a}b + a\bar{b} = (\bar{a} + \bar{b})(a + b)$$

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

- Puerta XNOR (NOR-Exclusiva)

$$z = \overline{a \oplus b} = ab + \bar{a}\bar{b} = (\bar{a} + b)(a + \bar{b})$$

a	b	$\overline{a \oplus b}$
0	0	1
0	1	0
1	0	0
1	1	1





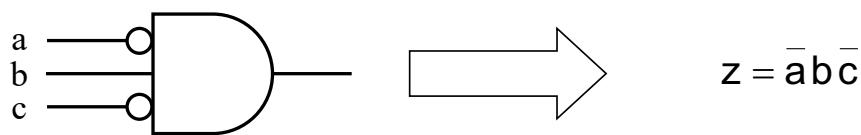
Generalización a n entradas

	Valor de la salida	
Puerta	0	1
AND	Alguna entrada = 0	Todas las entradas = 1
OR	Todas las entradas = 0	Alguna entrada = 1
NAND	Todas las entradas = 1	Alguna entrada = 0
NOR	Alguna entrada = 1	Todas las entradas = 0
XOR	Hay un nº par de entradas = 1	Hay un nº impar de entradas = 1
XNOR	Hay un nº impar de entradas = 1	Hay un nº par de entradas = 1



Otros símbolos

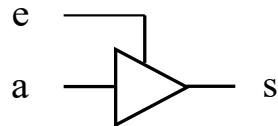
- Un círculo en una entrada o una salida indica negación





Buffer triestado

- Un tipo especial de puerta lógica que puede poner su salida en alta impedancia

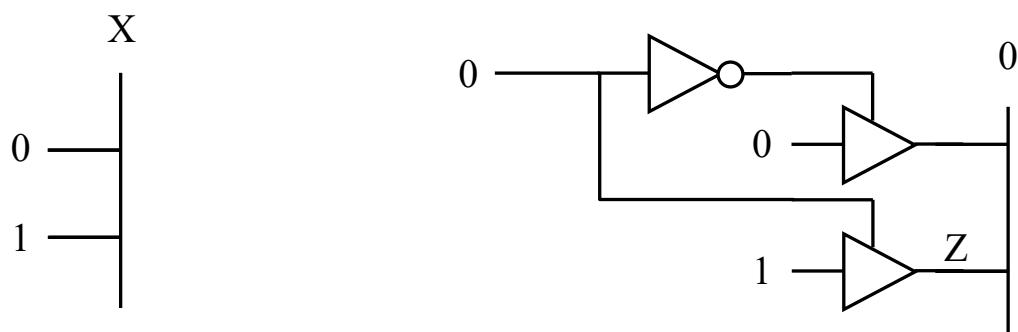


e	a	s
0	0	Z
0	1	Z
1	0	0
1	1	1



Buffer triestado

- Los buffers triestados son útiles para permitir múltiples conexiones a un mismo punto evitando cortocircuitos



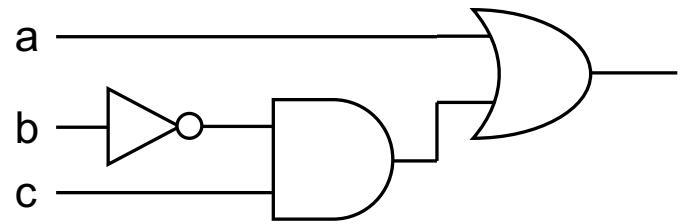
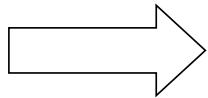
Cortocircuito!



Realización de una función lógica con puertas lógicas

- A partir de la expresión de la función, sustituimos las operaciones lógicas por puertas lógicas
- Ejemplo:

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





Conjuntos completos

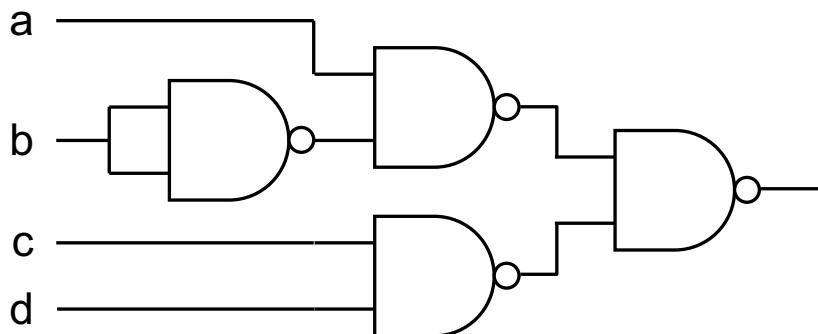
- Un conjunto de funciones es funcionalmente completo si cualquier función lógica puede realizarse con las funciones del conjunto solamente
 - {AND} **no** es un conjunto completo
 - {AND, NOT} es un conjunto completo
 - {OR, NOT} es un conjunto completo
 - {NAND} es un conjunto completo
 - {NOR} es un conjunto completo
- Los conjuntos {NAND} y {NOR} tienen la ventaja de que permiten realizar cualquier función lógica con un sólo tipo de puerta lógica



Realización de circuitos con puertas NAND

- Aplicación directa de las leyes de De Morgan
- Ejemplo: $f(a,b,c) = a\bar{b} + c\bar{d} =$

$$= \overline{\overline{a}\bar{b} + c\bar{d}} = \overline{\overline{a}\bar{b}} \bullet \overline{c\bar{d}}$$

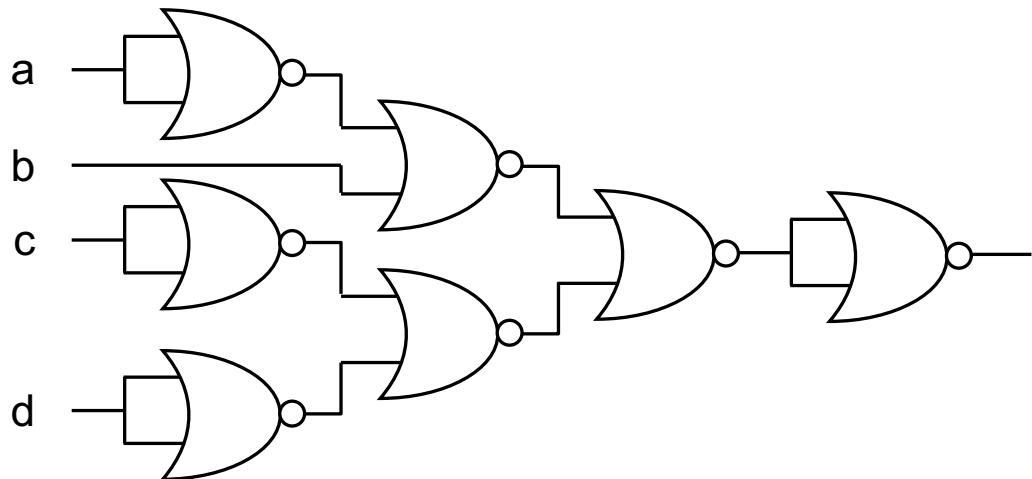




Realización de circuitos con puertas NOR

- Aplicación directa de las leyes de De Morgan
- Ejemplo: $f(a,b,c) = a\bar{b} + cd =$

$$= \overline{\overline{ab}} + \overline{\overline{cd}} = \overline{\overline{a} + b} + \overline{\overline{c} + d}$$





Minimización de funciones lógicas

- Una función lógica tiene múltiples expresiones equivalentes
 - La forma más sencilla dará lugar a una implementación mejor
- Criterios de optimización:
 - En tamaño o área:
 - Menor número de puertas lógicas
 - Puertas lógicas con el menor número de entradas
 - En velocidad o retardo:
 - Menor número de puertas lógicas desde una entrada hasta la salida
- Nos centraremos en la optimización en área



Minimización de funciones lógicas

• Métodos de optimización

- Manual: aplicación directa de las leyes del Álgebra de Boole
 - Muy difícil, no sistemático
- En dos niveles: el objetivo es obtener una expresión óptima en forma de suma de productos o productos de sumas
 - Existen soluciones sistemáticas y óptimas
 - Aplicable manualmente (para pocas variables) o con ayuda de un computador
- Multinivel
 - Mejor solución, aunque mucho más difícil
 - Sólo posible con ayuda de un computador



Métodos de los mapas de Karnaugh

- Método de optimización en dos niveles
- Se puede realizar manualmente hasta 6 variables
- Se basa en la Propiedad de adyacencia
 - $\forall E, x \in B \Rightarrow E x + E \bar{x} = E(x + \bar{x}) = E$
 $(E + x)(E + \bar{x}) = E + (x \bullet \bar{x}) = E \quad (\text{dual})$
 - Dos términos son adyacentes si son idénticos excepto por un literal, que aparece negado en un término y no negado en el otro
 - Los dos términos se simplifican en uno sólo con eliminación del literal que los diferencia



Aplicación de la propiedad de adyacencia

- Ejemplo:

$$\begin{aligned}f(a,b,c) &= \sum_3(0,1,2,3,7) = \overbrace{\bar{a}\bar{b}\bar{c}} + \overbrace{\bar{a}\bar{b}c} + \overbrace{\bar{a}b\bar{c}} + \overbrace{\bar{a}bc} + abc = \\&= \overbrace{\bar{a}\bar{b}} + \overbrace{\bar{a}b} + \overbrace{bc} \\&= \overbrace{\bar{a}} + \overbrace{bc}\end{aligned}$$

- La observación de las adyacencias puede ser difícil en la práctica



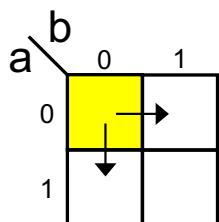
Mapas de Karnaugh

- Mapa que presenta la tabla de verdad de una función de manera que los términos adyacentes son contiguos:
 - Una casilla para cada combinación o término
 - Las casillas se numeran en código Gray
 - En un mapa de n variables, cada casilla tiene n casillas adyacentes que se corresponden con las combinaciones que resultan de invertir el valor de cada una de las n variables

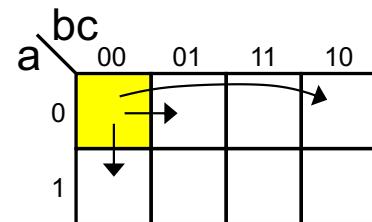
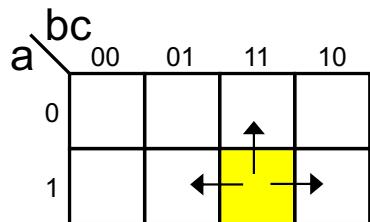


Mapas de Karnaugh: adyacencias

- Dos variables



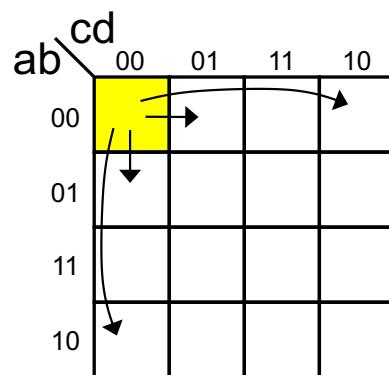
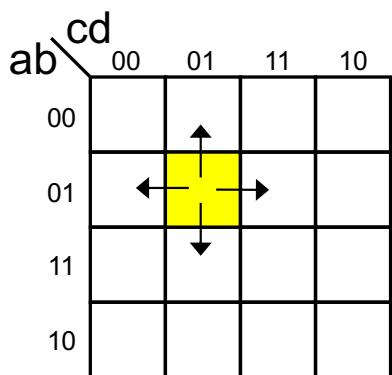
- Tres variables





Mapas de Karnaugh: adyacencias

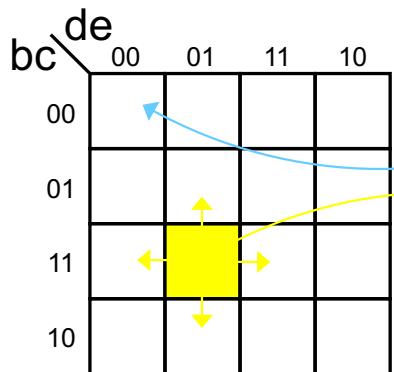
- Cuatro variables



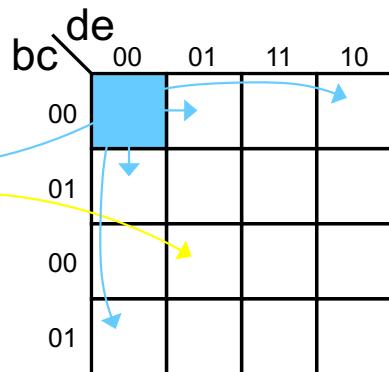


Mapas de Karnaugh: adyacencias

- Cinco variables



$a = 0$



$a = 1$



Mapas de Karnaugh: numeración de las casillas

- Dos variables

a\b	0	1
0	0	1
1	2	3

- Cuatro variables

ab\cd	00	01	11	10
00	0	1	3	2
01	4	5	7	6
11	12	13	15	14
10	8	9	11	10

- Tres variables

a\bc	00	01	11	10
0	0	1	3	2
1	4	5	7	6



Mapas de Karnaugh: numeración de las casillas

- Cinco variables

bc \ de	00	01	11	10
00	0	1	3	2
01	4	5	7	6
11	12	13	15	14
10	8	9	11	10

$a = 0$

bc \ de	00	01	11	10
00	16	17	19	18
01	20	21	23	22
11	28	29	31	30
10	24	25	27	26

$a = 1$



Representación de una función en el Mapa de Karnaugh

- Se marcan las casillas que corresponden a los mintérminos o los maxtérminos de la función
- Ejemplo:

$$\begin{aligned} f(a,b,c) &= \sum_3 (0,1,2,3,7) = \\ &= \prod_3 (4,5,6) \end{aligned}$$



		bc	00	01	11	10
		0	1	1	1	1
a	c	1			1	



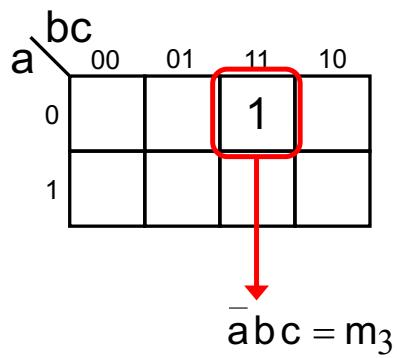
		bc	00	01	11	10
		0				
a	c	1	0	0		0



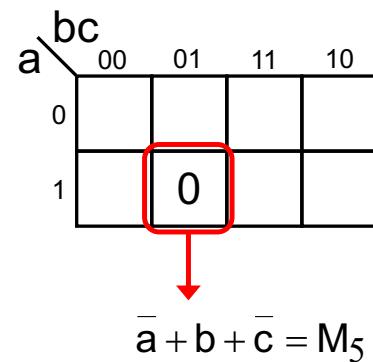
Obtención de una expresión a partir del Mapa de Karnaugh

- Se siguen las reglas para mintérminos y maxtérminos

- Regla para mintérminos
 - 0 → variable negada
 - 1 → variable sin negar



- Regla para maxtérminos
 - 0 → variable sin negar
 - 1 → variable negada



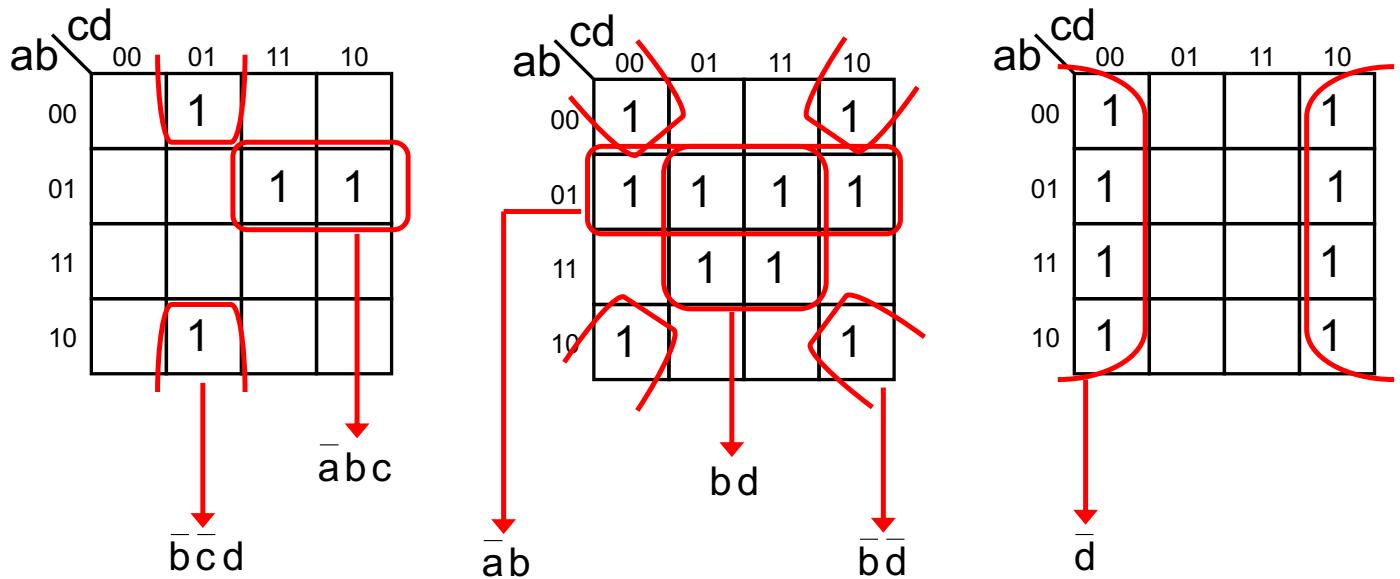


Simplificación mediante Mapas de Karnaugh

- Dos opciones
 - Por mintérminos (unos): se obtiene una suma de productos
 - Por maxtérminos (ceros): se obtiene un producto de sumas
- Buscar grupos de casillas adyacentes
 - Un grupo de 2 casillas adyacentes elimina 1 variable
 - Un grupo de 4 casillas adyacentes elimina 2 variables
 - Un grupo de 8 casillas adyacentes elimina 3 variables
 - Un grupo de 16 casillas adyacentes elimina 4 variables
 -
- Objetivo: cubrir todos los mintérminos (maxtérminos) con los grupos más grandes posibles y con el menor número de grupos
 - Se pueden repetir términos, si es necesario (propiedad de absorción)



Simplificación: formación de grupos





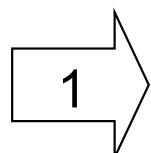
Simplificación mediante Mapas de Karnaugh: Algoritmo

- Algoritmo sistemático
 1. Cubrir las casillas que no pueden formar grupos de 2
 2. Cubrir las casillas que pueden formar grupos de 2, pero no de 4
 3. Cubrir las casillas que pueden formar grupos de 4, pero no de 8
 4. Cubrir las casillas que pueden formar grupos de 8, pero no de 16
 5. ...
- Si en algún paso hay más de una opción:
 - Comenzar siempre cubriendo las casillas que tienen menos opciones



Simplificación mediante Mapas de Karnaugh: Ejemplo

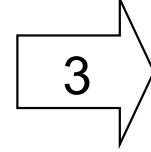
ab \ cd	00	01	11	10
00		1		
01			1	1
11			1	1
10			1	



ab \ cd	00	01	11	10
00		1		
01			1	1
11			1	1
10			1	



ab \ cd	00	01	11	10
00		1		
01			1	1
11			1	1
10			1	



ab \ cd	00	01	11	10
00		1		
01			1	1
11			1	1
10			1	



Funciones incompletas

- Una función incompletamente especificada (o simplemente incompleta) es aquella que no está especificada para alguna combinación de valores de sus entradas
- Las funciones incompletas se dan en la práctica:
 - Cuando las entradas provienen de otro circuito que no puede producir determinadas combinaciones por construcción
 - Cuando existen casos en que el valor de la función no tiene sentido o es indiferente
- Notación:
 - Un valor indiferente se representa con 'X' ó '-'
 - El conjunto de términos indiferentes ("don't cares") se denota con la letra Δ



Funciones incompletas

- Ejemplo: Función que determina si un número BCD es impar
 - Los números del 10 al 15 no tienen sentido en BCD

$$\begin{aligned}f(b_3, b_2, b_1, b_0) &= \sum_4 (1, 3, 5, 7, 9) + \Delta(10, 11, 12, 13, 14, 15) = \\&= \prod_4 (0, 2, 4, 6, 8) + \Delta(10, 11, 12, 13, 14, 15)\end{aligned}$$

Combinaciones indiferentes

b3	b2	b1	b0	f
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	1
1	0	1	0	X
1	0	1	1	X
1	1	0	0	X
1	1	0	1	X
1	1	1	0	X
1	1	1	1	X



Minimización de funciones incompletas

- Los términos indiferentes son “comodines”: se pueden cubrir o no, según convenga para formar grupos más grandes

		$b_1 b_0$	$b_3 b_2$			
		00	01	11	10	
		00	1	1		
		01	1	1		
		11	X	X	X	X
		10	1	X	X	

$$f(b_3, b_2, b_1, b_0) = \overline{b_3} b_0 + \overline{b_2} \overline{b_1} b_0$$

		$b_1 b_0$	$b_3 b_2$			
		00	01	11	10	
		00	1	1		
		01	1	1		
		11	X	X	X	X
		10	1	X	X	

$$f(b_3, b_2, b_1, b_0) = b_0$$

Correcto



Funciones múltiples

- En los circuitos digitales se implementan generalmente funciones múltiples: varias funciones a la vez o una función de múltiples salidas
- Las funciones múltiples se pueden implementar de forma óptima al considerarlas conjuntamente
 - Se pueden compartir términos o partes comunes para ahorrar lógica
- La descomposición de funciones múltiples de manera que se maximicen los términos comunes es difícil
 - Los algoritmos son difíciles de aplicar manualmente
 - Generalmente lo haremos por inspección

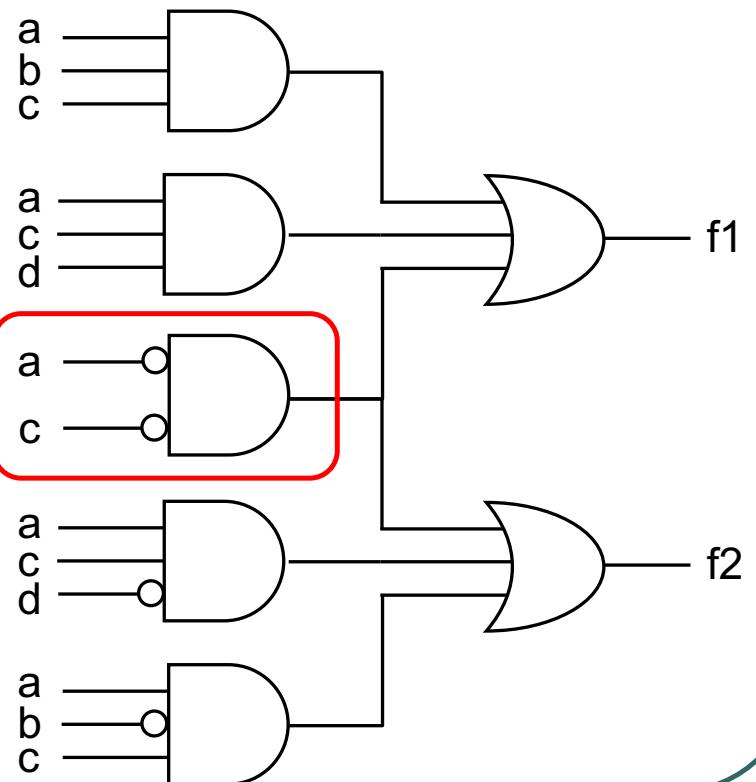


Funciones múltiples: Ejemplo

$$f_1(a,b,c,d) = \overline{a}\overline{c} + abc + acd$$

$$f_2(a,b,c,d) = \overline{a}\overline{c} + \overline{a}\overline{b}c + ac\overline{d}$$

Términos comunes





Funciones múltiples: Ejemplo

- Es posible encontrar más términos comunes

$$f_1(a,b,c,d) = \overline{a}\overline{c} + abc + acd = \overline{a}\overline{c} + \overline{a}\overline{c}\overline{d} + acd$$

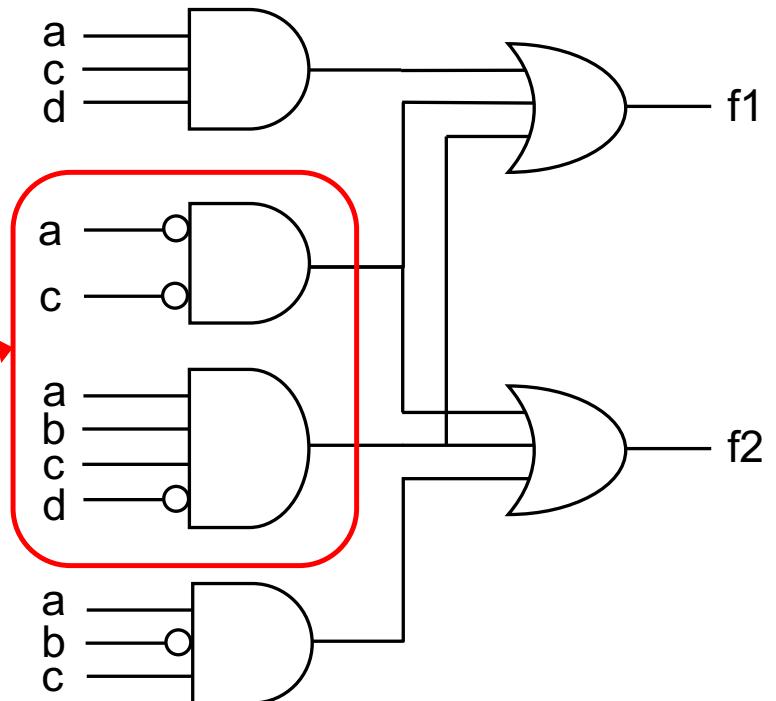
$$f_2(a,b,c,d) = \overline{a}\overline{c} + a\overline{b}c + a\overline{c}\overline{d} = \overline{a}\overline{c} + \overline{a}\overline{c}\overline{d} + a\overline{b}c$$

- Las expresiones de las funciones no son óptimas por separado, pero sí son óptimas en conjunto!
- Las herramientas de diseño incluyen algoritmos para minimizar funciones múltiples



Funciones múltiples: Ejemplo

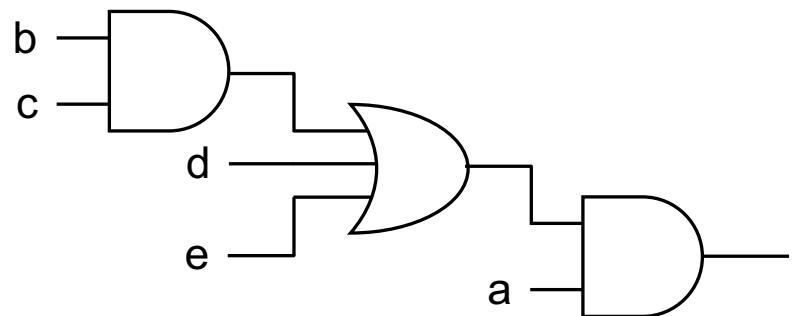
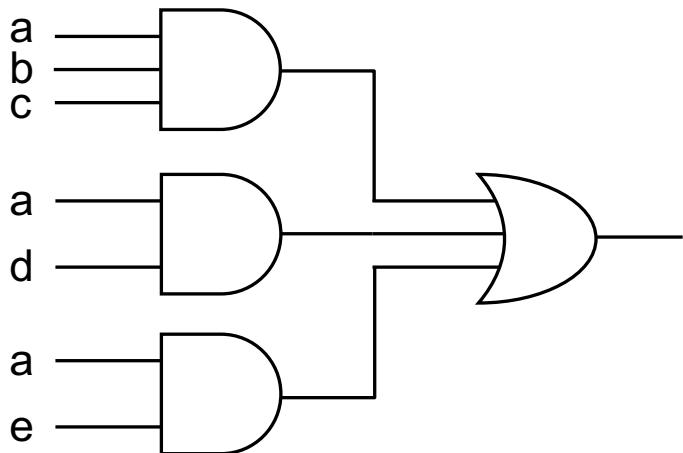
Términos comunes





Síntesis multinivel

- Si eliminamos la restricción a dos niveles, se pueden encontrar mejores soluciones
 - Se utilizan algoritmos heurísticos, con ayuda de un ordenador
- Ejemplo: $f(a,b,c,d,e) = abc + ad + ae = a(bc + d + e)$



Multinivel



Herramientas de optimización

- Métodos manuales:
 - Sólo en 2 niveles, pocas variables
- Herramientas software
 - Multinivel, múltiples funciones, muchas variables
 - Optimización en área o en retardo
 - Generalmente incorporadas en herramientas de síntesis lógica
- Herramientas de síntesis lógica
 - Funcionan como un compilador, a partir de la descripción del diseño en forma esquemática o mediante un Lenguaje de Descripción de Hardware
 - Optimizan el diseño y generan las puertas lógicas en una tecnología determinada



Referencias

- “Introducción al diseño lógico digital”. J. P. Hayes. Ed. Addison-Wesley
- “Circuitos y sistemas digitales”. J. E. García Sánchez, D. G. Tomás, M. Martínez Iniesta. Ed. Tebar-Flores



Extra



Tecnologías digitales

- Las puertas lógicas son circuitos electrónicos
- El nivel lógico (0 o 1) se representa mediante un nivel de tensión
- Generalmente se utiliza “lógica positiva”
 - Tensión alta (5V, 3.3V, 2.5 V, etc) → 1
 - Tensión baja (0V) → 0
- Existen muchas tecnologías, según la forma en que se realizan las puertas lógicas y las características que se obtienen



Familias lógicas

- El conjunto de componentes digitales básicos, tales como puertas lógicas y otros que estudiaremos a lo largo del curso, se conoce popularmente como *Serie* o *Familia 74*
- Existen numerosas subfamilias:
 - Segundo el rango de temperaturas de operación:
 - Serie 74: 0º a 70º
 - Serie 54: -55º a 125º
 - Segundo la tecnología utilizada:
 - LS
 - ALS
 - F
 - HC
 - AHC
 - G
 -



Familias lógicas

- Designación de componentes:
 - <Serie><Subfamilia><Componente>
- Ejemplo: 74HC00
 - Serie 74: rango de temperaturas convencional
 - Subfamilia HC (High speed CMOS)
 - Componente 00: 4 puertas NAND de 2 entradas
- Importante: las subfamilias no son compatibles entre sí
 - No se deben mezclar componentes de distintas subfamilias en un circuito



Hojas de catálogo

National Semiconductor

June 1989

54LS00/DM54LS00/DM74LS00 Quad 2-Input NAND Gates

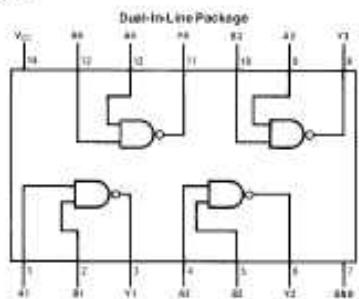
General Description

This device contains four independent gates each of which performs the logic NAND function.

Features

■ Alternate Military/Aerospace device (54LS00) is available. Contact a National Semiconductor Sales Office/Distributor for specifications.

Connection Diagram



Order Number 54LS00M08, 54LS00M08, 54LS00LM08, DM54LS00J, DM54LS00W, DM74LS00M or DM74LS00N
See NS Package Number E20A, J14A, M14A, H14A or W14B

54LS00/DM54LS00/DM74LS00 Quad 2-Input NAND Gates

Function Table

$Y = AB$

Inputs		Output
A	B	Y
L	L	H
L	H	H
H	L	H
H	H	L

H = High Logic Level

L = Low Logic Level



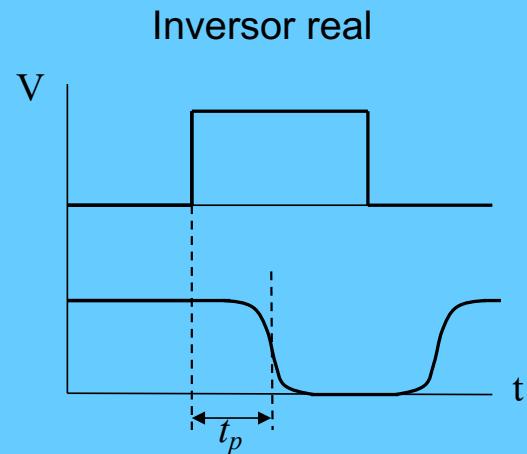
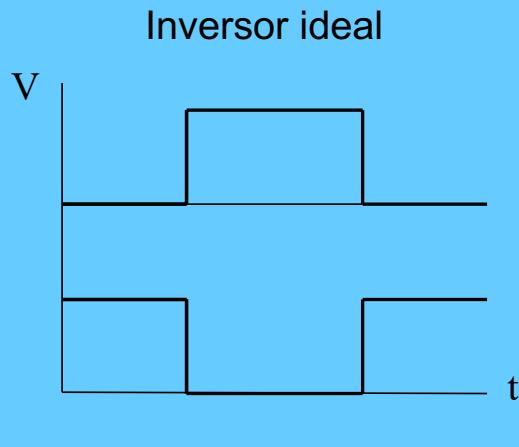
Características de las tecnologías digitales

- Principales características:
 - Margen de temperaturas de operación
 - Tensión de alimentación
 - Margen de ruido (intervalos de tensiones que se asocian a un nivel lógico determinado)
 - Retardo de conmutación
 - Consumo
 - Otros
- Cada tecnología o subfamilia presenta valores diferentes respecto a estos parámetros



Retardos

- Las puertas lógicas no comutan instantáneamente



- El retardo limita la velocidad de operación del circuito



Consumo

- Las puertas lógicas consumen energía:
 - Estática: la que se consume por tener alimentada la puerta lógica, sin cambiar los valores lógicos
 - Dinámica: la que se consume al comutar
- En la tecnología CMOS (la más utilizada actualmente), el consumo estático es muy pequeño. Sin embargo,
 - Los circuitos modernos pueden llegar a tener más de 10^8 puertas lógicas!
 - El consumo dinámico es proporcional a la frecuencia de comutación
- El consumo es un problema importante:
 - La energía consumida se transforma en calor, que hay que disipar. Si el circuito consume mucho, puede ser difícil disipar el calor
 - En dispositivos portátiles, el tamaño y el peso de la batería es limitado

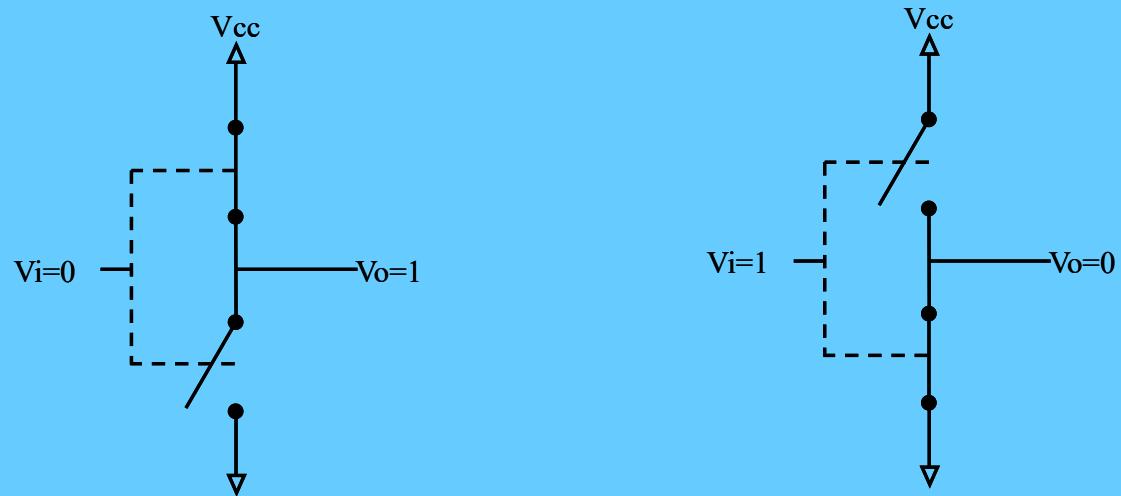


Tecnología CMOS

- La tecnología CMOS (Complementary Metal Oxide Semiconductor) es la tecnología más utilizada en la actualidad
- Basada en:
 - Transistores MOS: interruptores controlados por tensión
 - Complementarios: cada transistor o interruptor tiene su complementario, de manera que si un interruptor está abierto su complementario está cerrado y viceversa



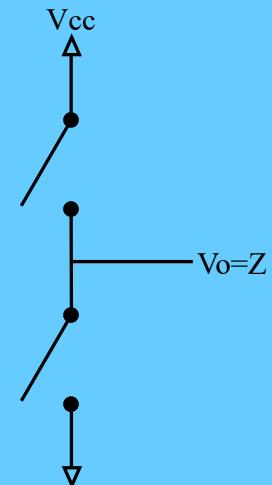
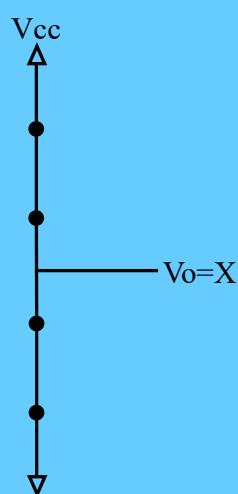
Inversor CMOS





Valores metalógicos

- Hay situaciones que no se corresponden con valores lógicos
 - Cortocircuito (X)
 - Alta impedancia o triestado (Z)





Aritmética Binaria

© Luis Entrena, Celia López, Mario García,
Enrique San Millán

Universidad Carlos III de Madrid



Índice

- Representación de números con signo
 - Sistemas de Signo y Magnitud, Complemento a 1 y Complemento a 2
 - Propiedades del sistema de Complemento a 2
- Aritmética Binaria
 - Adición y Sustracción
 - Multiplicación y División
- Representación de números reales



Representación de números con signo

- Los números con signo tienen dos partes: signo y magnitud
- Es necesario codificar el signo del número para poder representarlo utilizando sólo 0s y 1s:
 - Normalmente se codifica el signo con un 0 para números positivos y un 1 para números negativos
- Según diferentes formas de codificar la magnitud se obtienen 3 diferentes sistemas de representación:
 - Signo y magnitud
 - Complemento a 1
 - Complemento a 2



Representación Signo y Magnitud

- Los números en el sistema de Signo y Magnitud se codifican de la siguiente forma:
 - El MSB es el signo (0 si es positivo o 1 si es negativo)
 - El resto de los bits son la magnitud del número a representar, codificada en binario natural
- Ejemplos: $25_{10} = 11001_{\text{BIN}}$

$$+25_{10} = 011001_{\text{SM}}$$

$$-25_{10} = 111001_{\text{SM}}$$



Complemento a 1

- Los números en el sistema de Complemento a 1 se codifican:
 - Si el número es positivo:
 - El MSB es un 0 (signo)
 - El resto de los bits son la magnitud en binario natural
 - Si el número es negativo:
 - El MSB es un 1 (signo)
 - El resto de los bits son el complemento (a 1) de la magnitud
- Ejemplos:

$$+25_{10} = 011001_{\text{Ca1}}$$

$$-25_{10} = 100110_{\text{Ca1}}$$



Complemento a 2

- Los números en el sistema de Complemento a 2 se codifican:
 - Si el número es **positivo**:
 - El MSB es un 0 (signo)
 - El resto de los bits son la magnitud en binario natural
 - Si el número es **negativo**:
 - El MSB es un 1 (signo)
 - El resto de los bits son el complemento a 2 de la magnitud.
 - El complemento a dos de un número es su complemento + 1
 $Ca2(A) = \bar{A} + 1$
 - Equivalentemente, se puede definir como (siendo n el número de bits):
 $Ca2(A) = 2^n - A$
 $(2^n - A = 2^n - 1 + 1 - A = 11\dots11 - A + 1 = \bar{A} + 1)$
- Ejemplos:
 $+25_{10} = 011001_{Ca2}$
 $-25_{10} = 100111_{Ca2}$



Complemento a 2

- No hay que confundir los conceptos de “operación de complementar a 2” y “representación en complemento a 2” de un número:
 - Operación de complementar a 2 de un número A:
$$Ca2(A) = 2^n - A = \bar{A} + 1$$
 - Representación en complemento a 2 de un número A:
 - Hay que distinguir si es positivo o negativo, sólo en el caso de que sea negativo dicha representación se obtiene aplicando la operación de complementar a 2.
- La representación en complemento a 2 es la más utilizada en los sistemas digitales para números con signo.



Extensión del número de bits

- Un mismo número se puede representar con diferente número de bits:

$$+25_{10} = 011001_{SM} = \textcolor{red}{0000}11001_{SM} = \textcolor{red}{00000000}011001_{SM}$$

- Para extender el signo:

- En SM:
 - Añadir ceros justo después del signo
- En Ca1 y Ca2:
 - Si es positivo añadir ceros a la izquierda
 - Si es negativo añadir unos a la izquierda

$$+25_{10} = 011001_{Ca2} = \textcolor{red}{0000}011001_{Ca2} = \textcolor{red}{00000000}011001_{Ca2}$$

$$-25_{10} = 100111_{Ca2} = \textcolor{red}{1111}100111_{Ca2} = \textcolor{red}{1111111111}100111_{Ca2}$$



Complemento a 2

Codificación	SM	Ca1	Ca2
0 0 0	0	0	0
0 0 1	1	1	1
0 1 0	2	2	2
0 1 1	3	3	3
1 0 0	-0	-3	-4
1 0 1	-1	-2	-3
1 1 0	-2	-1	-2
1 1 1	-3	-0	-1

- En SM y en Ca1 hay dos codificaciones distintas que representan el 0
- En Ca2 la representación del 0 es única



Complemento a 2

- Otra propiedad del Ca2 es que el complemento a 2 del complemento a 2 de un número es el mismo número (la operación inversa del Ca2 es el propio complemento a 2):

$$\text{Ca2}(\text{Ca2}(A_{\text{Ca2}})) = A_{\text{Ca2}}$$

Dem: $\text{Ca2}(\text{Ca2}(A_{\text{Ca2}})) = 2^n - (\text{Ca2}(A_{\text{Ca2}})) = 2^n - (2^n - A_{\text{Ca2}}) = A_{\text{Ca2}}$

- De lo cual se deduce esta otra propiedad:

$$-A_{\text{Ca2}} = \text{Ca2}(A_{\text{Ca2}})$$

Dem: Si A_{Ca2} es positivo, entonces por la propia definición de representación en Ca2

Si A_{Ca2} es negativo, entonces $-A_{\text{Ca2}}$ se obtiene haciendo la operación inversa del Ca2, pero como $\text{Ca2}(\text{Ca2}(A_{\text{Ca2}})) = A_{\text{Ca2}}$ se tiene que $-A_{\text{Ca2}} = \text{Ca2}(A_{\text{Ca2}})$

- Ejemplo: Partiendo de un número positivo: 00001001 (+9)
Realizando la operación de Ca2: 11110111 (-9)
Realizando la operación de Ca2: 00001001 (+9)



Complemento a 2

- Otra forma de calcular el complemento a 2 de un número:
 - Empezando por el LSB, dejar iguales los bits hasta encontrar el primer 1 e invertir el resto:
 - $\text{Ca2}(11100\textcolor{red}{100}) = 00011\textcolor{red}{100}$
Comprobación: $\text{Ca2}(11100100) = 00011011 + 1 = 00011100$
- Otra forma de convertir de Ca2 a decimal:
 - Considerar el peso del signo como negativo
 - Ejemplos:
 - $1110_2 = 1*(-2^3) + 1*2^2 + 1*2^1 + 0*2^0 = -8 + 4 + 2 = -2_{10}$
 - $0110_2 = 0*(-2^3) + 1*2^2 + 1*2^1 + 0*2^0 = 4 + 2 = 6_{10}$
 - Demostración:
 - Si A es positivo el peso del signo será cero, y por tanto la magnitud es la suma del resto de pesos
 - Si A es negativo el peso será -2^n , y por tanto el número que se obtendrá es:
$$-2n + A = -\text{Ca2}(A) = -(-A) = A$$



Suma binaria

- Las sumas de números naturales en binario se realizan igual que en decimal:
$$\begin{array}{r} 1001 \quad (9) \\ + 1101 \quad (13) \\ \hline 10110 \quad (22) \end{array}$$
- Utilizando la representación de los números en Ca2, el método es válido también para números con signo, si se siguen las siguientes reglas.
 - Operandos con el mismo número de bits
 - Se descarta el acarreo final
 - Si los dos operandos tienen el mismo signo, y el resultado de la operación tiene signo diferente el resultado no es válido. Se dice que hay desbordamiento (“**overflow**”):
 - Esto sucede porque haría falta un bit adicional para poder representar el resultado.



Suma binaria en Ca2: Ejemplos

- Dos números positivos:
 $(+9) + (+4)$

$$\begin{array}{r} 0\ 1001_{\text{Ca2}} \ (+9) \\ +\ 0\ 0100_{\text{Ca2}} \ (+4) \\ \hline 0\ 1101_{\text{Ca2}} \ (+13) \end{array}$$

- Número positivo grande y número negativo pequeño:
 $(+9) + (-4)$

$$\begin{array}{r} 0\ 1001_{\text{Ca2}} \ (+9) \\ +\ 1\ 1100_{\text{Ca2}} \ (-4) \\ \hline 4\ 0\ 0101_{\text{Ca2}} \ (+5) \end{array}$$

- Dos números negativos:
 $(-9) + (-4)$

$$\begin{array}{r} 1\ 0111_{\text{Ca2}} \ (-9) \\ +\ 1\ 1100_{\text{Ca2}} \ (-4) \\ \hline 4\ 1\ 0011_{\text{Ca2}} \ (-13) \end{array}$$

- Número positivo pequeño y número negativo grande:
 $(-9) + (+4)$

$$\begin{array}{r} 1\ 0111_{\text{Ca2}} \ (-9) \\ +\ 0\ 0100_{\text{Ca2}} \ (+4) \\ \hline 1\ 1011_{\text{Ca2}} \ (-5) \end{array}$$

- Números iguales de signo contrario:
 $(+9) + (-9)$

$$\begin{array}{r} 0\ 1001_{\text{Ca2}} \ (+9) \\ +\ 1\ 0111_{\text{Ca2}} \ (-9) \\ \hline 4\ 0\ 0000_{\text{Ca2}} \ (0) \end{array}$$

- Overflow:
 $(+1) + (+1)$

$$\begin{array}{r} 0\ 1_{\text{Ca2}} \ (+1) \\ +\ 0\ 1_{\text{Ca2}} \ (+1) \\ \hline 1\ 0_{\text{Ca2}} \ (-2) \end{array}$$



Resta binaria

- Para realizar restas binarias se utiliza la propiedad vista anteriormente del complemento a 2:

$$-A_{Ca2} = Ca2(A_{Ca2})$$

Y por tanto:

$$A_{Ca2} - B_{Ca2} = A_{Ca2} + Ca2(B_{Ca2})$$

O equivalentemente:

$$A_{Ca2} - B_{Ca2} = A_{Ca2} + \overline{B_{Ca2}} + 1$$

- En los sistemas digitales se representan los números con signo en Ca2, y no se utilizan restadores (no hacen falta), sólo sumadores



Multiplicación binaria

- Las multiplicaciones en binario se realizan de forma similar a las multiplicaciones en decimal:

$$\begin{array}{r} 1001 \quad (9) \\ 1101 \quad (13) \\ \hline 1001 \\ 0000 \\ 1001 \\ \hline 1001 \\ \hline 1110101 \quad (117) \end{array}$$

- En el caso de números con signo se utiliza la representación en complemento a 2:

- Si los dos números son positivos, se realiza directamente la operación. Al resultado se añade un bit de signo 0 (el resultado es un número positivo).
- Si los dos números son negativos, se complementan a 2, se multiplican y al resultado se añade un bit de signo 0 (el resultado es un número positivo).
- Si uno de los números es negativo, se hace el Ca2 de ese número, se multiplica por el otro, y al resultado se le realiza el Ca2 y se le añade un bit de signo 1 (el resultado es negativo).



División binaria

- La división se realiza también como en sistema decimal:

$$\begin{array}{r} 1001 \quad / \quad 11 \\ -011 \\ \hline 0011 \\ -11 \\ \hline 0 \end{array}$$

- En el caso de números con signo se utiliza la representación en complemento a dos y las mismas reglas que para la multiplicación en cuanto a signos.



Representación de números reales

- Los números reales se pueden representar de dos formas:
 - Punto fijo
 - Punto flotante
- Punto fijo:
 - La coma decimal se considera fija en un punto.
 - Ejemplo: Datos de 32 bits, utilizar 20 bits para la parte entera y 12 bits para los decimales
 - Fácil realizar las operaciones de sumas, restas, multiplicaciones y divisiones vistas hasta ahora
 - Notación: $Q_{m,n}$
 - m: número de bits de la parte entera (opcional)
 - n: número de bits para la parte decimal
 - Se utiliza un bit adicional para el signo (en total hacen falta $m+n+1$ bits).
 - Ejemplos: Q16,16, Q.32, etc.



Representación de números reales

- Punto flotante:

- La coma decimal es “flotante”
- Se descompone el número en dos partes: mantisa y exponente:

$$N = M \times b^E$$

Ejemplos:

- $2547,35_{10} = 2,54735 \times 10^3$
- $0,0035_{10} = 3,5 \times 10^{-3}$
- $111,0110_2 = 1,11011_2 \times 2^2$
- $0,001101_2 = 1,101_2 \times 2^{-3}$
- Se utiliza un número fijo de bits para la mantisa, otro para el exponente y otro adicional para el signo
- Normalización: Fija la posición de la coma decimal en la descomposición (para tener una representación única)



Representación de números reales

- Standard IEEE 754: Precisión simple (32 bits)
 - Se utiliza 1 bit para el signo
 - Se utilizan 8 bits para el exponente (E)
 - Se utilizan 23 bits para la mantisa (M)
 - Números normalizados: $N = (-1)^s \cdot 2^{E-127} \cdot 1.M$
 - E puede tomar valores entre 1 y 254 (el exponente está desplazado por -127)
 - El cero se representa como todo ceros en los campos E y M (es una excepción)
 - Algunas otras excepciones: E = 255 denota infinito (utilizado en casos de overflow, por ejemplo)
 - También se pueden representar números no normalizados (E=0). La descomposición es diferente, la mantisa es 0.M
- Standard IEEE 754: Precisión doble (64 bits)
 - Representación similar
 - 1 bit para signo, 11 bits para exponente, 52 bits para mantisa



Representación de números reales

- Ejemplo: Representar -7.625_{10} en precisión sencilla

$$-7.625_{10} = -111.101_2$$

Descomponiendo en la forma $N = (-1)^s \cdot 2^{E-127} \cdot 1.M$:

$$-111.101_2 = (-1)^1 \cdot 1.11101 \cdot 2^2$$

$$S = 1$$

$$M = 11101$$

$$E = E - 127 \rightarrow E = 129_{10} = 10000001_2$$

Por tanto el número representado con precisión sencilla:

$$1\ 10000001\ 1110100000000000000000000$$



Representación de números reales

- Operaciones aritméticas con números en coma flotante: más complejas
 - Suma:
 - Modificar uno de los números para que tengan el mismo exponente (denormalizarlo)
 - Normalizar el resultado una vez realizada la operación
 - Resta:
 - Análogo a la suma. Añade complejidad de convertir a complemento a 2
 - Multiplicación:
 - Sumar exponentes
 - Multiplicar mantisas
 - Redondear y normalizar
 - División:
 - Análogo a la multiplicación
- Debido a la complejidad de las operaciones en coma flotante, muchos circuitos electrónicos digitales (microprocesadores por ejemplo) tienen módulos dedicados a realizar estas operaciones



Referencias

- Digital Systems Fundamentals. Thomas L. Floyd. Pearson Prentice Hall
- Introduction to Digital Logic Design. John P. Hayes. Addison-Wesley
- Digital Systems. Principles and Applications. Ronald J. Tocci. Pearson Prentice Hall.



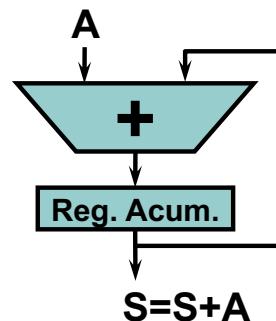
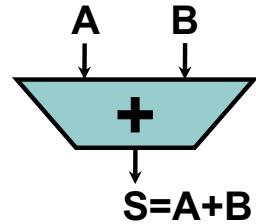
Circuitos combinacionales

© Luis Entrena, Celia López,
Mario García, Enrique San Millán

Universidad Carlos III de Madrid

Circuitos combinacionales y secuenciales

- Combinacionales:
 - Salida depende sólo de la entrada
 - Ejemplo: sumador de dos operandos
- Secuenciales:
 - Salida depende de las entradas y del estado
 - Ejemplo: sumador acumulador





Índice

- Codificadores
 - Decodificadores
 - Multiplexores
 - Demultiplexores
 - Comparadores
- Funcionalidad
 - Implementación
 - Asociación
 - Uso para implementación de funciones
 - Utilidad



1. Codificadores

- Definición:
 - Circuito combinacional que permite transformar un nivel activo en una de sus entradas en un valor codificado
- Ejemplo: teclado numérico
 - Entradas: dígitos 0-9
 - Salidas: codificación binaria (4 bits)

'0'	E0	S3	'0'
'0'	E1	S2	'1'
'0'	E2	S1	'0'
'0'	E3	S0	'1'
'0'	E4		
'1'	E5		
'0'	E6		
'0'	E7		
'0'	E8		
'0'	E9		

Activar E5 => S="0101" (=5)



Codificadores sin prioridad

- Características

- Suponen que sólo una entrada puede estar activa
- Si se activan varias entradas a la vez, la salida puede ser errónea.

- Funciones lógicas

- $S_3 = E_8 + E_9$
- $S_2 = E_4 + E_5 + E_6 + E_7$
- $S_1 = E_2 + E_3 + E_6 + E_7$
- $S_0 = E_1 + E_3 + E_5 + E_7 + E_9$

- Problemas:

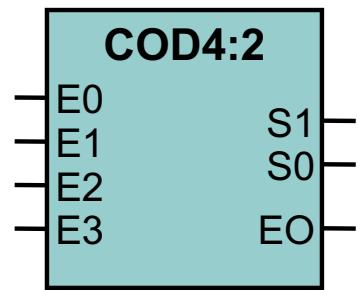
- E_1 y E_4 activas dan resultado 5
- Ninguna entrada activa da resultado 0

Entrada activa	$S_3S_2S_1S_0$
E_0	0000
E_1	0001
E_2	0010
E_3	0011
E_4	0100
E_5	0101
E_6	0110
E_7	0111
E_8	1000
E_9	1001



Ejemplo: codificador 4:2 sin prioridad

- M:N \Rightarrow 'M' entradas, 'N' salidas
- EO: "Enable Output"
 - Sirve para diferenciar el caso de activarse E_0 y el de que no haya nada activo
 - También sirve para asociar varios codificadores
- Casos no contemplados
 - Cualquier combinación de activación múltiple
 - Las salidas son indiferentes



E₃	E₂	E₁	E₀	S₁	S₀	EO
0	0	0	1	0	0	0
0	0	1	0	0	1	0
0	1	0	0	1	0	0
1	0	0	0	1	1	0
0	0	0	0	0	0	1
Resto de casos				X	X	X



Ejemplo: codificador 4:2 sin prioridad

E₃	E₂	E₁	E₀	S₁	S₀	EO
0	0	0	1	0	0	0
0	0	1	0	0	1	0
0	1	0	0	1	0	0
1	0	0	0	1	1	0
0	0	0	0	0	0	1
Resto de casos				X	X	X

E_{1E₀} E _{3E₂}	00	01	11	10
00	0	0	X	0
01	1	X	X	X
11	X	X	X	X
10	1	X	X	X

$$S_1 = E_2 + E_3$$

E_{1E₀} E _{3E₂}	00	01	11	10
00	1	0	X	0
01	0	X	X	X
11	X	X	X	X
10	0	X	X	X

$$EO = \overline{E}_3 \overline{E}_2 \overline{E}_1 \overline{E}_0$$

E_{1E₀} E _{3E₂}	00	01	11	10
00	0	0	X	1
01	0	X	X	X
11	X	X	X	X
10	1	X	X	X

$$S_0 = E_1 + E_3$$



Codificadores con prioridad

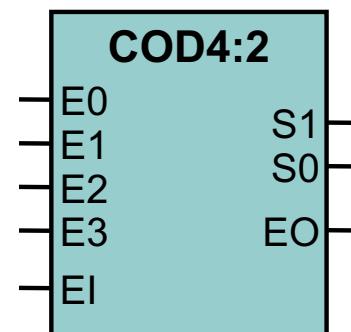
• Características

- Si se activan varias entradas a la vez, dan prioridad a una de ellas
- Prioridad:
 - Al bit más significativo: se da prioridad a la entrada mayor
Si se activan E1 y E5, el resultado es 5
 - Al bit menos significativo: se da prioridad a la entrada menor
Si se activan E1 y E5, el resultado es 1



Ejemplo: codificador 4:2 con prioridad al más significativo

- M:N \Rightarrow 'M' entradas, 'N' salidas
- EO: "Enable Output"
- EI ó E: "Enable Input" o "Enable".
Habilitación
 - Sirve para habilitar:
 - '0' (deshabilitado) implica que las salidas valen '0'
 - '1' (habilitado) indica funcionamiento normal
 - Junto con EO también sirve para asociar varios codificadores



EI	E3	E2	E1	E0	S1	S0	EO
0	X	X	X	X	0	0	0
1	0	0	0	0	0	0	1
1	0	0	0	1	0	0	0
1	0	0	1	X	0	1	0
1	0	1	X	X	1	0	0
1	1	X	X	X	1	1	0



Ejemplo: codificador 4:2 con prioridad al más significativo

E1	E3	E2	E1	E0	S1	S0	EO
0	X	X	X	X	0	0	0
1	0	0	0	0	0	0	1
1	0	0	0	1	0	0	0
1	0	0	1	X	0	1	0
1	0	1	X	X	1	0	0
1	1	X	X	X	1	1	0

- Recordatorio

- ‘X’ en las salidas \Rightarrow ‘X’ en el diagrama
- ‘X’ en las entradas \Rightarrow múltiples casos

E ₁ E ₀ E ₃ E ₂	00	01	11	10
00	0	0	0	0
01	1	1	1	1
11	1	1	1	1
10	1	1	1	1

$$S_1 = EI(E_2 + E_3)$$

E ₁ E ₀ E ₃ E ₂	00	01	11	10
00	0	0	1	1
01	0	0	0	0
11	1	1	1	1
10	1	1	1	1

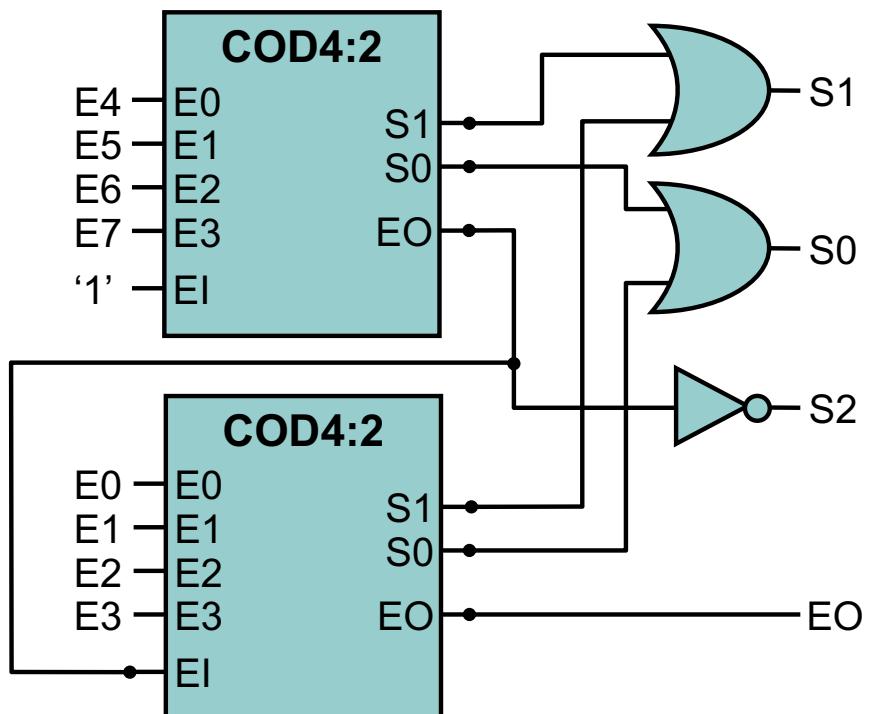
$$S_0 = EI(\overline{E_1} \overline{E_2} + E_3)$$

E ₁ E ₀ E ₃ E ₂	00	01	11	10
00	1	0	0	0
01	0	0	0	0
11	0	0	0	0
10	0	0	0	0

$$EO = EI(\overline{E_3} \overline{E_2} \overline{E_1} \overline{E_0})$$

Asociación de codificadores: COD8:3 con dos COD 4:2

- Se encadenan los EI y EO
- Cuando un COD está activo ($EI=1'$) y no tiene ninguna entrada activa, activa al siguiente COD ($EO=1'$).





Utilidad de los codificadores

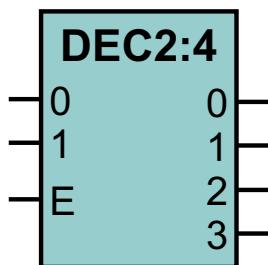
- Sensores de piso de un ascensor
 - Codifican cada sensor al número de piso
 - No necesita prioridad, ya que el ascensor sólo puede estar en un piso
- Botonera
 - Codifica el valor de la tecla pulsada
 - Necesita prioridad, ya que se pueden pulsar varios botones a la vez



2. Decodificadores

- Definición:

- Circuito combinacional que transforma un valor codificado en la activación de la salida correspondiente al dicho valor.
- Realizan la función inversa a los codificadores

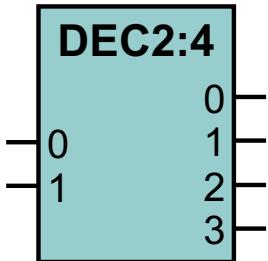


E	E ₁	E ₀	S ₃	S ₂	S ₁	S ₀
0	X	X	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0



Decodificadores

- Funciones lógicas:
 - Cada salida del decodificador es un mintérmino



E ₁	E ₀	S ₃	S ₂	S ₁	S ₀
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

$$S_0 = \overline{E}_1 \overline{E}_0$$

$$S_1 = \overline{E}_1 E_0$$

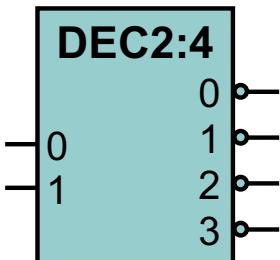
$$S_2 = E_1 \overline{E}_0$$

$$S_3 = E_1 E_0$$



Decodificadores

- Decodificador con salidas activas por nivel bajo:
 - Cada salida del decodificador es un maxtérmino



E ₁	E ₀	S ₃	S ₂	S ₁	S ₀
0	0	1	1	1	0
0	1	1	1	0	1
1	0	1	0	1	1
1	1	0	1	1	1

$$S_0 = E_1 + E_0$$

$$S_1 = E_1 + \overline{E}_0$$

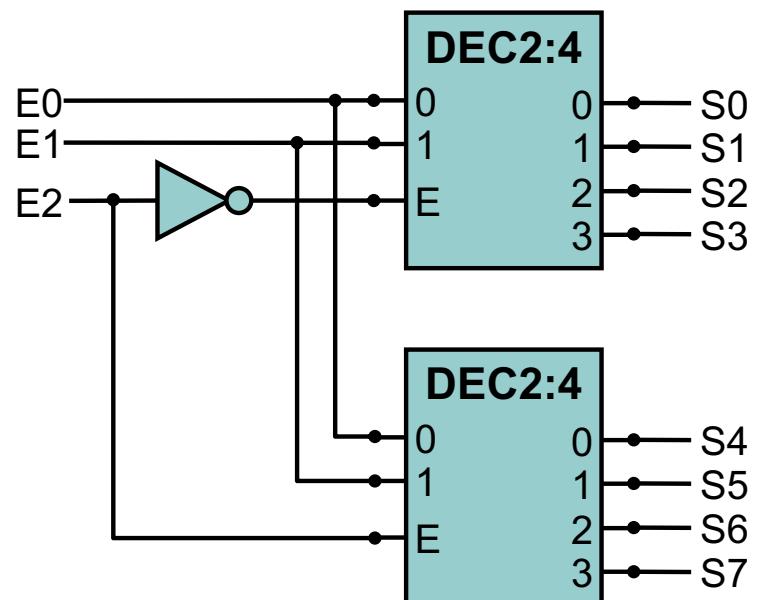
$$S_2 = \overline{E}_1 + E_0$$

$$S_3 = \overline{E}_1 + \overline{E}_0$$

Asociación de decodificadores

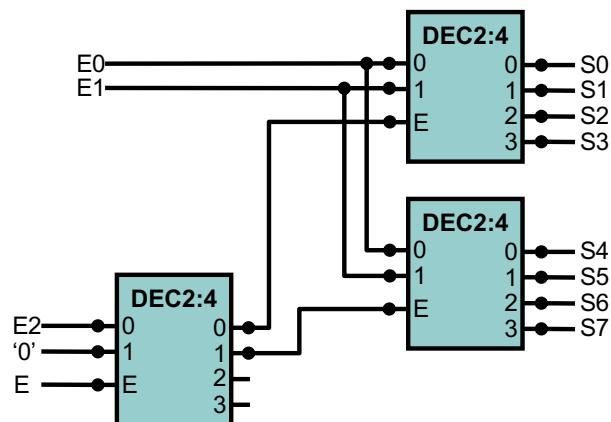
- DEC3:8 con DEC2:4

- Sólo uno de los decodificadores está activo, dependiendo del valor de E2
- El inversor hace la función de un DEC1:2
- No tiene Enable global



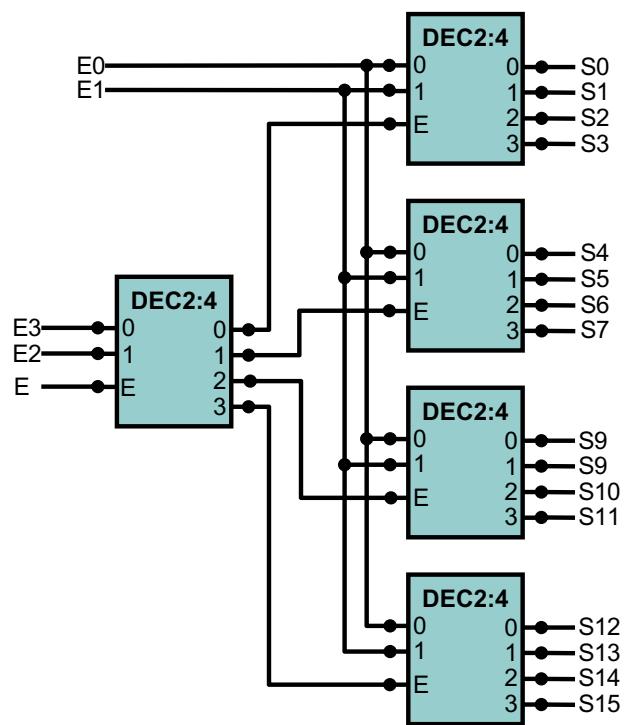
Asociación de decodificadores

- DEC4:16 con DEC2:4
 - Sólo uno de los decodificadores está activo, dependiendo del valor de E2
 - El decodificador de la izquierda se comporta como un DEC1:2
 - Tiene Enable Global. Si $E=0$, ningún decodificador se activa y las salidas valen '0'



Asociación de decodificadores

- DEC4:16 con DEC2:4
 - Sólo uno de los decodificadores está activo, dependiendo del valor de E3 y E2

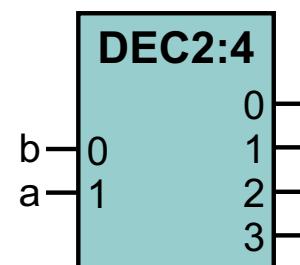


Implementación de funciones lógicas con decodificadores

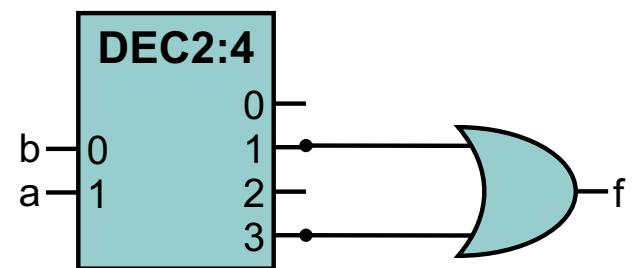
- Se pueden implementar funciones lógicas con un DEC y una puerta OR
- Las salidas del DEC son los mintérminos. Se suman las que valgan '1' en la tabla de verdad
- El dual se hace con DEC de salidas a nivel bajo y una puerta AND.

m	a	b	f
0	0	0	0
1	0	1	1
2	1	0	0
3	1	1	1

$$f = \bar{a}\bar{b} + ab = S_1 + S_3$$



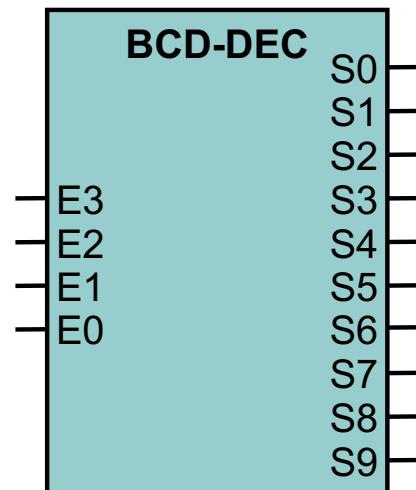
$$\begin{aligned}S_0 &= \bar{a}\bar{b} \\S_1 &= \bar{a}b \\S_2 &= a\bar{b} \\S_3 &= ab\end{aligned}$$





Decodificador BCD-decimal

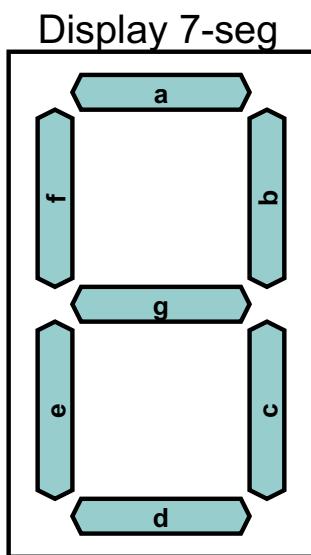
- Decodifica un dígito decimal codificado en BCD (natural) a 10 salidas que representan 0-9
- El comportamiento no está definido si la entrada no es un dígito decimal



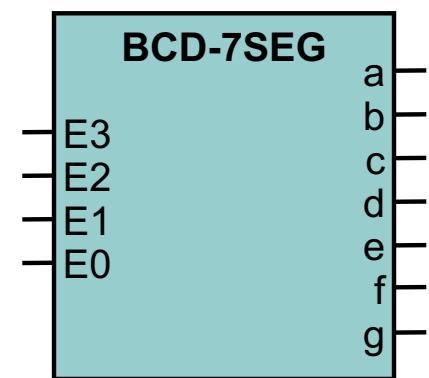


Decodificador BCD-7 segmentos

- Decodifica un dígito decimal codificado en BCD (natural) a los LEDs de un “display 7-segmentos”



E ₃	E ₂	E ₁	E ₀	a	b	c	d	e	f	g
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	0	1	1
Resto				X	X	X	X	X	X	X





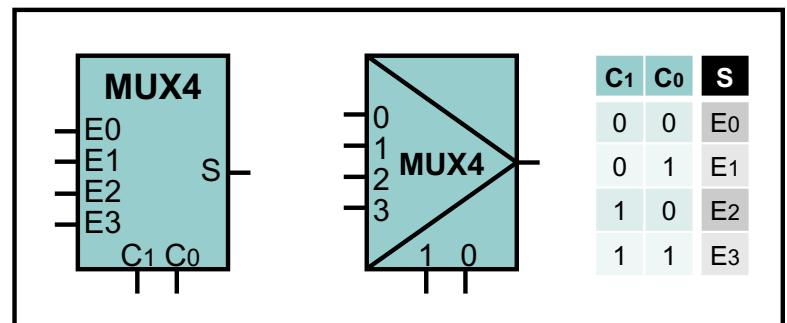
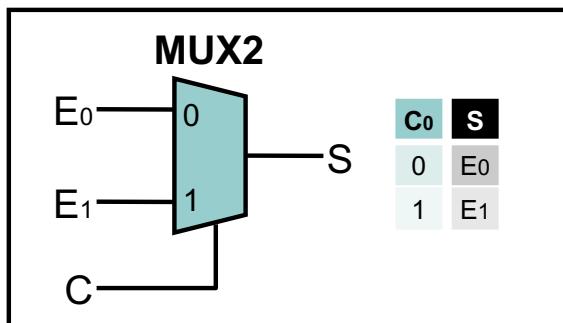
Utilidad de los decodificadores

- Microprocesadores:
 - Decodificación de instrucciones
 - Puertos de E/S, direcciones de memoria, etc.

3. Multiplexores

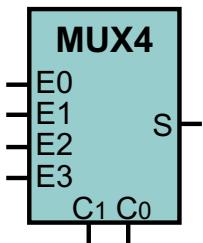
- Definición:

- Circuito que permite seleccionar una de las entradas y copiar su valor a la salida. La entrada seleccionada depende del valor que se dé a las entradas de control.
- Se denominan por el número de entradas de dato: MUX2, MUX4, ...
- $N = \text{entradas de datos}, n = \text{entradas de control} \Rightarrow 2^n = N$



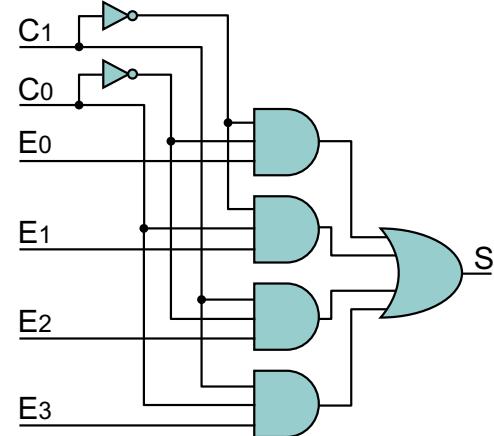
Multiplexores

- Función lógica
- Implementación con puertas



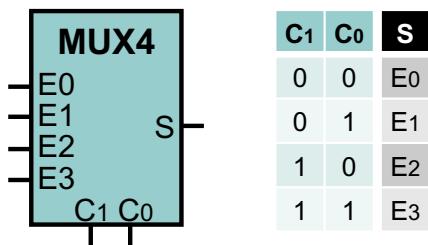
C ₁	C ₀	S
0	0	E ₀
0	1	E ₁
1	0	E ₂
1	1	E ₃

$$S = \overline{C_1} \overline{C_0} E_0 + \overline{C_1} C_0 E_1 + \\ + C_1 \overline{C_0} E_2 + C_1 C_0 E_3$$

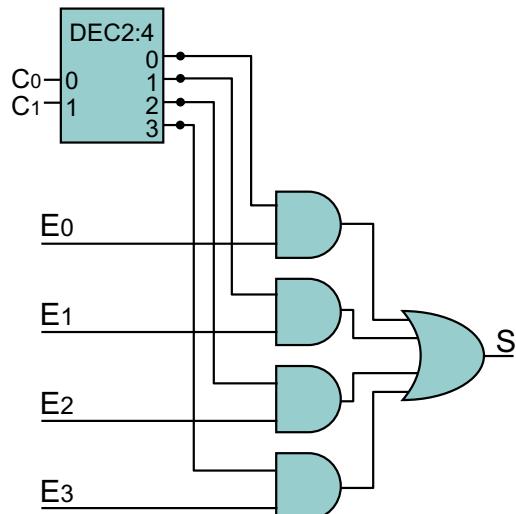


Multiplexores

- Función lógica
- Implementación con decodificador

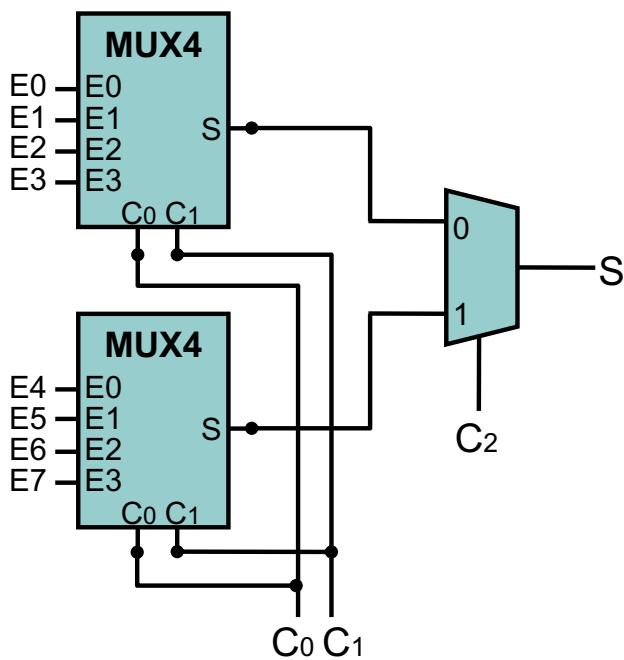


$$S = \overline{C_1} \overline{C_0} E_0 + \overline{C_1} C_0 E_1 + \\ + C_1 \overline{C_0} E_2 + C_1 C_0 E_3$$



Asociación de multiplexores

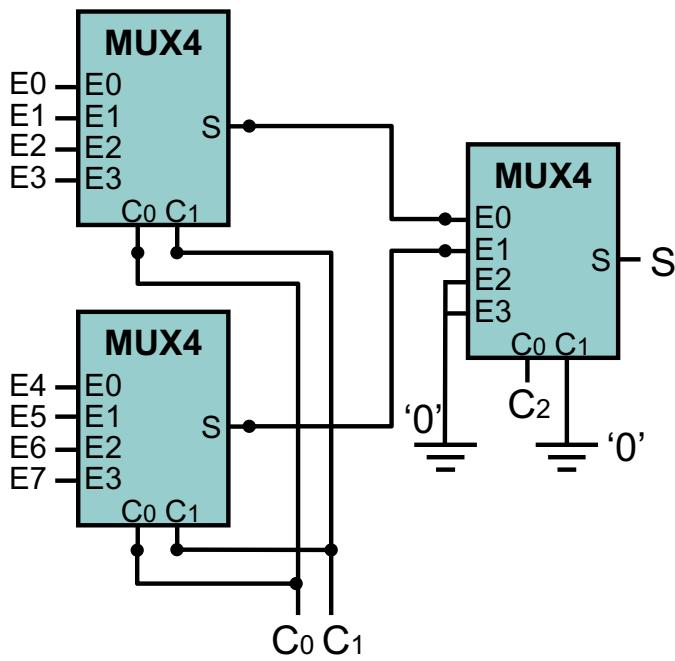
- MUX8 a partir de MUX4 y MUX2



- MUX2 selecciona entre los MUX4, dependiendo del valor del bit de control más significativo (C₂)
- Los bits de C y E deben asignarse según su peso

Asociación de multiplexores

- MUX8 a partir de MUX4

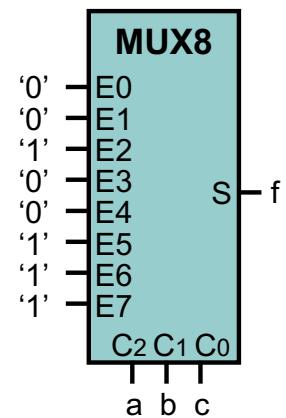


- El MUX4 de la derecha se comporta como un MUX2
- Recordatorio: las entradas de un circuito DEBEN estar conectadas; las salidas pueden quedar desconectadas

Implementación de funciones lógicas con multiplexores

- Con un MUX de tantas entradas de control como variables tiene la función
 - Las variables de la función van al control del MUX, ordenadas por peso
 - Los valores de la función en la tabla de verdad son las entradas de datos del MUX

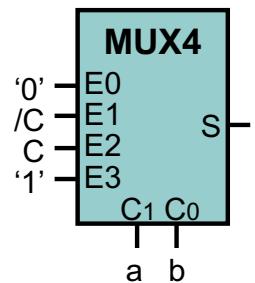
a	b	c	f
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1



Implementación de funciones lógicas con multiplexores

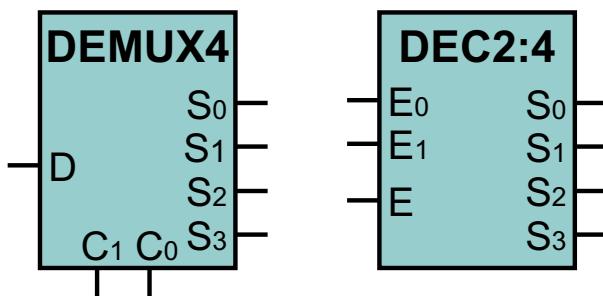
- Con un MUX de menos entradas de control que variables tiene la función
 - Agrupar la tabla de verdad según las variables menos significativas
 - Las variables de la función de mayor peso van al control del MUX, ordenadas por peso
 - Los valores de la función en la tabla de verdad son las entradas de datos del MUX

a	b	c	f	f(c)
0	0	0	0	0
0	0	1	0	
0	1	0	1	/C
0	1	1	0	
1	0	0	0	C
1	0	1	1	
1	1	0	1	1
1	1	1	1	



4. Demultiplexores

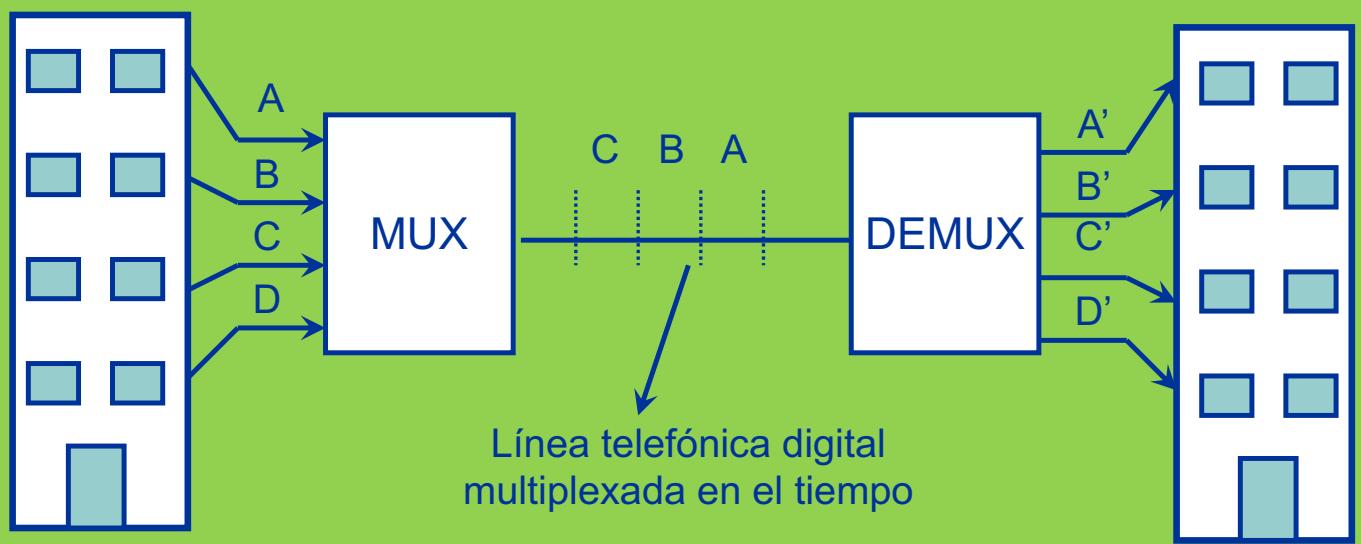
- Definición: circuito que copia el valor de la entrada de datos en la salida indicada por el valor de las señales de control.
- Son el circuito opuesto a los multiplexores
- Son equivalentes a decodificadores, si equiparamos las entradas de control (C_i) del DEMUX a las de datos (E_i) del DEC, y la señal de dato del DEMUX (D) al Enable del DEC (E)



D	C_1	C_0	S_3	S_2	S_1	S_0
E	E_1	E_0				
0	X	X	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

Utilidad de multiplexores y demultiplexores

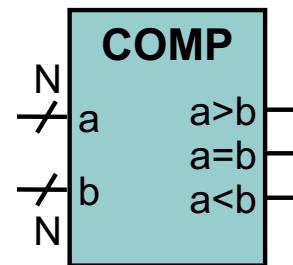
- Transmisión serie multiplexada





5. Comparadores

- Definición: circuito que permite determinar si dos datos son iguales, o si uno es mayor que otro.
- N es el número de bits de los datos



Comparador 1-bit

a	b	a=b	a>b	a<b
0	0	1	0	0
0	1	0	0	1
1	0	0	1	0
1	1	1	0	0

$$f_{a=b} = \overline{a \oplus b}$$

$$f_{a>b} = a\bar{b}$$

$$f_{a<b} = \bar{a}b$$

Comparadores

- Comparador 3-bit

$$f_{a=b} = \overline{(a_2 \oplus b_2)} \cdot \overline{(a_1 \oplus b_1)} \cdot \overline{(a_0 \oplus b_0)}$$

a₂=b₂ a₁=b₁ a₀=b₀

$$f_{a>b} = a_2 \overline{b_2} +$$

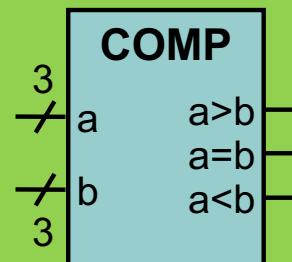
$$+ \overline{(a_2 \oplus b_2)} \cdot a_1 \overline{b_1} + \quad a_2=b_2 \text{ y } a_1 > b_1$$

$$+ \overline{(a_2 \oplus b_2)} \cdot \overline{(a_1 \oplus b_1)} \cdot a_0 \overline{b_0} \quad a_2=b_2, a_1=b_1 \text{ y } a_0 > b_0$$

$$f_{a>b} = \overline{a_2} b_2 + \quad a_2 < b_2$$

$$+ \overline{(a_2 \oplus b_2)} \cdot \overline{a_1} b_1 + \quad a_2=b_2 \text{ y } a_1 < b_1$$

$$+ \overline{(a_2 \oplus b_2)} \cdot \overline{(a_1 \oplus b_1)} \cdot \overline{a_0} b_0 \quad a_2=b_2, a_1=b_1 \text{ y } a_0 < b_0$$



- Se puede generalizar
- De este modo se reutilizan muchas puertas (XOR)



Bibliografía

- “Circuitos y Sistemas Digitales”. J. E. García Sánchez, D. G. Tomás, M. Martínez Iniesta. Ed. Tebar-Flores
- “Electrónica Digital”, L. Cuesta, E. Gil, F. Remiro, McGraw-Hill
- “Fundamentos de Sistemas Digitales”, T.L Floyd, Prentice-Hall



Circuitos combinacionales aritméticos (Parte II)

© Luis Entrena, Celia López,
Mario García, Enrique San Millán

Universidad Carlos III de Madrid



Contenidos

1. Circuitos sumadores y restadores

- Sumadores con propagación de acarreo serie
 - Semisumador. Sumador total. Sumador de n bits con acarreo serie
- Sumadores con acarreo anticipado
- Sumador/Restador en complemento a 2

2. Circuitos de multiplicación

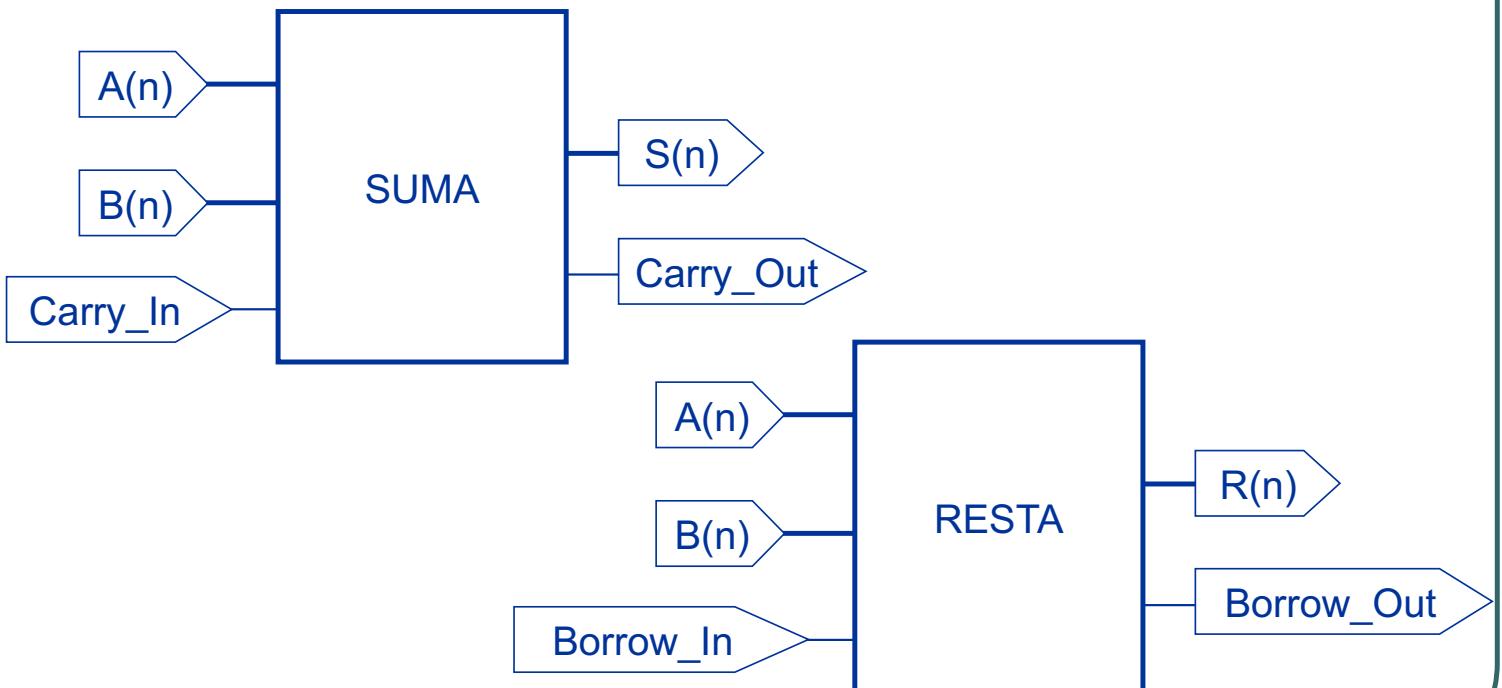
- Circuito multiplicador básico

3. Unidades Aritmético-Lógicas (ALUs)

- Concepto de ALU



Circuitos sumadores y restadores

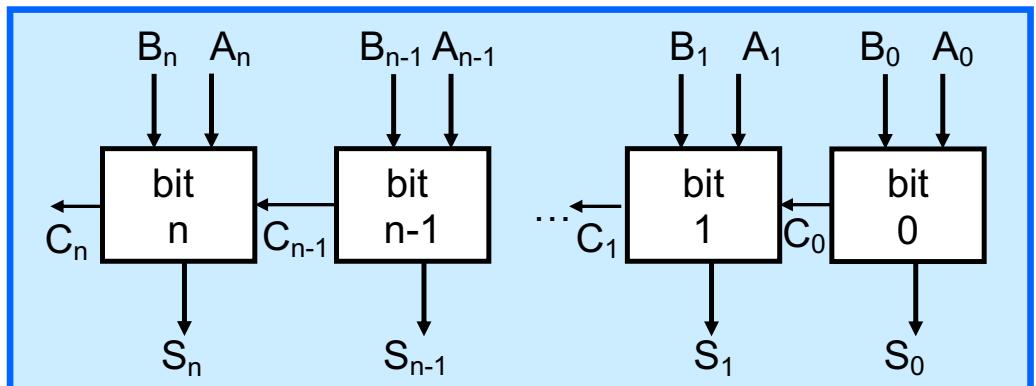


Sumador con propagación de acarreo serie.

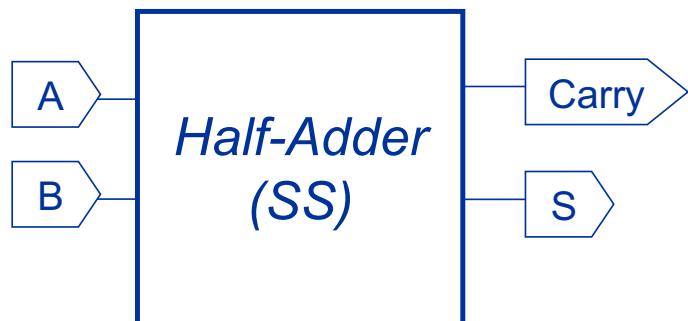
Suma decimal y binaria

$$\begin{array}{r}
 \textcolor{blue}{1} \textcolor{yellow}{1} \\
 86d \\
 + 25d \\
 \hline
 \textcolor{blue}{1} \textcolor{green}{1} \textcolor{blue}{1} d
 \end{array}
 \quad \rightarrow \quad
 \begin{array}{r}
 1010110b \\
 + 0011001b \\
 \hline
 11011111b
 \end{array}$$

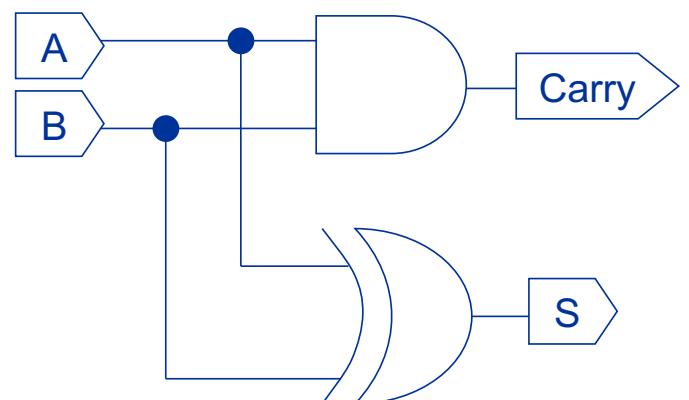
- Operandos: **n bits**
- Resultado: **n+1 bits**



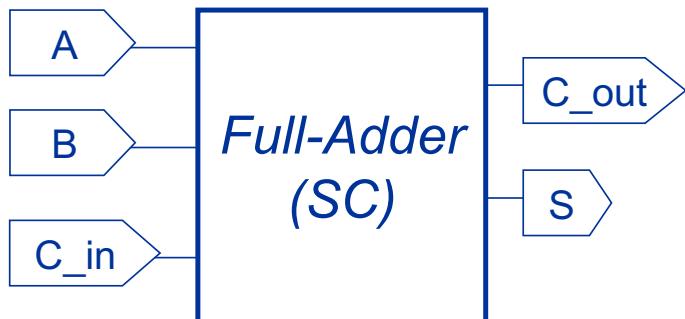
Sumador con propagación de acarreo serie. Semisumador



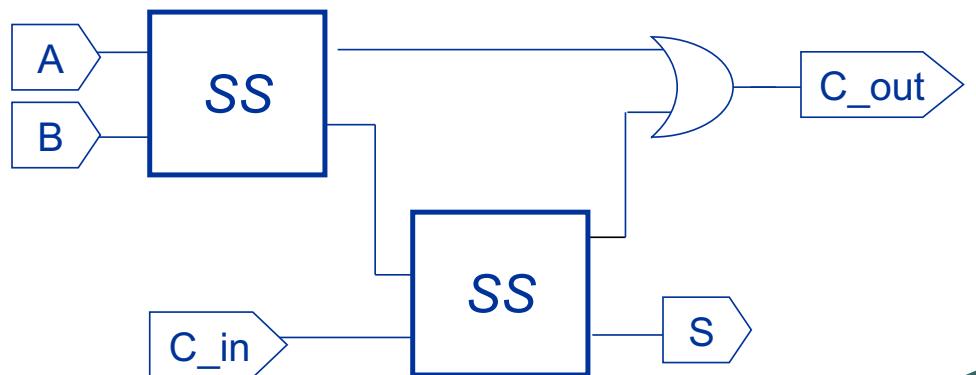
A	B	S	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



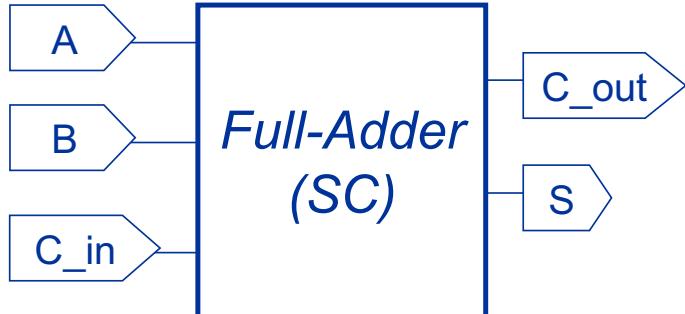
Sumador con propagación de acarreo serie. Sumador completo



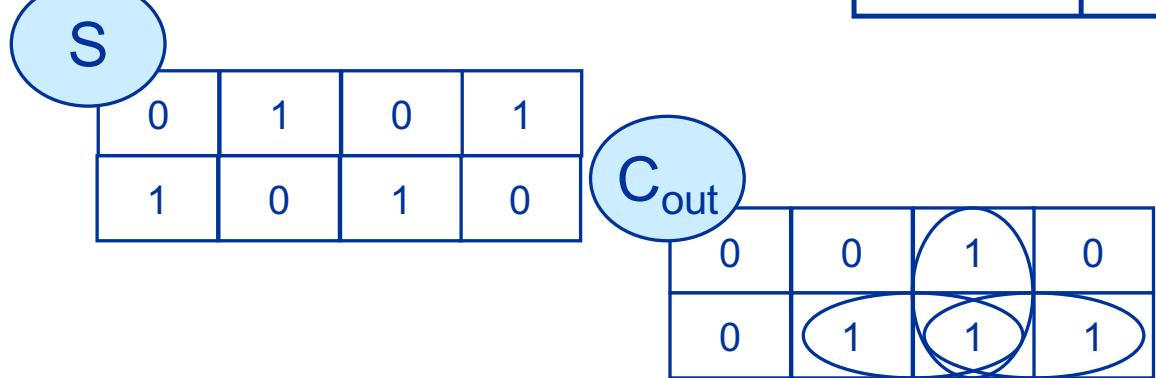
<i>A</i>	<i>B</i>	<i>C_{in}</i>	<i>S</i>	<i>C_{out}</i>
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



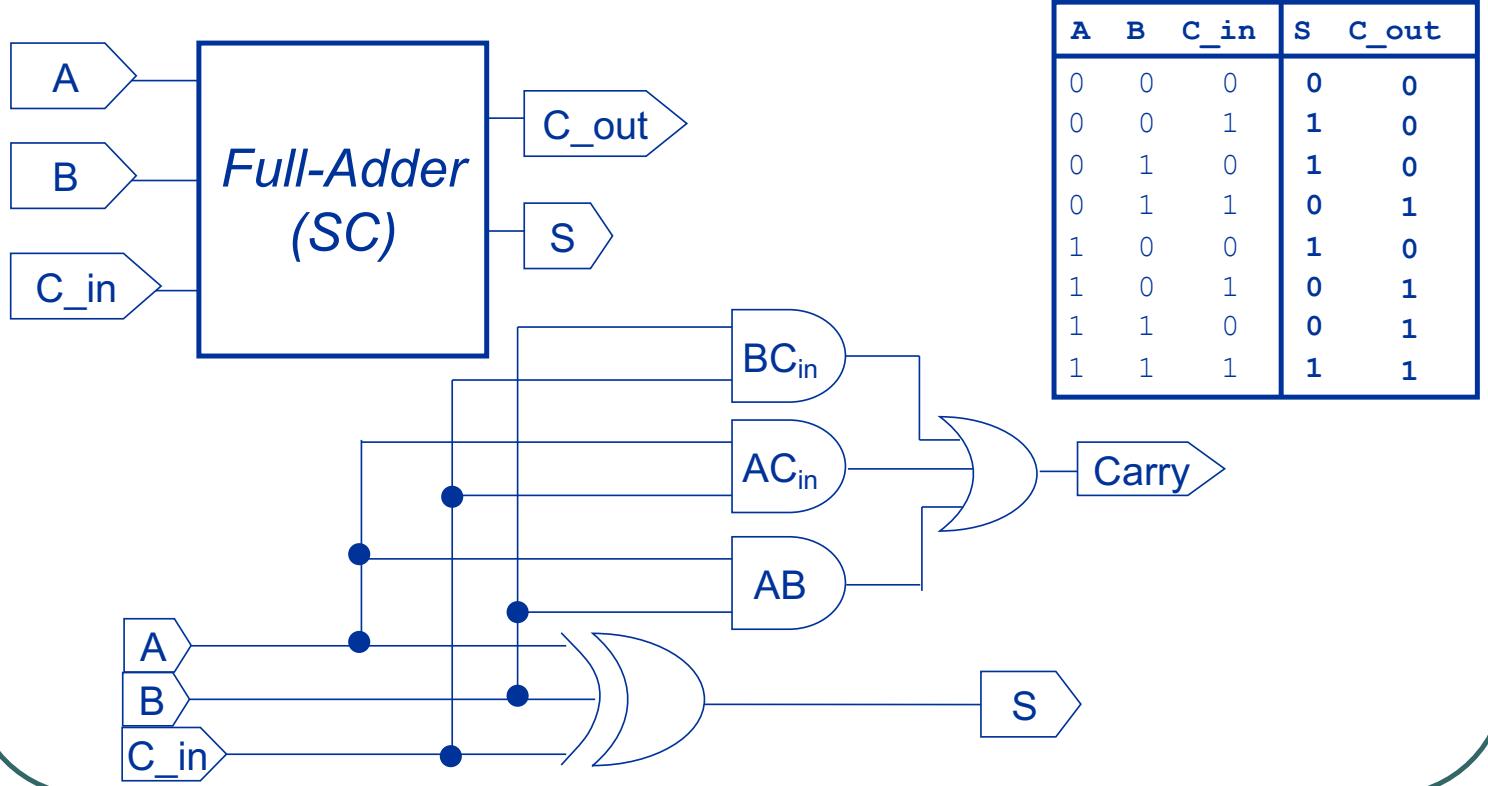
Sumador con propagación de acarreo serie. Sumador completo



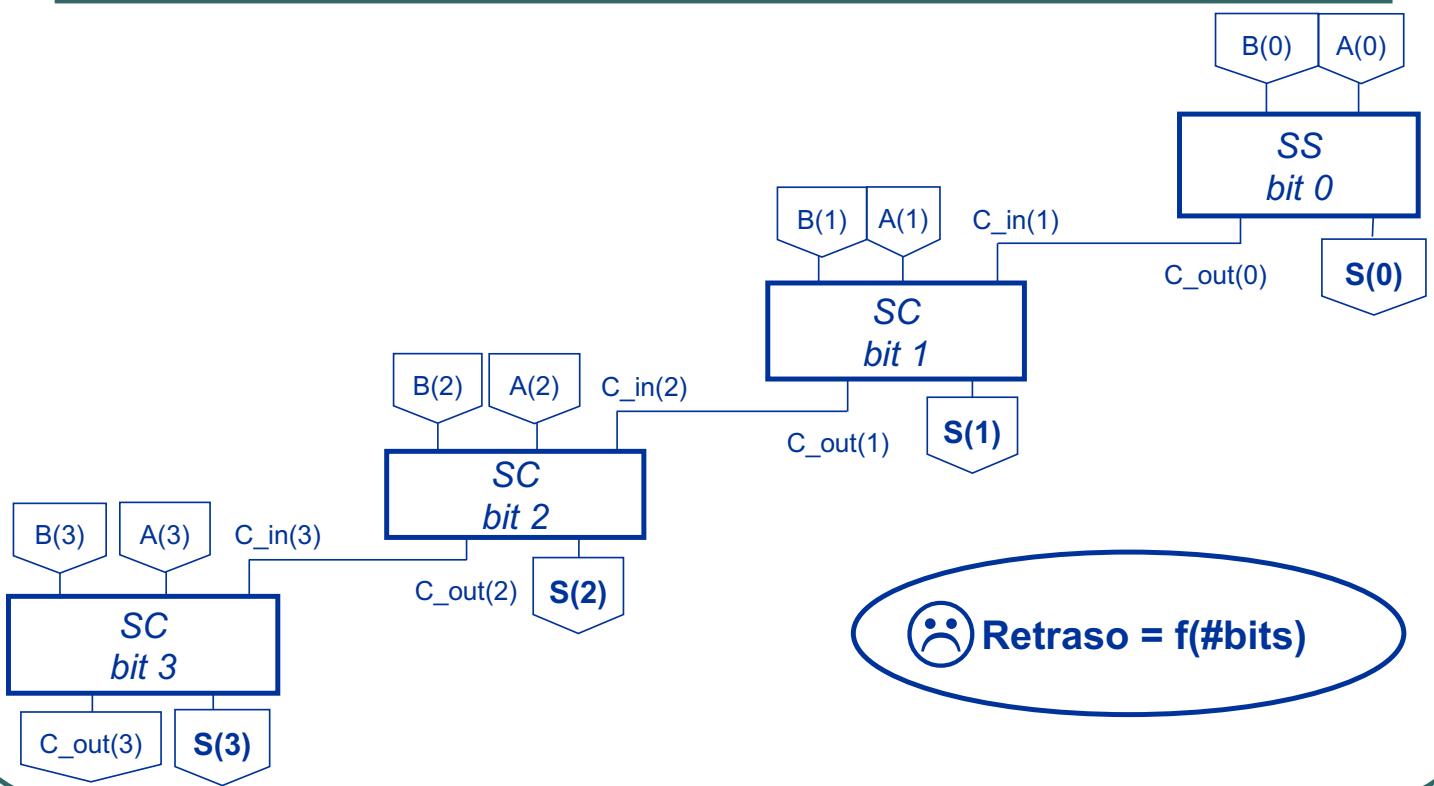
A	B	C _{in}	S	C _{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



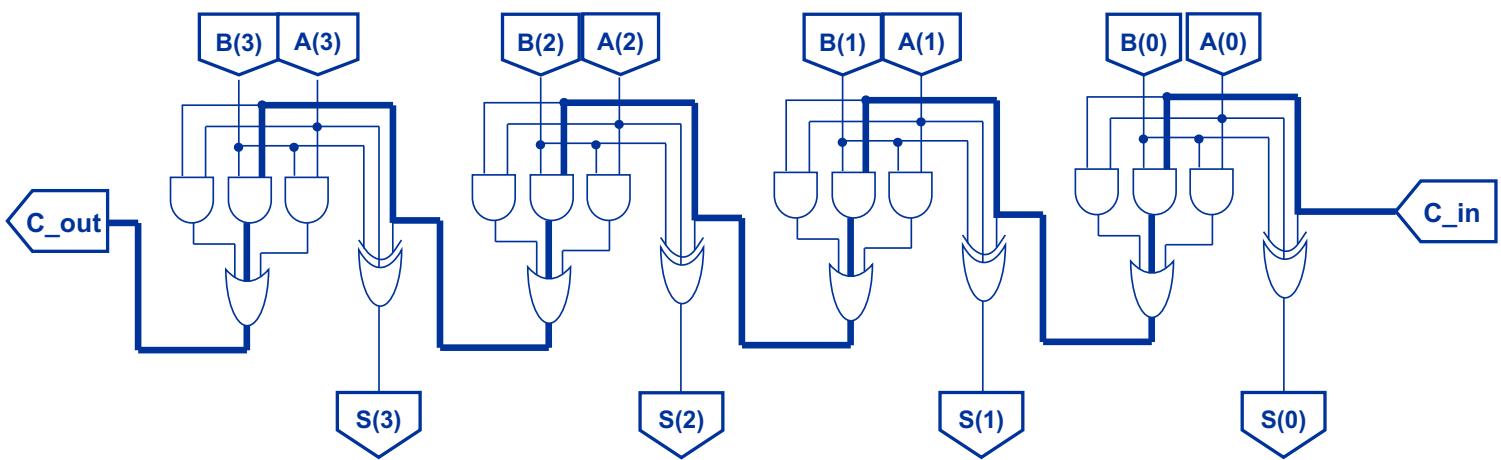
Sumador con propagación de acarreo serie. Sumador completo



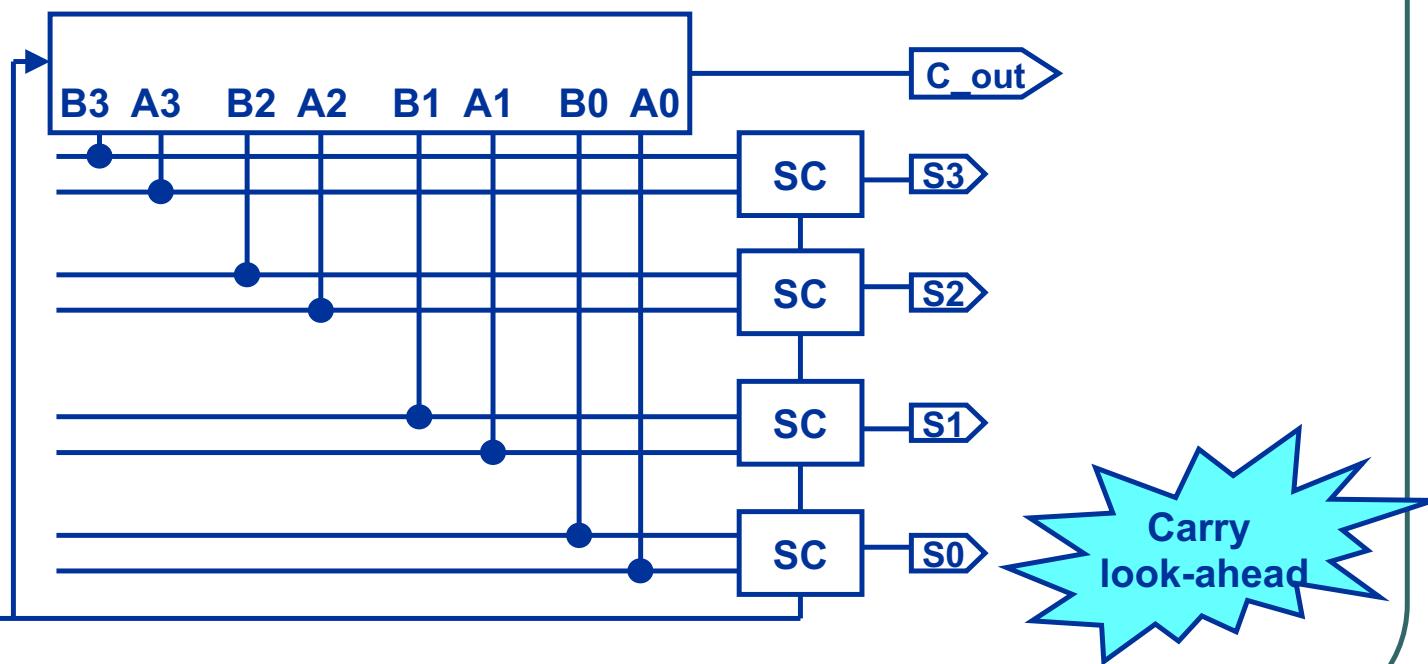
Sumador con propagación de acarreo serie. Sumador de varios bits



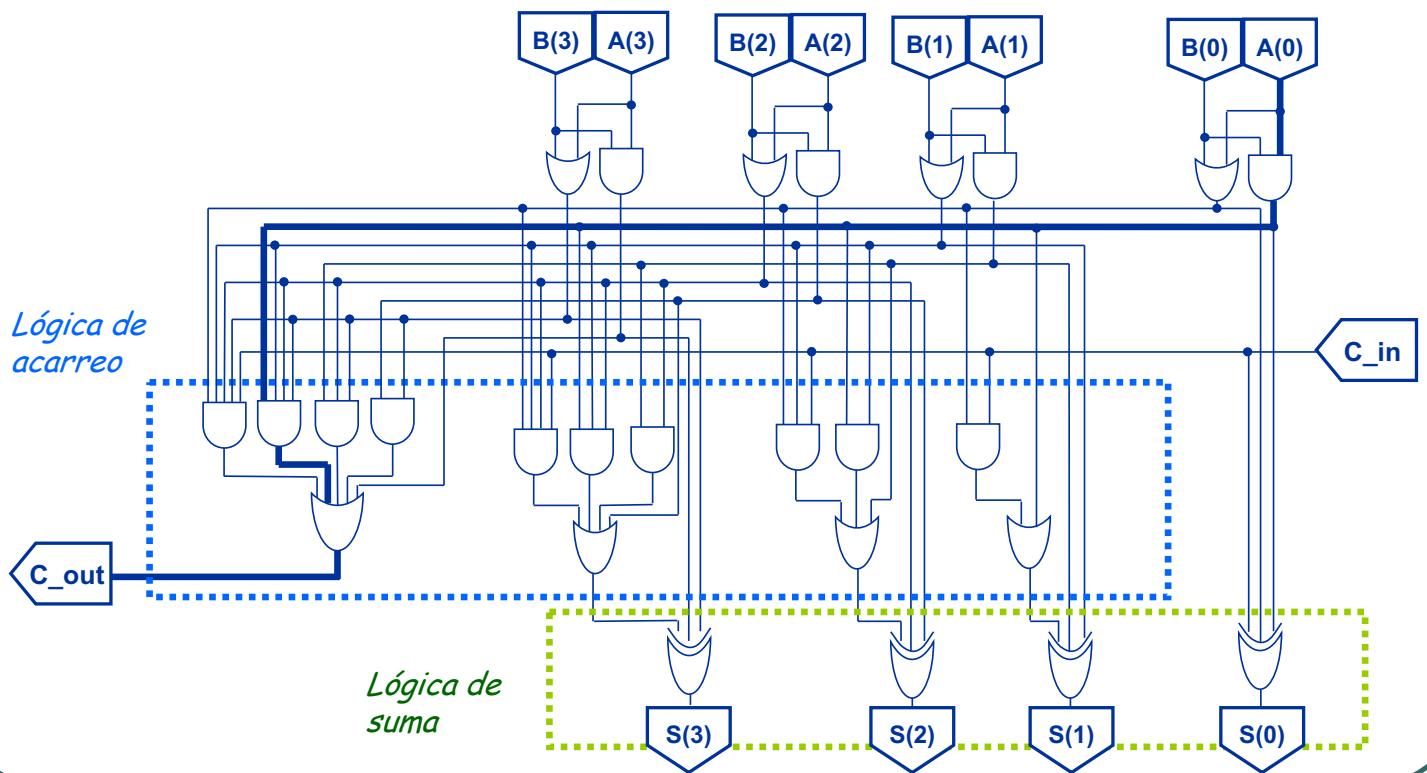
Sumador con propagación de acarreo serie. Sumador de varios bits



Sumador con acarreo anticipado.



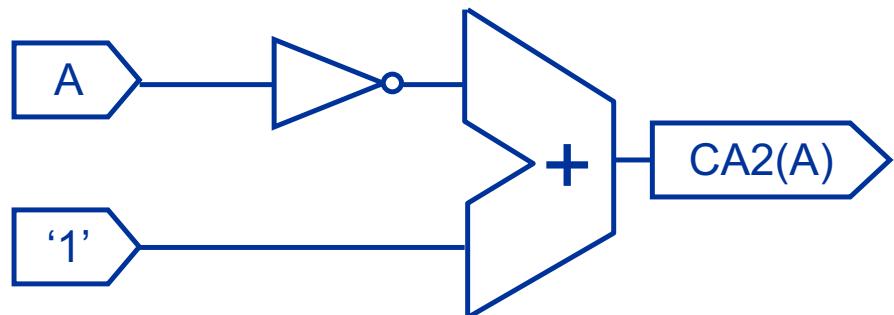
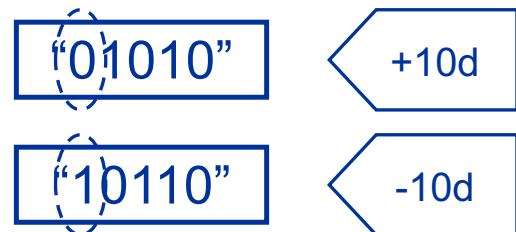
Sumador con acarreo anticipado.



Sumador/restador en CA2.

Complemento a 2

- Números positivos
- Números negativos

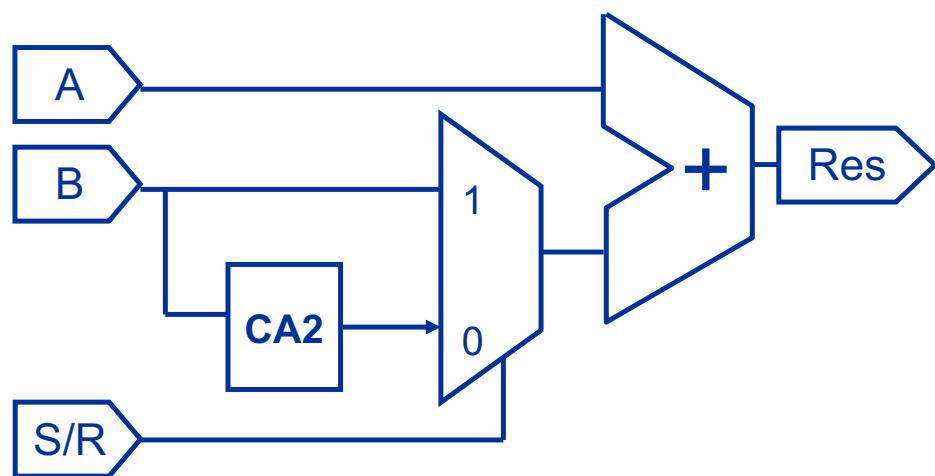




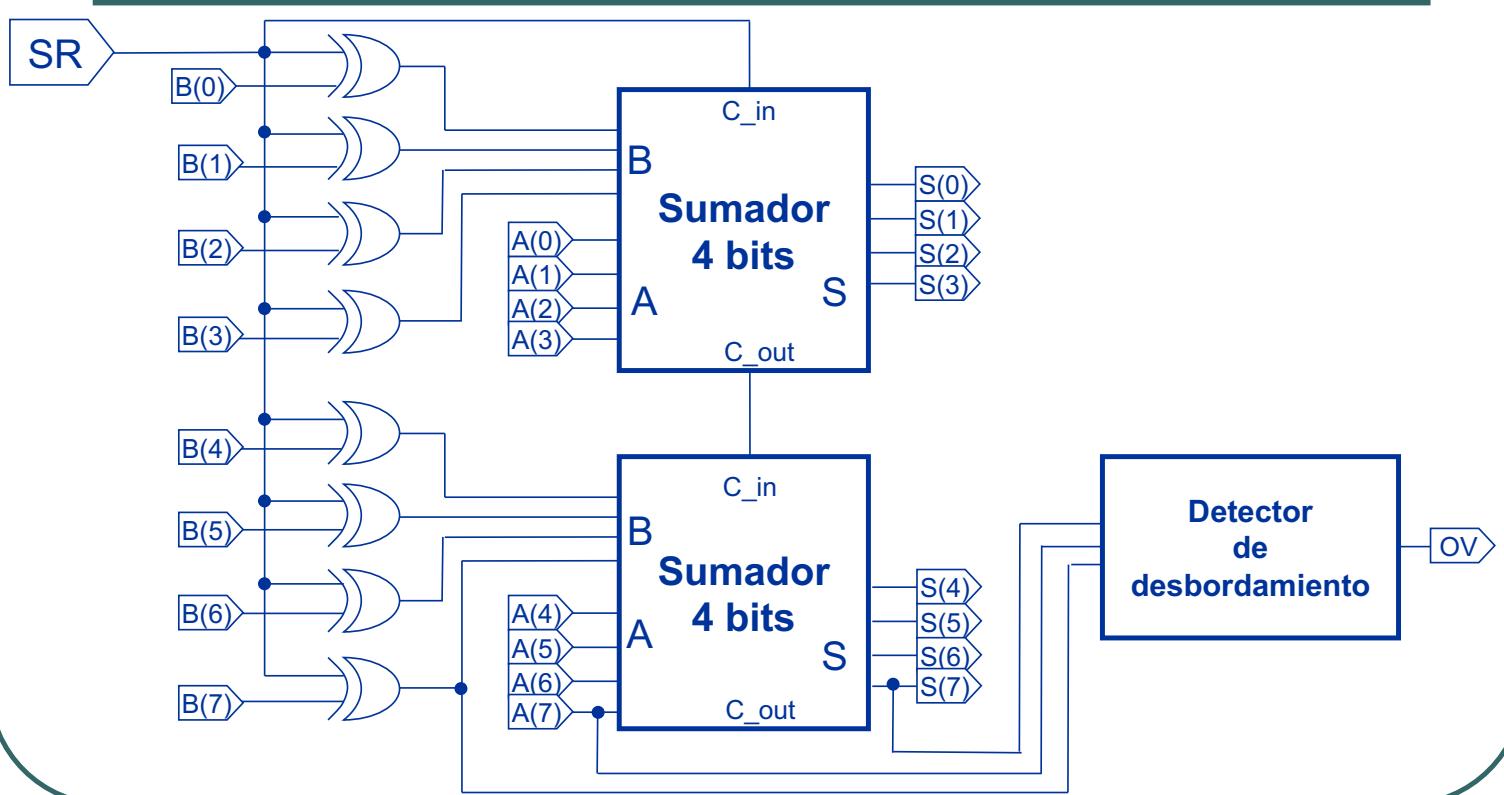
Sumador/restador en CA2.



$$A - B = A + (-B)$$



Sumador/restador en CA2.





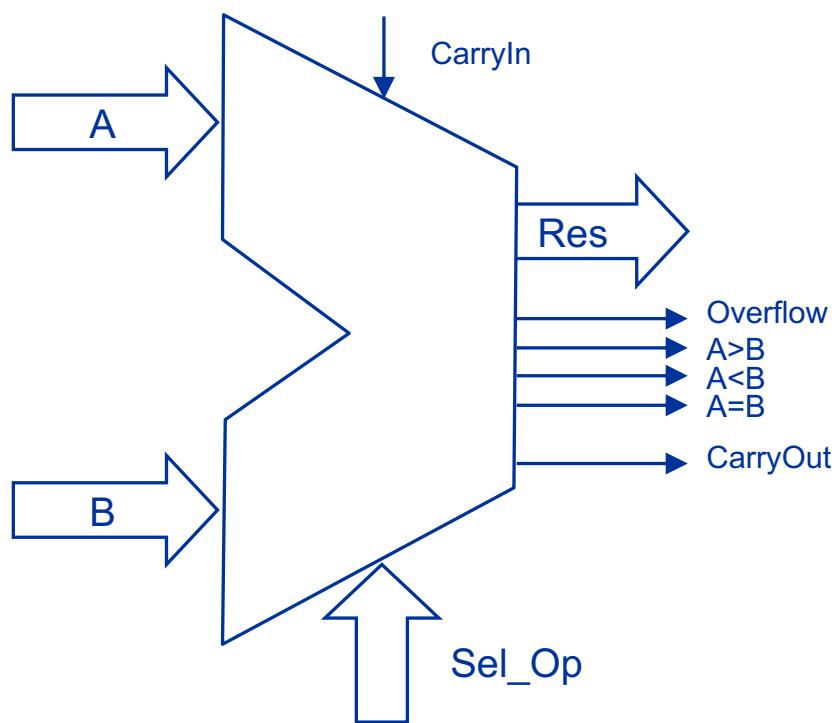
Sumador/restador en CA2.

Ejercicio





Unidad Aritmético-Lógica





Unidad Aritmético-Lógica

Combinacional

Bloque para la realización de operaciones aritmético-lógicas:

- Suma
 $A + B$
- Resta
 $A - B$
- Complemento a 2
 $- B$
- Comparación
 - $A > B$
 - $A < B$
 - $A = B$
- Desplazamiento a la izquierda
 $SHL(A) \leftarrow$
- Desplazamiento a la derecha
 $SHR (A) \rightarrow$

Operaciones lógicas (bit a bit)

- AND
- OR
- XOR
- XNOR
- NOT





Referencias

- “Circuitos y Sistemas Digitales”. J. E. García Sánchez, D. G. Tomás, M. Martínez Iniesta. Ed. Tebar-Flores
- “Electrónica Digital”, L. Cuesta, E. Gil, F. Remiro, McGraw-Hill
- “Fundamentos de Sistemas Digitales”, T.L Floyd, Prentice-Hall



Extra



Multiplicador.



Multiplicador decimal y binario

$$\begin{array}{rcl} 86d & \rightarrow & 1010110b \\ 15d & \rightarrow & 0001111b \\ \hline 1290d & \rightarrow & 10100001010b \end{array}$$

Decimal	
86d	
15d	
30	5x6
40	5x8 desplazado a izqda 1 p.
6	6x1 desplazado a izqda 1 p.
8	8x1 desplazado a izqda 2 p.
1290	

- Operando: **n bits**
- Resultado: **2*n bits**



Multiplicador.

Binario

$$A * B = A * (b_{n-1} * 2^{n-1} + b_{n-2} * 2^{n-2} + \dots + b_1 * 2^1 + b_0 * 2^0)$$

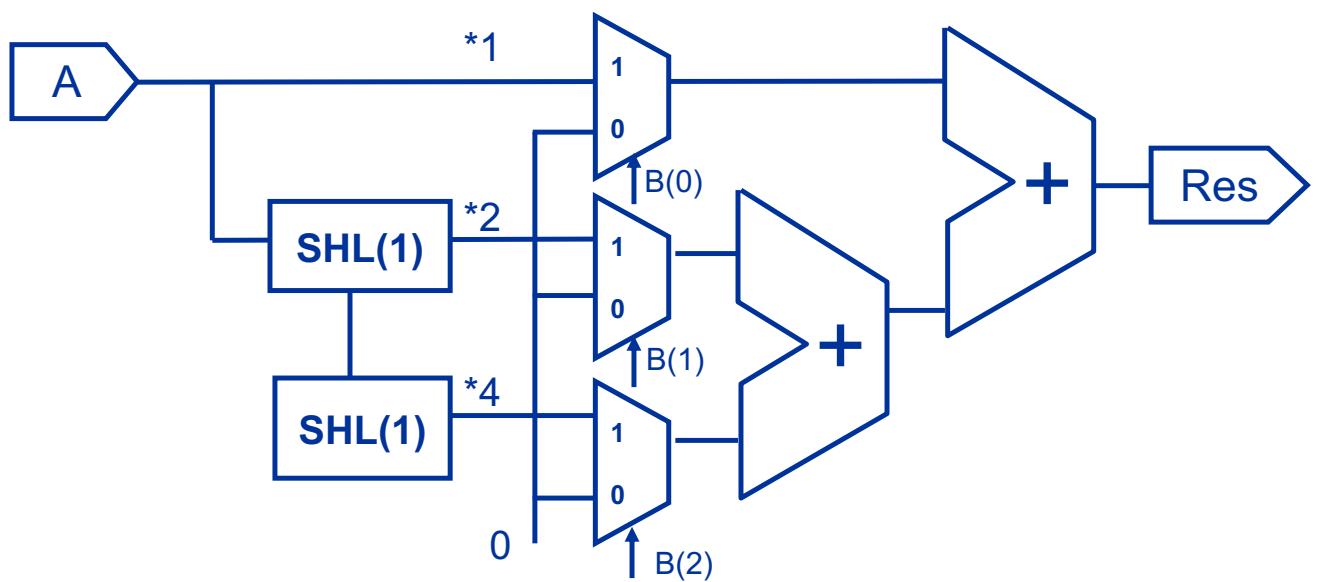
¡¡ '1s' o '0s' !!

La multiplicación binaria de dos números A (m bits) y B (n bits) consiste en una suma de tantos elementos como bits tenga B (n). Cada elemento i es el número A desplazado a la izquierda i veces si el peso correspondiente de B vale '1'. En caso contrario el elemento i es '0'.

Multiplicador.



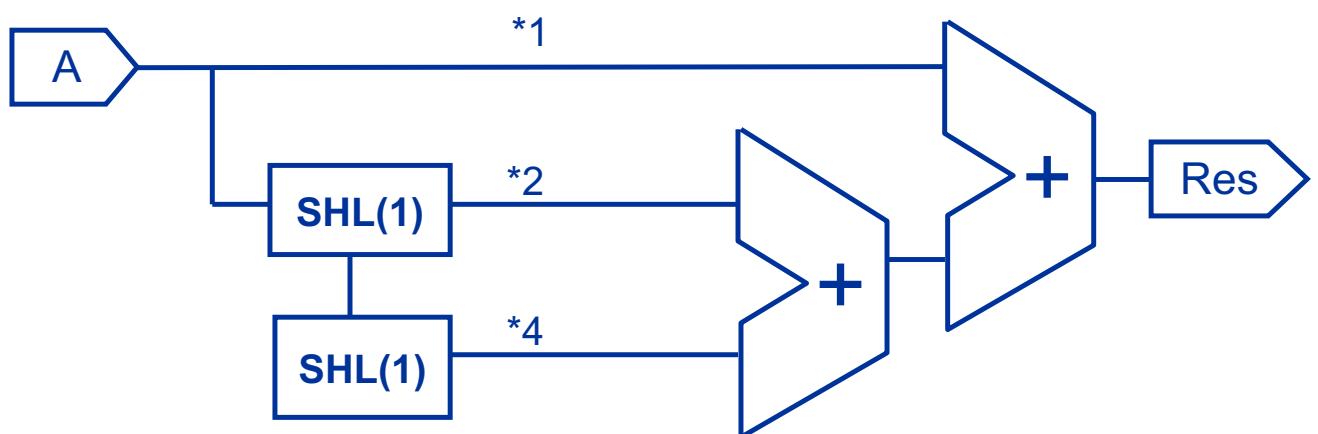
$$A * B = A * (4*B(2) + 2*B(1) + 1*B(0))$$



Multiplicador.



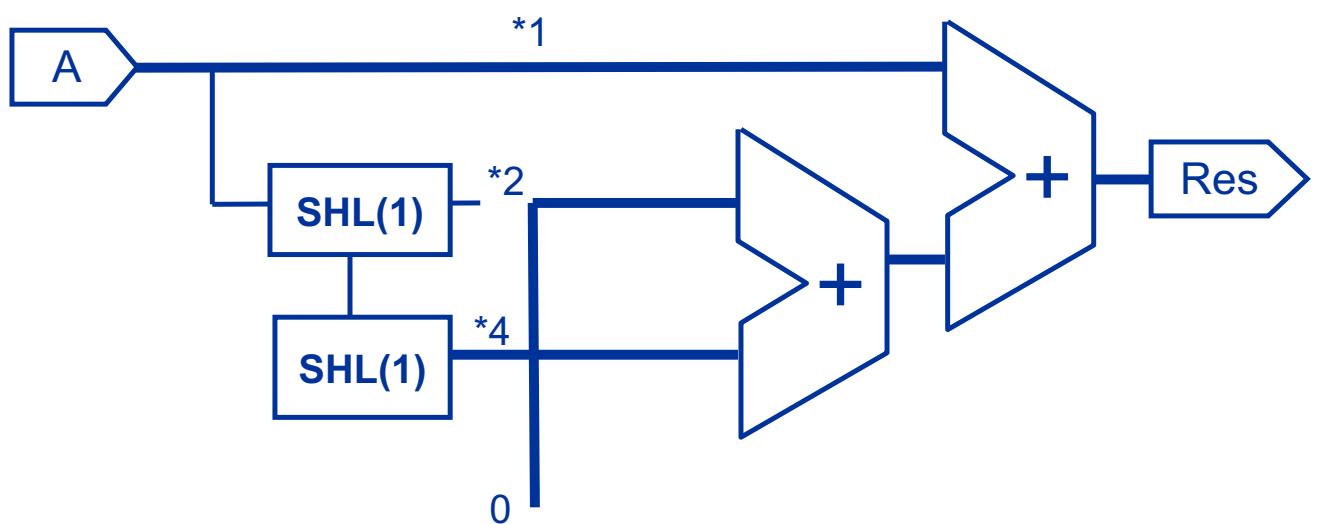
$$A * 7 = A * (4+2+1)$$



Multiplicador.

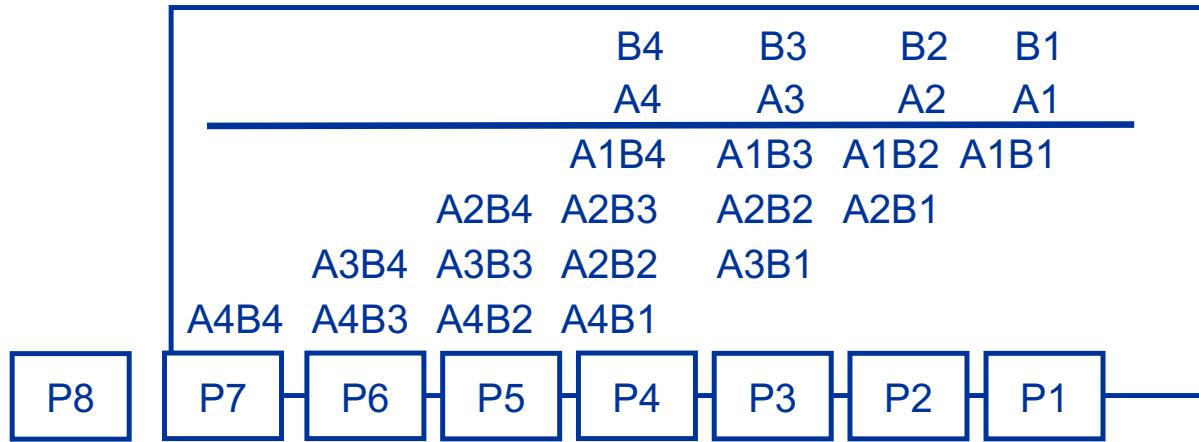


$$A * 5 = A * (4+0+1)$$



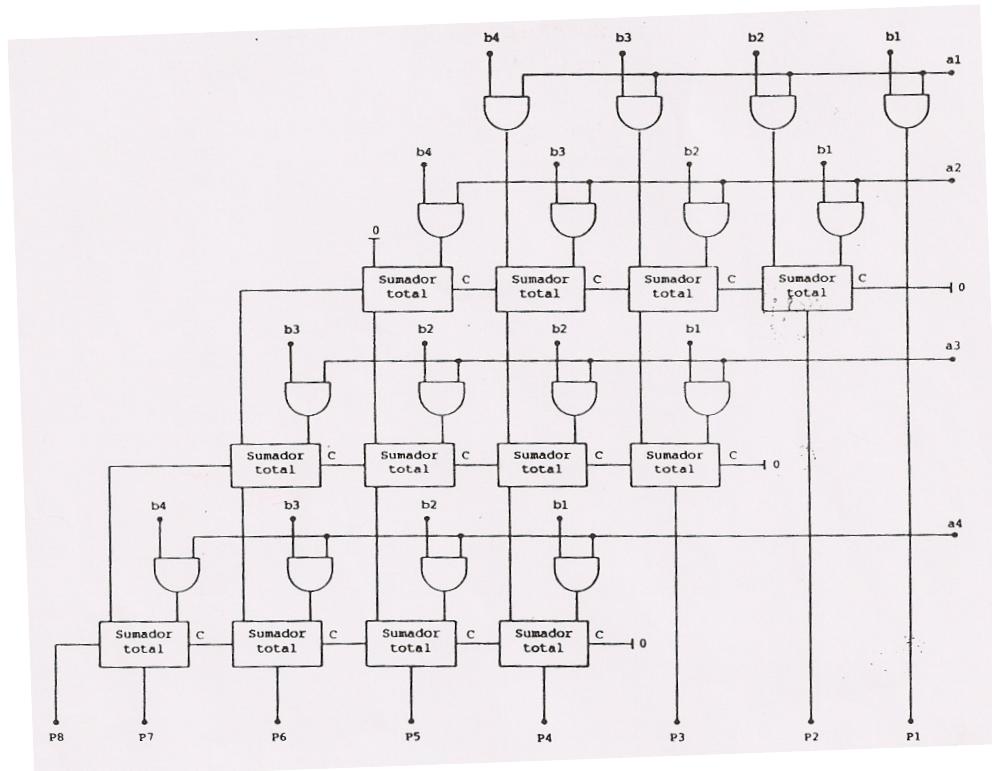


Multiplicador.





Multiplicador.





Circuitos Secuenciales Síncronos

© Luis Entrena, Celia López, Mario García,
Enrique San Millán

Universidad Carlos III de Madrid



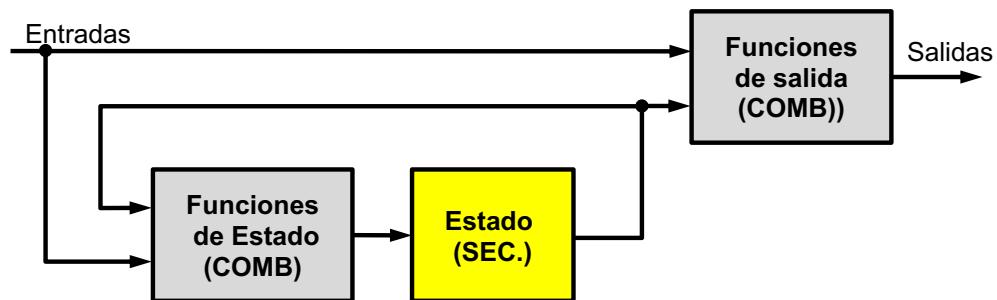
Índice

- Introducción
- Máquinas de estados finitos
 - Modelo de Moore
 - Modelo de Mealy
- Análisis de circuitos secuenciales síncronos
- Síntesis de circuitos secuenciales síncronos
- Ejemplos



Introducción

- Esquema general de un circuito secuencial síncrono:



- El bloque “Estado” está formado por biestables, todos ellos sincronizados con la misma señal de reloj



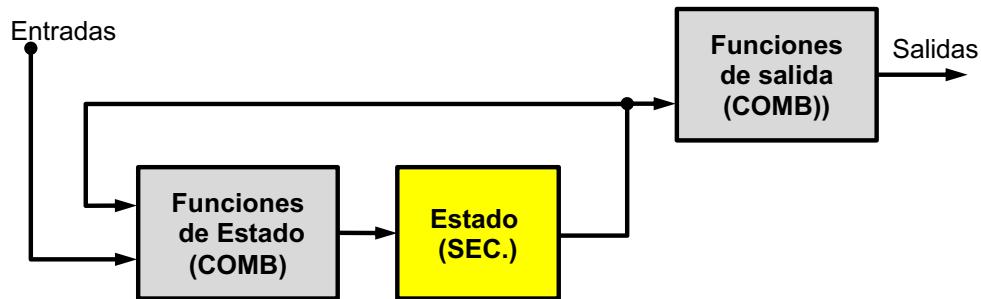
Máquinas de estados

- El comportamiento de un circuito síncrono se puede representar mediante una máquina de estados (FSM, o “Finite State Machine”)
- Una máquina de estados tiene los siguientes elementos:
 - X = Entradas
 - Y = Salidas
 - Z = Estados (valores de los biestables, cambian con cada flanco de reloj)
 - δ = Funciones de estado (funciones combinacionales de entrada de los biestables)
 - λ = Funciones de salida (combinacionales)
- Una FSM se define como una secuencia de eventos en tiempos discretos. El estado Z cambia en cada evento (el cambio está definido por δ).



Modelo de Moore

- En el modelo de Moore las salidas dependen únicamente de los estados (no de las entradas)
- Máquina de estados de Moore:
 - $Z = \delta (X, Z)$
 - $Y = \lambda (Z)$
- Estructura de un circuito asociado a un modelo de Moore:





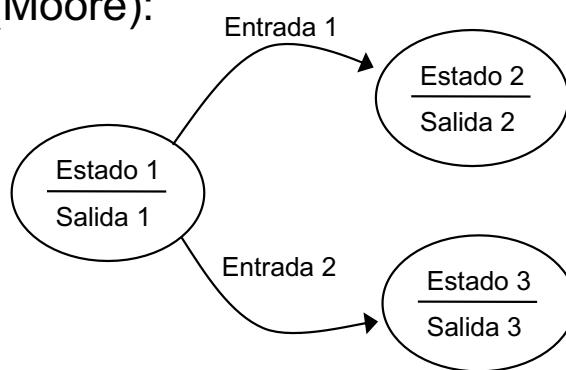
Modelo de Moore

- El reloj y el reset no aparecen en las máquinas de estados, la asociación entre estas señales en un circuito y la máquina de estados es:
 - En cada flanco de reloj se produce una transición o cambio de estado
 - El reset se utiliza únicamente para establecer el estado inicial
- En las máquinas de estados de Moore las salidas cambian únicamente si hay un cambio de estado:
 - Las salidas están sincronizadas con el reloj



Modelo de Moore

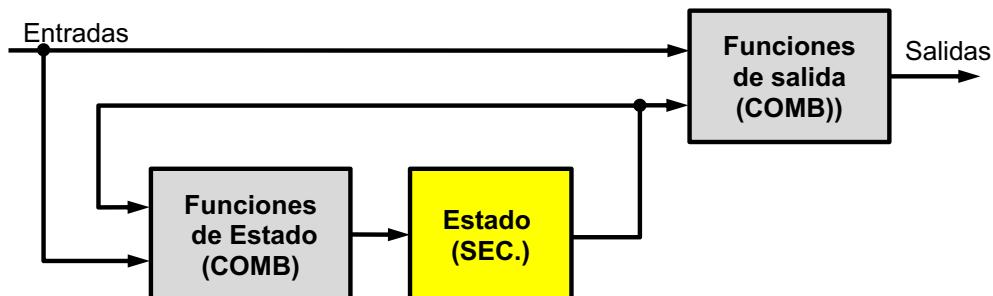
- Una FSM se puede representar también mediante un **diagrama de estados (STG o “State Transition Graph”)**:
 - Cada estado se representa con un círculo
 - Cada transición de estado se representa con una flecha
 - Los diferentes valores de las entradas se representan en las flechas
 - En el caso del modelo de Moore, las salidas se representan dentro de cada estado
- Diagrama de estados (Moore):





Modelo de Mealy

- En el modelo de Mealy las salidas dependen tanto de los estados como de las entradas (caso general)
- Máquina de estados de Mealy:
 - $Z = \delta(X, Z)$
 - $Y = \lambda(X, Z)$
- Estructura de un circuito asociado a un modelo de Mealy:

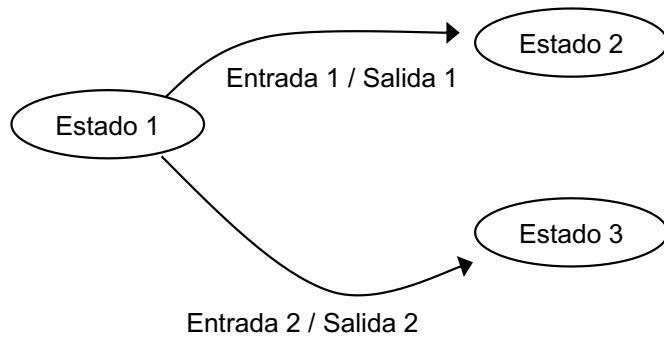




Modelo de Mealy

- **Diagrama de estados de Mealy:**

- Cada estado se representa con un círculo
- Cada transición de estado se representa con una flecha
- Los diferentes valores de las entradas se representan en las flechas
- En el caso del modelo de Mealy, las salidas se representan también en las flechas (dependen del estado y de las entradas)





Modelo de Mealy

- Igual que en Moore, el reloj y el reset no aparecen en el STG, están implícitos
- En las máquinas de estados de Mealy las salidas pueden cambiar en cualquier momento (basta con que cambie una entrada del circuito):
 - Las salidas **no** están sincronizadas con el reloj
 - NOTA: Aunque las salidas no estén sincronizadas con el reloj, el circuito sigue siendo síncrono (todos los biestables están sincronizados con el mismo reloj)



Análisis y Síntesis de Circuitos Secuenciales Síncronos

- **Análisis:** A partir de un circuito obtener su funcionalidad
 - **Circuitos Combinacionales:**
 - Obtener tablas de verdad o funciones booleanas de las salidas
 - **Circuitos Secuenciales:**
 - Obtener diagrama de estados, o funciones de estado y salidas (δ y λ)
- **Síntesis:** Dada una funcionalidad, obtener la implementación de un circuito
 - **Circuitos Combinacionales:**
 - Obtener expresiones booleanas, implementar con puertas lógicas, multiplexores, decodificadores, etc.
 - **Circuitos Secuenciales:**
 - Obtener diagrama de estados e implementar las funciones de estado y de salida (δ y λ) con puertas lógicas, multiplexores, decodificadores



Análisis de Circuitos Secuenciales Síncronos

- Análisis: Obtener tabla de transiciones, calcular δ y λ , y obtener diagrama de estados.
- Ejemplo:

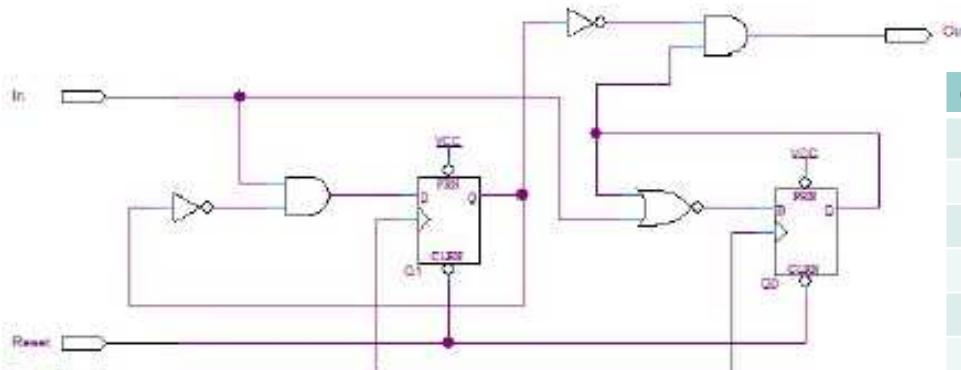


Tabla de transiciones:

Q1	Q0	In	D1	D0	Q1+	Q0+	Out
0	0	0	0	1	0	1	0
0	0	1	1	0	1	0	0
0	1	0	0	0	0	0	1
0	1	1	1	0	1	0	1
1	0	0	0	1	0	1	0
1	0	1	0	0	0	0	0
1	1	0	0	0	0	0	0
1	1	1	0	0	0	0	0

$$\delta \Rightarrow \begin{cases} D_0 = \overline{Q_0 + In} \\ D_1 = \overline{Q_1} \cdot In \end{cases}$$

$$\lambda \Rightarrow Out = \overline{Q_1} \cdot Q_0$$



Análisis de Circuitos Secuenciales Síncronos

Tabla de transiciones:

Q1	Q0	In	D1	D0	Q1+	Q0+	Out
0	0	0	0	1	0	1	0
0	0	1	1	0	1	0	0
0	1	0	0	0	0	0	1
0	1	1	1	0	1	0	1
1	0	0	0	1	0	1	0
1	0	1	0	0	0	0	0
1	1	0	0	0	0	0	0
1	1	1	0	0	0	0	0

Diagrama estados (Mealy):

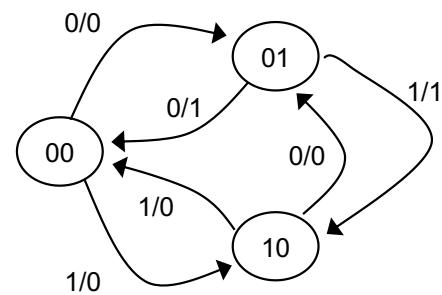
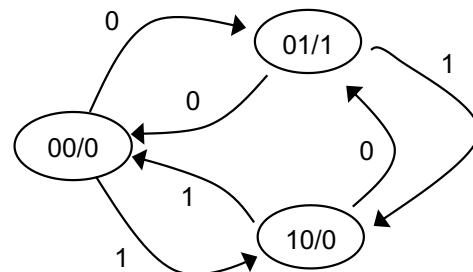


Diagrama estados (Moore):





Síntesis de Circuitos Secuenciales Síncronos

- A partir de la descripción de la funcionalidad de un circuito secuencial, los pasos a seguir para obtener la implementación son:
 1. Obtener diagrama de estados
 2. Codificación de estados
 3. Obtener Tablas de salidas y de transiciones de estados
 4. Tabla inversa de biestables (o tabla de excitación)
 5. Obtener funciones de salida
 6. Obtener funciones de estado
 7. Implementación
- La diferencia entre Moore y Mealy está en las funciones de salida



Tabla de excitación (o tablas inversas) de biestables

- Tablas inversas o tablas de excitación:
 - Describen todas las posibles combinaciones de entradas que permiten pasar del estado actual Q al estado siguiente Q+

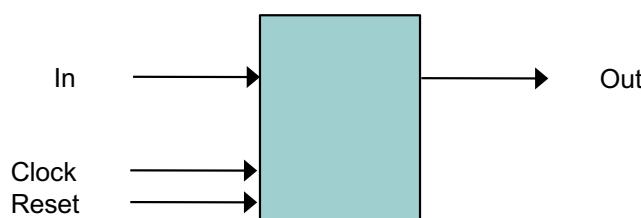
R-S latch				J-K flip-flop			
Q	Q+	S	R	Q	Q+	J	K
0	0	0	X	0	0	0	X
0	1	1	0	0	1	1	X
1	0	0	1	1	0	X	1
1	1	X	0	1	1	X	0

D flip-flop			T flip-flop		
Q	Q+	D	Q	Q+	T
0	0	0	0	0	0
0	1	1	0	1	1
1	0	0	1	0	1
1	1	1	1	1	0



Síntesis de Circuitos Secuenciales Síncronos

- Problema: Diseñar un circuito secuencial síncrono que permita detectar una secuencia de tres o más unos consecutivos a través de una entrada serie.
 - La entrada se lee en cada flanco ascendente de reloj
 - La salida se activa cuando se detecta la secuencia



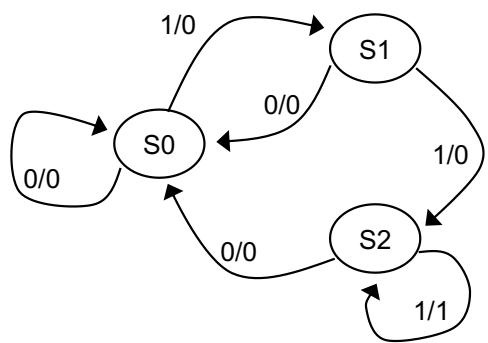
- Ejemplo de secuencia de entradas y salidas:
 - X : 0 0 1 1 0 1 1 1 1 0 0 1 1 1
 - Z : 0 0 0 0 0 0 1 1 1 0 0 0 0 1



Síntesis de Circuitos Secuenciales Síncronos

- Ejemplo 1: Mealy con biestables D:

1. Diagrama de estados:



2. Codificación de estados:

Estado	Q1	Q0
S0	0	0
S1	0	1
S2	1	1
	1	0

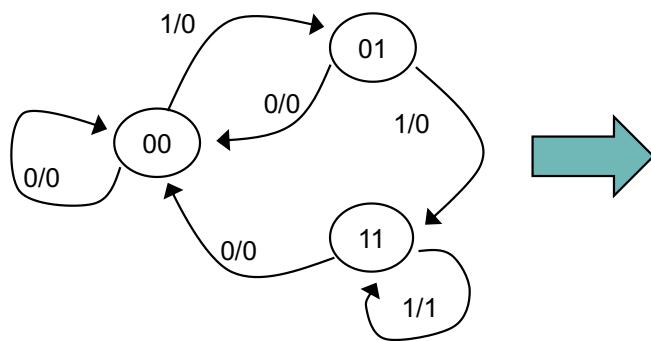
← Estado no alcanzable



Síntesis de Circuitos Secuenciales Síncronos

- Mealy con biestables D :

3. Tabla de transiciones y tabla de salidas (combinadas juntas):



In	Q1	Q0	Q1+	Q0+	Out
0	0	0	0	0	0
0	0	1	0	0	0
0	1	0	X	X	X
0	1	1	0	0	0
1	0	0	0	1	0
1	0	1	1	1	0
1	1	0	X	X	X
1	1	1	1	1	1



Síntesis de Circuitos Secuenciales Síncronos

- Mealy con biestables D :

- Tabla inversa de biestables (biestables D):

In	Q1	Q0	Q1+	Q0+	Out	D1	D0
0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0
0	1	0	X	X	X	X	X
0	1	1	0	0	0	0	0
1	0	0	0	1	0	0	1
1	0	1	1	1	0	1	1
1	1	0	X	X	X	X	X
1	1	1	1	1	1	1	1

- Función de salida:

In	Q1 Q0	00	01	11	10
0					x
1			1	x	

$$Out = Q_1 In$$

- Funciones de estado

In	Q1 Q0	00	01	11	10
0					x
1		1	1		x

$$D_1 = Q_0 In$$

In	Q1 Q0	00	01	11	10
0					x
1		1	1	1	x

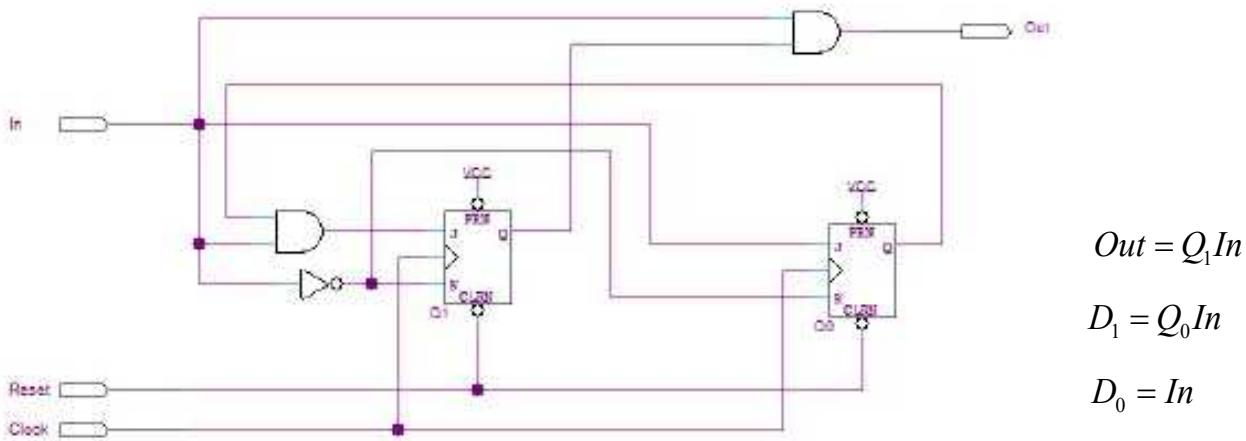
$$D_0 = In$$



Síntesis de Circuitos Secuenciales Síncronos

- Mealy con biestables D:

7. Implementación





Síntesis de Circuitos Secuenciales Síncronos

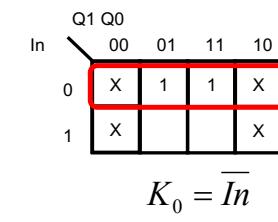
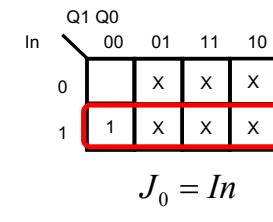
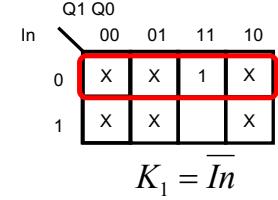
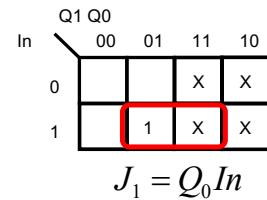
- Ejemplo 2.** Mealy con biestables J-K :

4. Tabla inversa de biestables
(biestables J-K):

In	Q1	Q0	Q1+	Q0+	Out	J1	K1	J0	K0
0	0	0	0	0	0	0	X	0	X
0	0	1	0	0	0	0	X	X	1
0	1	0	X	X	X	X	X	X	X
0	1	1	0	0	0	X	1	X	1
1	0	0	0	1	0	0	X	1	X
1	0	1	1	1	0	1	X	X	0
1	1	0	X	X	X	X	X	X	X
1	1	1	1	1	1	X	0	X	0

5. Función de salida: $Out = Q_1 In$

6. Funciones de estado

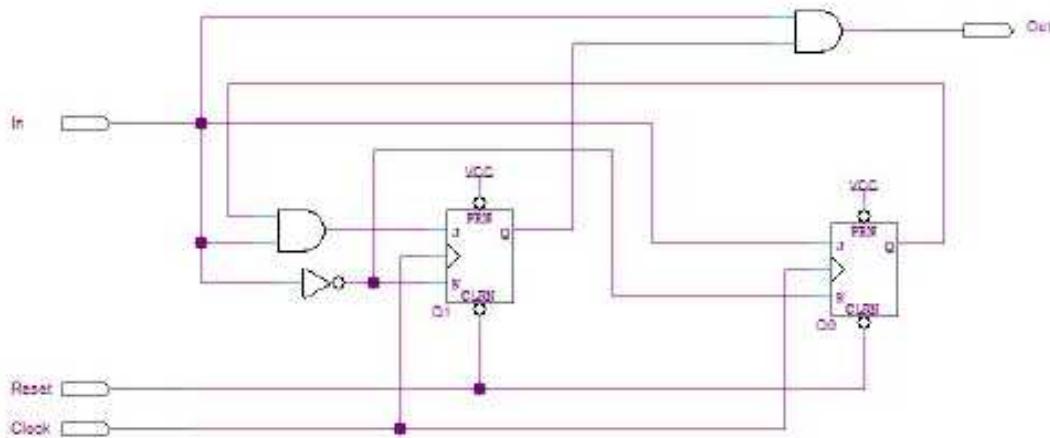




Síntesis de Circuitos Secuenciales Síncronos

- Mealy con biestables J-K:

7. Implementación



$$\begin{aligned} Out &= Q_1 In \\ J_0 &= In \\ K_0 &= \bar{In} \\ J_1 &= Q_0 In \\ K_1 &= \bar{In} \end{aligned}$$



Síntesis de Circuitos Secuenciales Síncronos

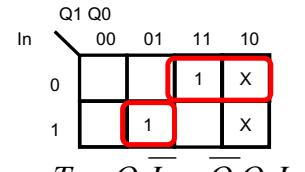
● Ejemplo 3. Mealy con biestables T :

4. Tabla inversa de biestables (biestables T):

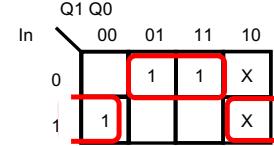
In	Q1	Q0	Q1+	Q0+	Out	T1	T0
0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	1
0	1	0	X	X	X	X	X
0	1	1	0	0	0	1	1
1	0	0	0	1	0	0	1
1	0	1	1	1	0	1	0
1	1	0	X	X	X	X	X
1	1	1	1	1	1	0	0

5. Función de salida: $Out = Q_1 In$

6. Funciones de estado



$$T_1 = Q_1 \overline{In} + \overline{Q_1} Q_0 In$$



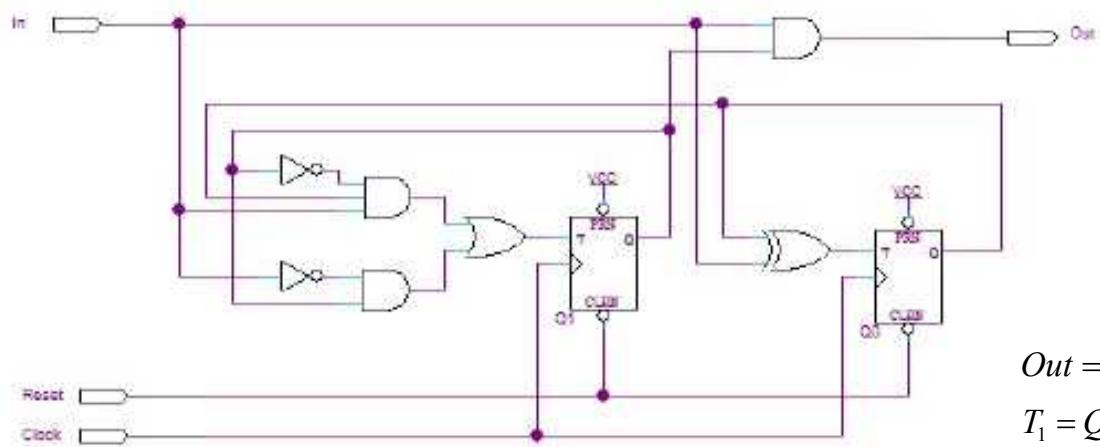
$$T_0 = \overline{In} Q_0 + In \overline{Q_0} = In \oplus Q_0$$



Síntesis de Circuitos Secuenciales Síncronos

- Mealy con biestables T:

7. Implementación



$$Out = Q_1 In$$

$$T_1 = Q_1 \bar{In} + \bar{Q}_1 Q_0 In$$

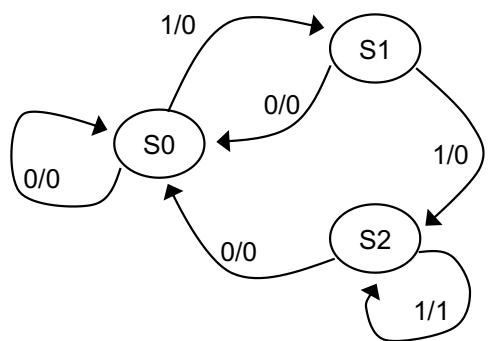
$$T_0 = \bar{In} Q_0 + In \bar{Q}_0 = In \oplus Q_0$$



Síntesis de Circuitos Secuenciales Síncronos

- Ejemplo 4: Mealy, otra codificación diferente:

1. Diagrama de estados:



2. Codificación de estados:

Estado	Q1	Q0
S0	0	0
S1	0	1
S2	1	0
	1	1



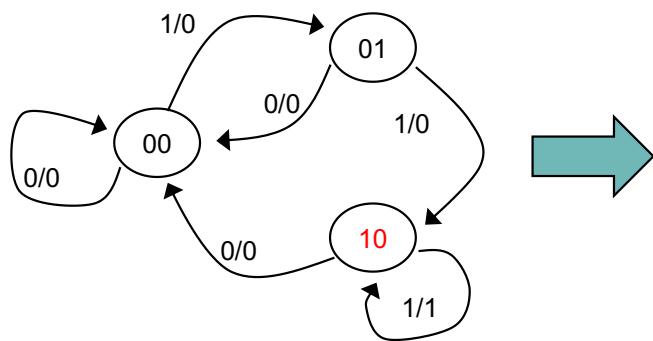
Ahora codificamos S2 de forma diferente



Síntesis de Circuitos Secuenciales Síncronos

- Mealy con biestables D (codificación diferente):

3. Tablas de transiciones y salidas (combinadas en una sola):



In	Q1	Q0	Q1+	Q0+	Out
0	0	0	0	0	0
0	0	1	0	0	0
0	1	0	0	0	0
0	1	1	X	X	X
1	0	0	0	1	0
1	0	1	1	0	0
1	1	0	1	0	1
1	1	1	X	X	X



Síntesis de Circuitos Secuenciales Síncronos

- Mealy con biestables D :

4. Tabla inversa de biestables (biestables D):

In	Q1	Q0	Q1+	Q0+	Out	D1	D0
0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0
0	1	0	0	0	0	0	0
0	1	1	X	X	X	X	X
1	0	0	0	1	0	0	1
1	0	1	1	0	0	1	0
1	1	0	1	0	1	1	0
1	1	1	X	X	X	X	X

5. Función de salida:

In	Q1 Q0	00	01	11	10
0				X	
1			X	1	

$Out = Q_1 In$

6. Funciones de estado

In	Q1 Q0	00	01	11	10
0				X	
1		1	X	1	

$$D_1 = Q_0 In + Q_1 In = In(Q_0 + Q_1)$$

In	Q1 Q0	00	01	11	10
0	1			X	

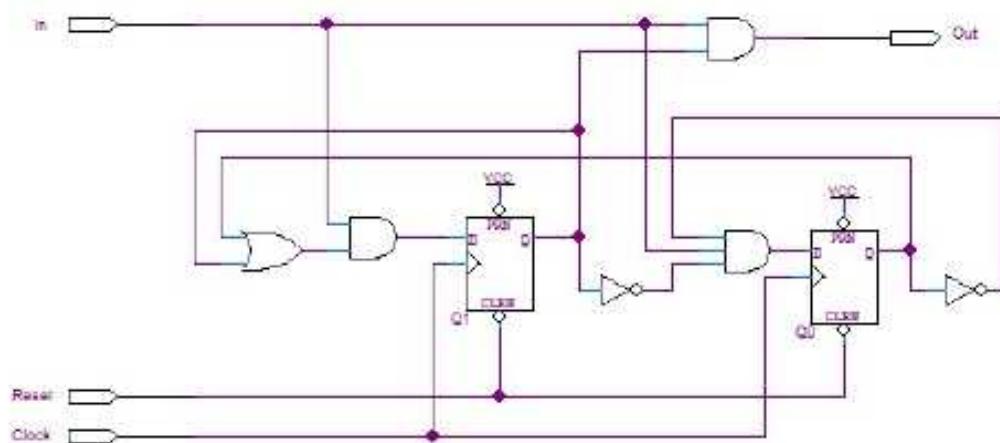
$$D_0 = \overline{Q_1} \overline{Q_0} In$$



Síntesis de Circuitos Secuenciales Síncronos

- Mealy con biestables D (codificación diferente):

7. Implementación



Con esta otra codificación sale más complejo y se requieren más puertas lógicas para la implementación

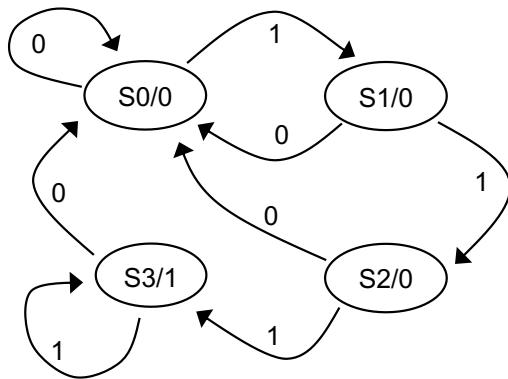
$$\begin{aligned}Out &= Q_1 In \\D_1 &= In(Q_0 + Q_1) \\D_0 &= \overline{Q_1 Q_0} In\end{aligned}$$



Síntesis de Circuitos Secuenciales Síncronos

- Ejemplo 5: Moore con biestables D:

1. Diagrama de estados:



2. Codificación de estados:

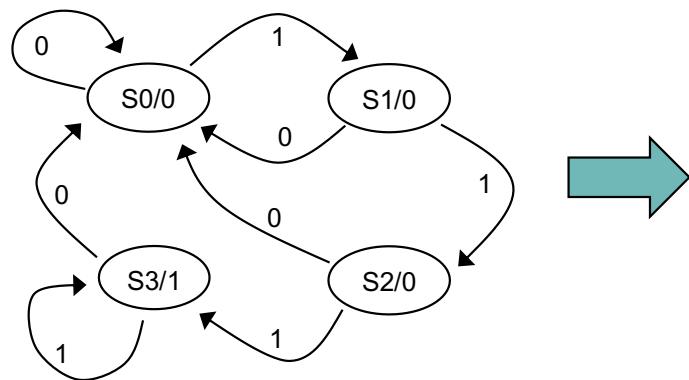
Estado	Q1	Q0
S0	0	0
S1	0	1
S2	1	1
S3	1	0



Síntesis de Circuitos Secuenciales Síncronos

- Moore con biestables D :

- Tablas de transiciones y salidas:



In	Q1	Q0	Q1+	Q0+
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	0	0
1	0	0	0	1
1	0	1	1	1
1	1	0	1	0
1	1	1	1	0

Q1	Q0	Out
0	0	0
0	1	0
1	0	0
1	1	1

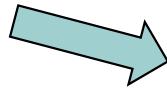


Síntesis de Circuitos Secuenciales Síncronos

- Moore con biestables D :

- Tabla inversa de biestables (biestables D):

In	Q1	Q0	Q1+	Q0+	D1	D0
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	0	0	0	0
0	1	1	0	0	0	0
1	0	0	0	1	0	1
1	0	1	1	1	1	1
1	1	0	1	0	1	0
1	1	1	1	0	1	0



- Función de salida:

Q1	Q0	Out
0	0	0
0	1	0
1	0	0
1	1	1

$$\rightarrow \text{Out} = Q_1 Q_0$$

- Funciones de estado

In	Q1 Q0	00	01	11	10
0					
1		1	1	1	

$$D_1 = Q_0 In + Q_1 In = \\ = (Q_0 + Q_1) In$$

In	Q1 Q0	00	01	11	10
0					
1		1	1		

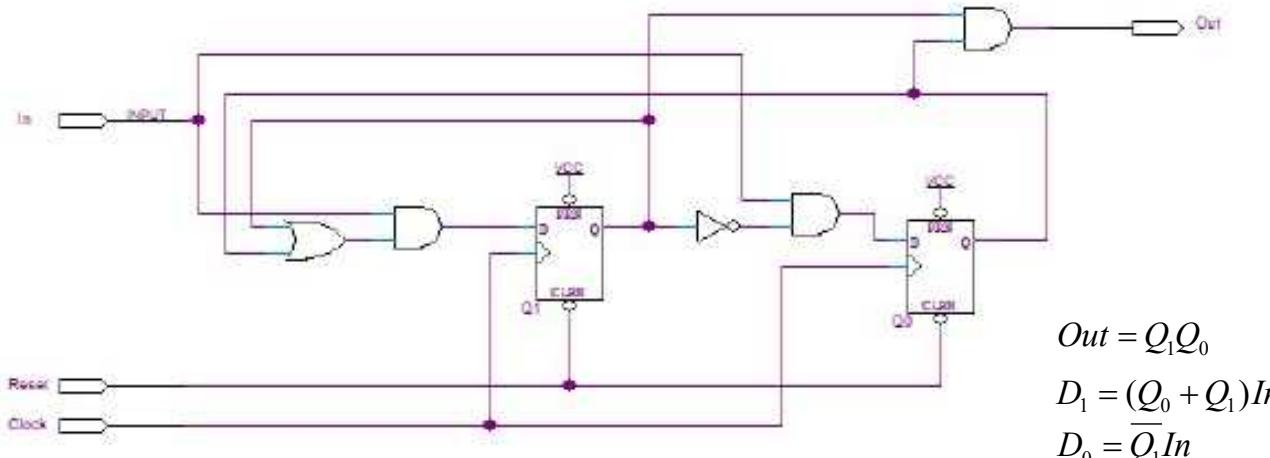
$$D_0 = \overline{Q}_1 In$$



Síntesis de Circuitos Secuenciales Síncronos

- Moore con biestables D:

7. Implementación





Bibliografía

- “Circuitos y Sistemas Digitales”. J. E. García Sánchez, D. G. Tomás, M. Martínez Iniesta. Ed. Tebar-Flores
- “Electrónica Digital”, L. Cuesta, E. Gil, F. Remiro, McGraw-Hill
- “Fundamentos de Sistemas Digitales “, T.L Floyd, Prentice-Hall



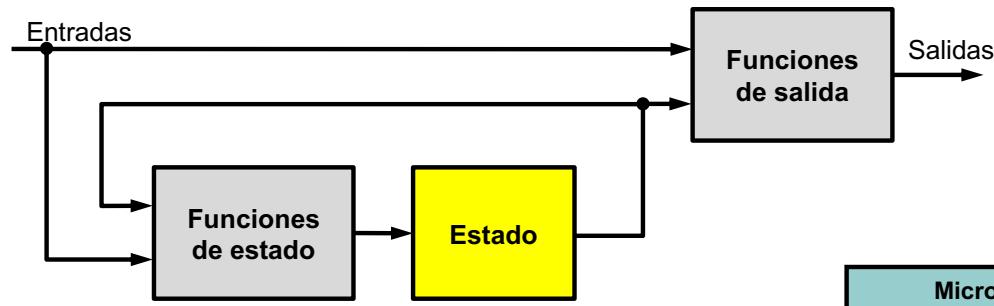
Biestables

© Luis Entrena, Celia López,
Mario García, Enrique San Millán

Universidad Carlos III de Madrid

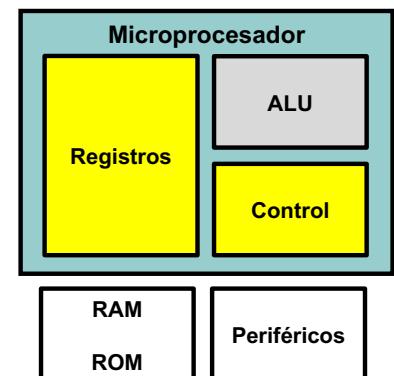


Circuitos digitales y microprocesadores



Combinacional

Secuencial





Índice

- Introducción
 - El biestable como elemento básico de memoria
 - Tipos de biestables
- Biestables síncronos
- Biestables síncronos con entradas asíncronas
- Lógicas de control de biestables
- Características temporales
- Circuitos síncronos
- Circuitos con biestables: cronogramas

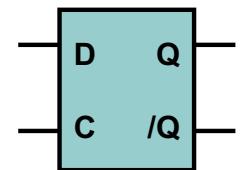
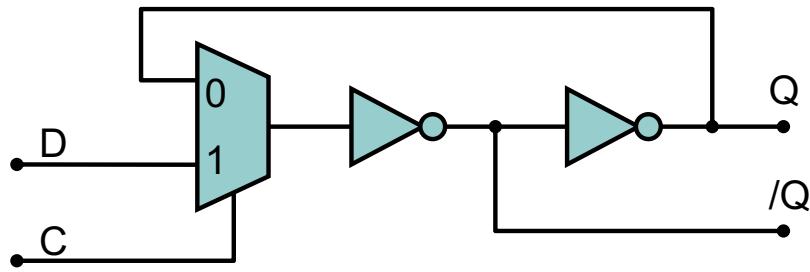


1. Introducción: biestables

- Definición:
 - Circuito capaz de almacenar un bit de información. Tiene dos estados estables, 0 y 1 lógicos. Dicho estado se mantiene hasta que sus señales de control indiquen un cambio
- Clasificación
 - Lógica de control: entradas que determinan el nuevo estado
 - D, T, SR, JK
 - Sincronismo:
 - Asíncronos: pueden cambiar al cambiar cualquier entrada
 - Síncronos: tienen una señal de control que indica cuándo pueden cambiar de valor
 - Activos por nivel
 - Activos por flanco

2. Biestables síncronos: activos por nivel

- Tiene una señal de control que permite que el biestable cambie de estado
- Biestable D síncrono activo por nivel (**latch-D**)
 - $C=1$ => el biestable toma el valor de la entrada D
 - $C=0$ => el biestable mantiene su valor

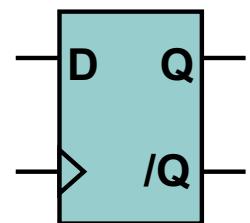
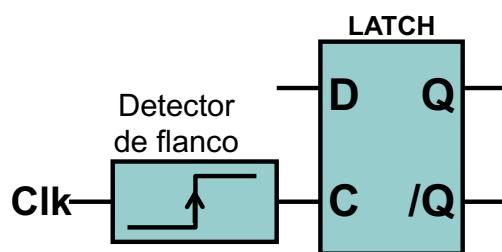


C	D	Q	/Q
0	X	Q	/Q
1	0	0	1
1	1	1	0

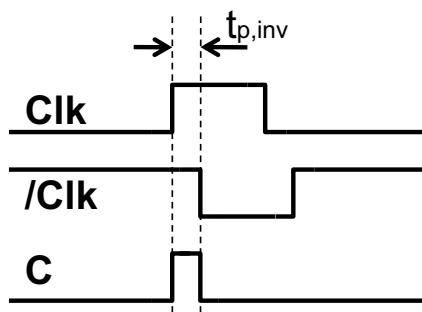
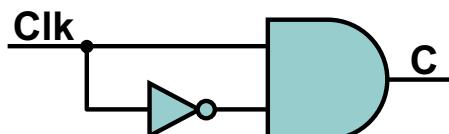
→ Mantener estado
→ Asignar '0'
→ Asignar '1'

Biestables síncronos: activos por flanco

- Biestable D síncrono, activo por flanco



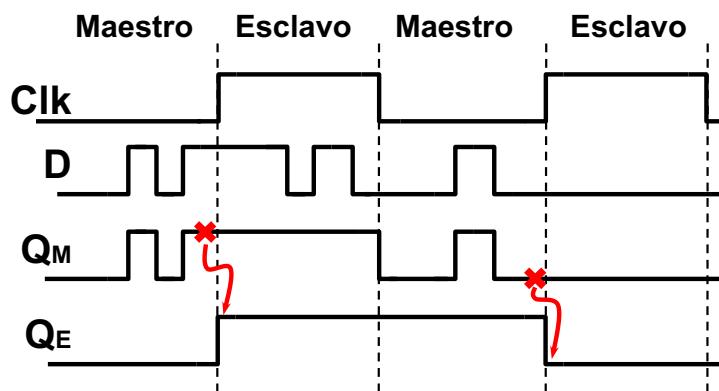
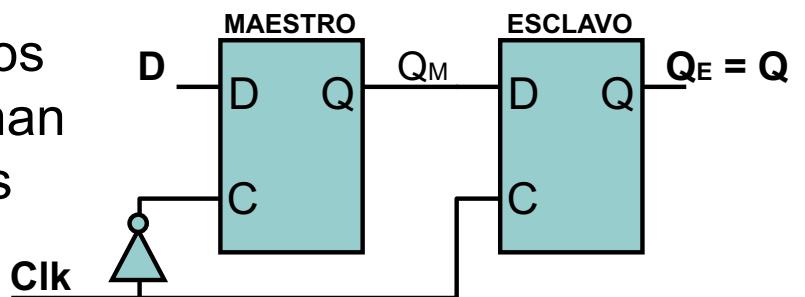
Detector de flanco



Solución mala
debido a la
tecnología: el
retraso del inversor
no es controlable

Biestables síncronos: maestro esclavo

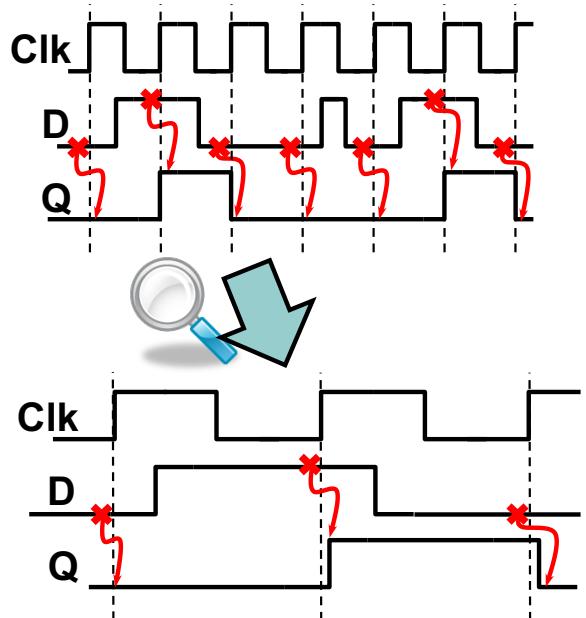
- Dos biestables activos por nivel que funcionan con niveles opuestos



- La salida Q_E sólo cambia en los flancos de subida del reloj
- La salida toma el valor de D justo antes del flanco

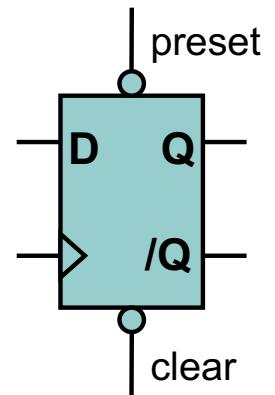
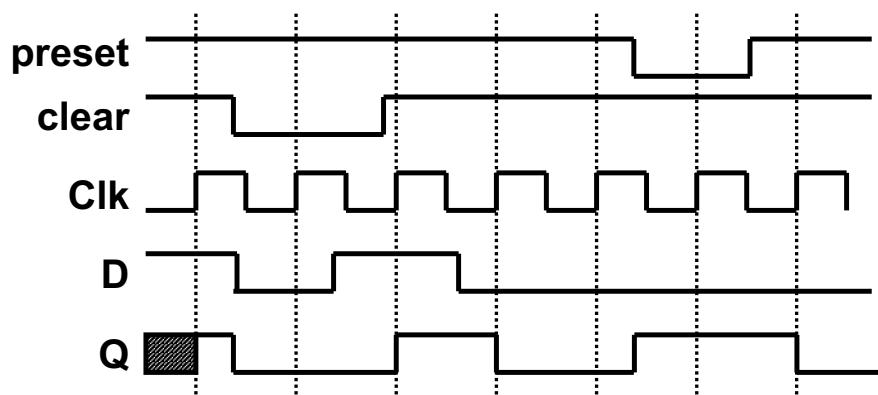
Biestable D síncrono activo por flanco

- Es el que más se usa para diseñar
- Sólo cambia de valor en los flancos de reloj (normalmente en el flanco de subida)
- El cambio a la salida del biestable se produce después del flanco de reloj
- El valor del biestable tras el flanco es el valor de su entrada D justo antes del flanco



3. Biestables síncronos con entradas asíncronas

- Biestables síncronos, que disponen de señales asíncronas para su inicialización
 - Clear: inicialización a '0' asíncrona
 - Preset: inicialización a '1' asíncrona
 - Normalmente activas por nivel bajo





4. Lógicas de control de biestables

- Señales que permiten controlar el cambio de estado del biestable
 - Tipos de biestable:
 - D,T,JK,SR
 - Señal de habilitación:
 - Habilita el cambio de estado.
 - Si no se habilita, el estado se mantiene.
 - Inicialización síncrona
 - Puesta a '0' y/o puesta a '1' atendiendo a la señal de reloj



Lógicas de control de biestables

- Tabla de funcionamiento
 - Describe funcionalidad

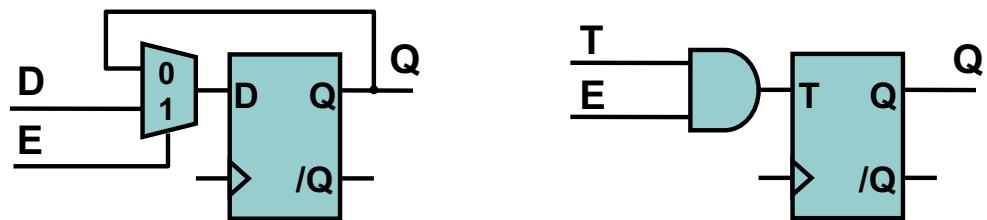
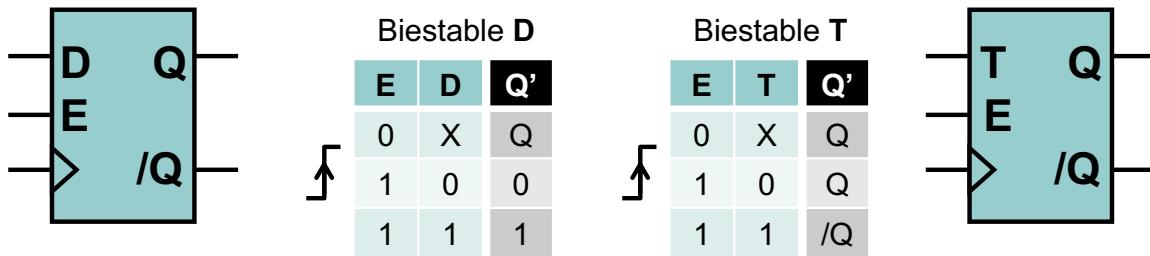
Biestable D (Data)		Biestable T (Toggle)	
D	Q'	T	Q'
0	0	0	Q
1	1	1	/Q

- Tabla de transiciones
 - Describe el próximo estado en función del estado actual y las entradas

Biestable D			Biestable T		
D	Q	Q'	T	Q	Q'
0	0	0	0	0	0
0	1	0	0	1	1
1	0	1	1	0	1
1	1	1	1	1	0

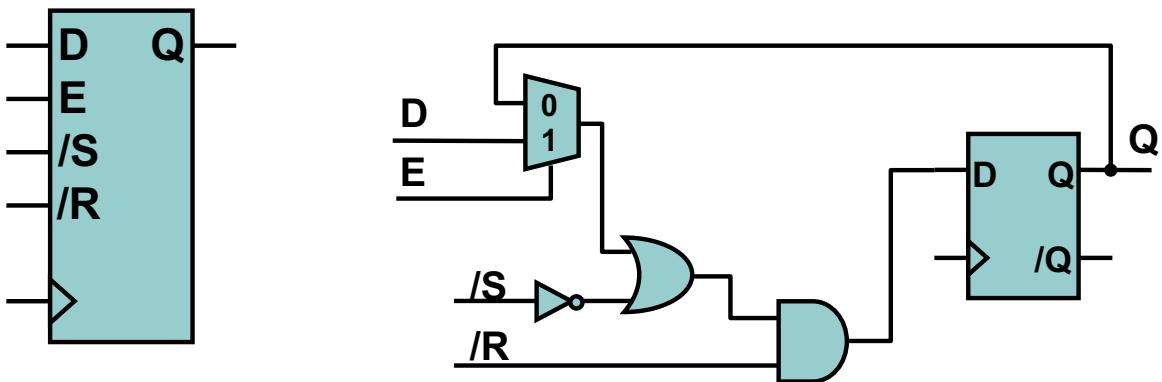
Lógicas de control de biestables

- Biestables con señal de habilitación



Lógicas de control de biestables

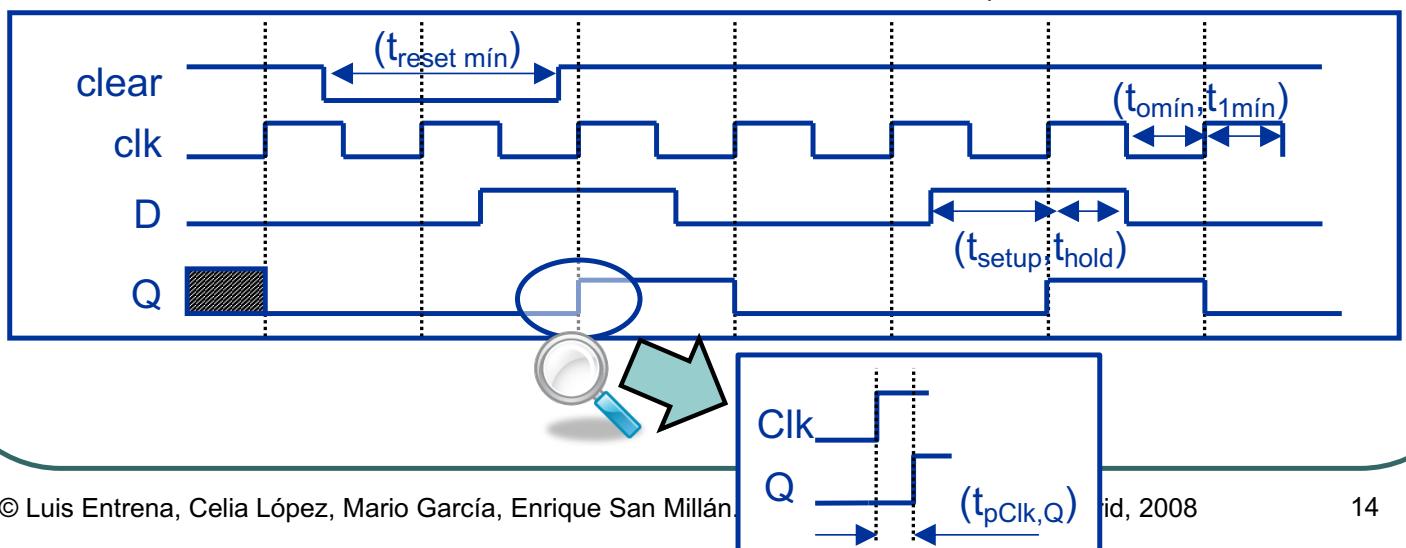
- Biestables con inicialización síncrona
 - Set: inicializa a '1'
 - Reset: inicializa a '0'
- Ejemplo: biestable D con habilitación, Set y Reset
 - Orden de prioridad: Reset, Set, Enable



5. Características temporales

- Restricciones de los biestables

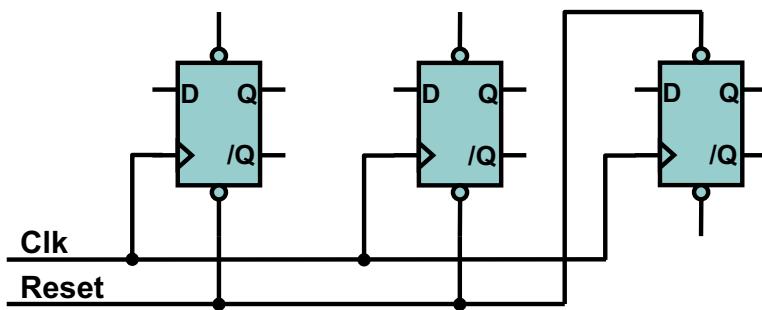
- Duración de los niveles de la señal de reloj → $(t_{0\text{mín}}, t_{1\text{mín}})$
- Duración de las señales de inicialización asíncrona → $(t_{\text{reset m\'ın}})$
- Tiempo de inserción de señales de datos → $(t_{\text{setup}}, t_{\text{hold}})$
- Tiempo de propagación de la salida → $(t_{p\text{Clk}, Q})$





6. Circuitos síncronos

- Circuito síncrono
 - Todos sus biestables usan la misma señal de reloj
 - Los biestables son activos por el mismo flanco de reloj (normalmente el de subida)
 - Los biestables usan una señal común de inicialización llamada *Reset*



Circuitos síncronos: el ciclo de reloj

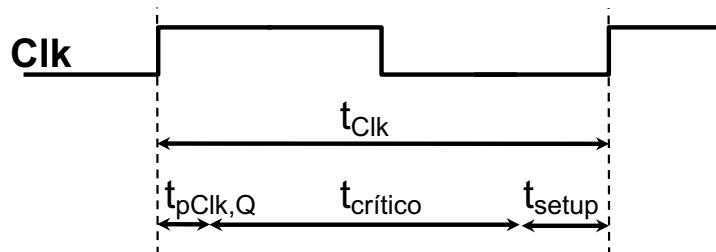
- Camino crítico:

- Camino entre dos biestables cuyo retraso es el mayor de todo el circuito
- Camino más lento entre dos biestables, que determina la máxima frecuencia de reloj a la que el circuito puede funcionar

$$t_{Clk} > t_{pClk,Q} + t_{crítico} + t_{setup}$$

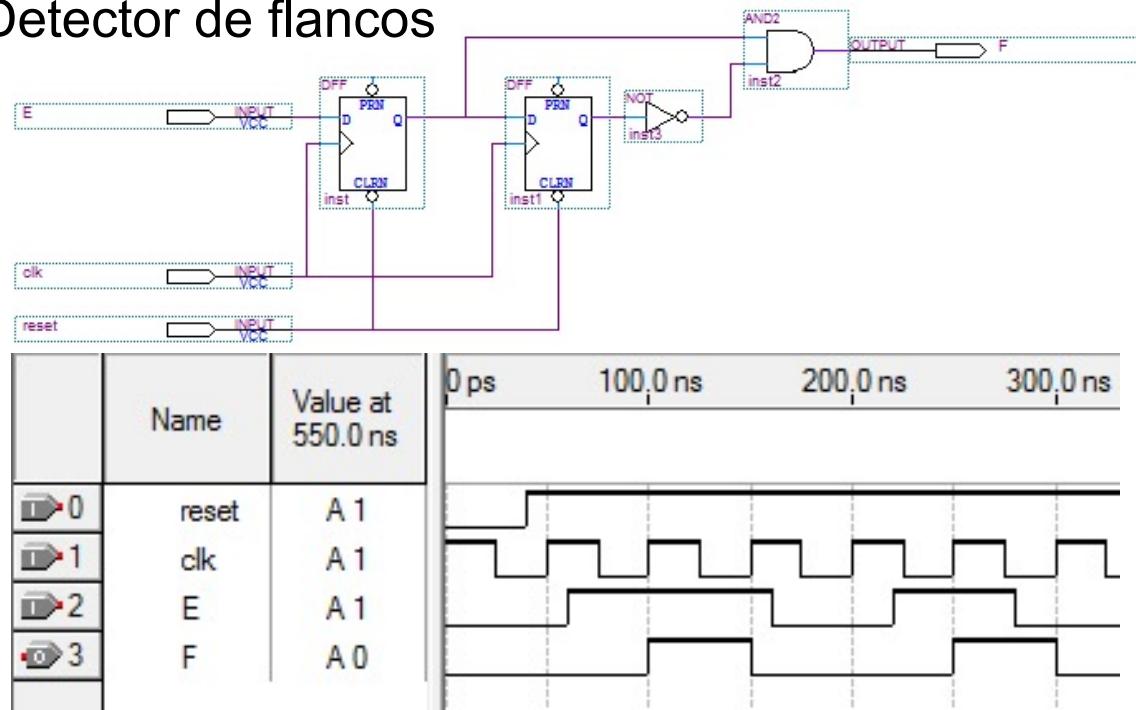
$$f_{Clk} = 1 / t_{Clk}$$

Ej: $t_{Clk} = 1\text{ns} \rightarrow f_{Clk} = 1\text{GHz}$



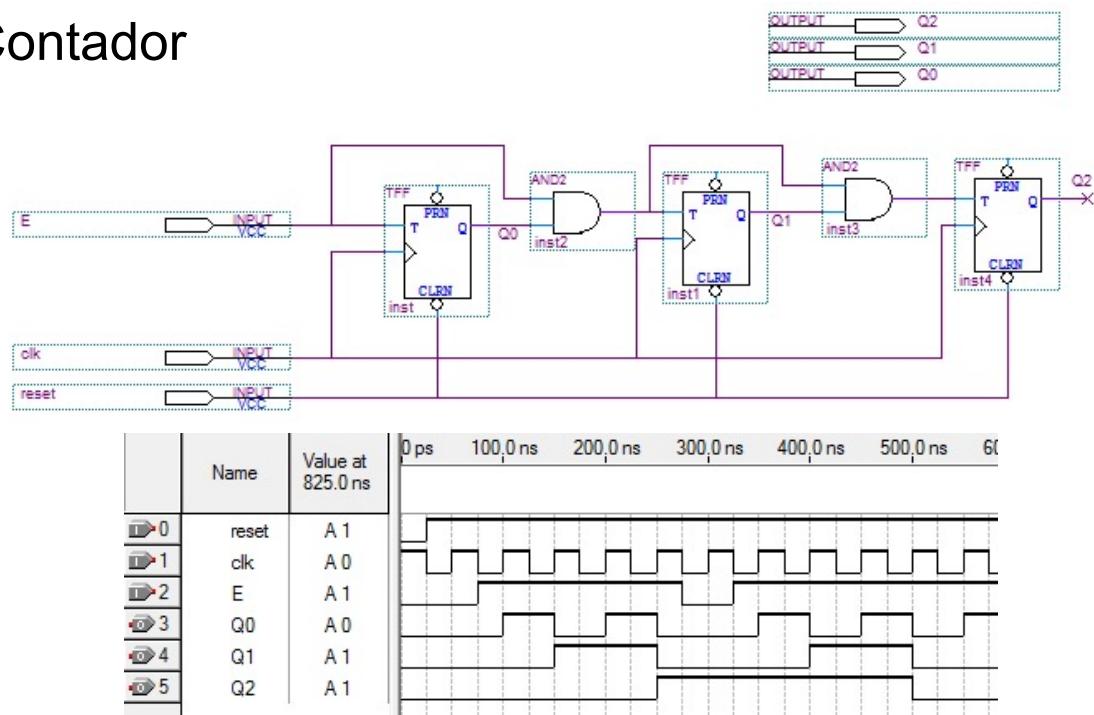
7. Cronogramas con biestables

- Detector de flancos



Cronogramas con biestables

- Contador



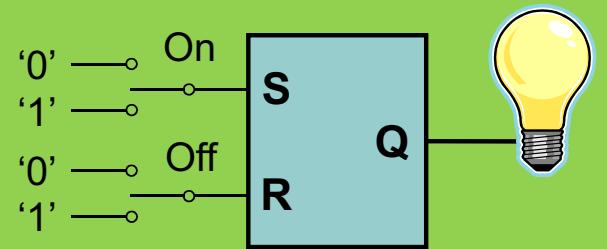


Bibliografía

- “Circuitos y Sistemas Digitales”. J. E. García Sánchez, D. G. Tomás, M. Martínez Iniesta. Ed. Tebar-Flores
- “Electrónica Digital”, L. Cuesta, E. Gil, F. Remiro, McGraw-Hill
- “Fundamentos de Sistemas Digitales “, T.L Floyd, Prentice-Hall

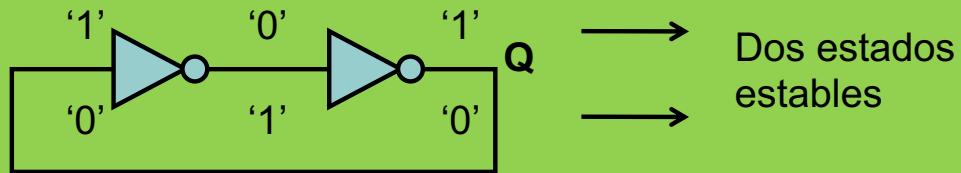
Extra: Biestables asíncronos

- Biestable SR asíncrono
 - $S=1$ => Encender (Set)
 - $R=1$ => Apagar (Reset)
 - $S=R=0$ => Mantener estado
- Características
 - **Memoria:** si no se activan las entradas, mantiene su estado
 - **Asíncrono:** cambia inmediatamente si se activan sus entradas (R o S)



Extra: Biestables asíncronos

- Circuito que mantiene su valor



- Con entradas de control



Extra: Otros tipos de biestables Lógicas de control

- Tabla de funcionamiento
 - Describe funcionalidad

Biestable D (Data)	
D	Q'
0	0
1	1

Biestable T (Toggle)	
T	Q'
0	Q
1	/Q

Biestable SR (Set-Reset)		
S	R	Q'
0	0	Q
0	1	0
1	0	0
1	1	1

Biestable JK (Jump & Kill)		
J	K	Q'
0	0	Q
0	1	0
1	0	1
1	1	/Q

- Tabla de transiciones
 - Describe el próximo estado en función del estado actual y las entradas

Biestable D		
D	Q	Q'
0	0	0
0	1	0
1	0	1
1	1	1

Biestable T		
T	Q	Q'
0	0	0
0	1	1
1	0	1
1	1	0



Circuitos combinacionales aritméticos (Parte II)

© Luis Entrena, Celia López,
Mario García, Enrique San Millán

Universidad Carlos III de Madrid



Contenidos

1. Circuitos sumadores y restadores

- Sumadores con propagación de acarreo serie
 - Semisumador. Sumador total. Sumador de n bits con acarreo serie
- Sumadores con acarreo anticipado
- Sumador/Restador en complemento a 2

2. Circuitos de multiplicación

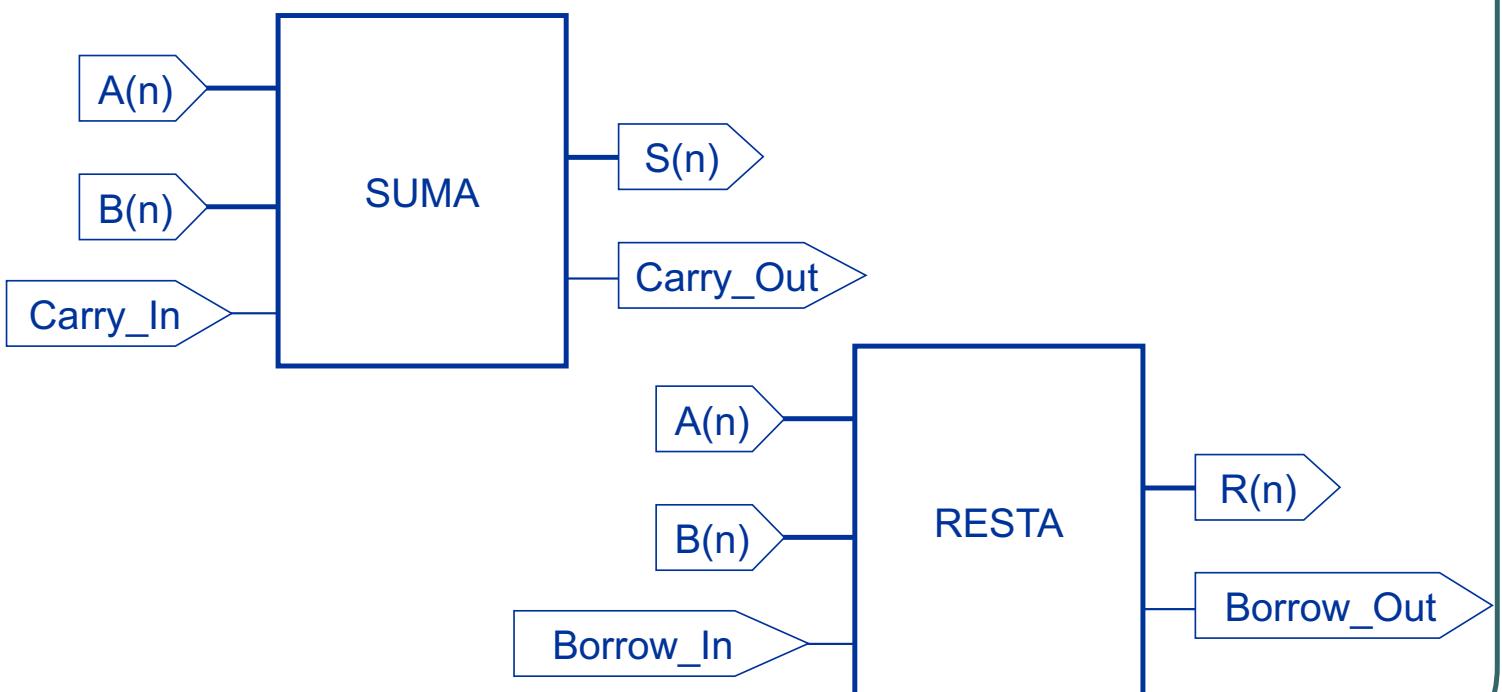
- Circuito multiplicador básico

3. Unidades Aritmético-Lógicas (ALUs)

- Concepto de ALU



Circuitos sumadores y restadores

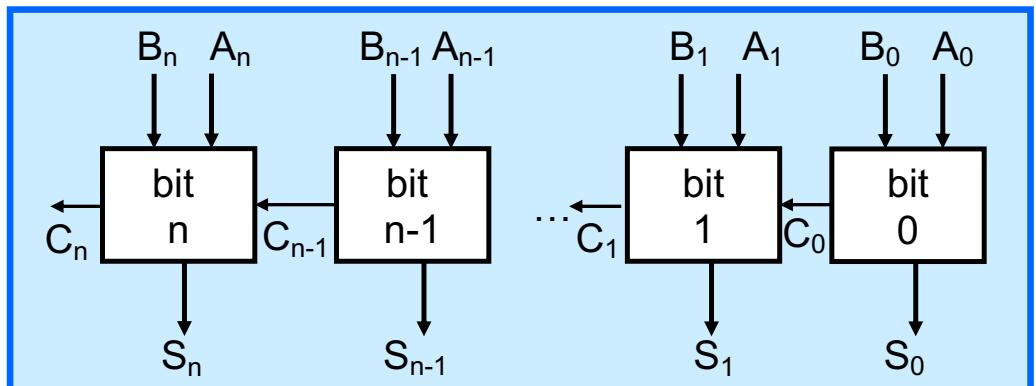


Sumador con propagación de acarreo serie.

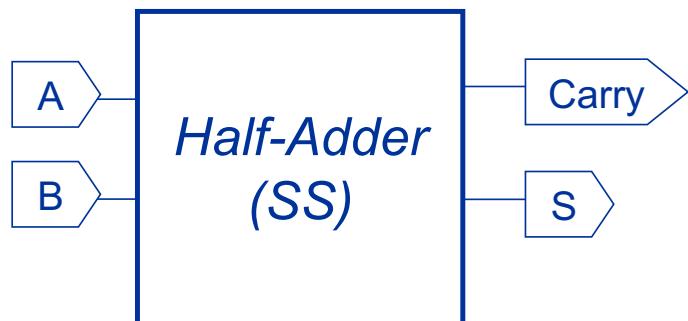
Suma decimal y binaria

$$\begin{array}{r}
 \textcolor{blue}{1} \textcolor{yellow}{1} \\
 86d \\
 + 25d \\
 \hline
 \textcolor{blue}{1} \textcolor{green}{1} \textcolor{blue}{1} d
 \end{array}
 \quad \rightarrow \quad
 \begin{array}{r}
 1010110b \\
 + 0011001b \\
 \hline
 11011111b
 \end{array}$$

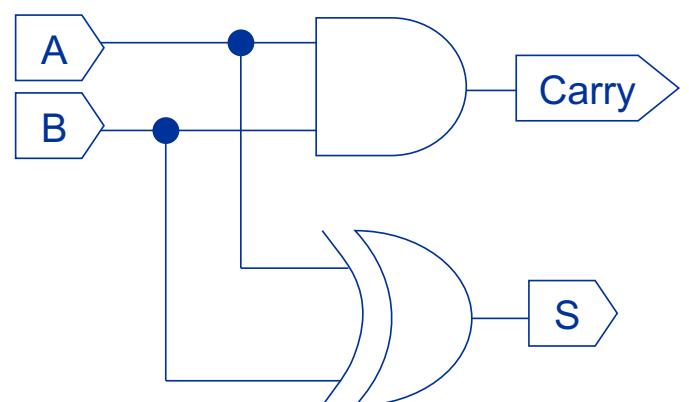
- Operandos: **n bits**
- Resultado: **n+1 bits**



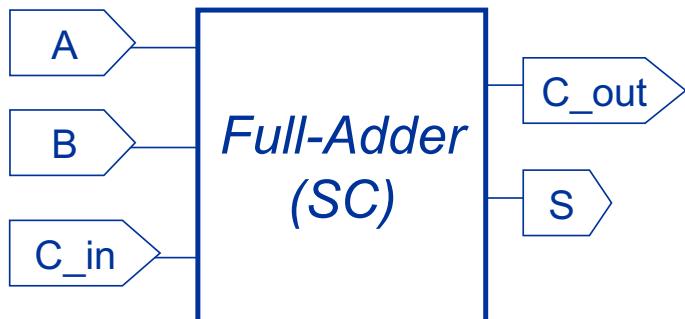
Sumador con propagación de acarreo serie. Semisumador



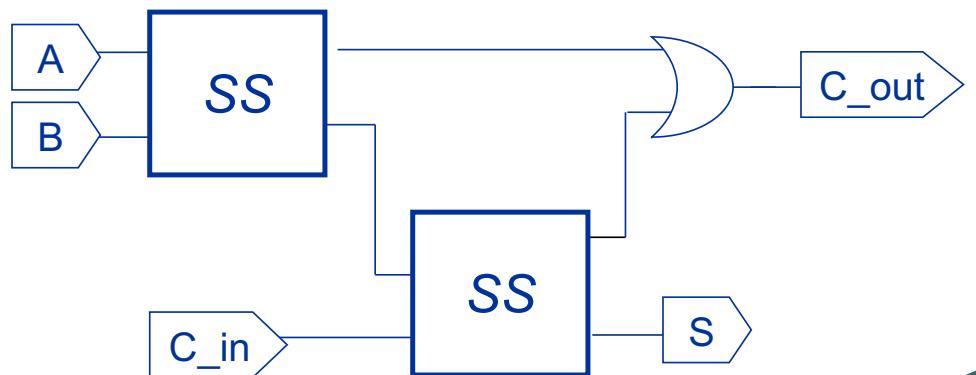
A	B	S	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



Sumador con propagación de acarreo serie. Sumador completo



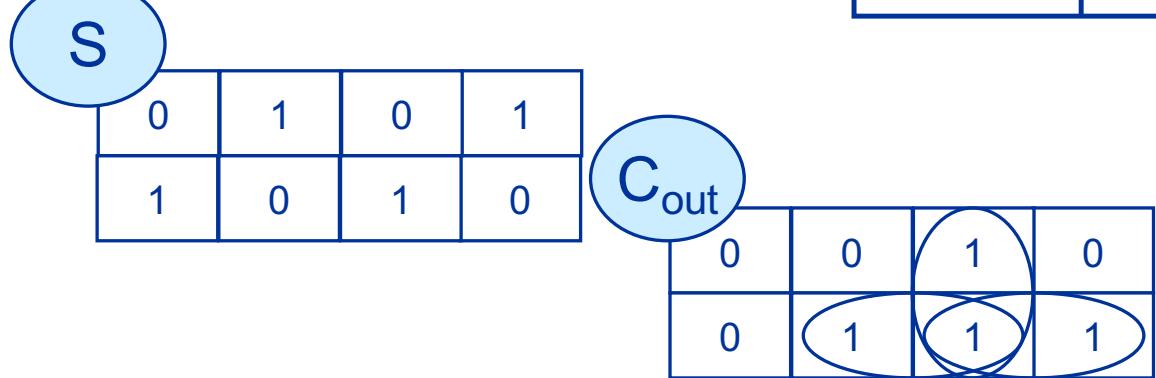
<i>A</i>	<i>B</i>	<i>C_{in}</i>	<i>S</i>	<i>C_{out}</i>
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



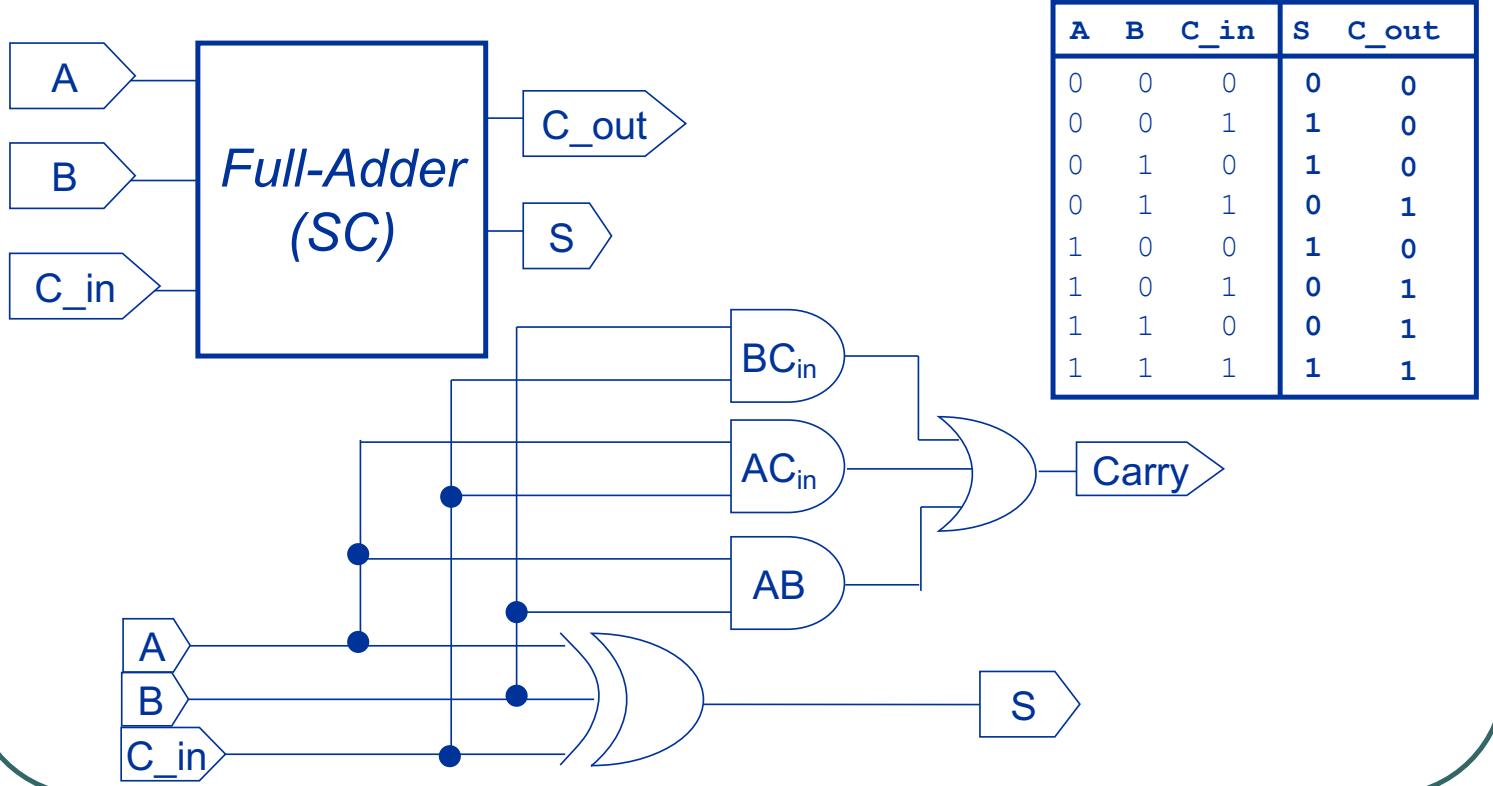
Sumador con propagación de acarreo serie. Sumador completo



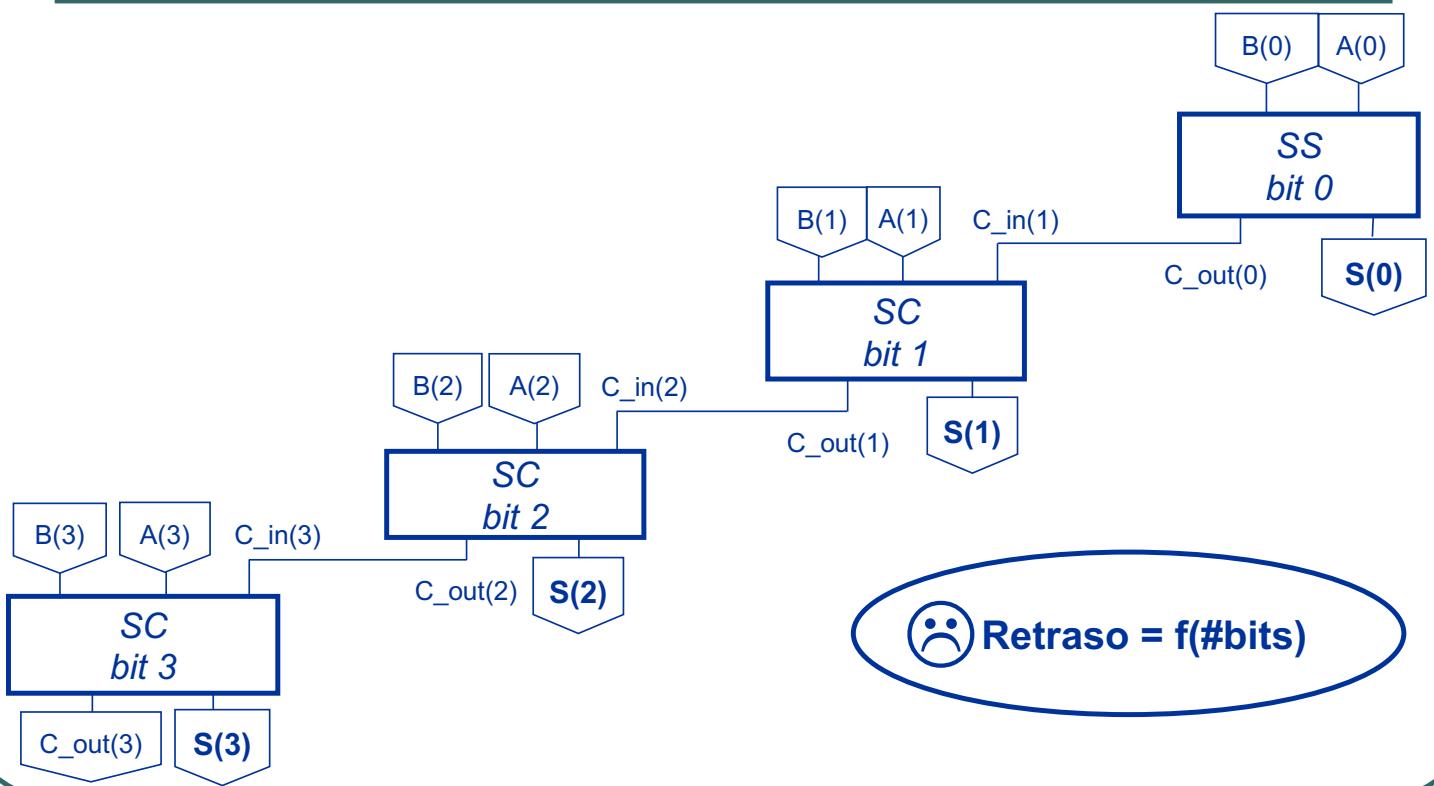
A	B	C _{in}	S	C _{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



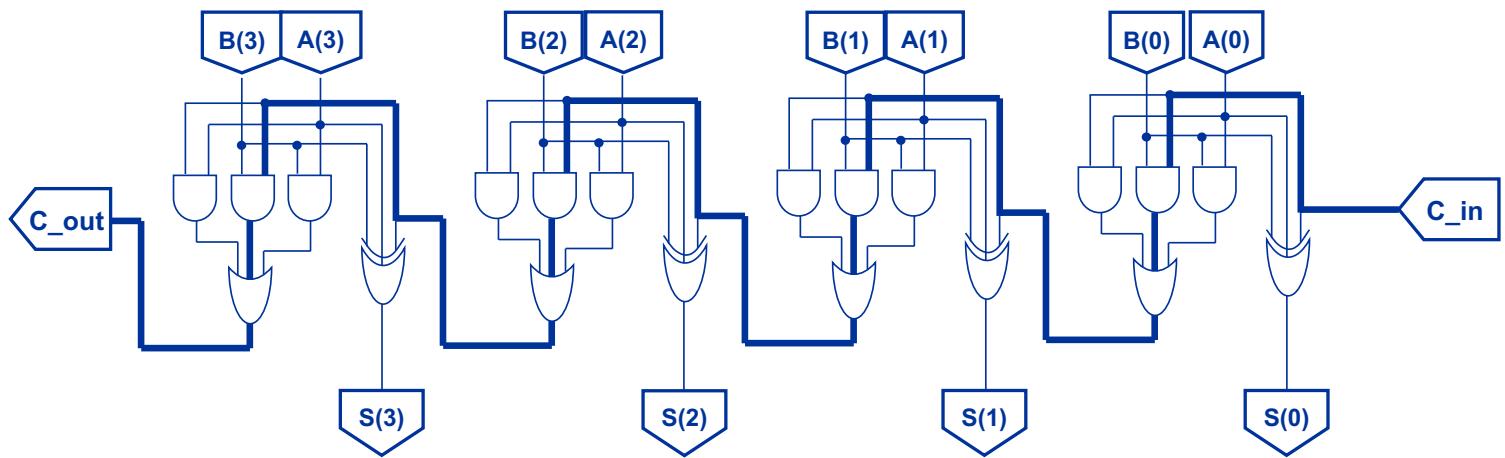
Sumador con propagación de acarreo serie. Sumador completo



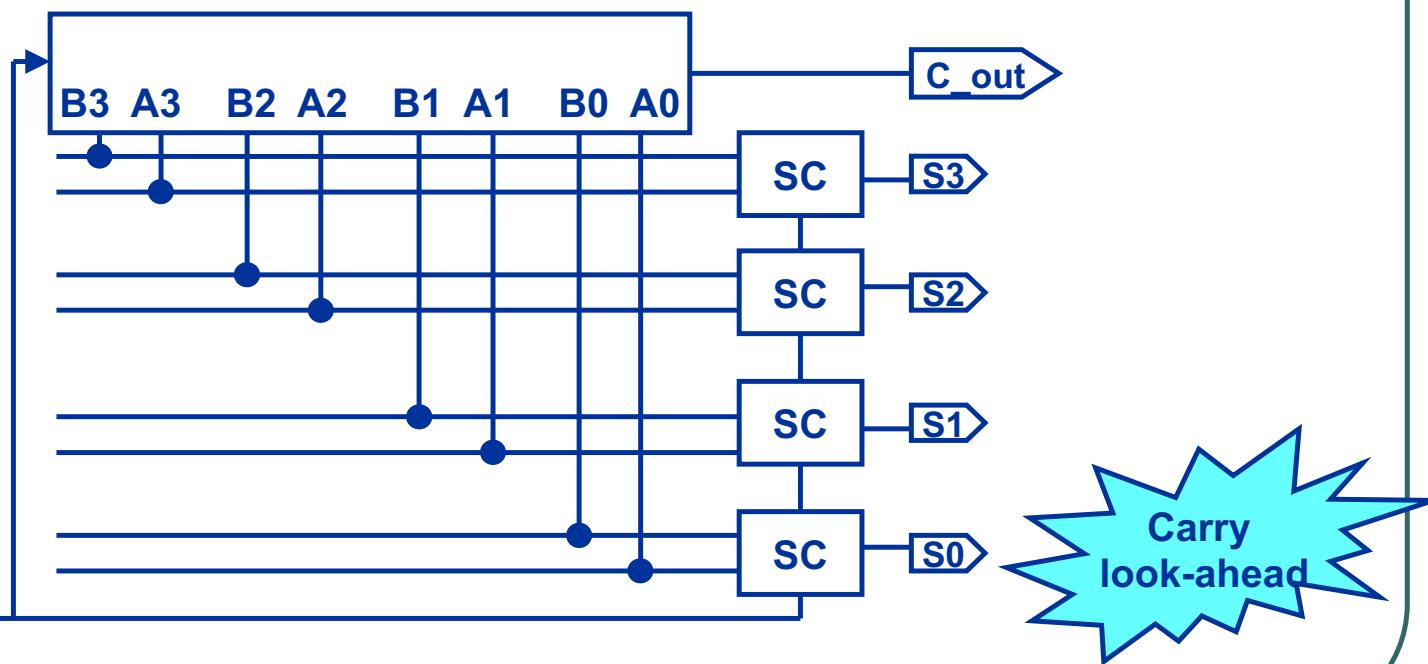
Sumador con propagación de acarreo serie. Sumador de varios bits



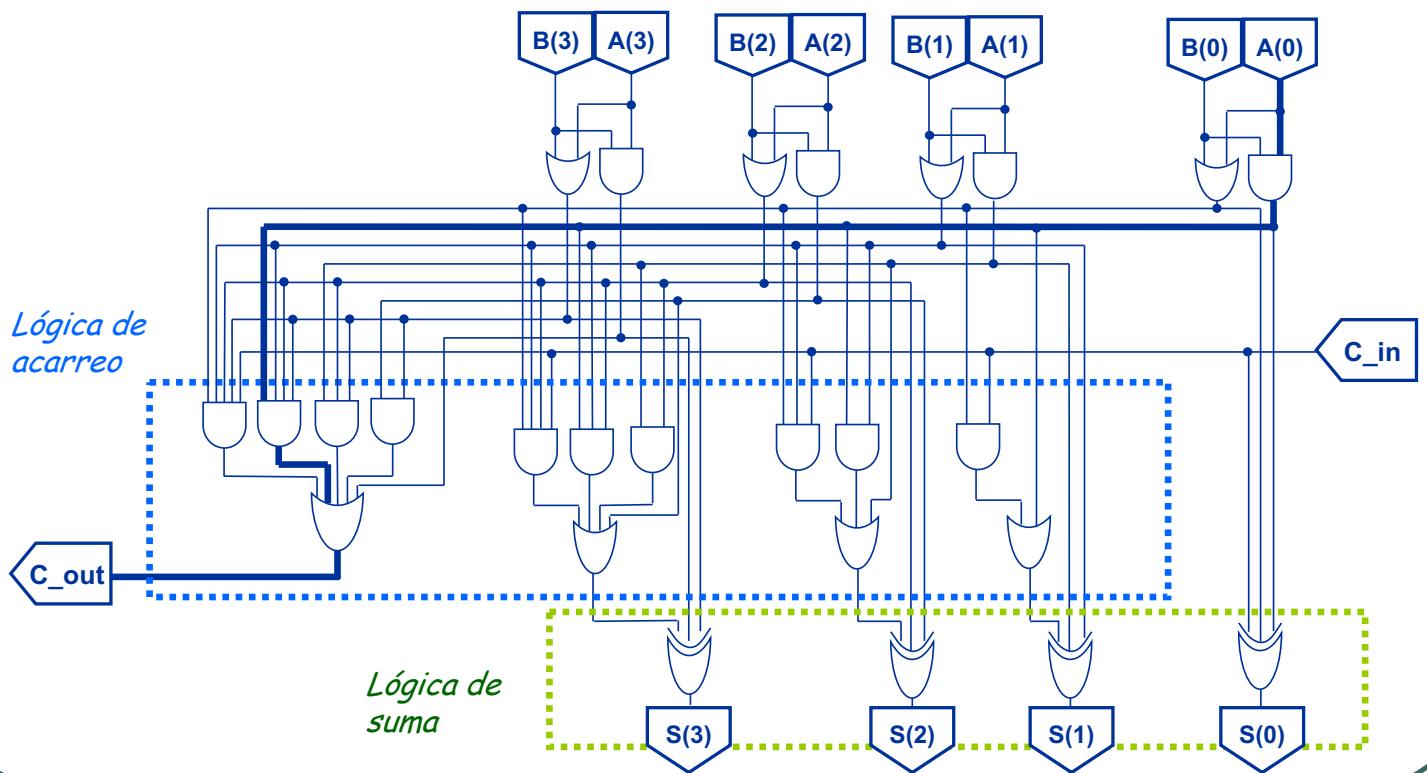
Sumador con propagación de acarreo serie. Sumador de varios bits



Sumador con acarreo anticipado.



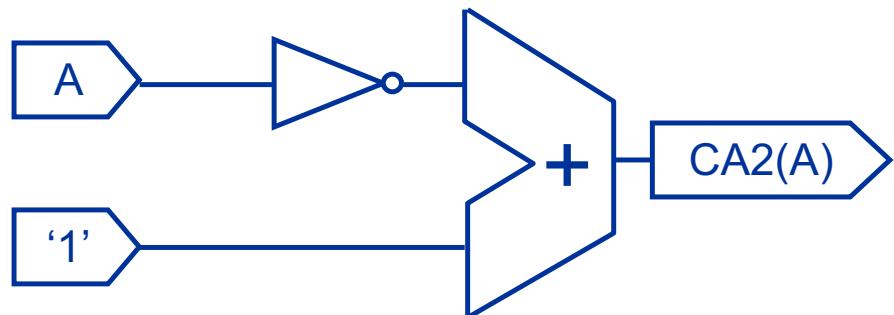
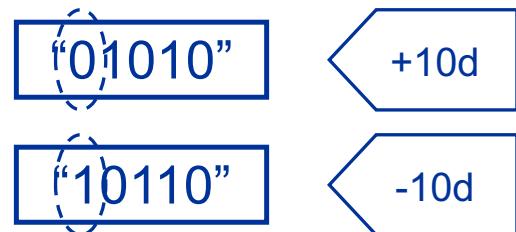
Sumador con acarreo anticipado.



Sumador/restador en CA2.

Complemento a 2

- Números positivos
- Números negativos

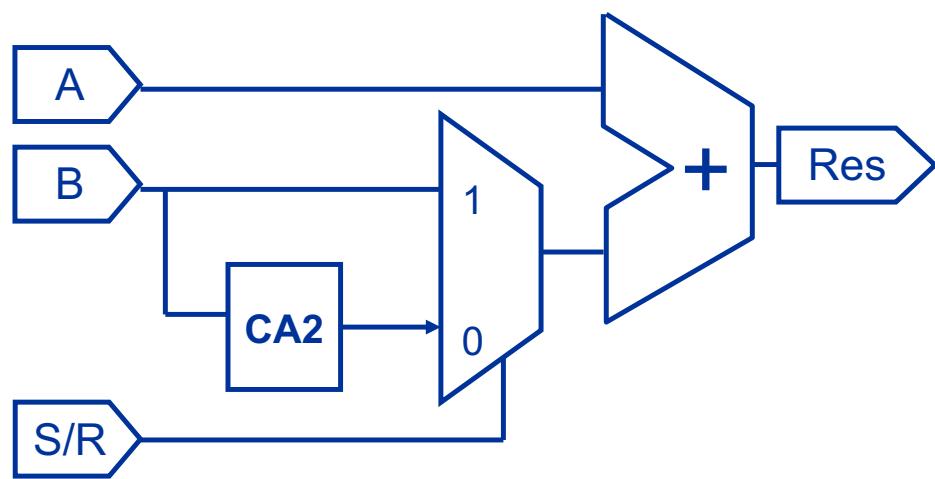




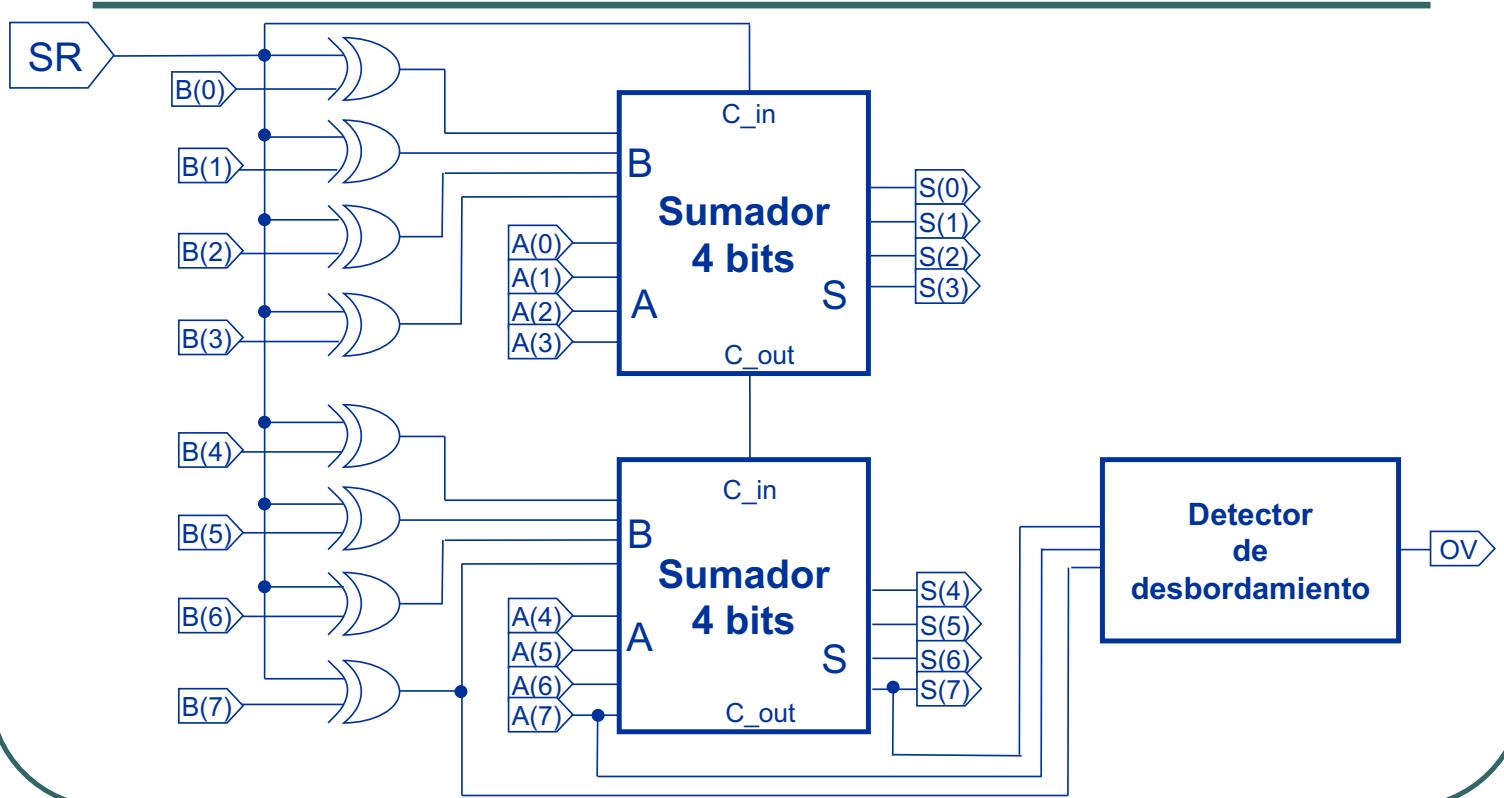
Sumador/restador en CA2.



$$A - B = A + (-B)$$



Sumador/restador en CA2.





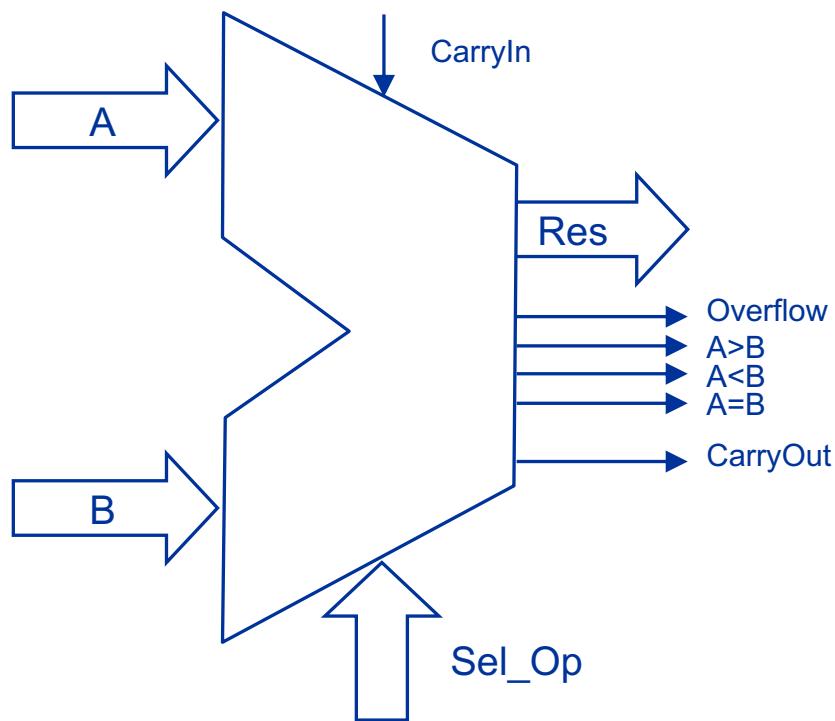
Sumador/restador en CA2.

Ejercicio





Unidad Aritmético-Lógica





Unidad Aritmético-Lógica

Combinacional

Bloque para la realización de operaciones aritmético-lógicas:

- Suma
 $A + B$
- Resta
 $A - B$
- Complemento a 2
 $- B$
- Comparación
 - $A > B$
 - $A < B$
 - $A = B$
- Desplazamiento a la izquierda
 $SHL(A) \leftarrow$
- Desplazamiento a la derecha
 $SHR (A) \rightarrow$

Operaciones lógicas (bit a bit)

- AND
- OR
- XOR
- XNOR
- NOT





Referencias

- “Circuitos y Sistemas Digitales”. J. E. García Sánchez, D. G. Tomás, M. Martínez Iniesta. Ed. Tebar-Flores
- “Electrónica Digital”, L. Cuesta, E. Gil, F. Remiro, McGraw-Hill
- “Fundamentos de Sistemas Digitales”, T.L Floyd, Prentice-Hall



Tema 7

Registros y Contadores

© Luis Entrena, Celia López,
Mario García, Enrique San Millán

Universidad Carlos III de Madrid



Contenidos

1. *Registros*

- **Registros con entrada serie y salida serie y paralelo**
- **Registros con entrada paralelo y salida serie y paralelo**
- **Registro universal de desplazamiento**

2. *Contadores*

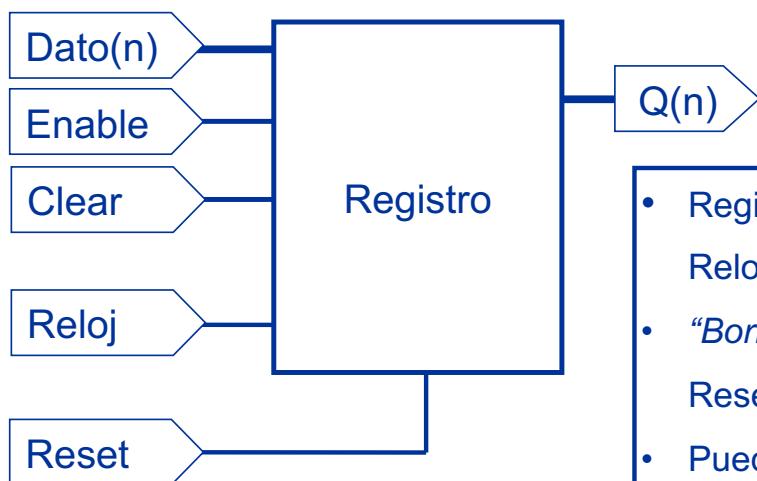
- **Contadores síncronos**
 - Concepto de contador síncrono.
 - Síntesis como máquina de estados con biestables T.
 - Contador ascendente-descendente.
 - Contadores con entradas de precarga, acarreo/habilitación y salida de acarreo.
 - Aplicaciones con contadores síncronos: secuenciadores.
- **Contadores basados en registros de desplazamiento**
 - Contador en anillo. Contador Johnson.



Registro

“Circuito digital con dos funciones básicas: almacenamiento de datos y movimiento de datos” (Floyd)

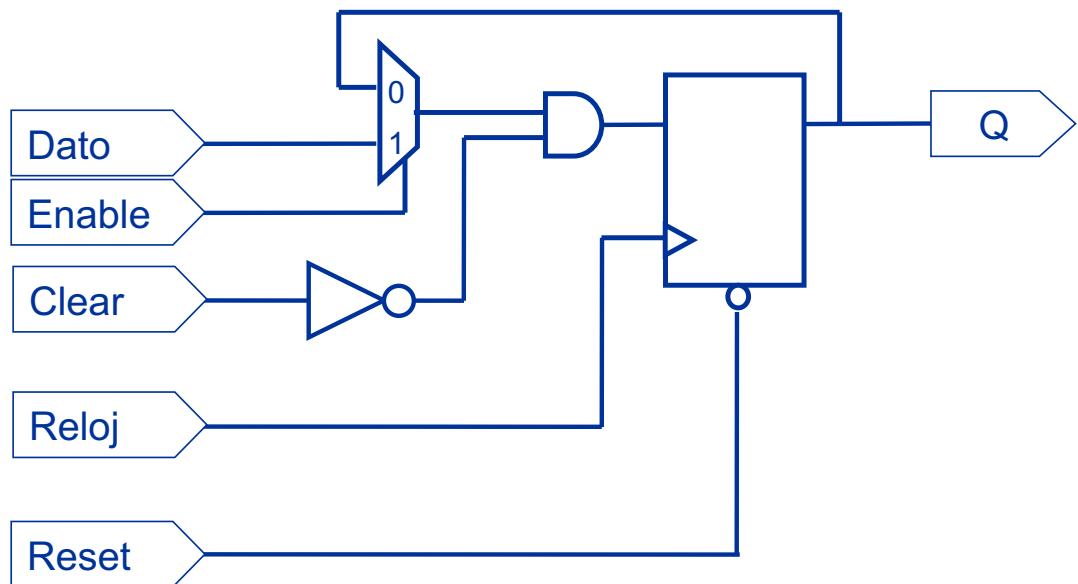
Es una colección de dos o más biestables tipo-D con una entrada común. Se utiliza para almacenar una serie de bits relacionados, como un byte (8 bits) de datos.



- Registra datos en los flancos activos del Reloj
- “Borra” el contenido ante el nivel activo del Reset
- Puede tener señales de Habilitación y Clear síncronos

Registro (1 bit)

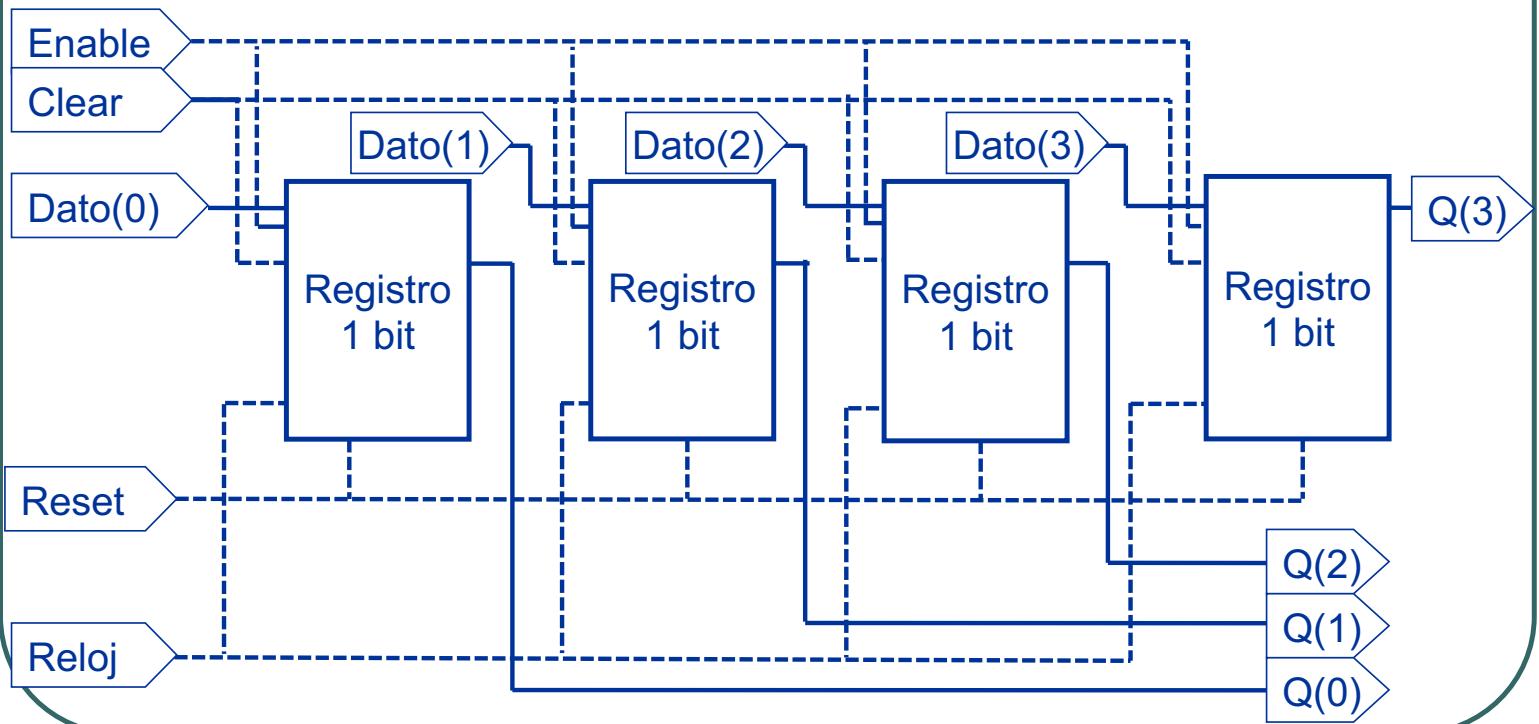
Esquema



Registro (4 bits)

Entrada paralelo/salida paralelo

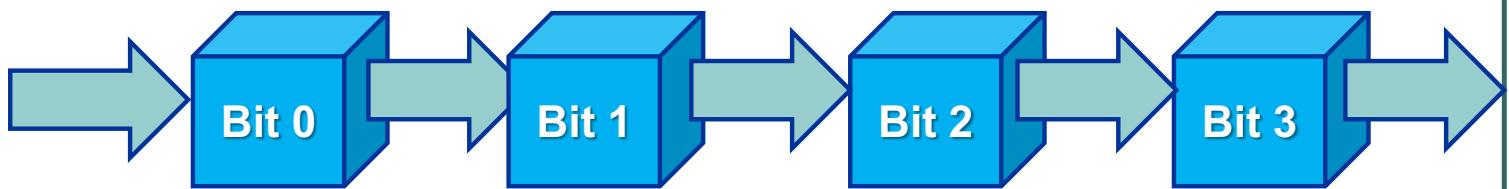
Esquema





Registro de desplazamiento

Es un registro que almacena y desplaza la información

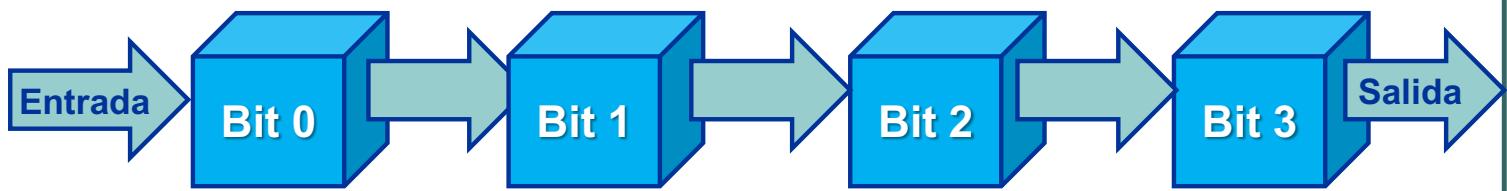


- Registra datos en los flancos activos del Reloj. Desplaza los bits..
- “Borra” el contenido ante el nivel activo del Reset
- Puede tener señales de Habilitación y Clear síncronos



Registro de desplazamiento

Entrada y salida SERIE

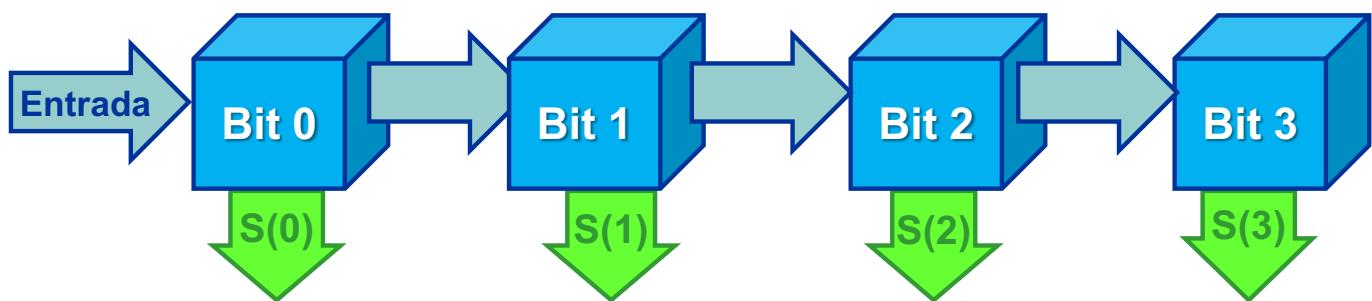


SISO

- Un bit de entrada. Un bit de salida.
- Carga SERIE. Desplaza los bits.
- 4 Ciclos de reloj en cargar un dato
- 4 Ciclos de reloj en leer un dato

Registro de desplazamiento

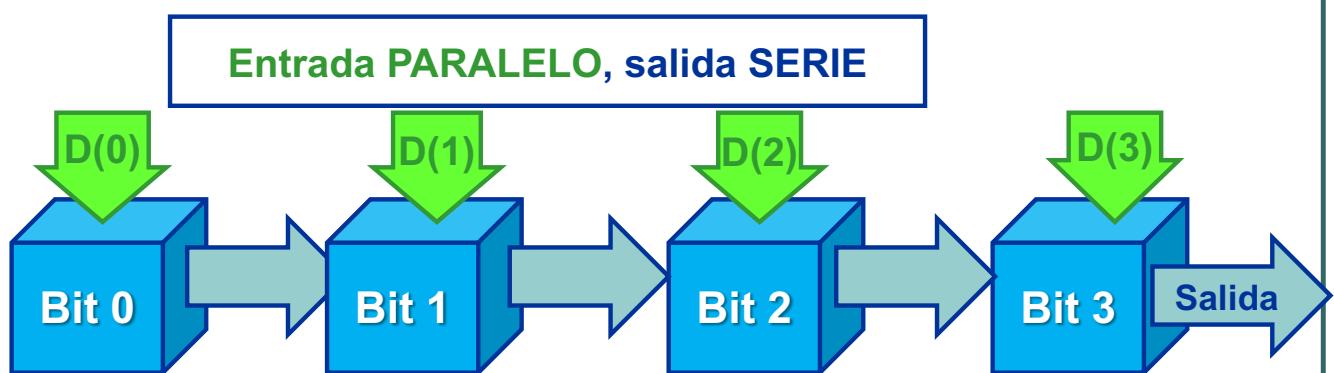
Entrada SERIE, salida PARALELO



- Un bit de entrada. Cuatro bits de salida.
- Carga SERIE. Desplaza los bits.
- 4 Ciclos de reloj en cargar un dato
- 1 Ciclo de reloj en leer un dato

SIPO

Registro de desplazamiento



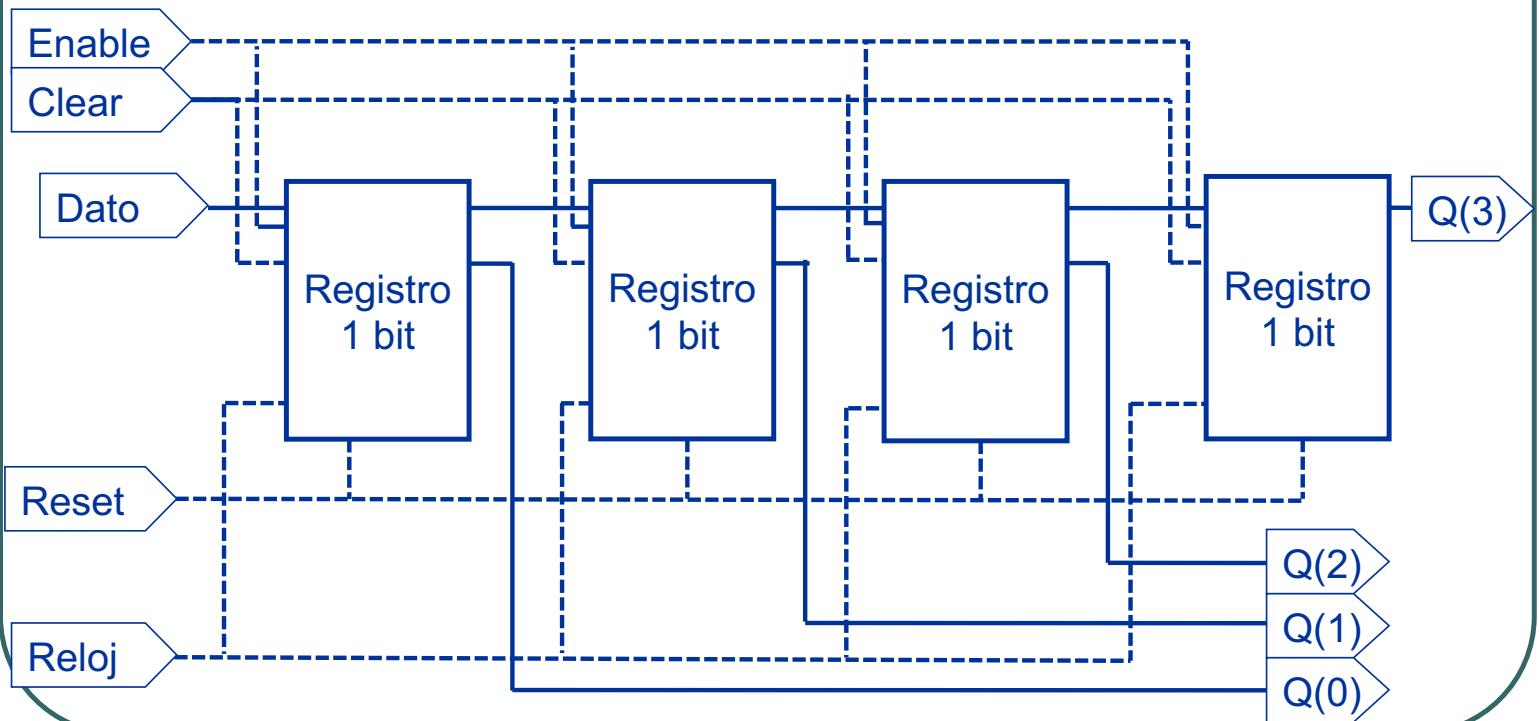
PISO

- Cuatro bits de entrada. Un bit de salida.
- Carga PARALELO. Salida SERIE.
- 1 Ciclo de reloj para almacenar el dato
- 4 Ciclos de reloj para leer el dato

Registro de desplazamiento

Entrada serie/ salida paralelo-serie

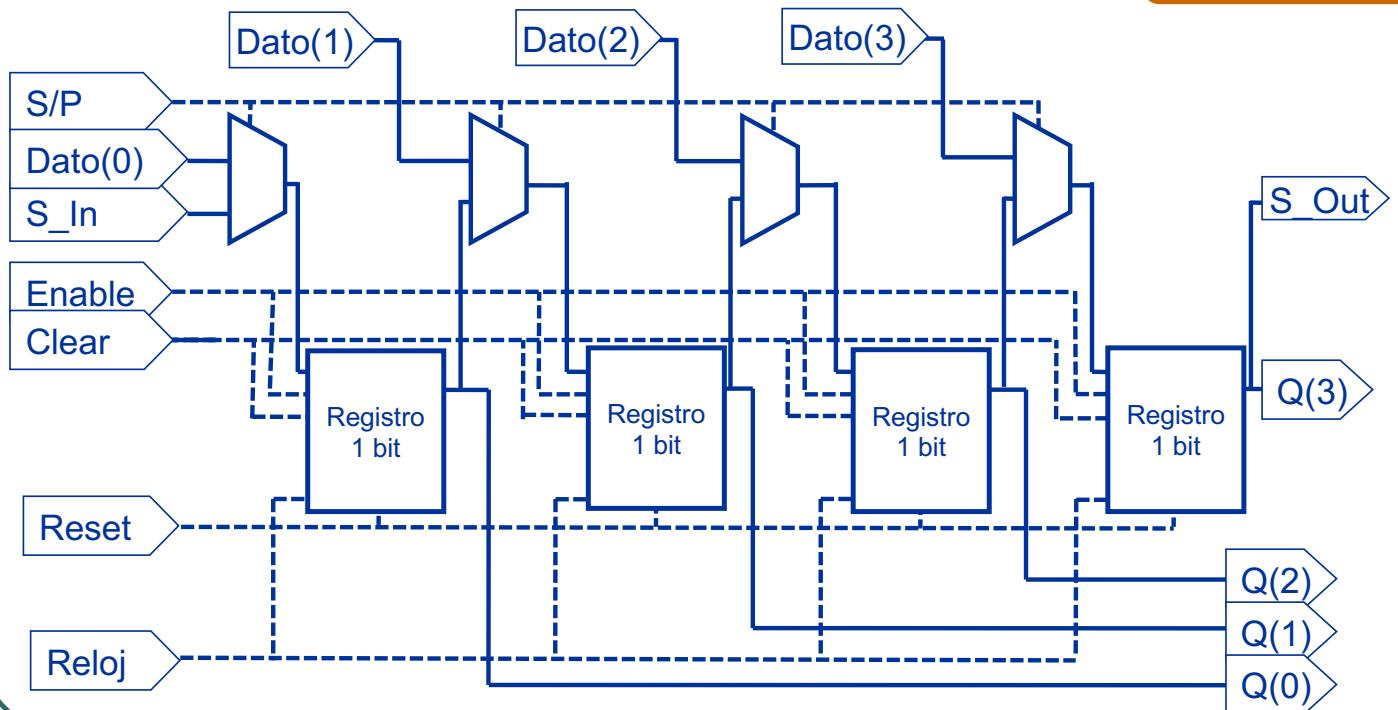
Esquema



Registro de desplazamiento

Entrada serie-paralelo / salida paralelo-serie

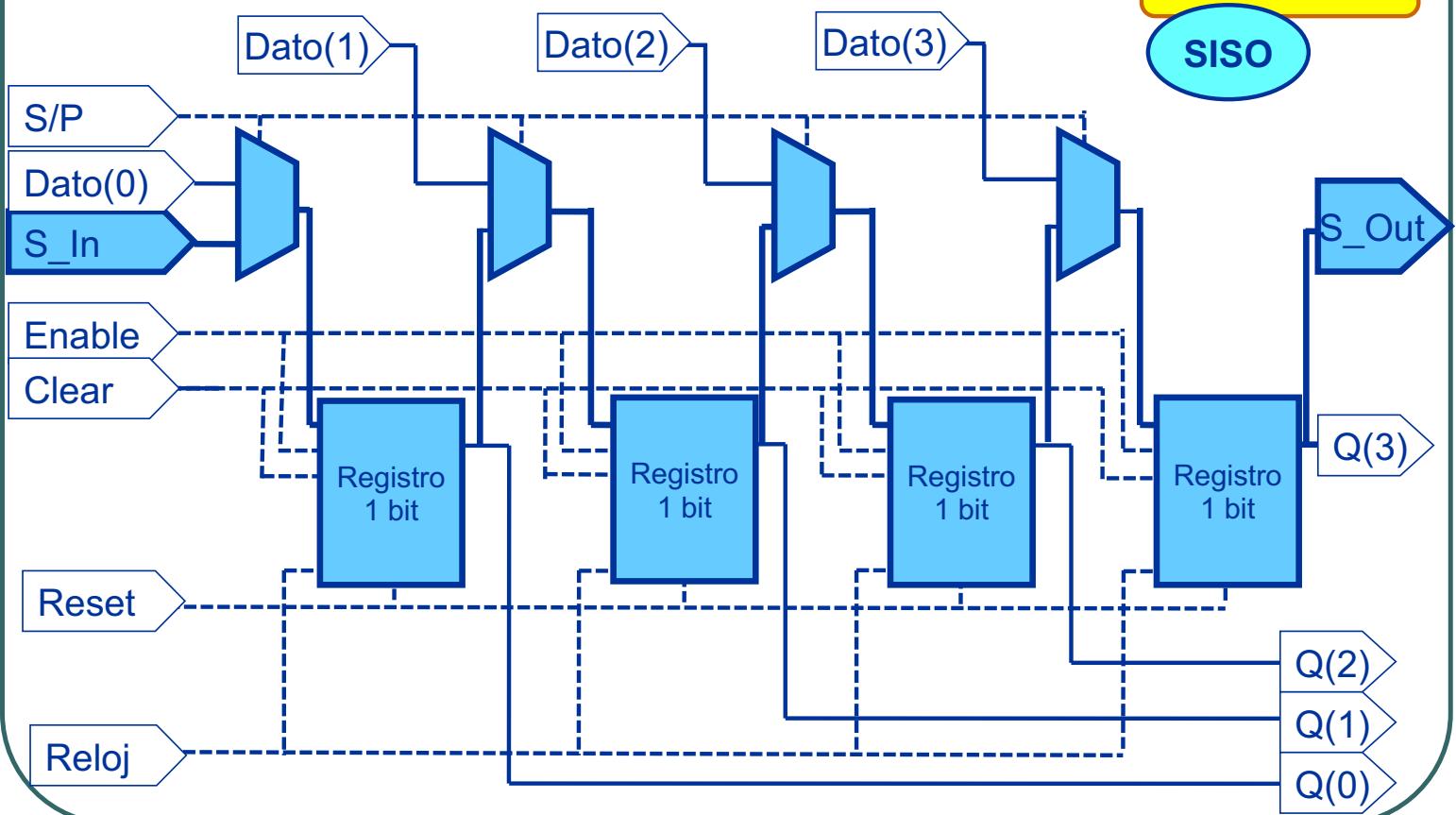
Esquema



Registro de desplazamiento

Esquema

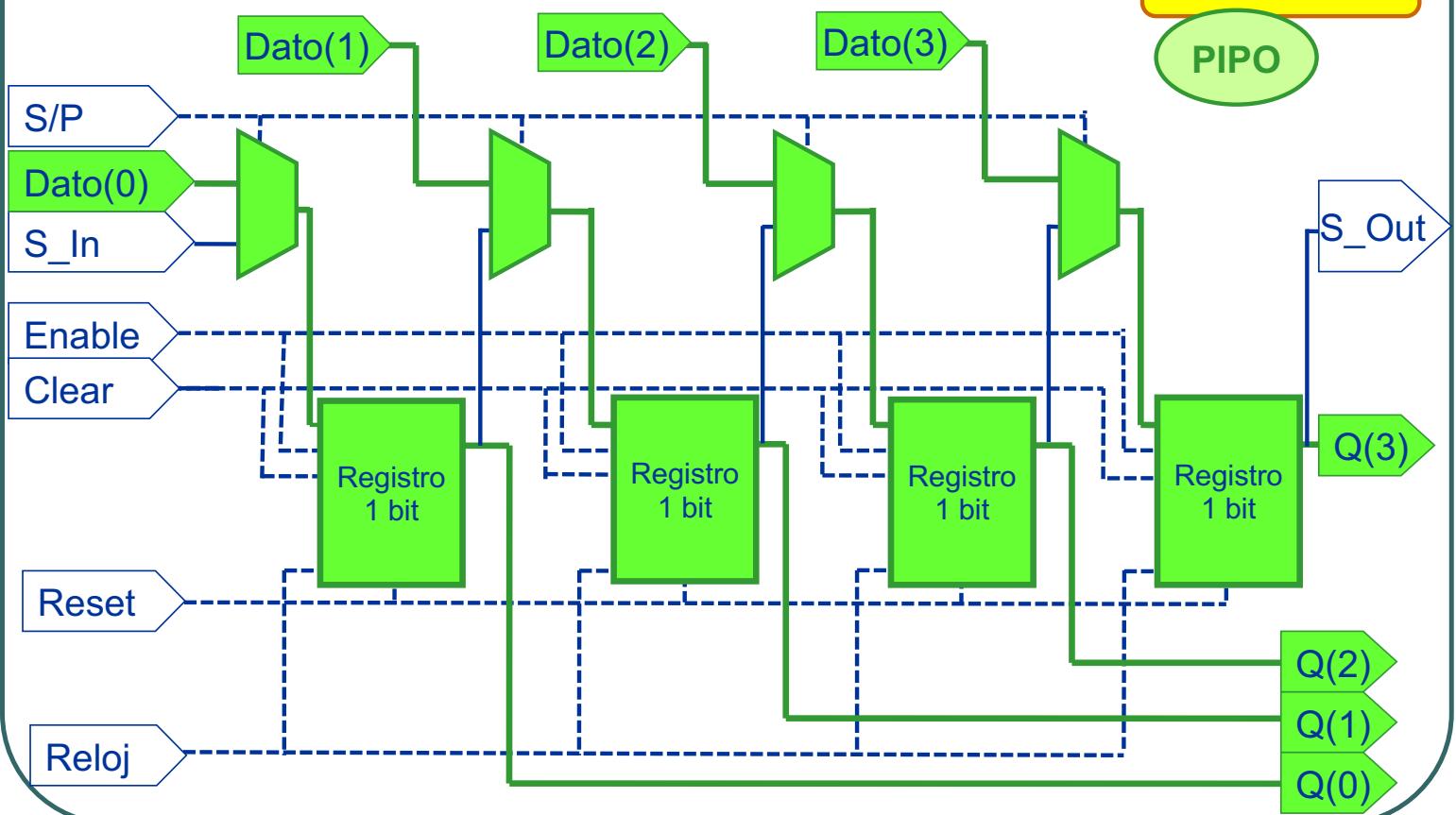
SISO



Registro de desplazamiento

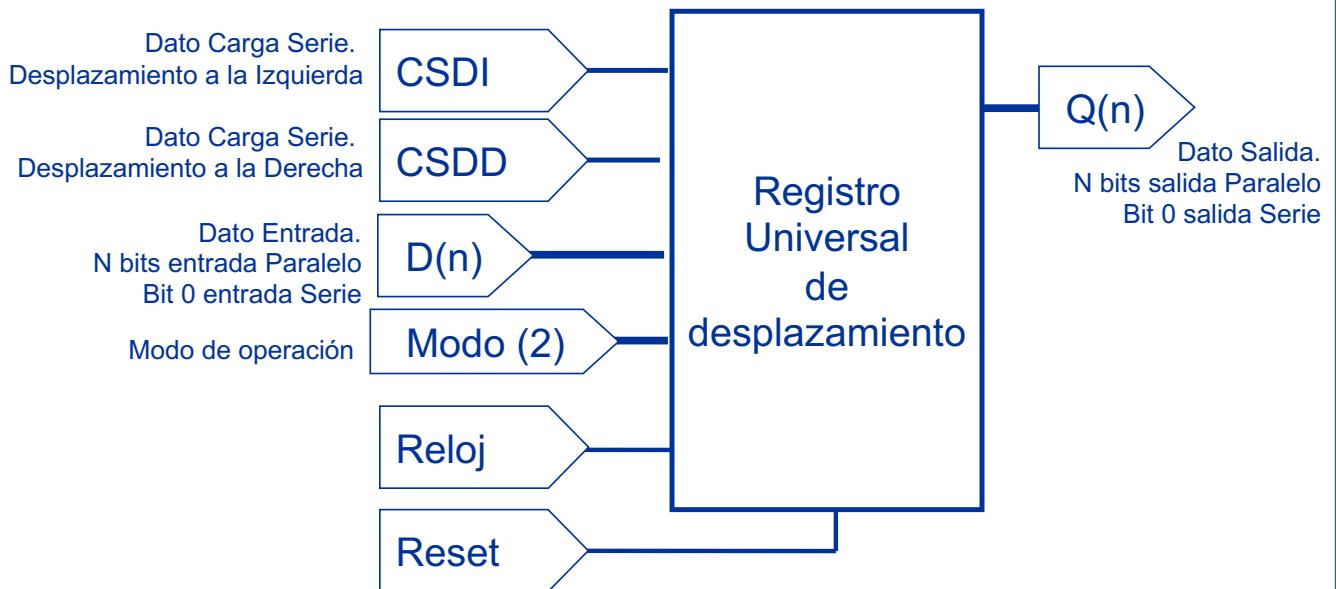
Esquema

PIPO





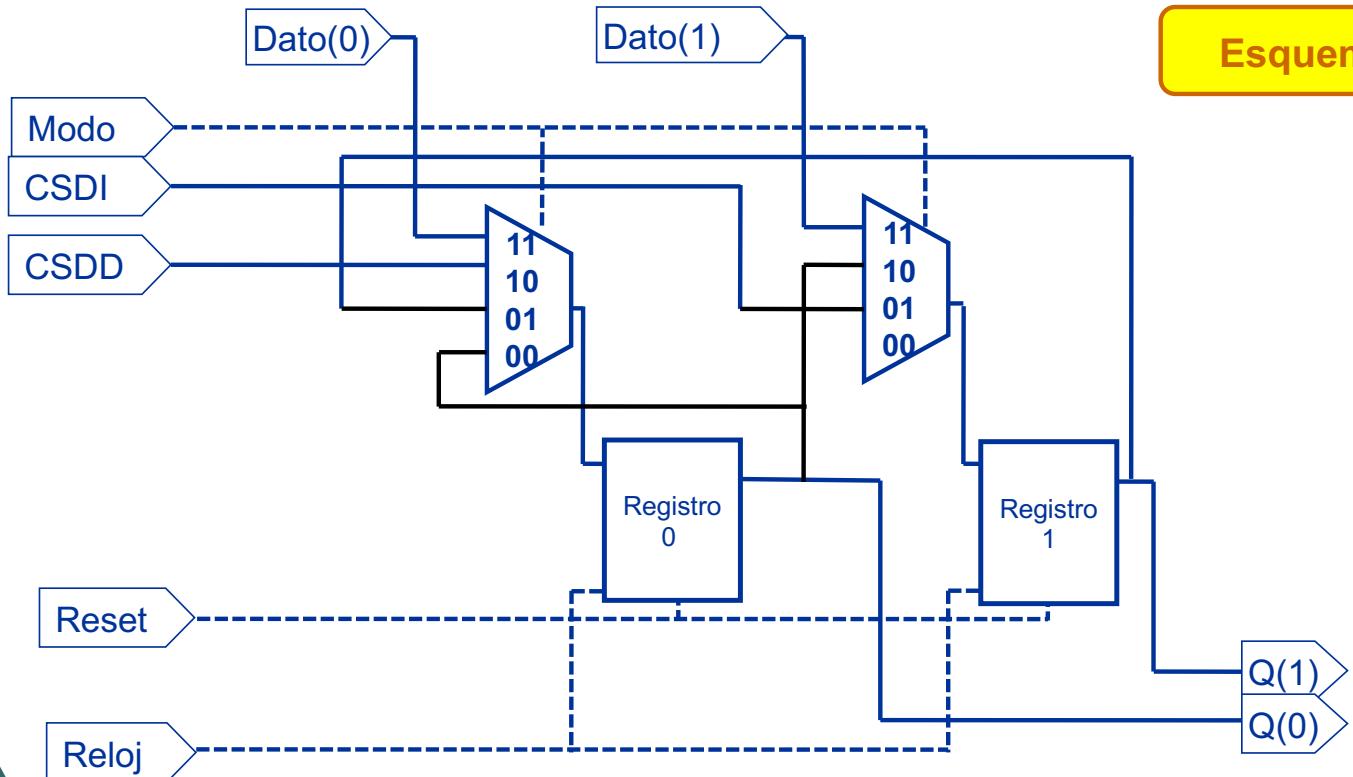
Registro universal de desplazamiento



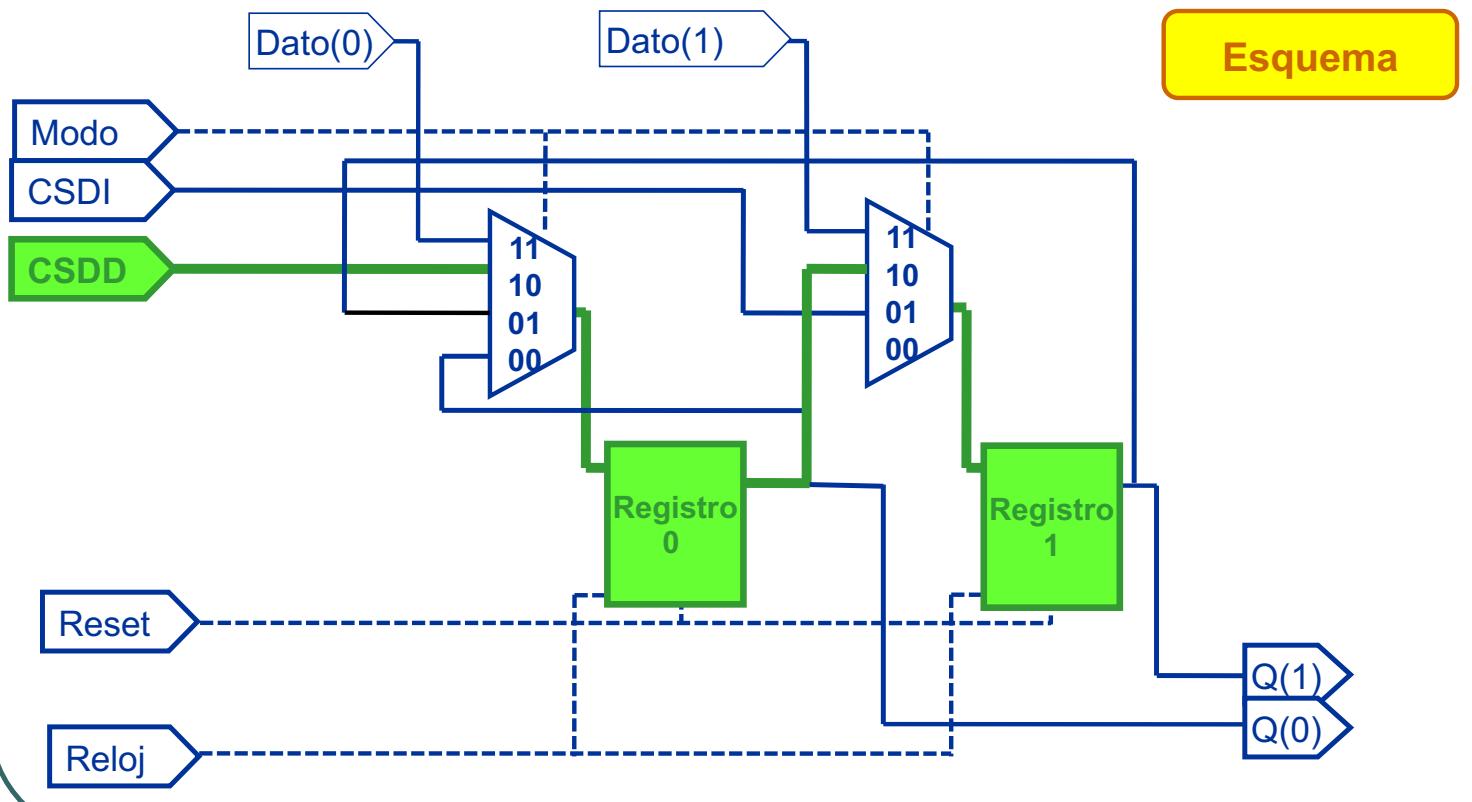
Registro de desplazamiento con entrada serie y paralelo. Los datos se pueden desplazar a izquierda o a derecha.

Registro universal de desplazamiento

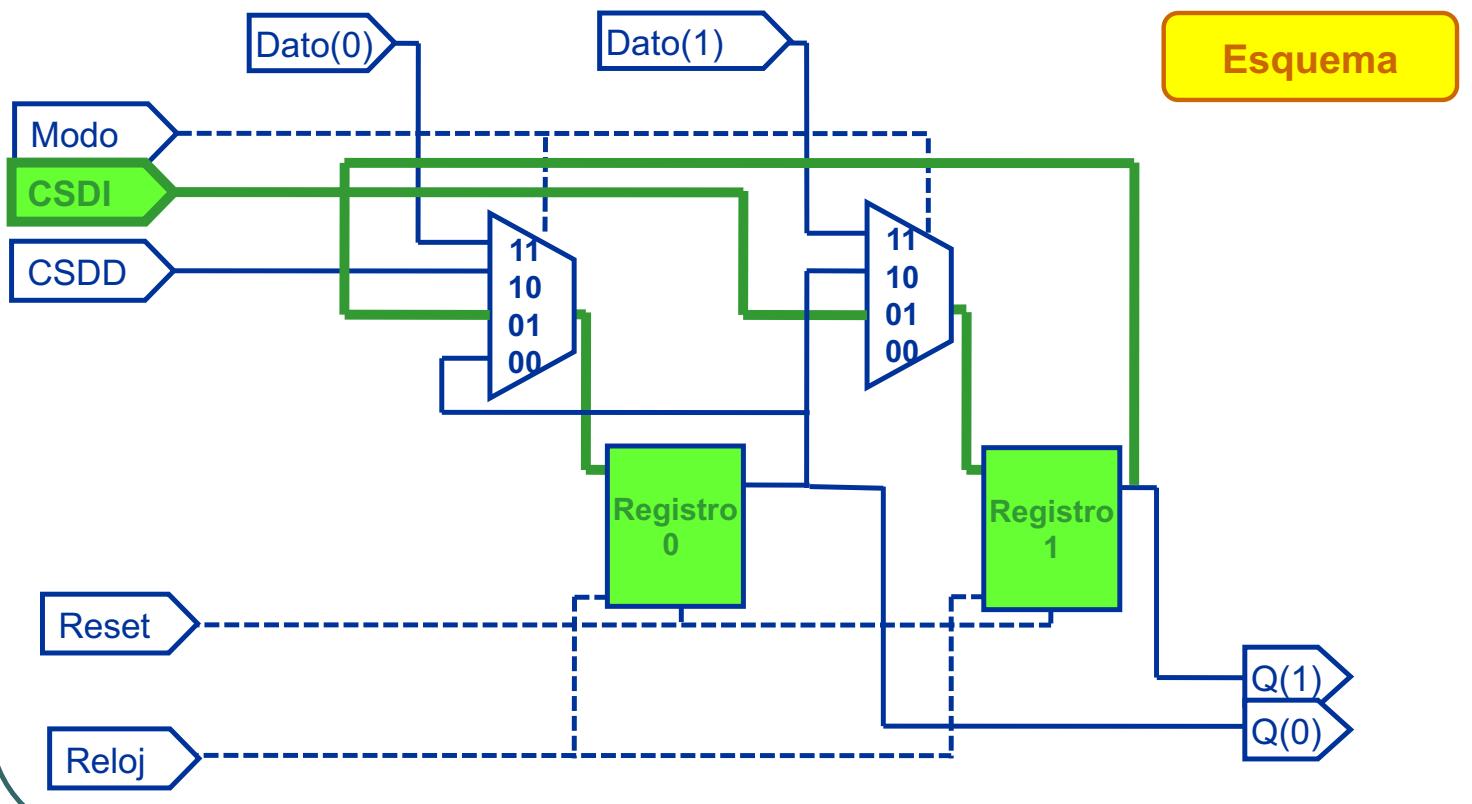
Esquema



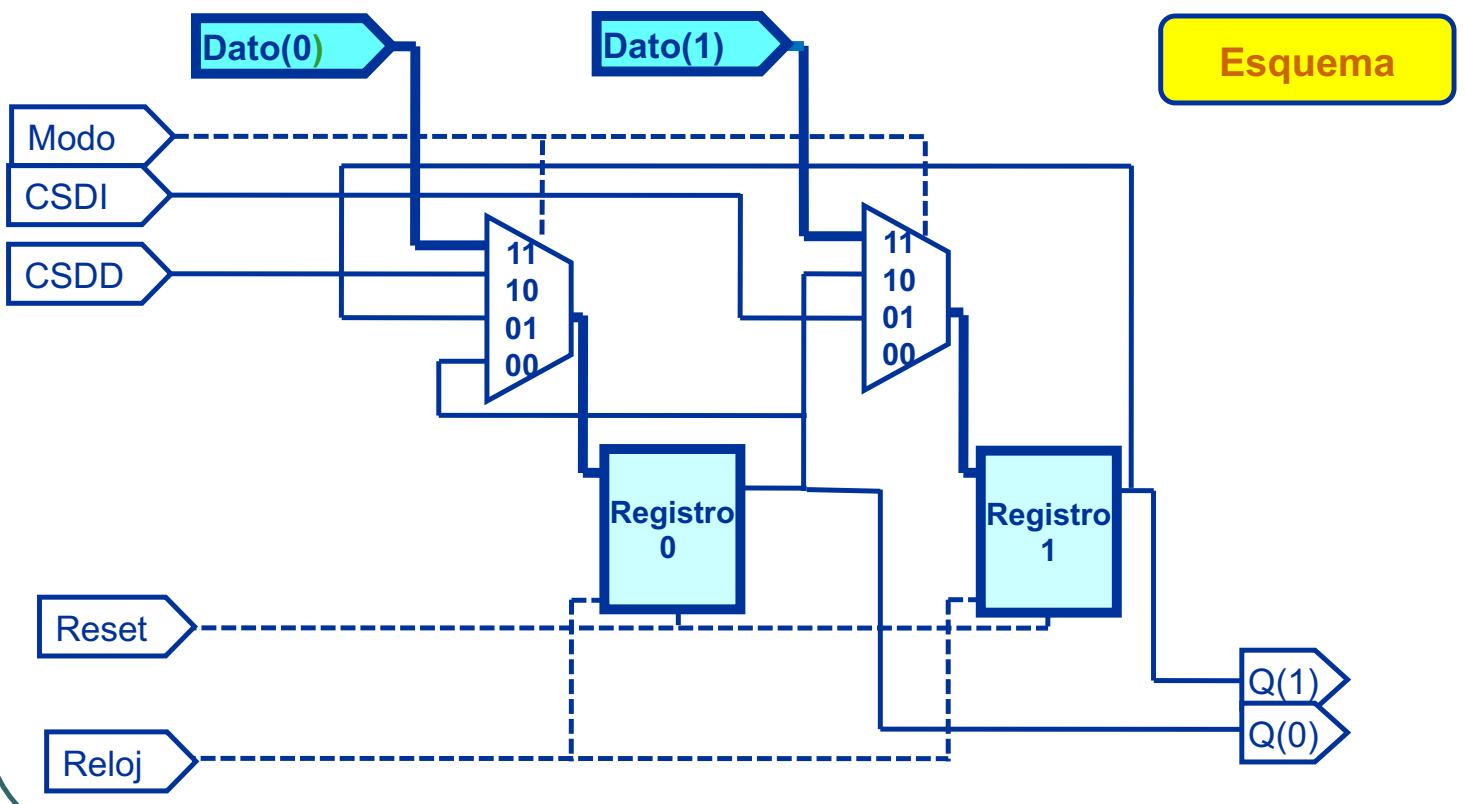
Registro universal de desplazamiento



Registro universal de desplazamiento



Registro universal de desplazamiento

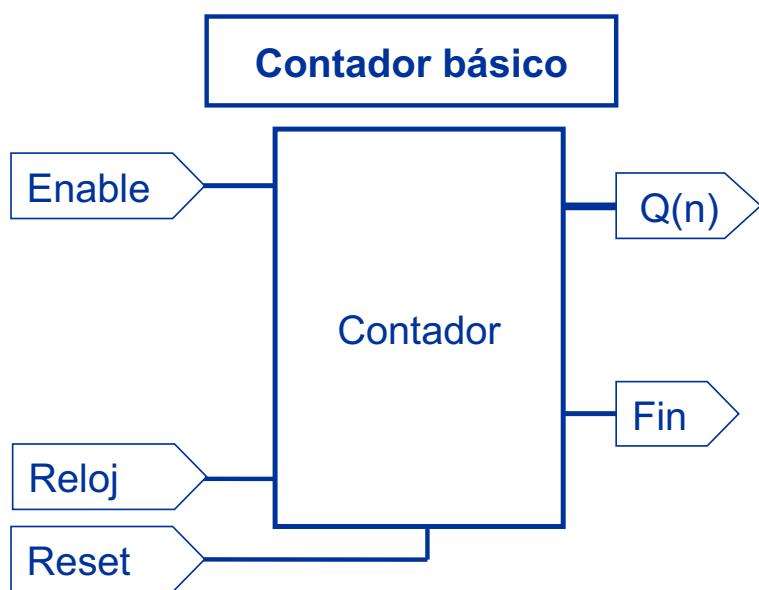




Contadores síncronos



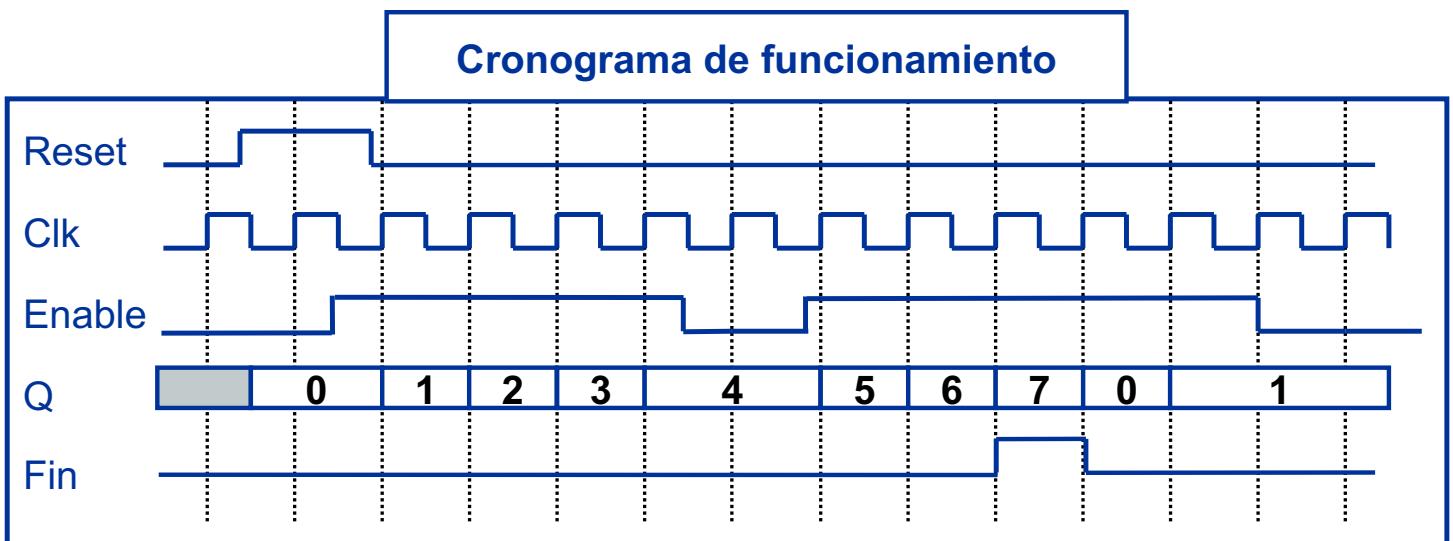
Contadores síncronos



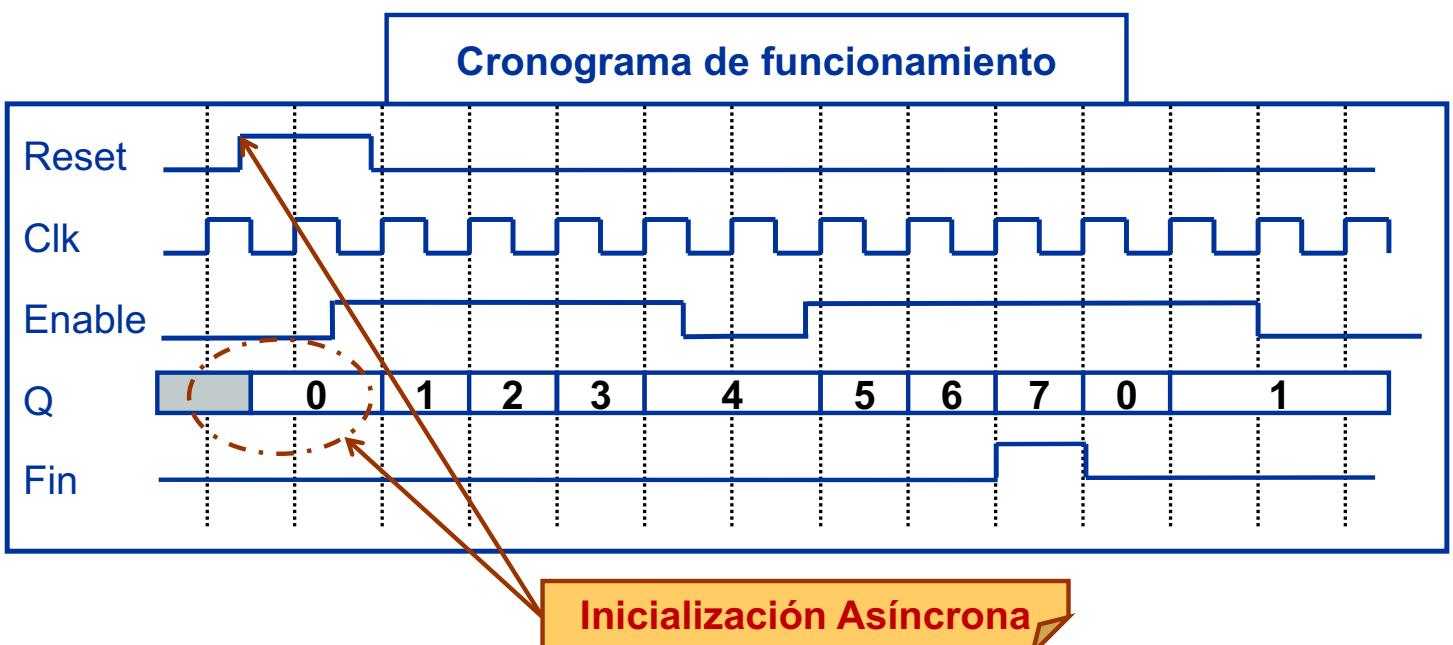


Contadores síncronos

Cronograma de funcionamiento

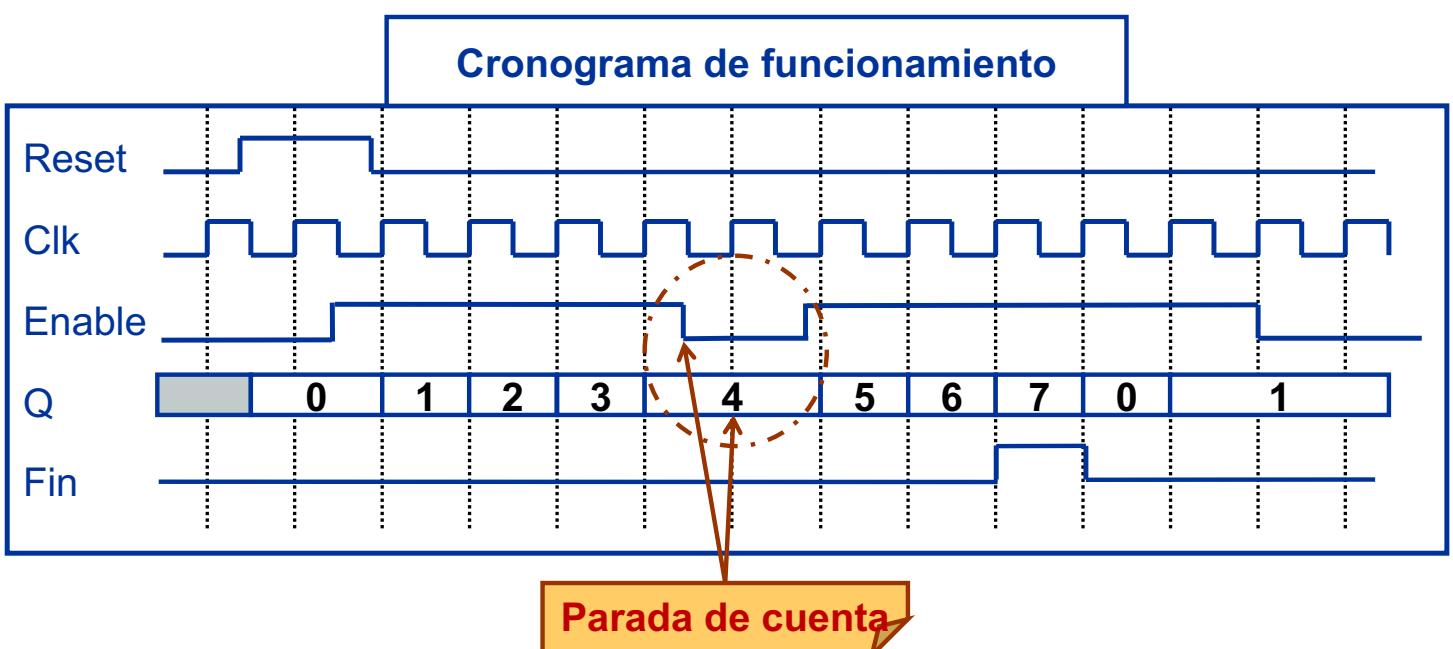


Contadores síncronos



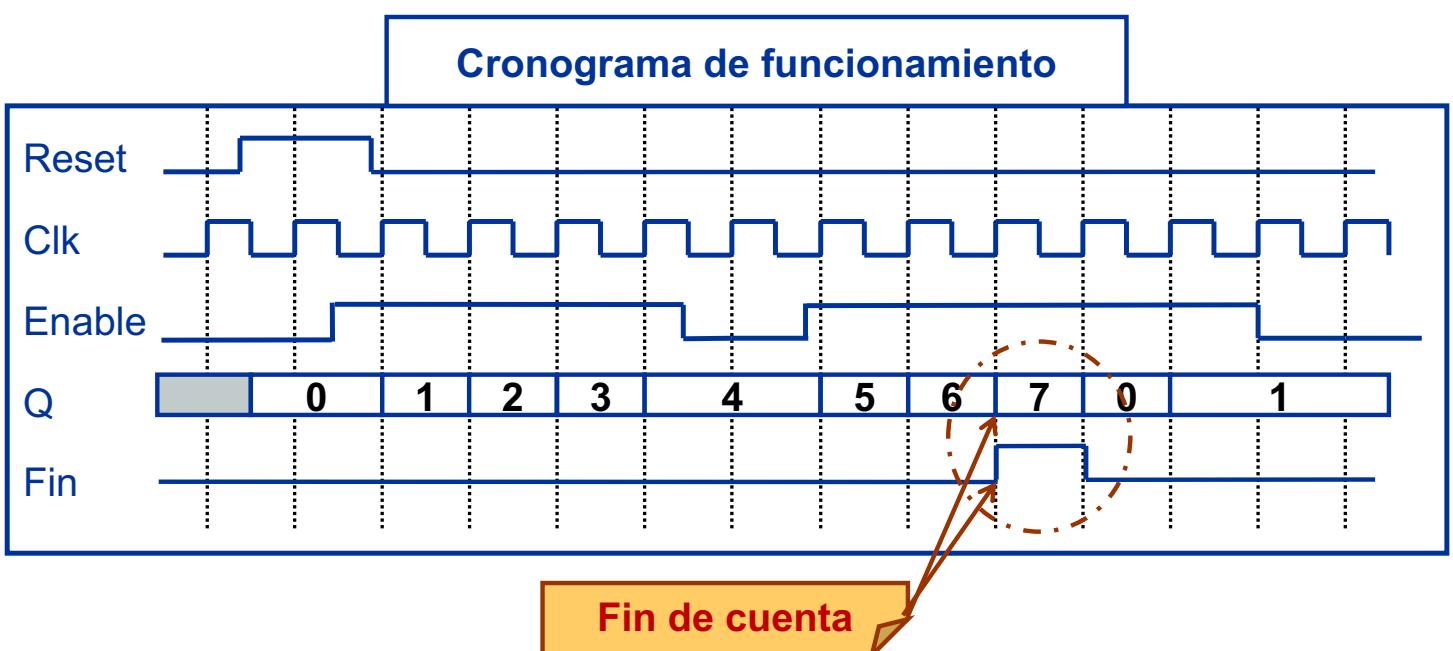


Contadores síncronos





Contadores síncronos





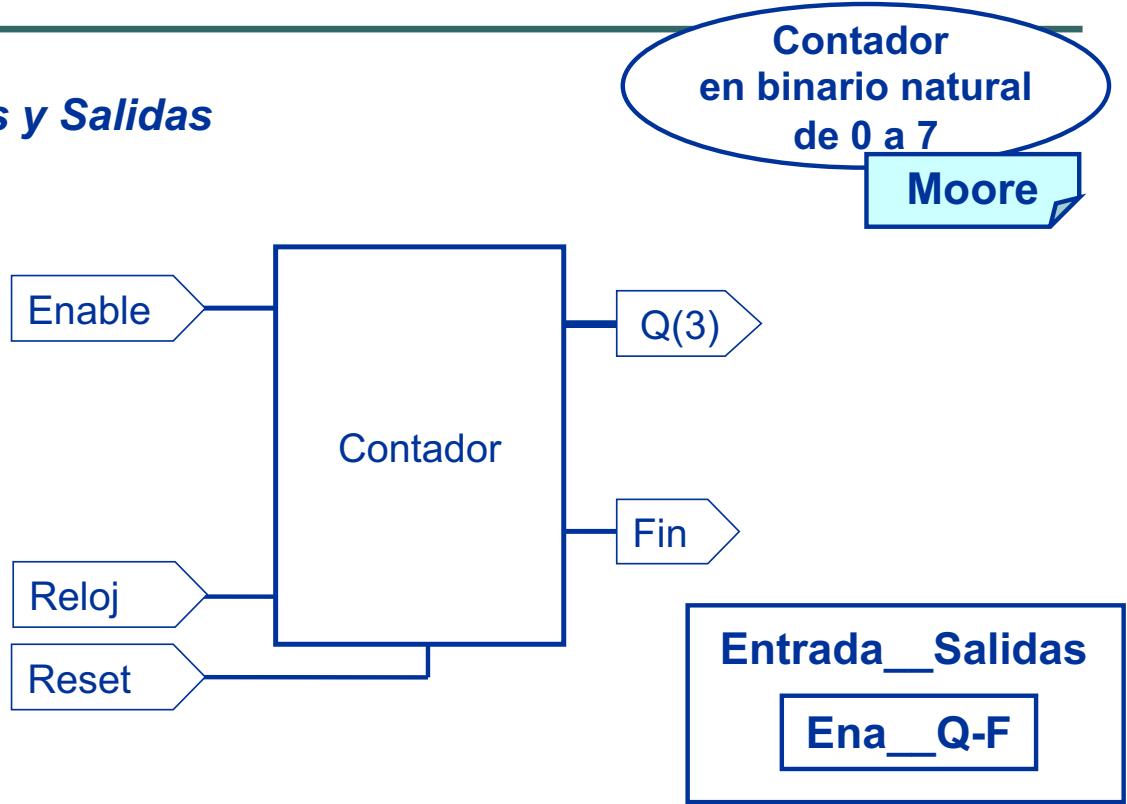
Contadores síncronos como FSM

1. *Entradas y Salidas*
2. *Diagrama de estados. Asignación de estados. Biestables*
3. *Tabla de transiciones*
4. *Optimización*
5. *Esquemático*



Contadores síncronos como FSM

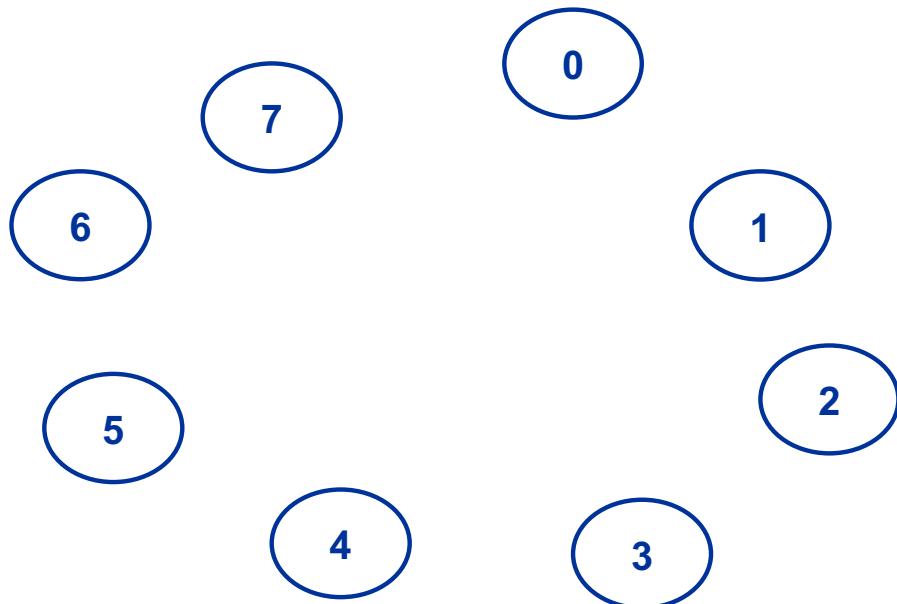
1. Entradas y Salidas





Contadores síncronos como FSM

2. Diagrama de estados

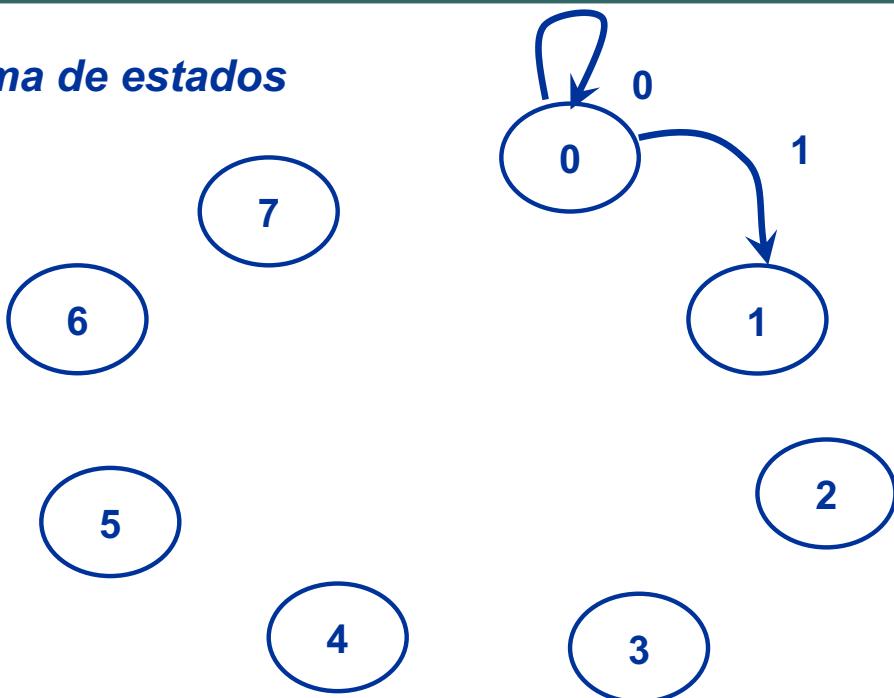




Contadores síncronos como FSM

2. Diagrama de estados

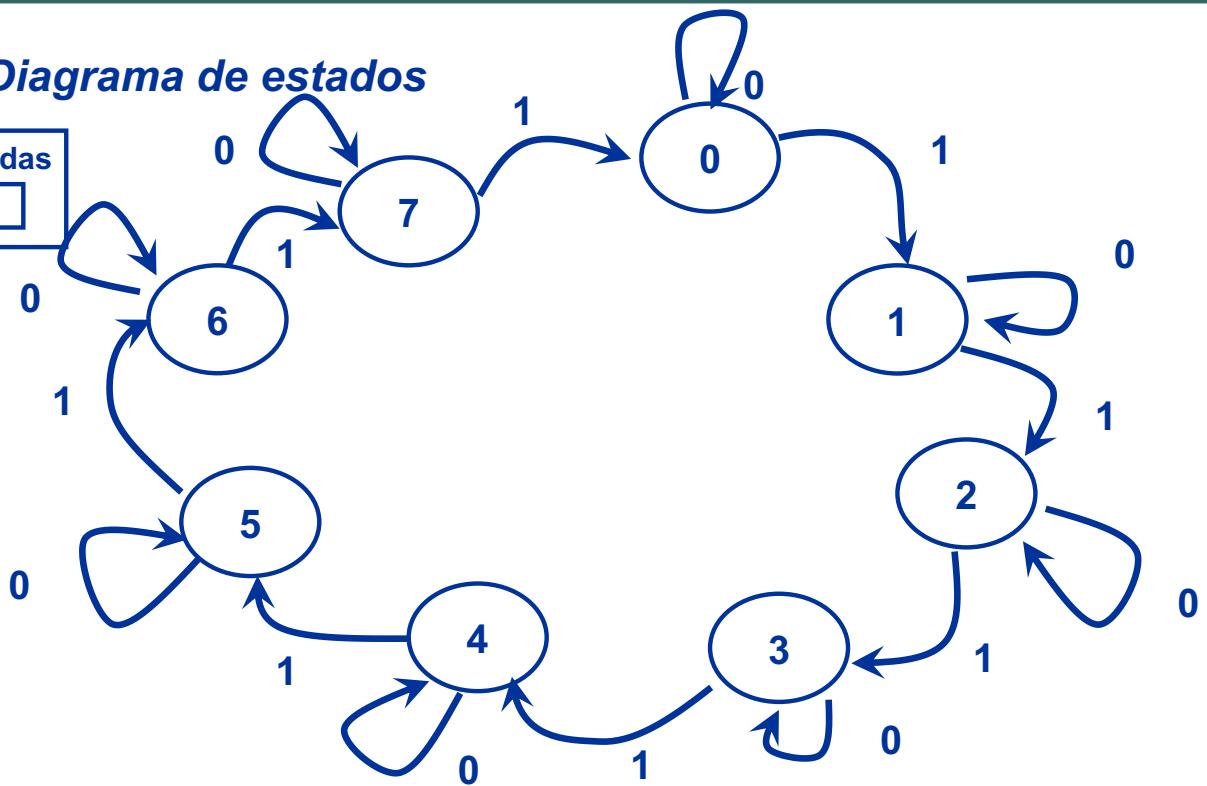
Entrada_Salidas
Ena_Q-F



Contadores síncronos como FSM

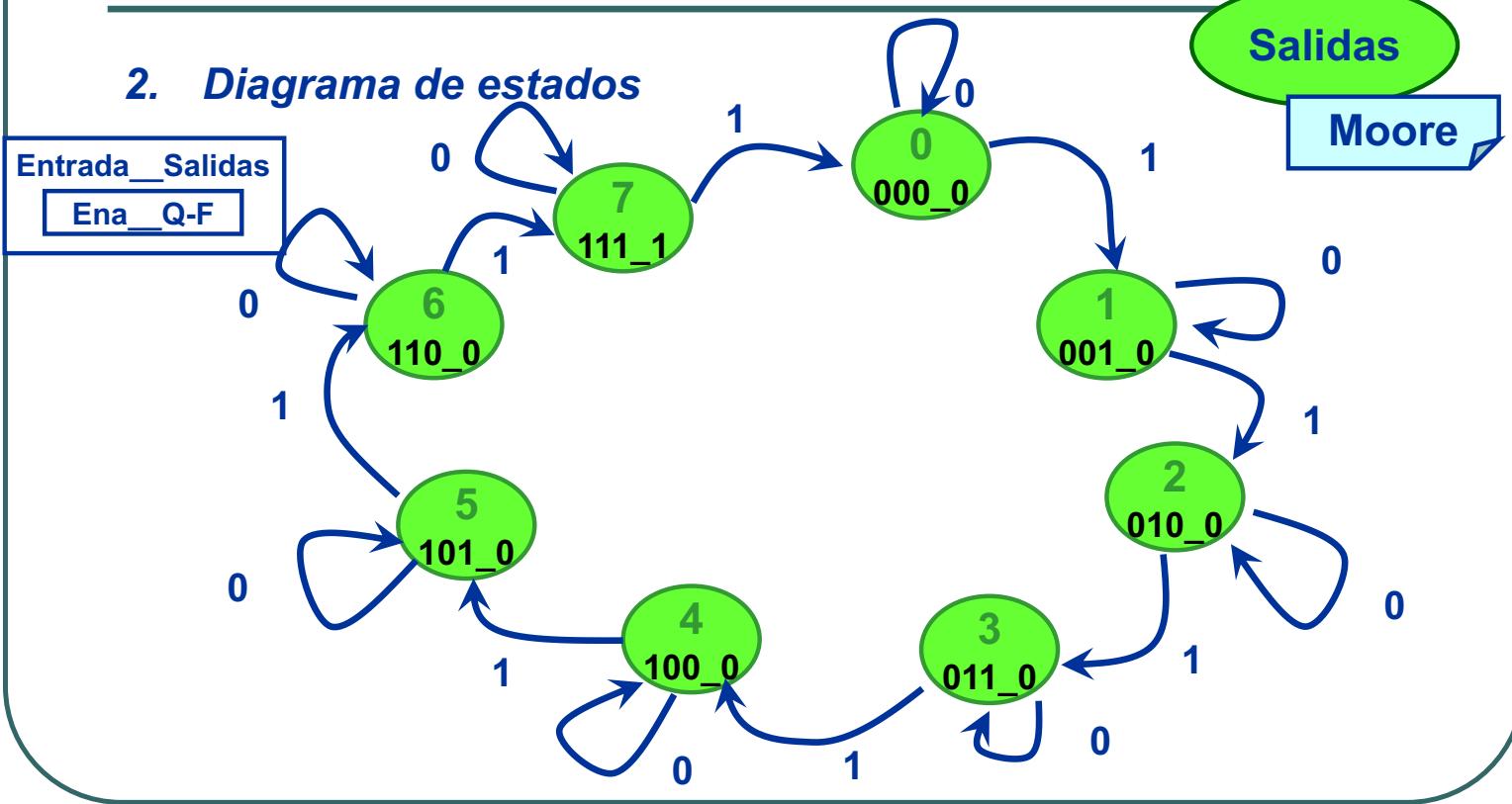
2. Diagrama de estados

Entrada_Salidas
Ena_Q-F



Contadores síncronos como FSM

2. Diagrama de estados





Contadores síncronos como FSM

2. Asignación de estados

7 estados → 3 biestables

Estado	Codificación
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111



Contadores síncronos como FSM

Entrada +Estado				Estado siguiente			Entradas T			Salidas			
E	Q2	Q1	Q0	Q2	Q1	Q0	T2	T1	T0	Q2	Q1	Q0	Fin
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	1	0	0	0	0	0	1	0
0	0	1	0	0	1	0	0	0	0	0	1	0	0
0	0	1	1	0	1	1	0	0	0	0	1	1	0
0	1	0	0	1	0	0	0	0	0	1	0	0	0
0	1	0	1	1	0	1	0	0	0	1	0	1	0
0	1	1	0	0	1	1	0	0	0	1	1	0	0
0	1	1	1	1	1	1	0	0	0	1	1	1	0



Contadores síncronos como FSM

Entrada +Estado				Estado siguiente			Entradas T			Salidas			
E	Q2	Q1	Q0	Q2	Q1	Q0	T2	T1	T0	Q2	Q1	Q0	Fin
1	0	0	0	0	0	1	0	0	1	0	0	0	0
1	0	0	1	0	1	0	0	1	1	0	0	1	0
1	0	1	0	0	1	1	0	0	1	0	1	0	0
1	0	1	1	1	0	0	1	1	1	0	1	1	0
1	1	0	0	1	0	1	0	0	1	1	0	0	0
1	1	0	1	1	1	0	0	1	1	1	0	1	0
1	1	1	0	1	1	1	0	0	1	1	1	0	0
1	1	1	1	0	0	0	1	1	1	1	1	1	1



Contadores síncronos como FSM

4. Optimización

T2

Q1Q0 EnQ2 \	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	0	0	1	0
10	0	0	1	0

T1

Q1Q0 EnQ2 \	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	0	1	1	0
10	0	1	1	0



Contadores síncronos como FSM

4. Optimización

T0

Q1Q0 EnQ2	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	1	1	1	1
10	1	1	1	1

Fin

Q1Q0 EnQ2	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	0	0	1	0
10	0	0	0	0



Contadores síncronos como FSM

4. Optimización

T0 = Enable

T1 = Q0·Enable

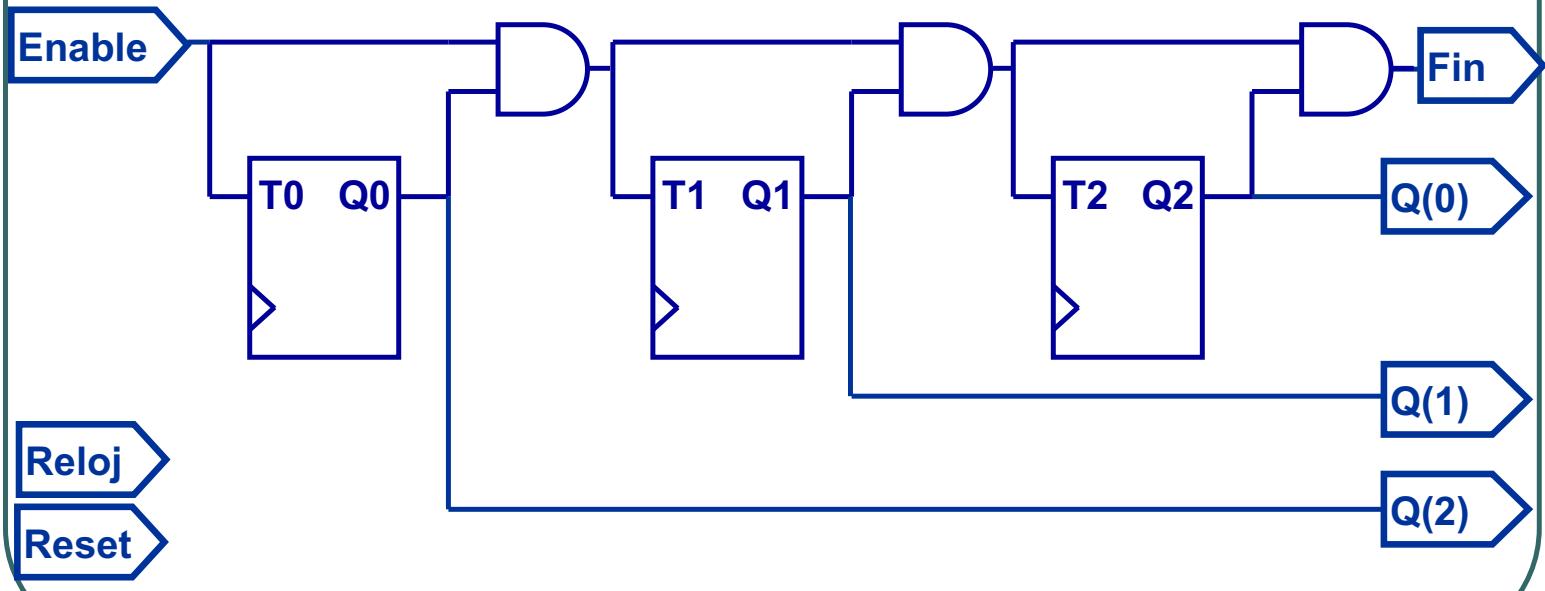
T2 = Q2·Q1·Enable

Fin = Q2·Q1·Q0·Enable

Contadores síncronos como FSM

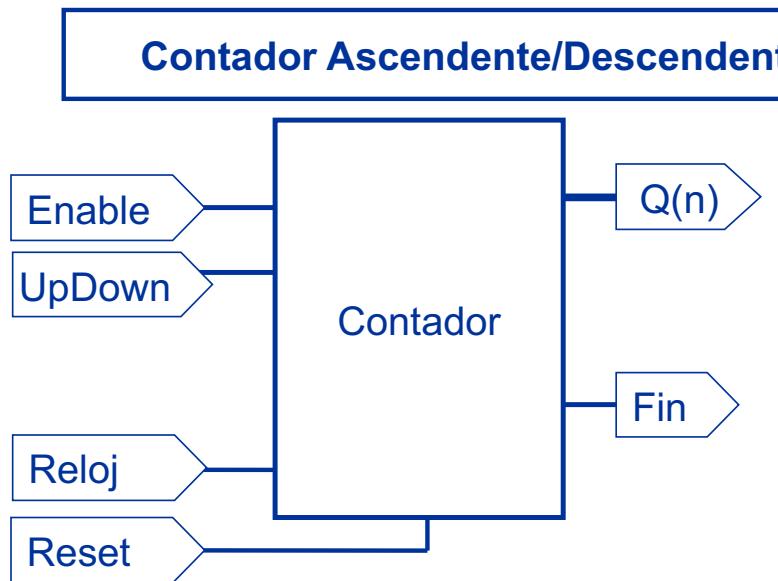
Esquema

5. Esquemático



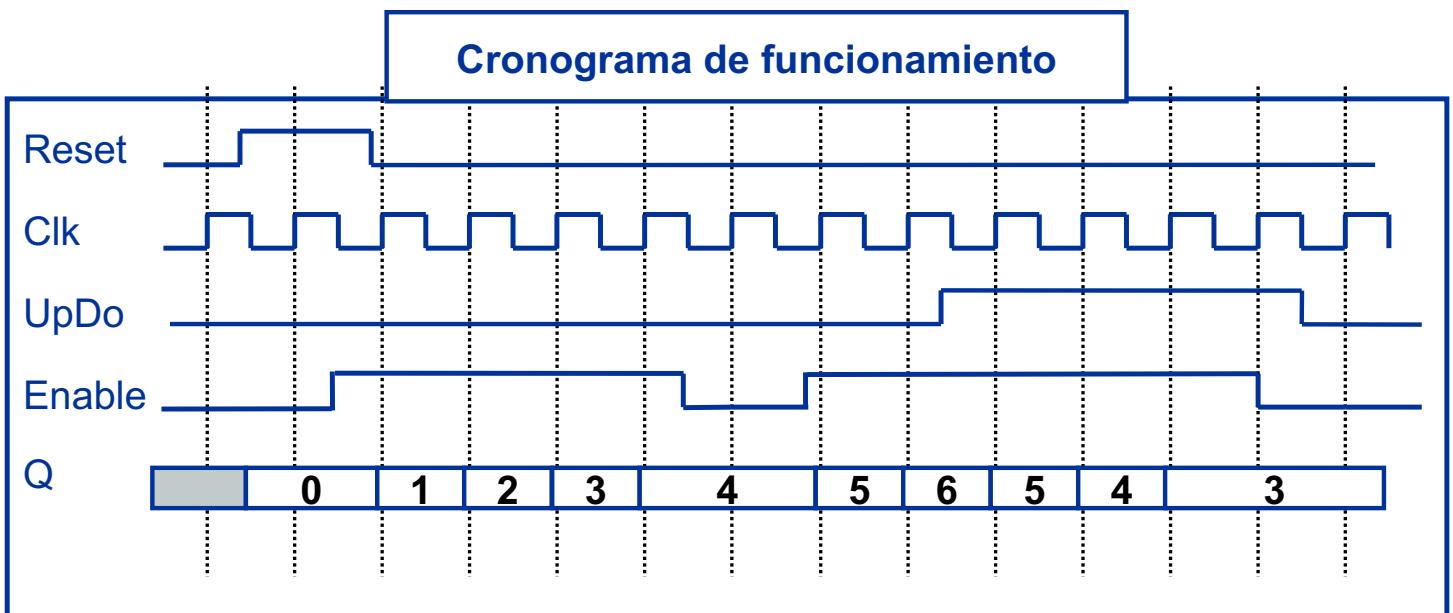


Contadores síncronos

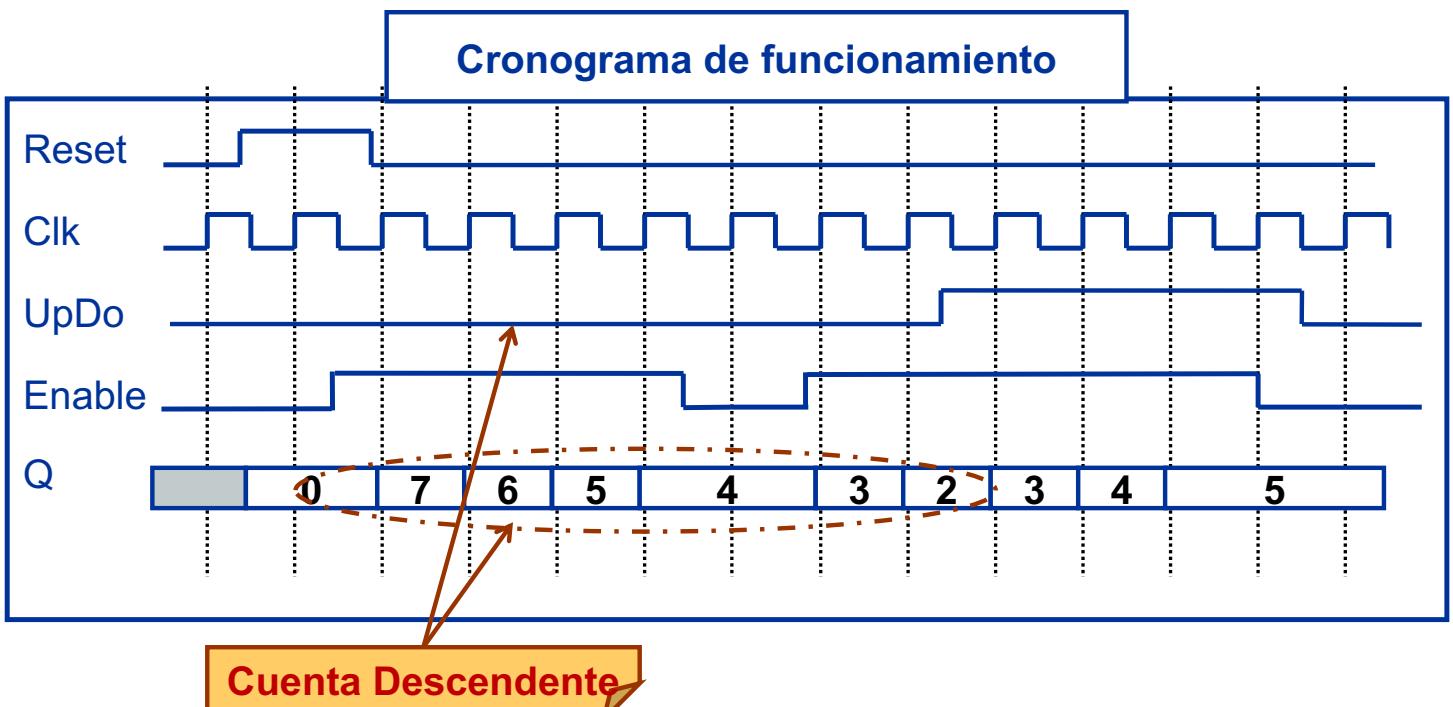




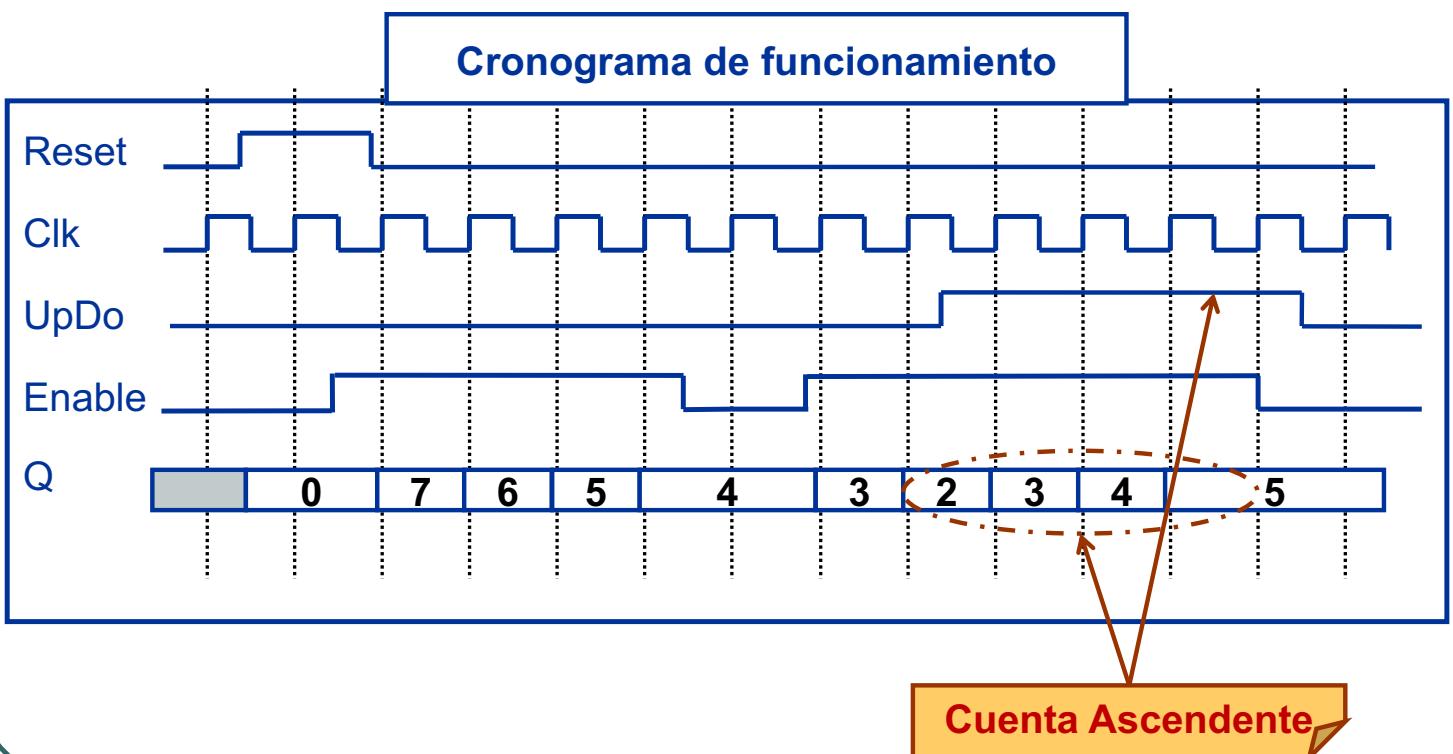
Contadores síncronos (Asc/Desc)



Contadores síncronos (Asc/Desc)



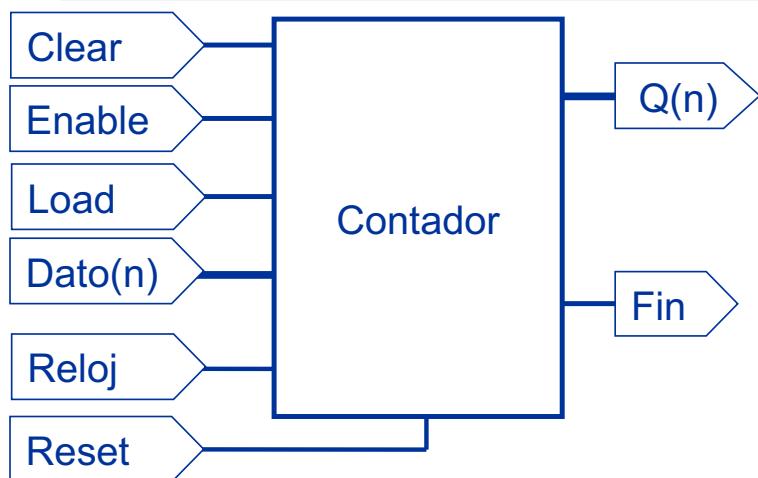
Contadores síncronos (Asc/Desc)





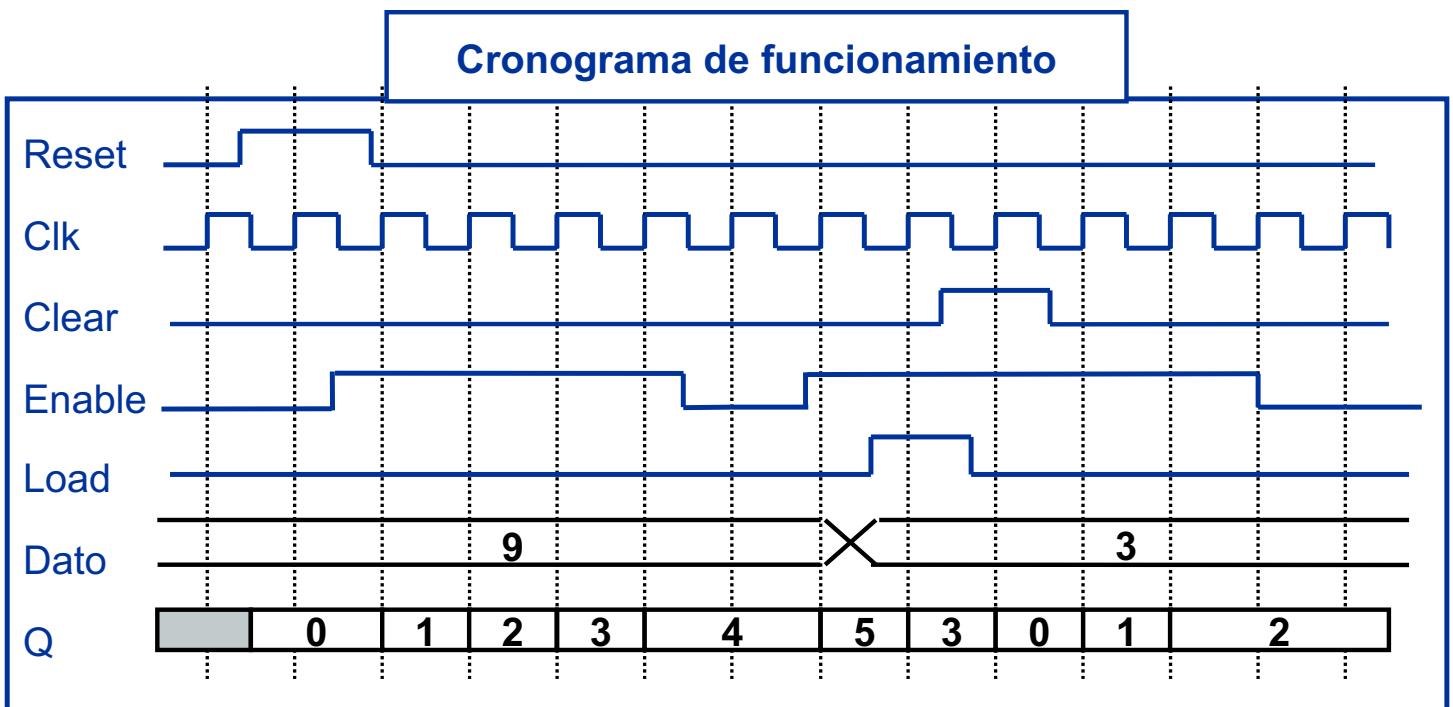
Contadores síncronos

Contador con Precarga /Clear síncrono



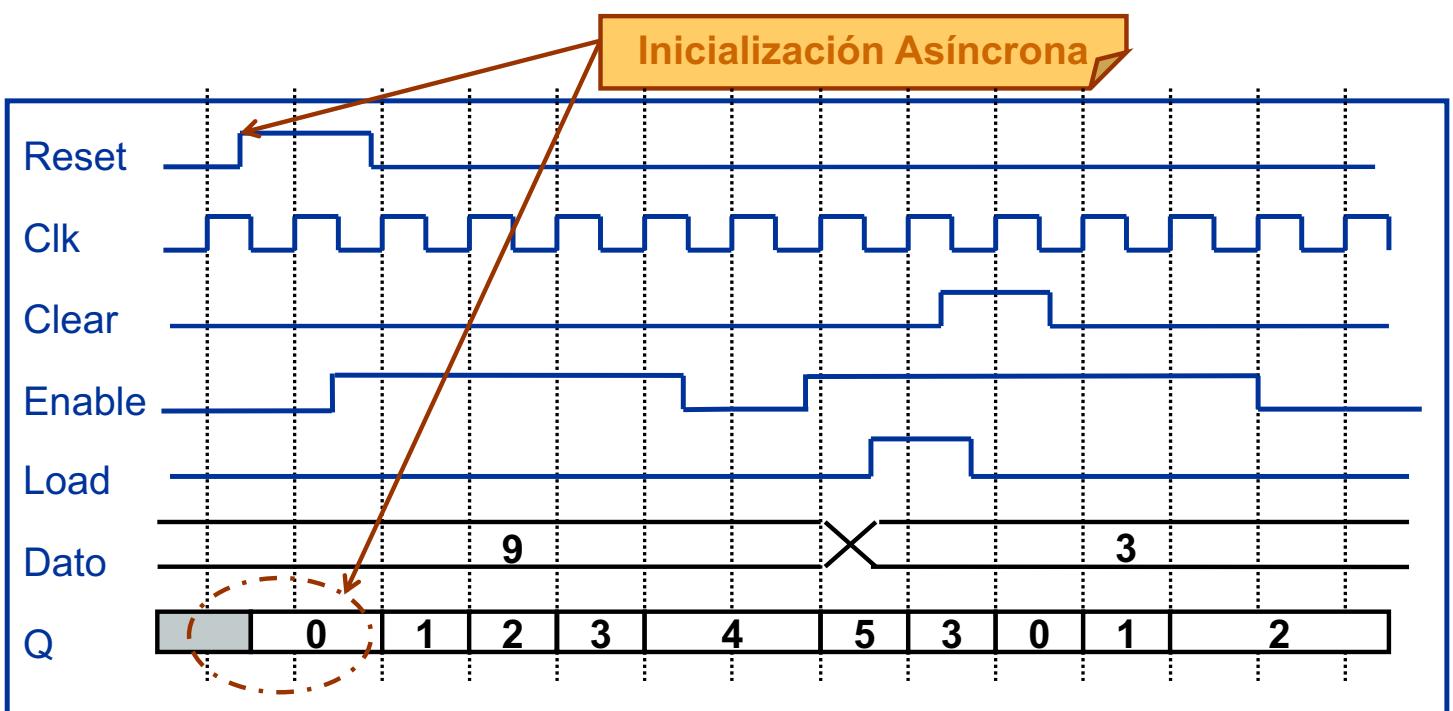


Contadores síncronos

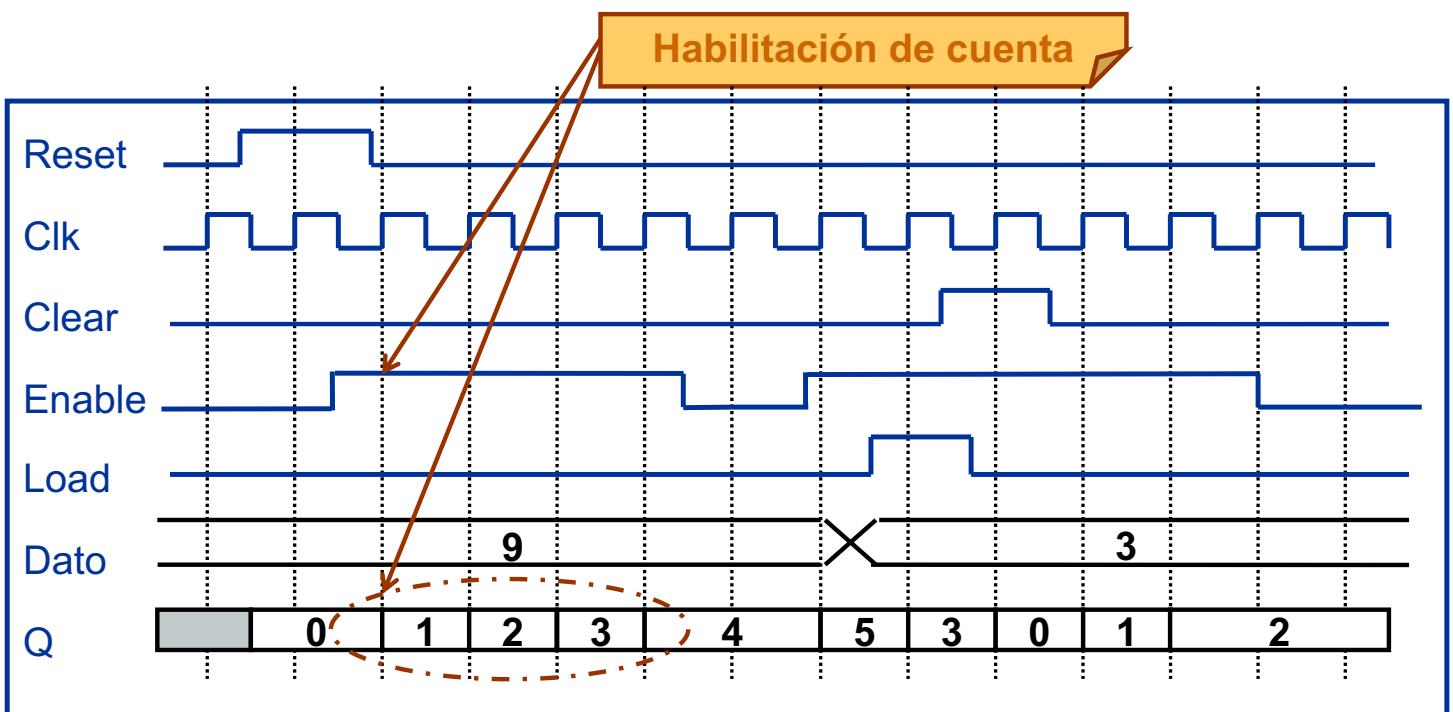




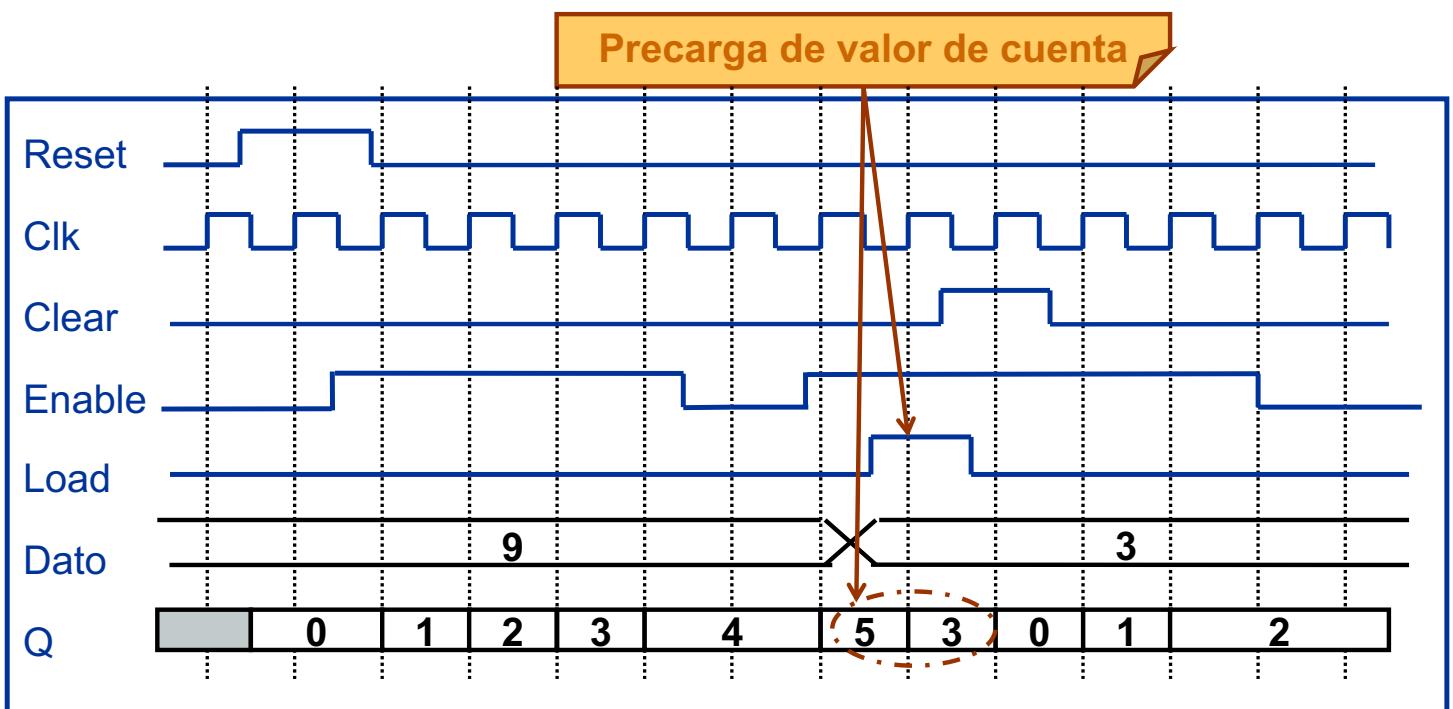
Contadores síncronos



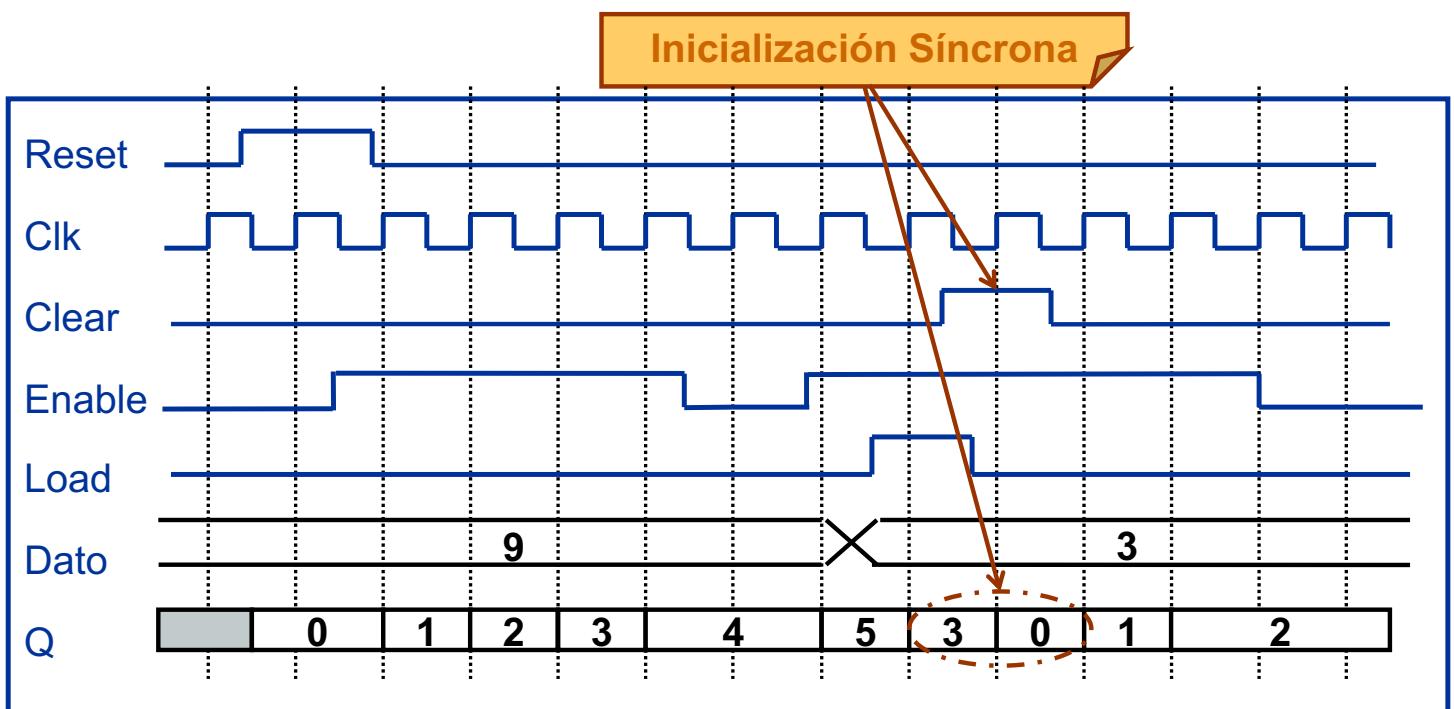
Contadores síncronos



Contadores síncronos



Contadores síncronos

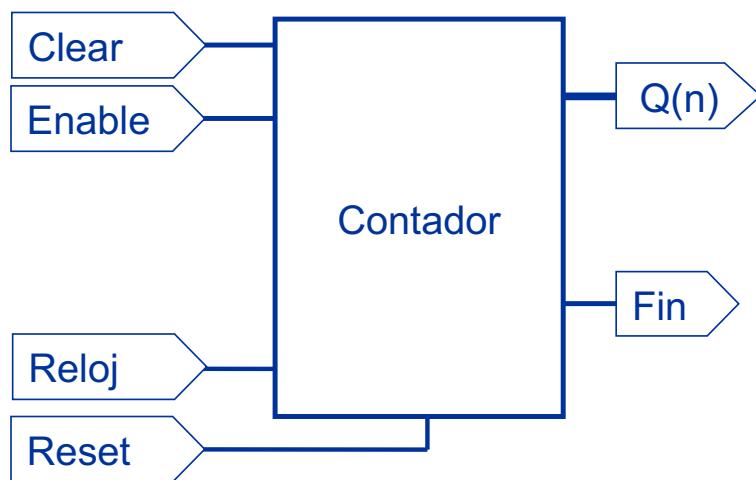




Contadores síncronos (FSM)

Contador con entrada Clear

Ejercicio



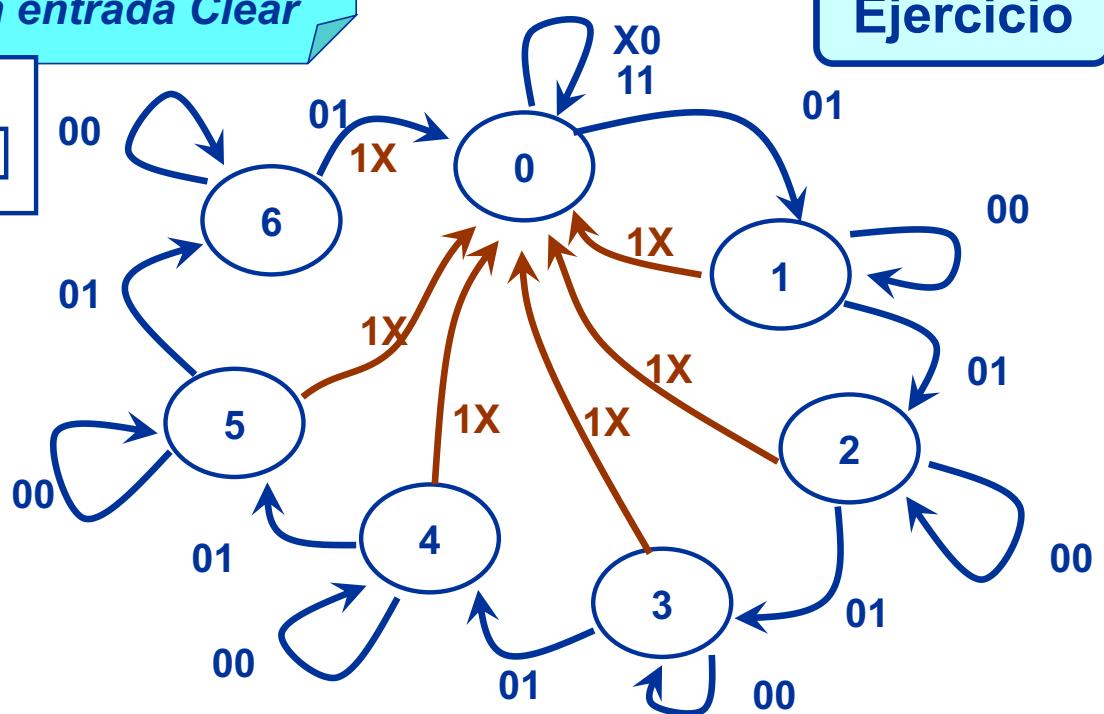
Contadores síncronos (FSM)

Contador con entrada Clear

Entradas_Salidas

Clear-Ena_Q-Fin

Ejercicio





Contadores síncronos (FSM)

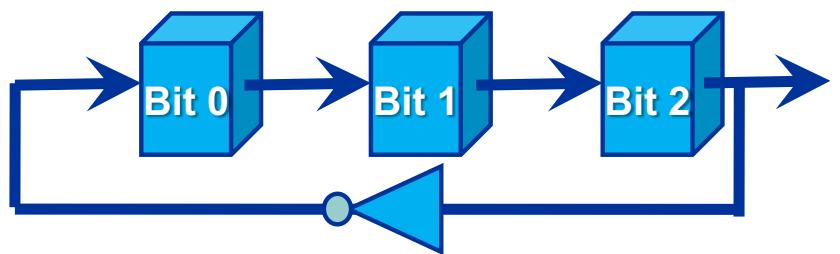
Contador con entrada Clear

Ejercicio

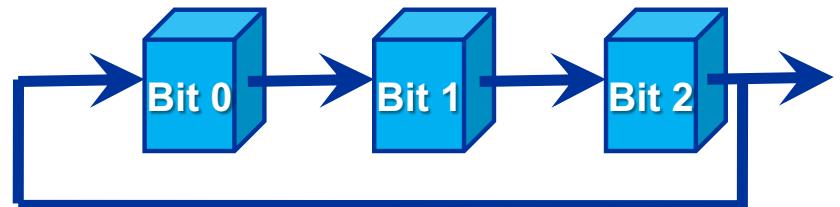
Contadores basados en registros de desplazamiento

“Un contador basado en un registro de desplazamiento es básicamente un registro de desplazamiento con la salida serie realimentada a la entrada serie, de modo que se generen secuencias especiales” (Floyd)

Contador Johnson



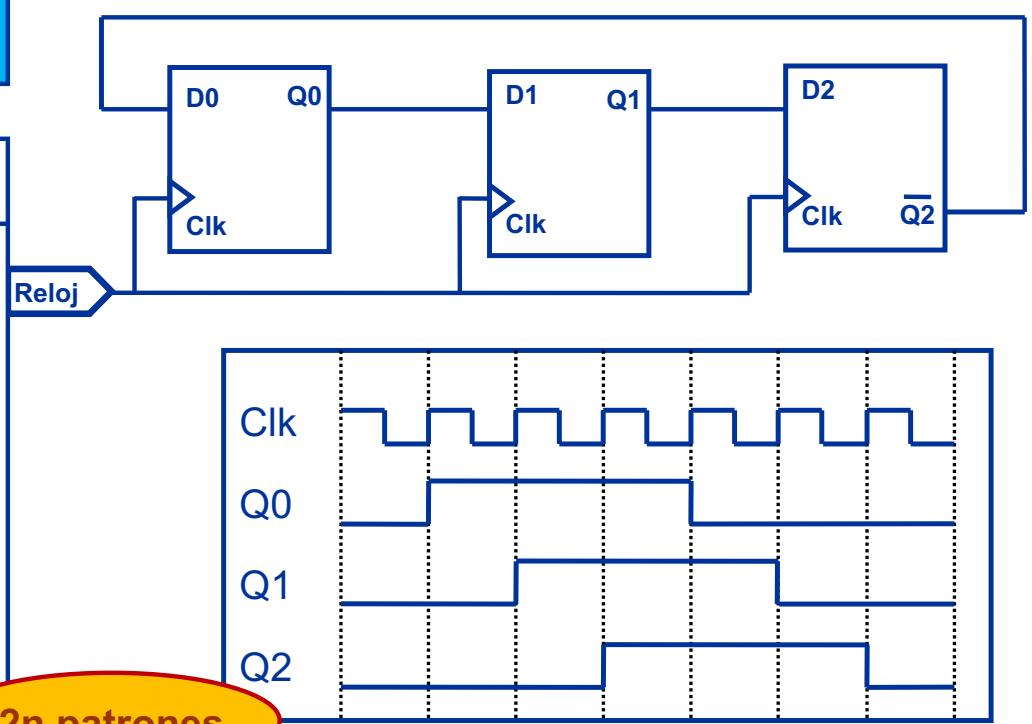
Contador en anillo



Contadores basados en registros de desplazamiento

Contador Johnson

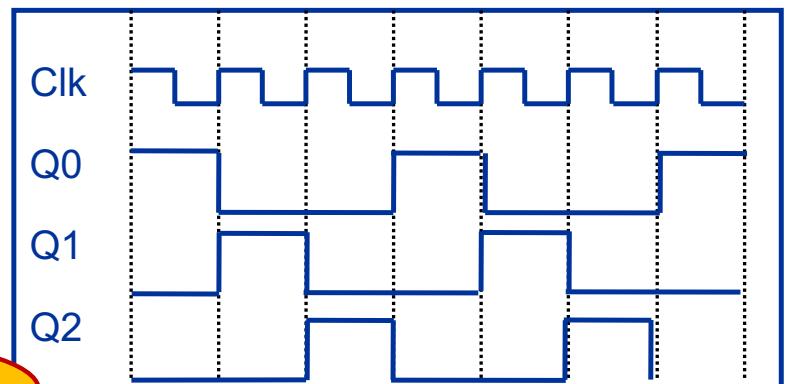
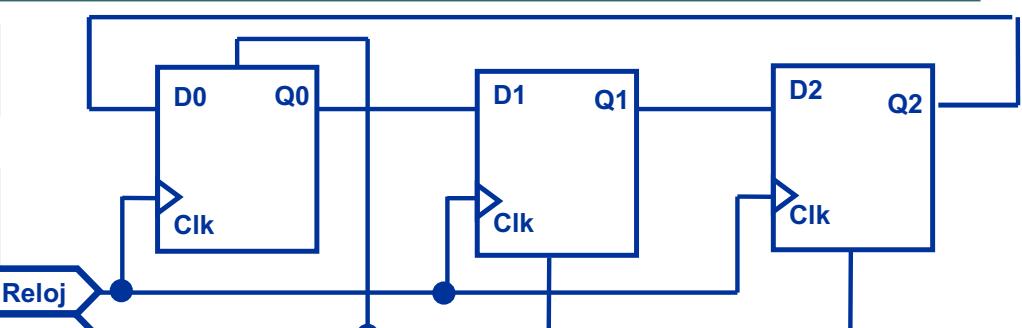
Ciclo	Q2	Q1	Q0
0	0	0	0
1	0	0	1
2	0	1	1
3	1	1	1
4	1	1	0
5	1	0	0
6	0	0	0
7	0	0	1



Contadores basados en registros de desplazamiento

Contador en anillo

Ciclo	Q2	Q1	Q0
0	0	0	1
1	0	1	0
2	1	0	0
3	0	0	1
4	0	1	0
5	1	0	0
6	0	0	1
7	0	1	0





Bibliografía

- “Circuitos y Sistemas Digitales”. J. E. García Sánchez, D. G. Tomás, M. Martínez Iniesta. Ed. Tebar-Flores
- “Electrónica Digital”, L. Cuesta, E. Gil, F. Remiro, McGraw-Hill
- “Fundamentos de Sistemas Digitales “, T.L Floyd, Prentice-Hall



Memorias

© Luis Entrena, Celia López, Mario García,
Enrique San Millán

Universidad Carlos III de Madrid



Índice

- Introducción. Tipos de memorias
- Características de las memorias
- Organización interna de una memoria
- Memorias de lectura y escritura (RAM)
- Memorias de sólo lectura (ROM)
- Expansión del tamaño de palabra y de capacidad de las memorias
- Cronogramas de acceso a memoria
- Otras aplicaciones de las memorias



Memorias

- Dispositivos para almacenamiento masivo de información
- Son un componente fundamental de los sistemas digitales
- Existen numerosos tipos de memorias. Se pueden clasificar según diferentes parámetros:
 - Propiedad física utilizada para el almacenamiento de la información
 - Características de acceso, permanencia de la información, etc.



Tipos de memorias

- Memorias magnéticas:
 - Patrones de magnetización sobre una superficie cubierta de un material magnetizable
 - Ejemplos: disco duro, disquete, cinta magnética, etc.
- Memorias ópticas:
 - La información se graba con un láser que genera minúsculas perforaciones sobre una superficie. La lectura se realiza iluminando con un láser y midiendo la reflexión
 - Ejemplos: CD, DVD
- Memorias de semiconductores
 - Circuitos electrónicos
 - *Nos centraremos en este tipo de memorias*



Memorias de semiconductores

- Memorias RAM (Random Access Memory)
 - Memorias de Lectura y Escritura
 - Ejemplo: memoria principal de un ordenador
- Memorias ROM (Read Only Memory)
 - Memorias de sólo lectura
 - Los contenidos están fijados de fábrica o pueden ser programados
 - Ejemplo: memoria Flash



Características de las memorias

- Capacidad: cantidad de información que es capaz de almacenar
 - Tamaño de palabra: número de bits que se pueden acceder de una vez
 - Normalmente potencias de 2: 1, 2, 4, 8, 16, 32
 - 1 Byte = 1B = 8 bits
 - Número de palabras: Normalmente potencias de 2
 - $2^{10} = 1.024 = 1K$ (Kilo)
 - $2^{20} = 1.048.576 = 1M$ (Mega)
 - $2^{30} = 1.073.741.824 = 1G$ (Giga)
 - $2^{40} = 1.099.511.627.776 = 1T$ (Tera)
 - Capacidad = <número de palabras>x<tamaño de palabra>
 - Ejemplo: 16Mx8



Características de las memorias

- Tiempo de acceso:
 - Tiempo que se necesita para acceder un dato en la memoria
 - Puede ser diferente para lectura y para escritura
- Modo de acceso:
 - Secuencial o serie: sólo se puede acceder en un orden determinado. El tiempo de acceso varía dependiendo de la posición que se desea acceder
 - Ejemplo: cinta magnética
 - Aleatorio: se puede acceder en cualquier orden. El tiempo de acceso es el mismo para todas las posiciones de memoria
 - Ejemplo: memoria principal de un ordenador



Características de las memorias

- Permanencia o estabilidad de los datos:
 - Memoria no volátil: Mantiene la información almacenada aunque esté desconectada de una fuente de alimentación
 - Ejemplo: memoria Flash
 - Memoria volátil: Si se desconecta de la fuente de alimentación, se borra
 - Ejemplo: memoria RAM estática (memoria cache)
 - Memoria dinámica: Pierde la información al cabo de un cierto tiempo, incluso aunque esté alimentada.
 - Necesita refresco periódico de la información almacenada
 - Ejemplo: memoria RAM dinámica (memoria principal)

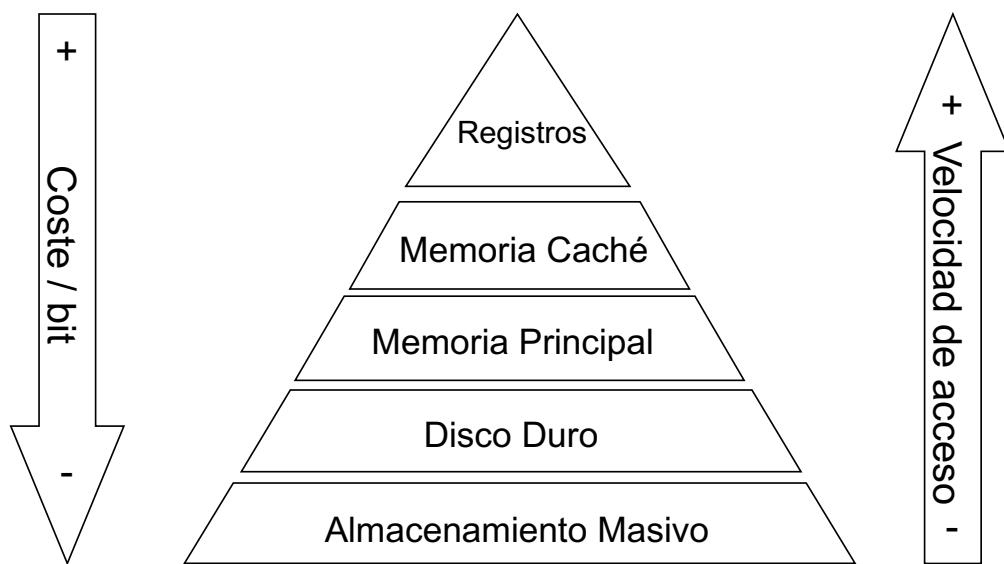


Características de las memorias

- Otras características que pueden determinar la elección de una memoria para una aplicación determinada:
 - Coste/bit
 - Consumo
- ¡No hay ningún tipo de memoria que sea la mejor respecto a todas las características!
 - Es necesario utilizar la memoria más adecuada para cada aplicación o una combinación jerárquica de memorias

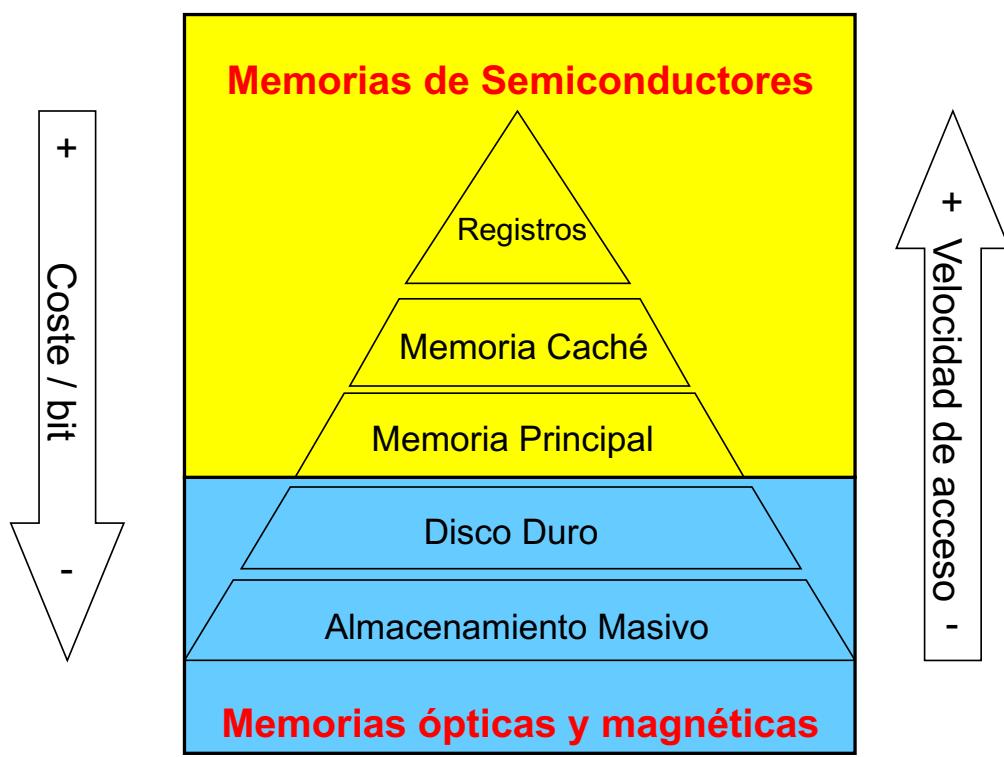


Jerarquía de memoria



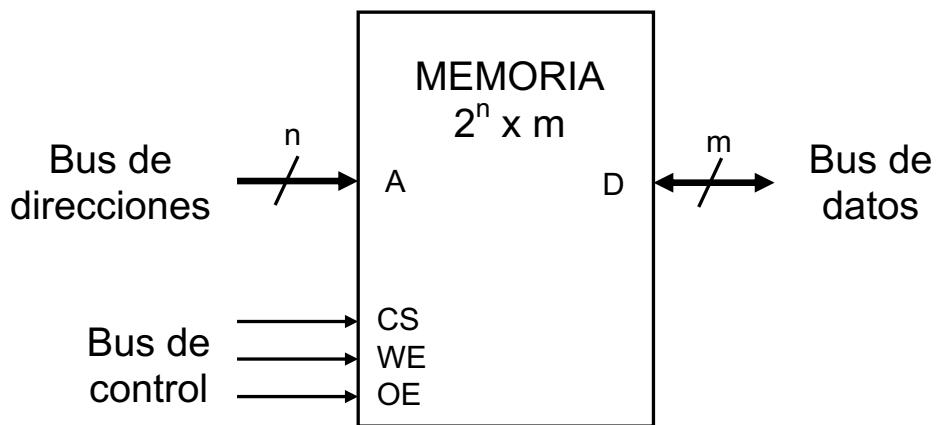


Jerarquía de memoria





Organización de una memoria: interfaz





Organización de una memoria: interfaz

- Bus de direcciones (A): Indica la posición que se desea acceder
 - Tiene n bits para una memoria de 2^n posiciones
 - Ejemplo: 20 bits para 1M, 30 bits para 1G
- Bus de Datos (D): Proporciona el dato
 - El ancho del bus de datos (m) es igual al tamaño de palabra
 - En escritura es un dato de entrada
 - En lectura es un dato de salida
 - Puede ser un único bus bidireccional, o dos buses, uno de entrada y otro de salida

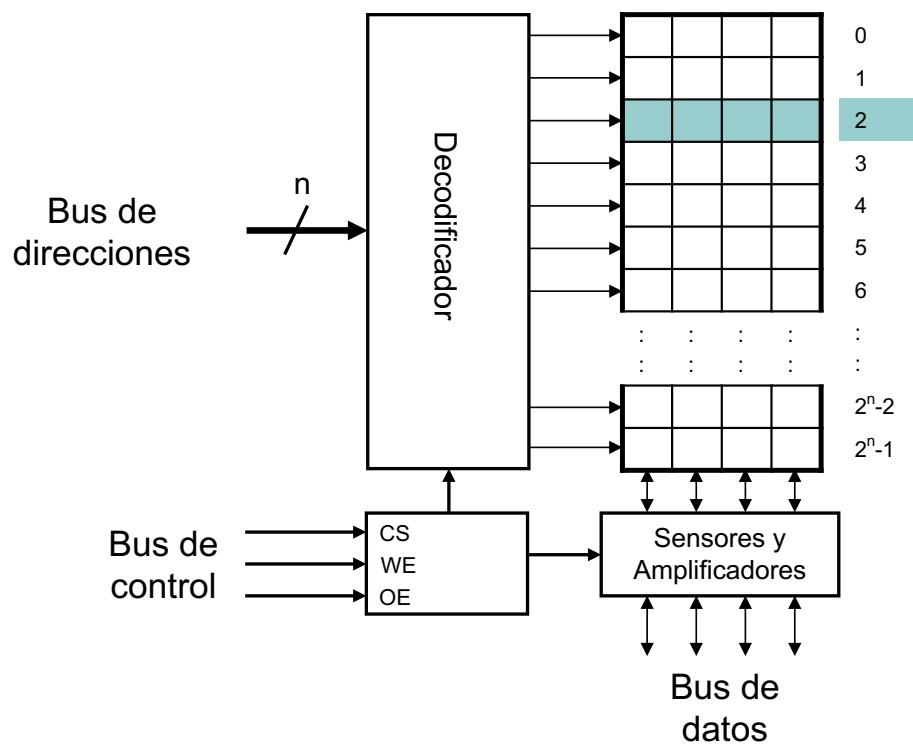


Organización de una memoria: interfaz

- Bus de control: señales que controlan la operación de la memoria. Algunas señales típicas:
 - CS (Chip Select) o CE (Chip Enable): Habilita el acceso. Si no se activa, el bus de datos se pone típicamente en triestado
 - R/W (Read/Write) o WE (Write Enable): Selecciona la operación a realizar (lectura o escritura)
 - OE (Output Enable): Habilita la salida de datos. Si no se activa, el bus de datos se pone típicamente en triestado
 - RAS (Row Address Strobe) y CAS (Column Address Strobe): En memorias con organización 3D
 - Otras señales: CLK en memorias síncronas, BE (Byte Enable) para selección de bytes dentro de una palabra, etc...

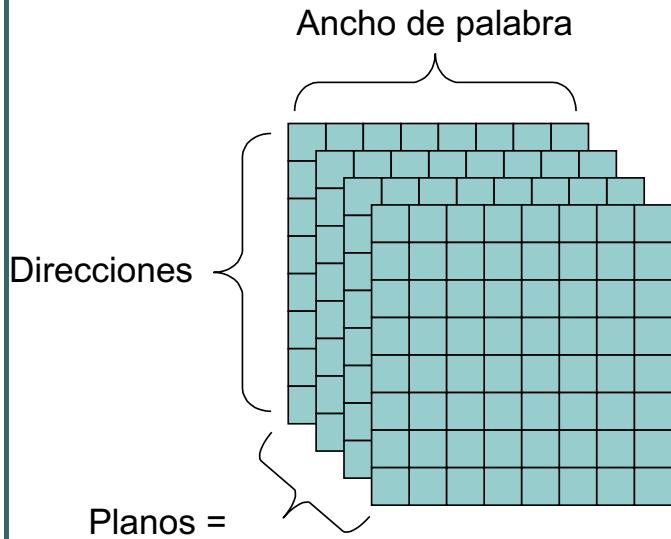


Organización de una memoria: estructura interna





Organización 3D

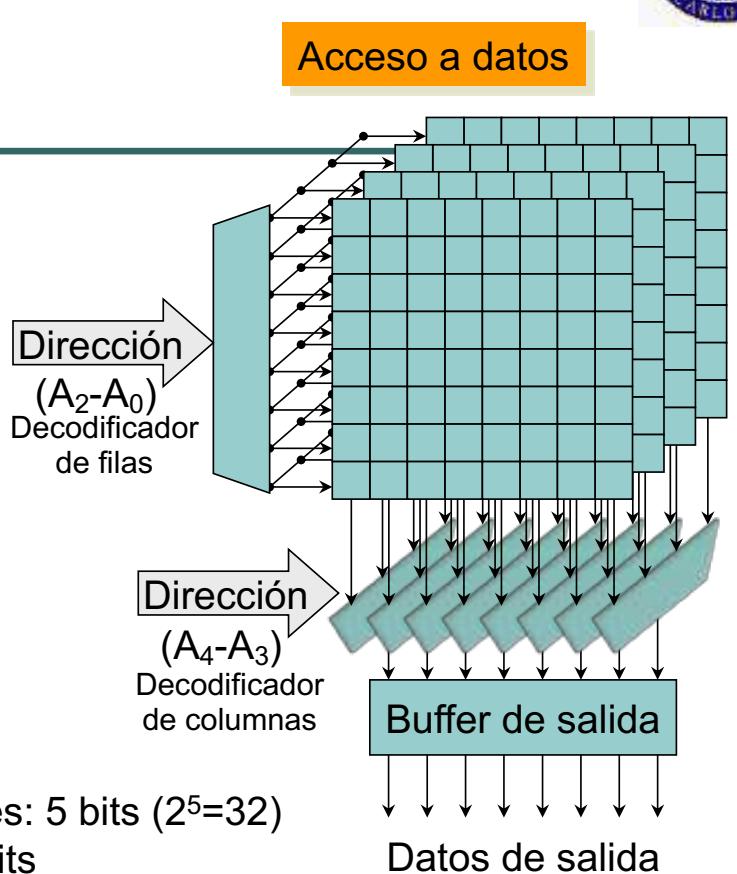


Memoria: 32×8

Bus de direcciones: 5 bits ($2^5=32$)

Bus de datos: 8 bits

Capacidad: $32 \cdot 8 = 256$ bits





Memorias RAM

- Memorias de lectura y escritura
- Dos tipos:
 - RAM estática (SRAM): cada bit se almacena en un biestable
 - RAM dinámica (DRAM): cada bit se almacena en un condensador
- La memoria DRAM más usada actualmente es del tipo DDR (Double Data Rate) SDRAM
 - Organización 3D
 - Utiliza reloj (Synchronous DRAM)
 - Accede a los datos en los dos flancos de reloj (Double Data Rate)
 - Ancho de 64 bits
 - Versiones evolucionando en el tiempo: DDR, DDR2, DDR3, DDR4



Memorias RAM

- Ventajas e inconvenientes:

	SRAM	DRAM
Tamaño (nº de bits)		✓
Velocidad	✓	
Coste/bit		✓
Consumo		✓
Volátil	SI	SI
Refresco	NO	SI
Ejemplo de utilización	Cache	Memoria principal



Memorias ROM

- Memorias de sólo lectura
- Son no volátiles
- Tipos
 - No programables: contenidos fijados en fabricación
 - Programables: el usuario puede fijar los contenidos
 - Borrables o reprogramables: puede borrarse para almacenar otros contenidos
- Conviene distinguir entre los conceptos de “escritura” y “programación”, aunque la frontera es cada vez más difusa:
 - Escritura: es una operación similar a la lectura
 - Programación: utiliza un mecanismo físico diferente que la lectura, suele ser mucho más lenta y aplicable solo por bloques o incluso para el chip entero



Tipos de memorias ROM

Tipo	Significado	Lectura	Programable	Borrable (Reprogramable)
ROM	Read Only Memory	Aleat.	No (por máscara)	No
PROM	Programmable ROM	Aleat.	Una vez	No
EPROM	Erasable Programmable ROM	Aleat.	Electricamente	Por luz UV
EEPROM	Electrically Erasable Programmable ROM	Aleat.	Electricamente	Electricamente
NOR Flash	Celdas en paralelo	Aleat.	Electricamente	Electricamente
NAND Flash	Celdas en serie	Serie	Electricamente	Electricamente



Expansión de memorias

- ¿Cómo construir memorias grandes a partir de unas pastillas de memoria?
 - Expansión del bus de datos
 - Expansión del bus de direcciones
 - Expansión del bus de datos y del bus de direcciones
- La expansión es una práctica habitual (ejemplo: modulos DIMM)



Expansión del tamaño de palabra

- Buses de direcciones y control comunes
- Bus datos se forma con la unión de los buses de datos:
 - Cada pastilla aporta una porción de los datos

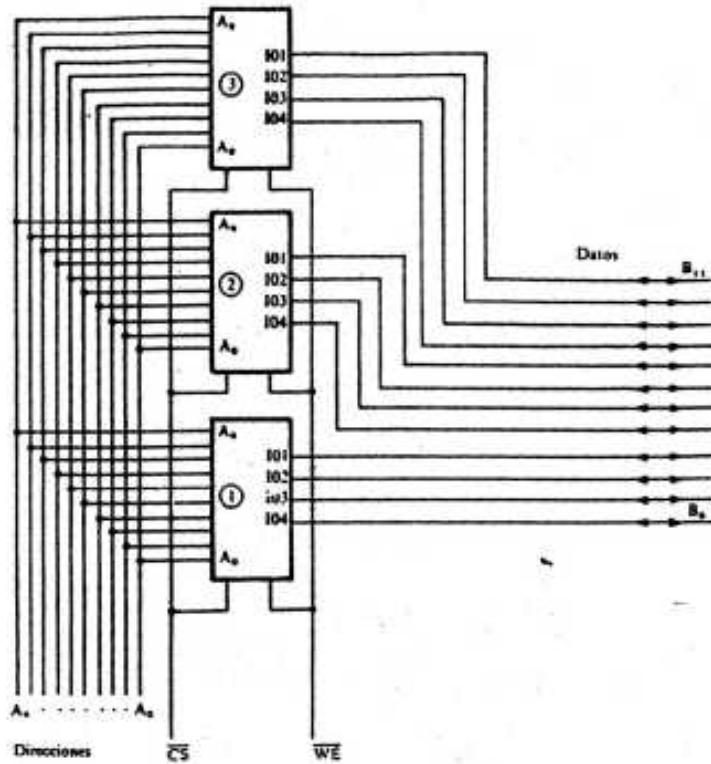


Figura 1.13. Con 3 chips 2114 (1K x 4) se construye un subsistema de memoria de 1K x 12 bits.

Expansión de la capacidad

- Buses de control y de datos comunes
- Bus de direcciones:
 - Cada pastilla aporta una porción del espacio de direcciones
 - Parte alta se decodifica para seleccionar la pastilla
 - Parte baja común: selecciona la dirección dentro de la pastilla

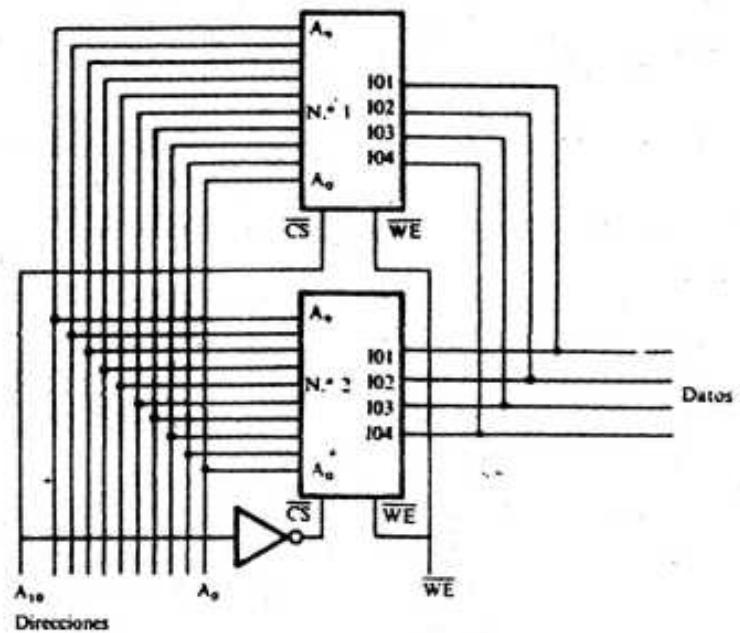


Figura 1.14. Agrupación de dos chips de $1K \times 4$ (tipo 2114) para formar un conjunto de $2K \times 4$.

Expansión de tamaño de palabra y capacidad

- Combinación de las anteriores

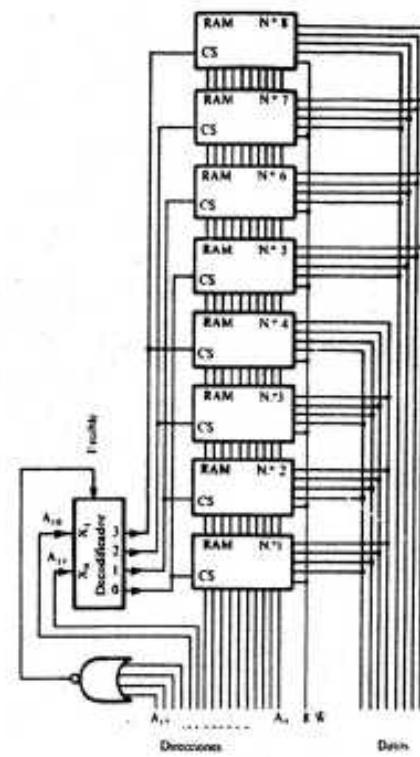


Figura 1.15. Con chips 2114 (de 1K × 4) se realiza un subsistema de memoria RAM de 4K × 8.



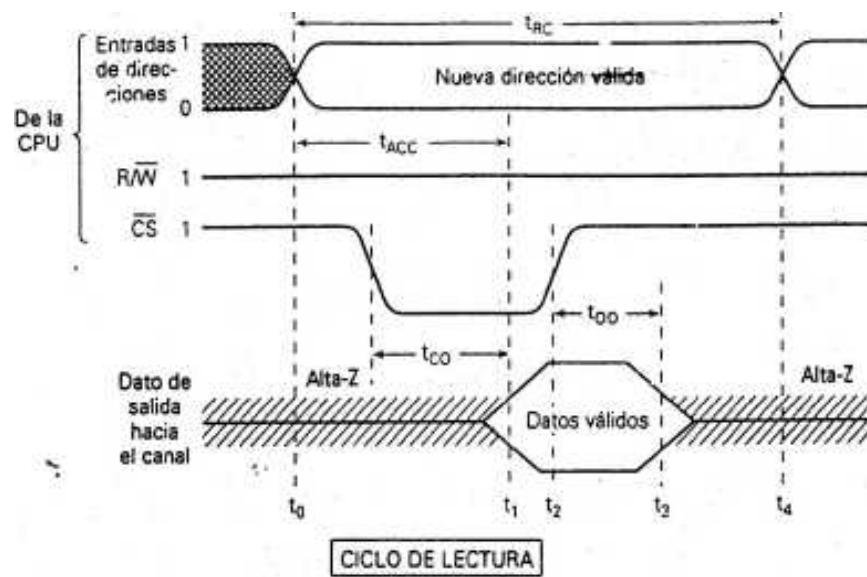
Mapa de memoria

- Rangos de direcciones correspondientes a cada circuito de memoria

	Dirección (hex)	Dirección (bin)	
		A16 A16 A14...A0	
ROM (64K)	00000h	0 0 0...0	
	0FFFFh	0 1 1...1	
RAM (32K)	10000h	1 0 0...0	
	17FFFh	1 0 0...0	
RAM (32K)	18000h	1 1 0...0	
	1FFFFh	1 1 1...1	$64K = 2^6 \cdot 2^{10} = 1 \cdot 2^{16} = 10000h$ $32K = 2^5 \cdot 2^{10} = 2^3 \cdot 2^{12} = 08000h$

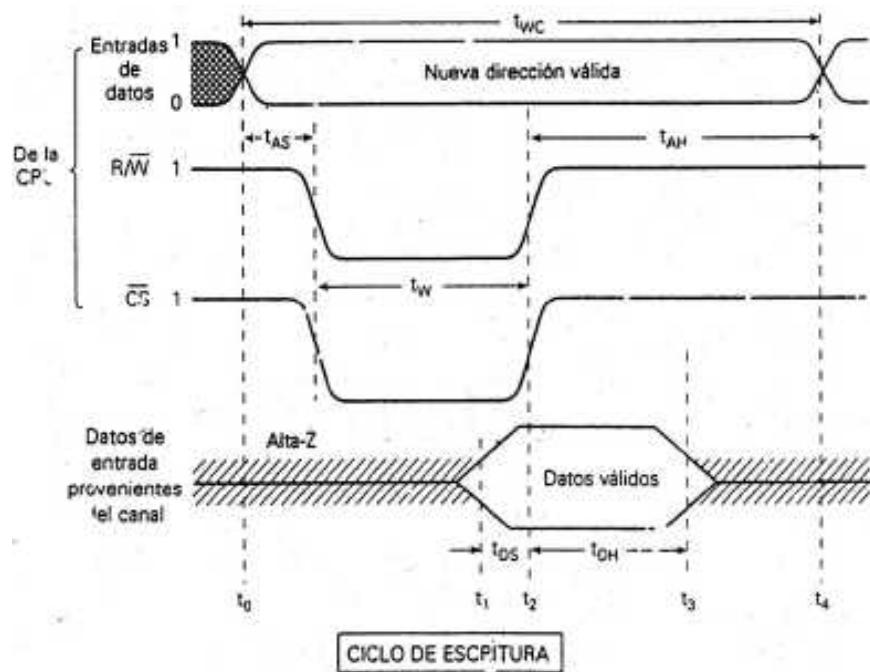
Cronogramas: Ciclo de lectura

- t_{RC} : Tiempo mínimo de ciclo de lectura
- t_{ACC} : Tiempo de acceso de lectura



Cronogramas: Ciclo de escritura

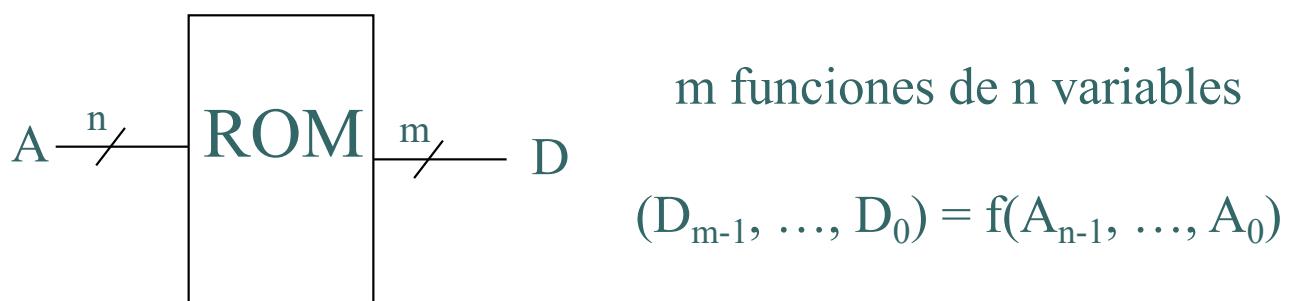
- t_{WC} : Tiempo mínimo de ciclo de escritura
- t_W : Tiempo mínimo del pulso de escritura
- t_{DS} : Tiempo de establecimiento o setup
- t_{DH} : Tiempo de mantenimiento o hold





Otras aplicaciones de las memorias

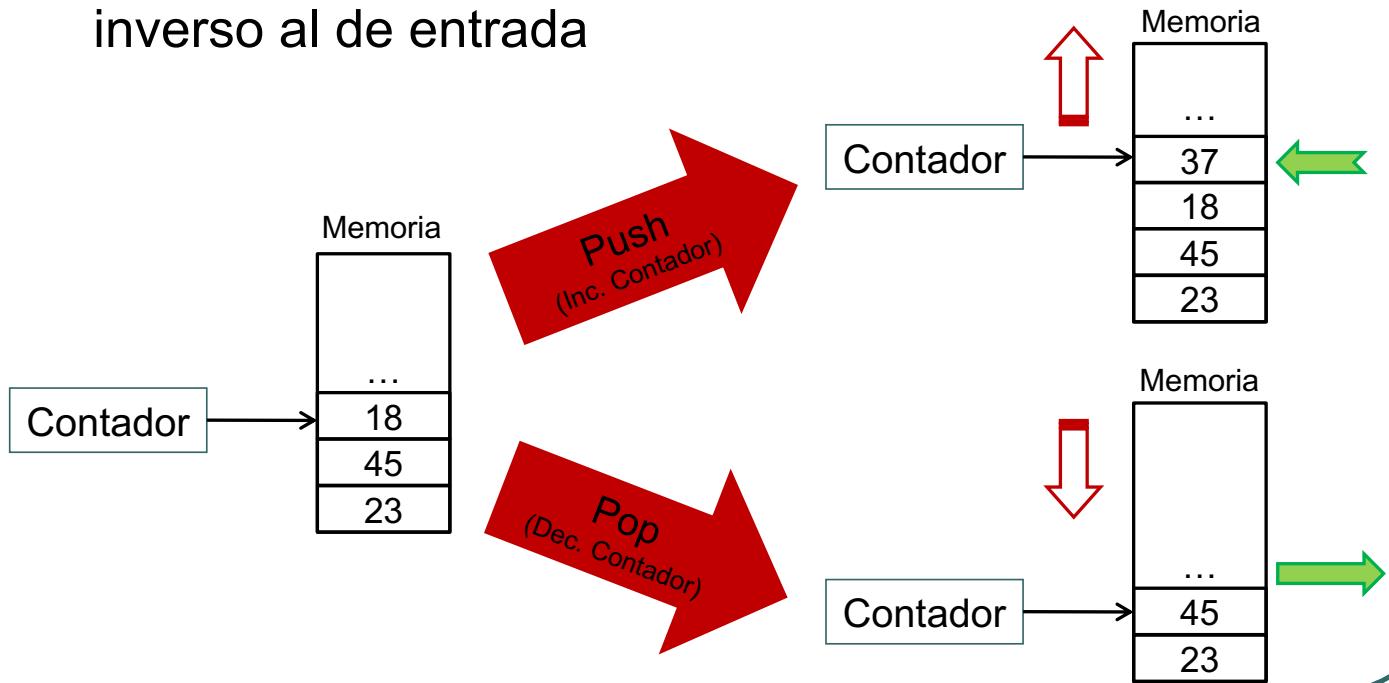
- Se pueden utilizar memorias para realizar funciones lógicas
- Una función se implementa almacenando su tabla de verdad en una memoria
- Una memoria que se utiliza para almacenar la tabla de verdad de una función se denomina Look-Up Table (LUT)
- El tamaño de la memoria necesaria aumenta exponencialmente con el número de variables de entrada





Pilas de inserción/extracción (LIFO)

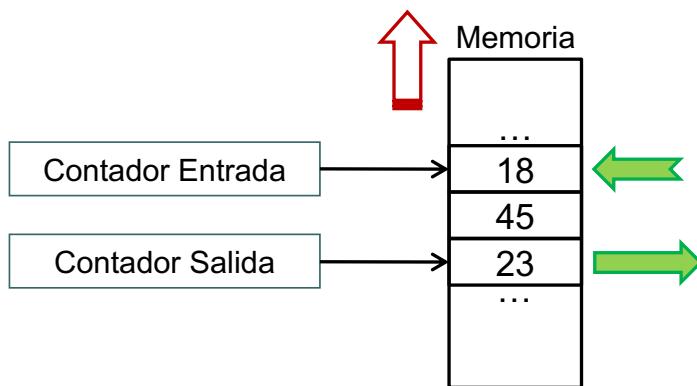
- LIFO (Last In First Out): Los datos salen en orden inverso al de entrada





Colas FIFO

- FIFO (First In First Out): Los datos salen y entran en el mismo orden, pero en instantes diferentes
- Aplicación: Almacenamiento temporal para ajuste de ráfagas de datos





Bibliografía

- “Circuitos y sistemas digitales”. J. E. García Sánchez, D. G. Tomás, M. Martínez Iniesta. Ed. Tebar-Flores
- “Principios de Diseño Digital”. D. Gajski. Ed. Prentice-Hall
- “Fundamentos de Sistemas Digitales”. Thomas L. Floyd. Pearson Prentice Hall



Dispositivos Lógicos Programables

© Luis Entrena, Celia López,
Mario García, Enrique San Millán

Universidad Carlos III de Madrid



Índice

- Tecnologías de implementación de circuitos programables
- Circuitos programables simples
- Circuitos programables complejos (CPLD, FPGA)



Implementación de circuitos digitales

- Lógica discreta
- Circuitos integrados a medida (ASIC, Application Specific Integrated Circuits)
- Circuitos programables (PLD, Programmable Logic Devices)
 - Simples
 - PROM: Programmable Read Only Memory
 - PLA: Programmable Logic Array
 - PAL: Programmable Array Logic
 - GAL: Generic Array Logic
 - Complejos
 - CPLD: Complex Programmable Logic Device
 - FPGA: Field Programmable Gate Array



Tecnologías

- Transistor MOS de puerta flotante (EPROM-FLASH)
 - Transistores que, al aplicarles sobretensión, pueden mantener su tensión de puerta (conexiones programables)
- Memoria RAM estática (SRAM)
 - La memoria permite implementar funciones lógicas
 - Se usan LUTs (Look-Up Tables) de 4 o 5 entradas
- Antifusibles
 - Al fundirse un antifusible se produce un cortocircuito
 - Los cortocircuitos tienen menor resistencia que los diodos-fusibles, proporcionando mayores prestaciones



Circuitos programables simples

PLDs (Programmable Logic Devices)

Entradas + Inversores

Matriz
AND

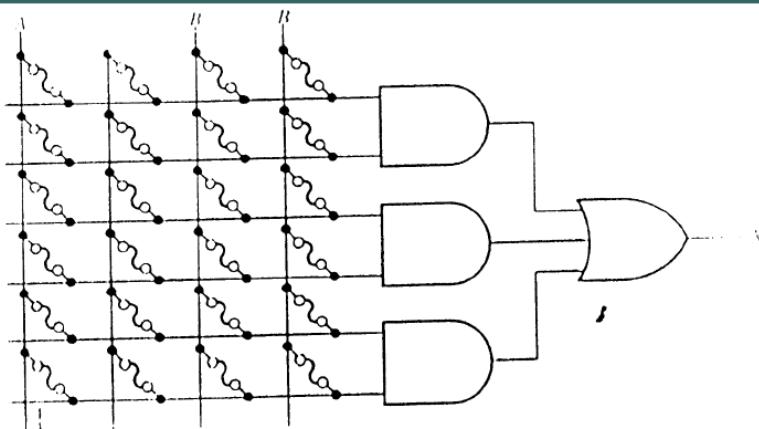
Matriz
OR

Biestables (opcional)

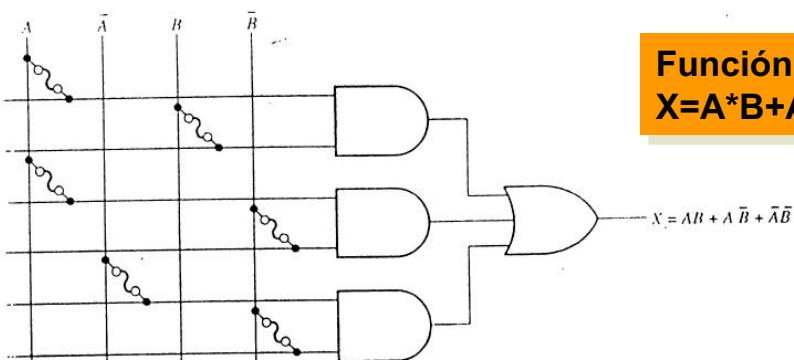
Inversores + Salidas



Matrices programables



Matriz AND
con OR fija

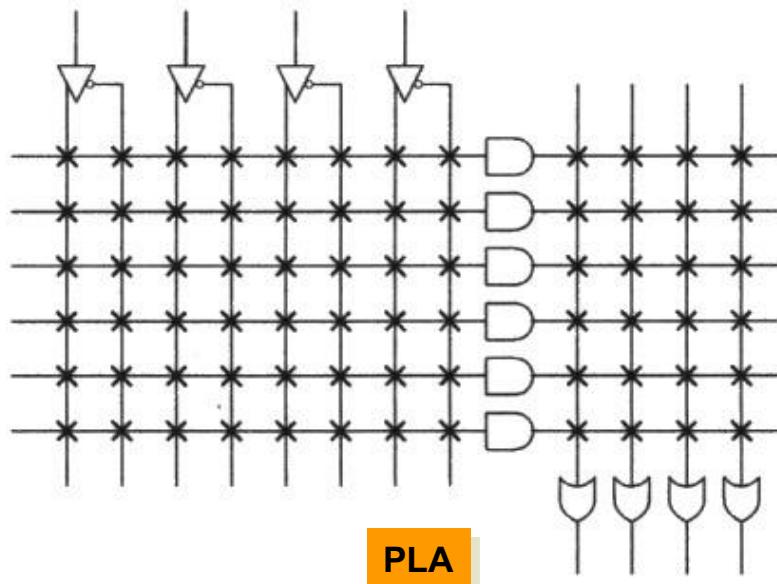


Función
 $X = A * B + A * \text{NOT}(B) + \text{NOT}(A) * \text{NOT}(B)$

Matrices programables

Matriz AND

Matriz OR

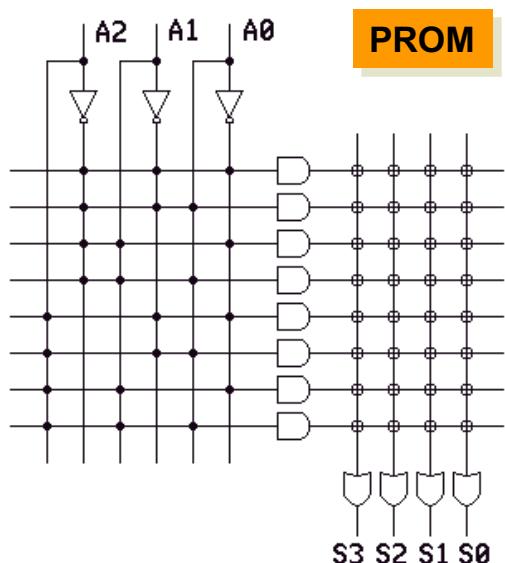


Tipos de PLDs

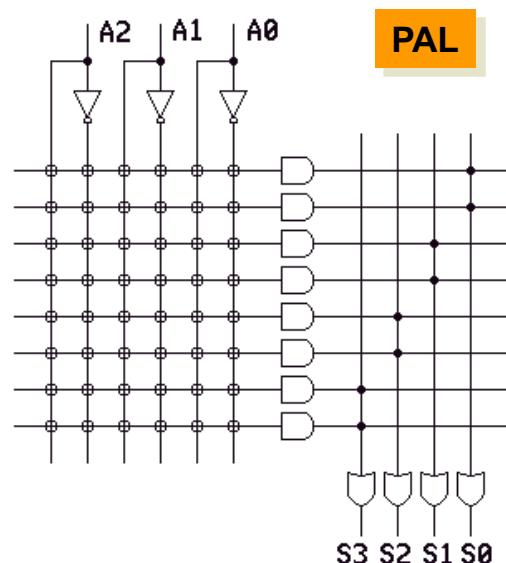
	Matriz AND	Matriz OR
PROM	Fija	Programable
PLA	Programable	Programable
PAL	Programable	Fija
GAL	Programable	Fija

- Notación simplificada para las conexiones

Tipos de PLDs



PROM

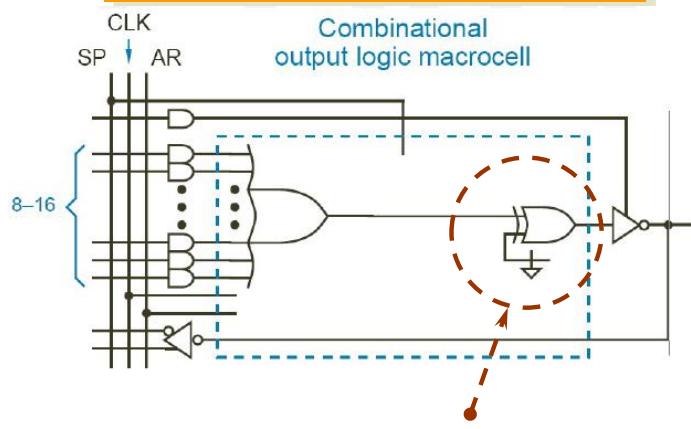


PAL

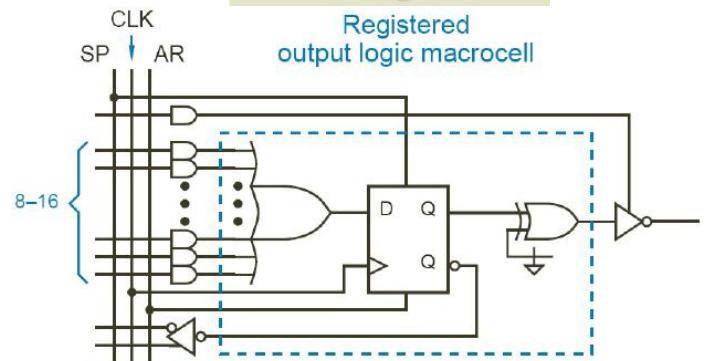
- Matriz AND fija (decodificador direcciones)
- Matriz OR fija
- Matriz AND programable
- Matriz OR programable (datos)

Bloques de salida

Entrada-Salida combinacional



Salida registrada



entradas

PAL 16 R 8

salidas

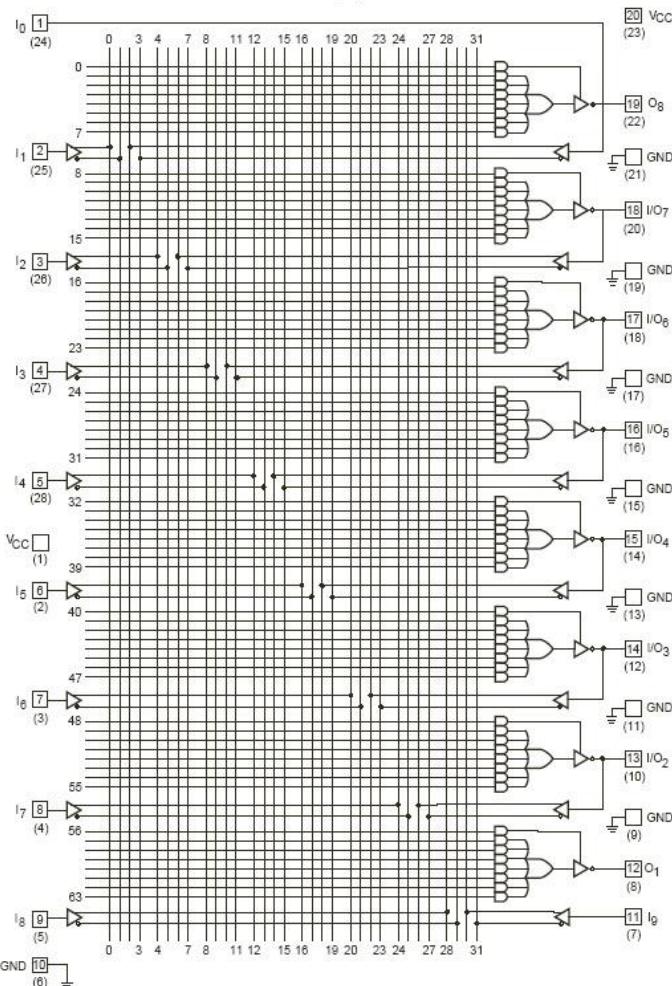
tipo salida

• Nomenclatura

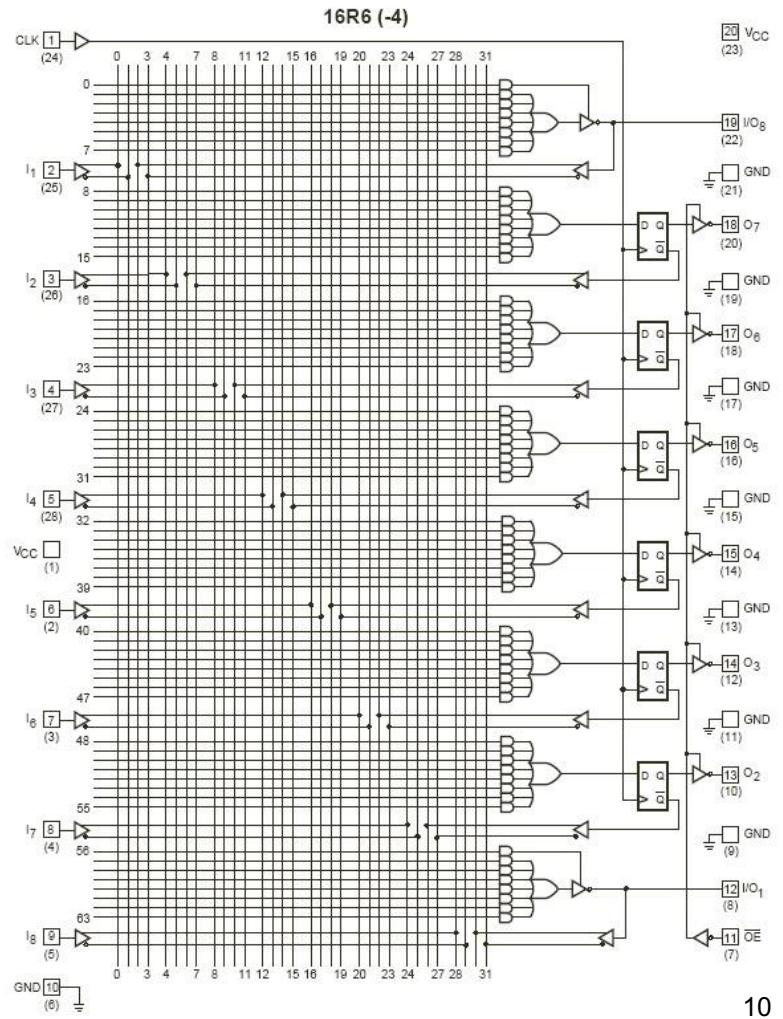
- L: active Low
- H: active High
- P: polaridad programable
- R: registrada

PALs reales

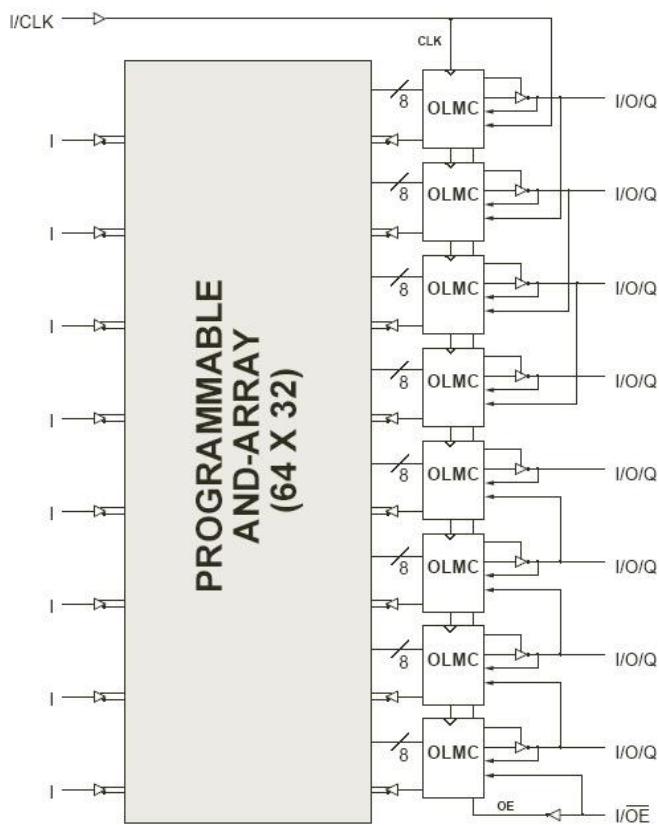
16L8 (-4)



16R6 (-4)

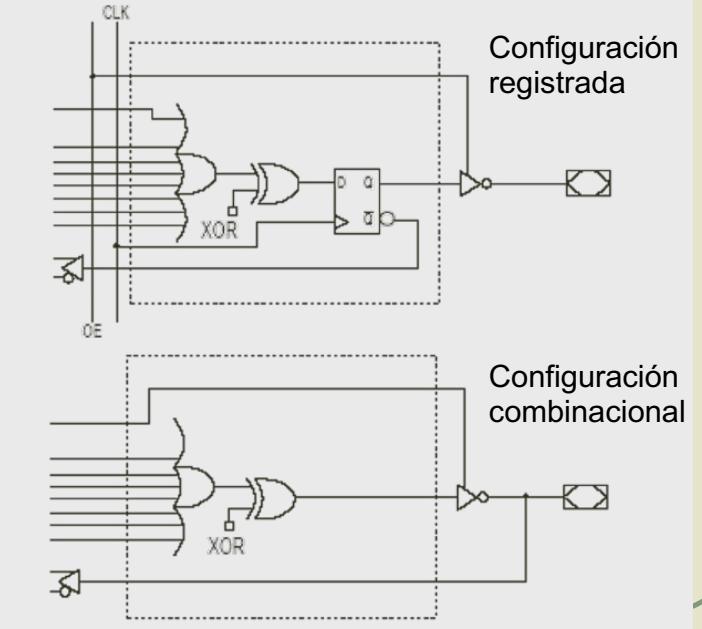


GAL (Generic Array Logic)



Arquitectura como la de las PAL, pero con funciones de salida programables.

OLMC: Output Logic Macrocell



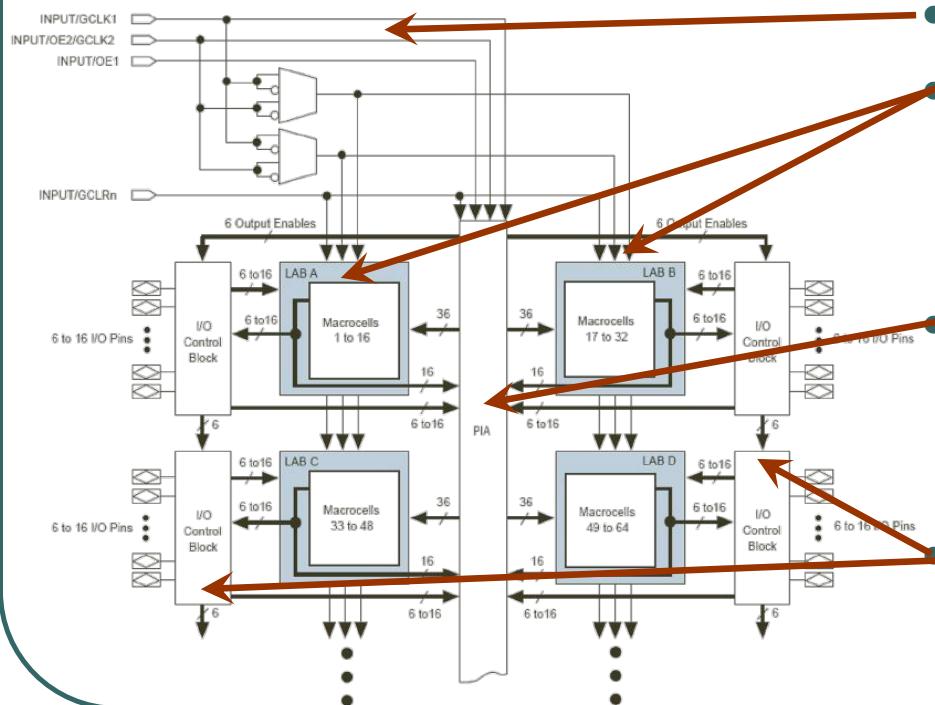


Circuitos programables complejos

- CPLD:
Complex Programmable Logic Devices
- FPGA:
Field Programmable Gate Array
- Diferencias con los PLDs simples
 - Arquitectura
 - Cantidad de recursos lógicos
- Fabricantes de CPLDs/FPGAs
 - Xilinx
 - Altera
 - Actel
 - Atmel
 - Lattice
 - Cypress

CPLD: arquitectura

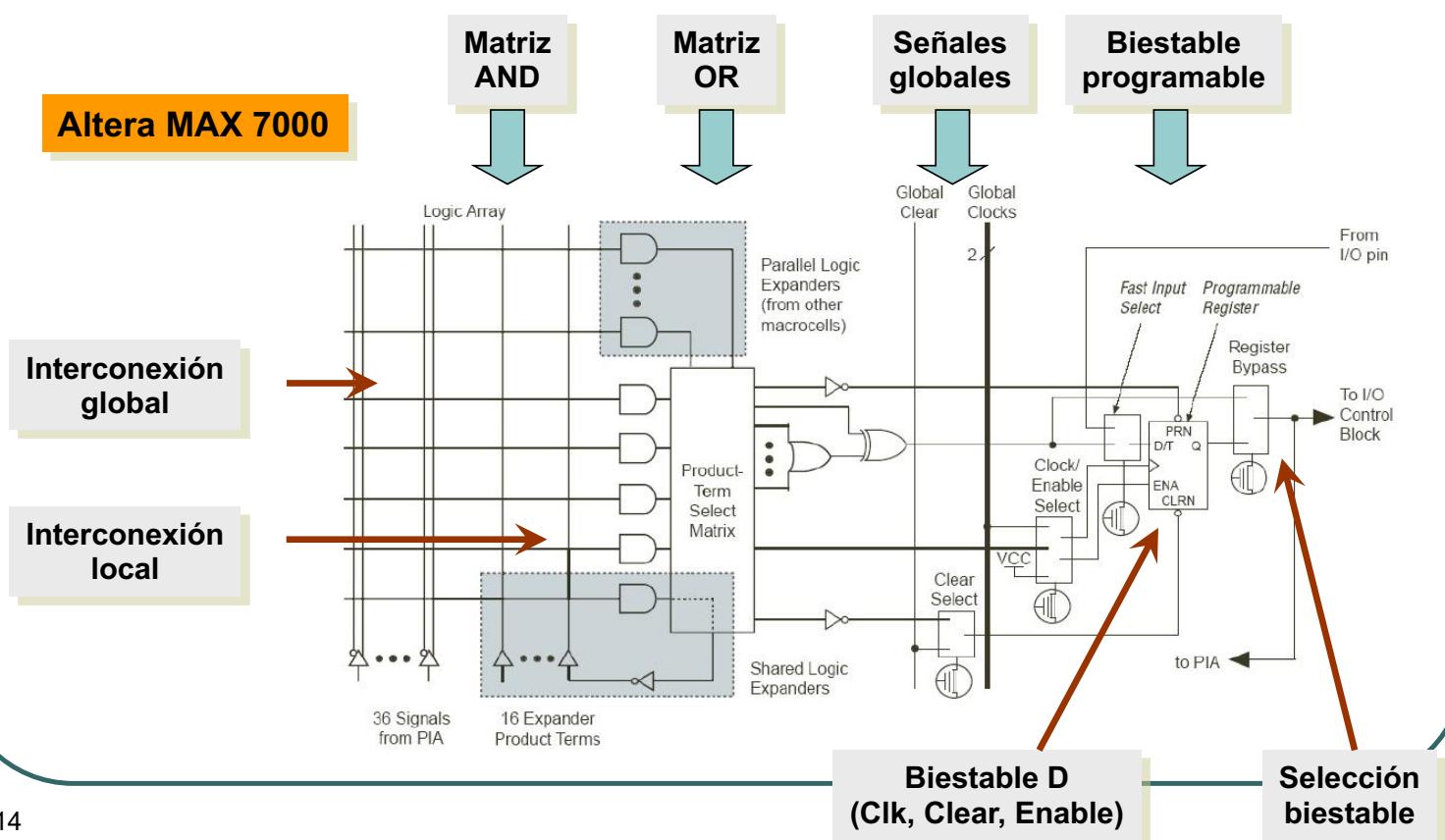
Altera MAX 7000



- Señales globales
- Bloques de matrices lógicas (LAB, Logic Array Blocks).
 - 1 LAB = 16 macroceldas
- Matriz de interconexión programable (PIA, Programmable Interconnect Array)
- Bloques E/S

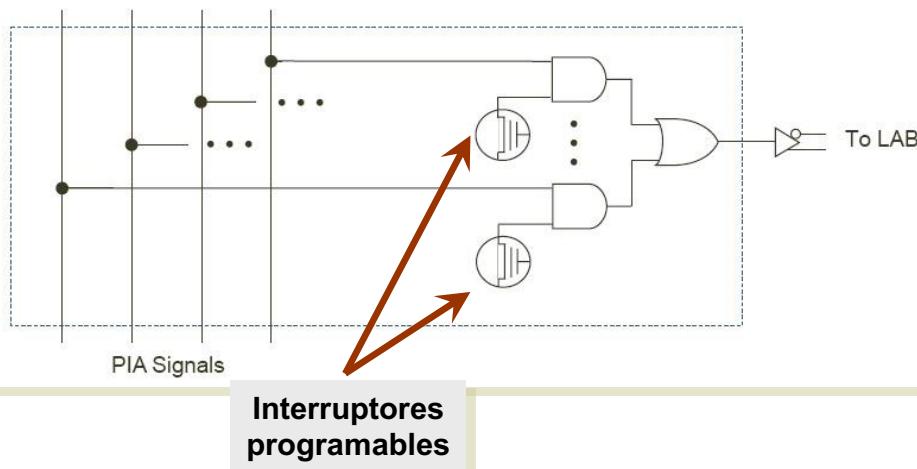


CPLD: macrocelda



CPLD: matriz de interconexión

Matriz de interconexión global



- Entradas PIA
 - Pines E/S
 - Salidas LABs
- Salidas PIA
 - Entradas LABs



CPLD: resumen de características

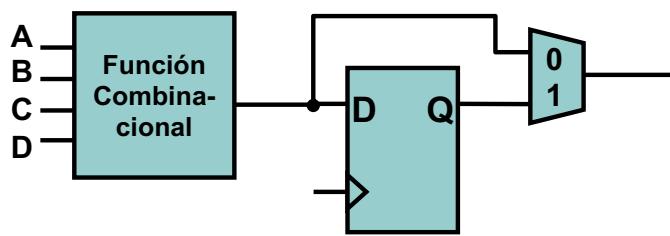
- Estructura de PAL con registros y lógica de interconexión
- Capacidad media (hasta 25000 puertas)
- Velocidad media/alta
- Consumo alto
- Tecnología EPROM (reprogramable, no volátil)
- Precio bajo
- La matriz de interconexión global limita el tamaño
- ISP (In-System Programming). JTAG.



FPGAs

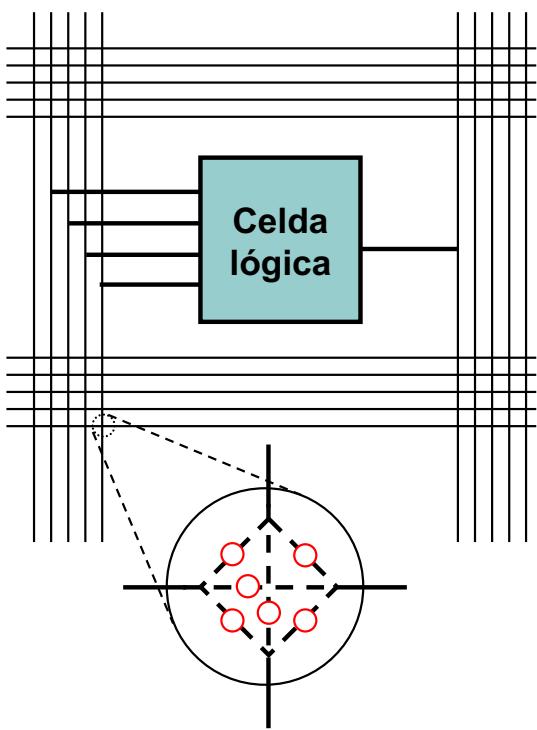
- Field Programmable Gate Arrays (Matrices de puertas programables en campo)
- Superan las limitaciones en tamaño de las CPLDs, mediante arquitecturas avanzadas
- Ofrecen mayor variedad de recursos lógicos
 - Lógica combinacional
 - Lógica secuencial
 - Memoria RAM
 - Conformadores de reloj
 - Señales globales
 - Multiplicadores
- Fabricantes
 - Xilinx
 - Altera
 - Actel
 - Atmel

FPGA: celda lógica básica



- Función combinacional + Biestable
- Otras variaciones:
 - 2 FC + 1 biestable
 - 2 FC + 2 biestables
- Funcionalidad adicional:
 - Lógica de acarreos
 - FC de 6 u 8 entradas
 - Varias señales de reloj y reset
 - Diferentes configuraciones del biestable: nivel, flanco de subida, flanco de bajada
- Función combinacional:
 - LUT (Look-Up Table): SRAM, volátil

FPGA: interconexiones

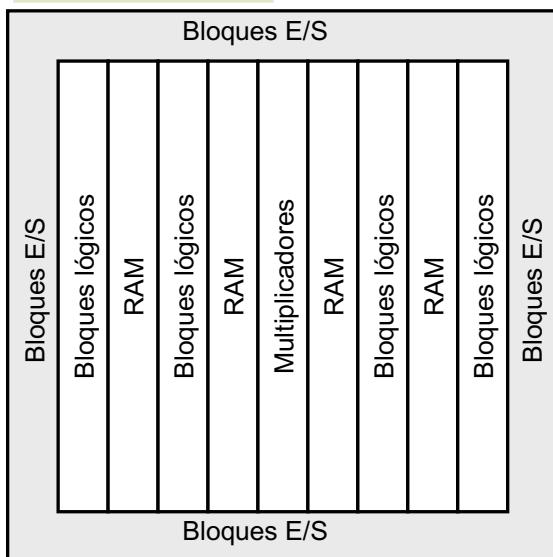


- Interconexiones programables
 - Locales:
 - Abundantes y rápidas
 - Para conectar celdas cercanas
 - Globales
 - Para conectar zonas lejanas



Arquitectura general

FPGA (Xilinx)



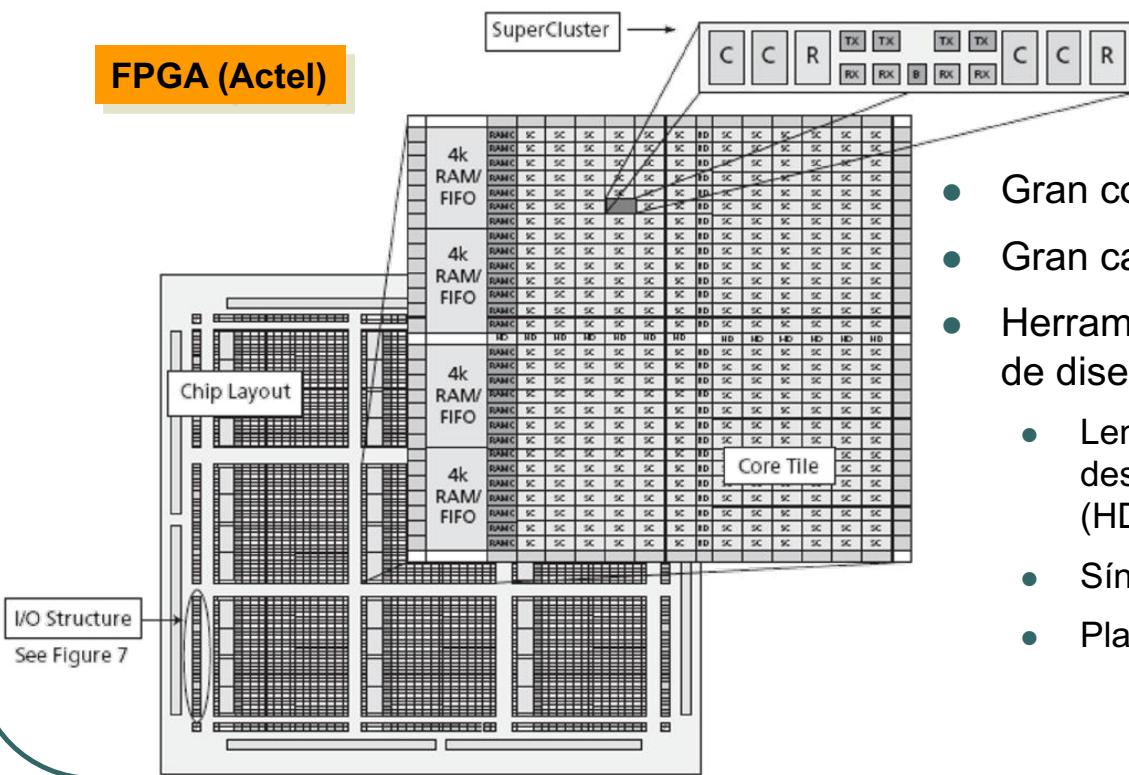
• Elementos básicos

- Bloques lógicos
- Bloques de E/S
- Matrices de interconexión programables

• Elementos avanzados

- Memoria RAM
- Gestores de reloj
- Multiplicadores

Actel Axcelerator FPGA (antifusibles)



- Gran complejidad
- Gran cantidad de recursos
- Herramientas automáticas de diseño:
 - Lenguajes de descripción de hardware (HDL)
 - Síntesis
 - Place & Route

Figure 1-6 • AX Device Architecture (AX1000 shown)



Bibliografía

- Webs de fabricantes:
 - Xilinx: www.xilinx.com
 - Altera: www.altera.com
 - Actel: www.actel.com
 - Lattice: www.latticesemi.com
- “Fundamentos de Sistemas Digitales”. Thomas L. Floyd.
Pearson Prentice Hall
- “Sistemas digitales: principios y aplicaciones”, Tocci, Ronald J.
Pearson Prentice Hall
- “Dispositivos lógicos programables (PLD): diseño práctico de
aplicaciones”. García Iglesias, José Manuel. RaMa



Introducción a los sistemas digitales

© Luis Entrena, Celia López, Mario García,
Enrique San Millán

Universidad Carlos III de Madrid



Índice

- Diseño de sistemas digitales
 - Proceso de diseño de sistemas digitales
 - Estructura de un sistema digital
 - La ruta de datos
 - La unidad de control
- Circuitos integrados
 - Tecnologías digitales
 - ASICs
 - Dispositivos programables
 - Microprocesadores



Sistemas digitales

- Los sistemas digitales procesan información digital, de acuerdo con un algoritmo determinado
- Algoritmo: conjunto ordenado y finito de operaciones que permite hallar la solución a un problema
- Objetivo de la lección: introducción a los sistemas digitales



Diseño de sistemas digitales

- El diseño con componentes tan básicos como las puertas lógicas no es práctico para diseñar sistemas digitales
- Diseño en el Nivel de Transferencia entre Registros (Register Transfer Level, RTL):
 - Componentes más abstractos: ALUs, registros o multiplexores
 - El sistema se describe como operaciones que se realizan entre datos almacenados en registros

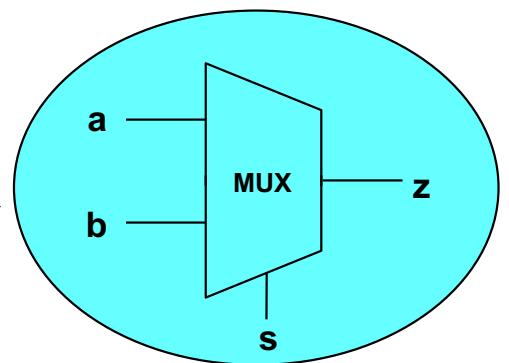


Lenguajes de Descripción de Hardware

- Los Lenguajes de Descripción de Hardware o HDLs (Hardware Description Languages) permiten diseñar un circuito digital desde un mayor nivel de abstracción
- Ejemplo (VHDL)

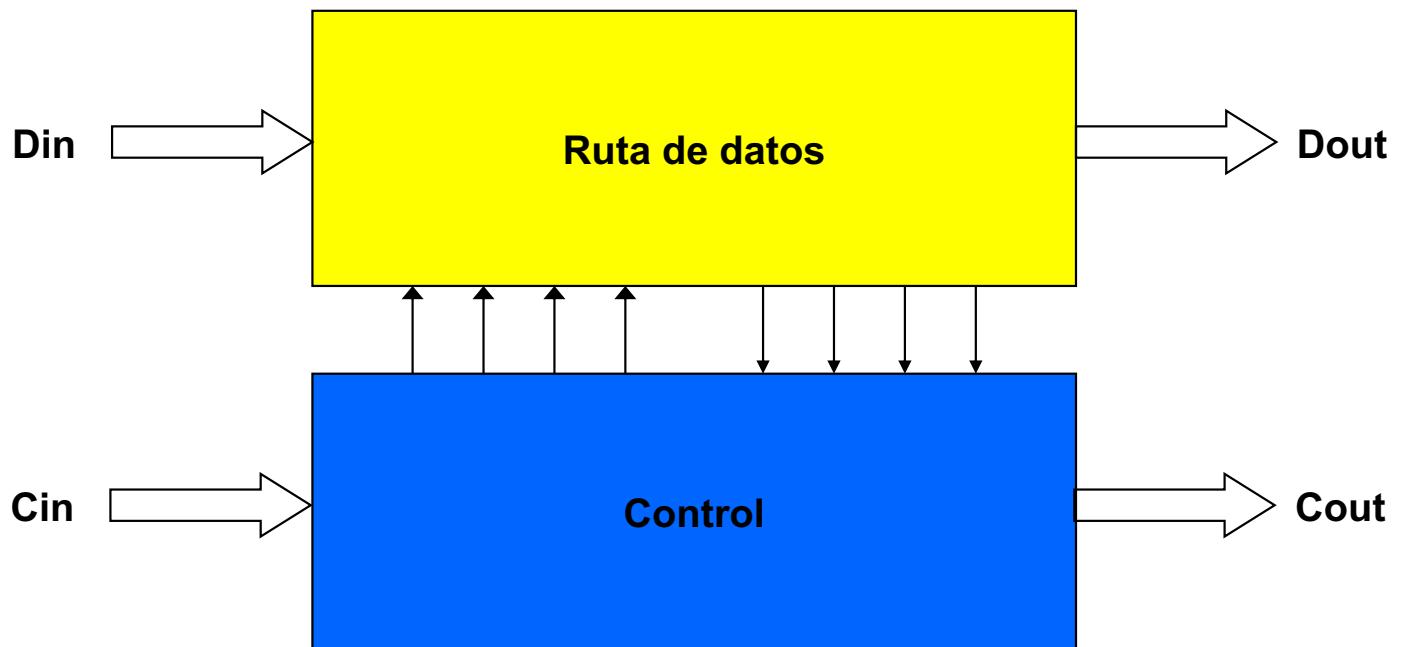
```
if s = '0' then  
    z <= a;  
else  
    z <= b;  
end if;
```

Sintetizador
(Compilador)





Estructura de un sistema digital





La ruta de datos

- Todo algoritmo se descompone en una serie de operaciones básicas:
 - Aritméticas
 - Lógicas
 - Desplazamientos y rotaciones
- La ruta de datos (“datapath”) es el conjunto de unidades funcionales que procesan los datos



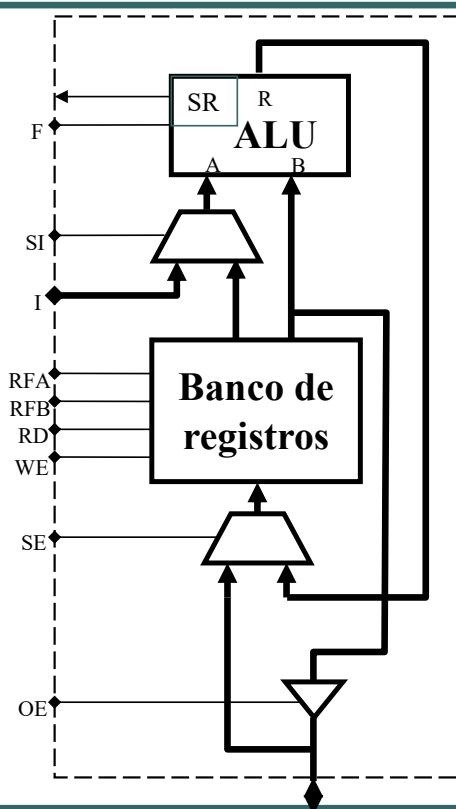
La ruta de datos

- Componentes típicos de la ruta de datos
 - ALUs: realizan las operaciones necesarias
 - Registros y memorias: almacenan datos temporales
 - Buses: conectan los elementos de la ruta de datos
 - Multiplexores: seleccionan los datos que se deben procesar en cada momento



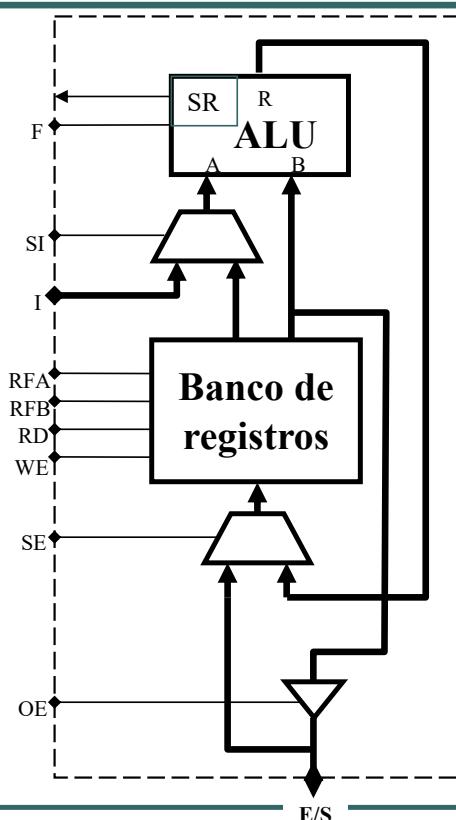
Ejemplo de ruta de datos

- ALU realiza operaciones
 - F: Función de la ALU
- SR: Registro de estado. Indicadores (flags) de la operación realizada.
Ejemplos:
 - C: Acarreo
 - O: Overflow
 - Z: Cero
 - S: Signo
- Operandos inmediatos a través de I, seleccionados con SI



Ejemplo de ruta de datos

- Banco de registros (memoria de triple puerto) almacena valores intermedios
- Selección de registros
 - RFA: Registro Fuente A
 - RFB: Registro Fuente B
 - RD: Registro Destino
- E/S para conexión externa
 - SE selecciona la fuente para RD (externa o interna)
 - OE habilita la salida





Unidad de control

- Determina la correcta secuenciación y utilización de las operaciones sobre los datos
- Elementos típicos:
 - Máquinas de estados
 - Contadores
 - Registros y memorias con datos de control, etc.



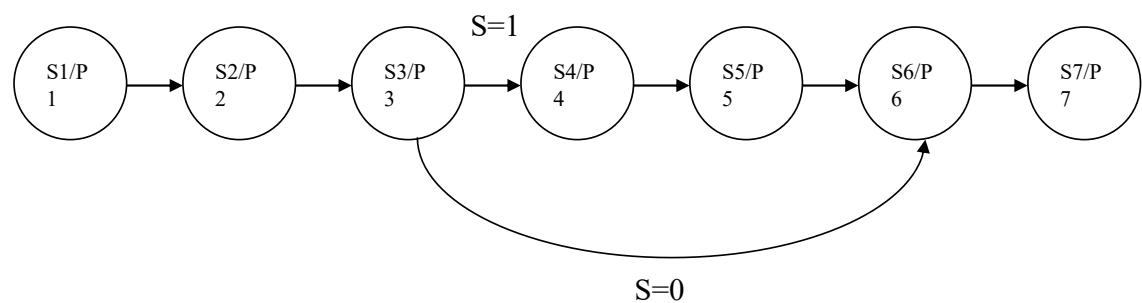
Unidad de control

- Ejemplo: realizar la operación siguiente
 $y = \text{abs}(x_1 - x_2)/2$
- Pasos:
 - 1. Cargar x_1 en el registro R1
 - 2. Cargar x_2 en el registro R2
 - 3. Restar x_2 de x_1 y colocar el resultado en el registro R3 ($R3 = R1 - R2$).
 - Si el resultado es positivo, saltar al paso 6. En caso contrario, seguir con el paso 4.
 - 4. Complementar R3 ($R3 = \text{NOT } R3$)
 - 5. Incrementar R3 ($R3 = R3 + 1$)
 - 6. Desplazar R3 a la derecha ($R3 = R3/2$)
 - 7. Enviar R3 a la salida



Unidad de control

- Máquina de Estados de la unidad de control





Unidad de control

- Salidas de la Máquina de Estados

Paso	Función ALU (F)	RFA	SI	I	RFB	RD	WE	SE	OE
1	X	X	X	X	X	R1	0	1	0
2	X	X	X	X	X	R2	0	1	0
3	RESTA	R1	0	X	R2	R3	0	0	0
4	NOT	R3	0	X	X	R3	0	0	0
5	SUMA	X	1	1	R3	R3	0	0	0
6	SHR	R3	0	X	X	R3	0	0	0
7	X	X	X	X	R3	X	1	X	1



Circuitos Integrados

- Componentes discretos (estándar)
 - 74xx, 54xx
- Circuitos integrados a medida
 - ASIC: Application Specific Integrated circuit
- Circuitos programables
 - PLD, SPLD: (Simple) Programmable Logic Devices
 - CPLD: Complex Programmable Logic Devices
 - FPGA: Field Programmable Gate Array
- Microprocesadores



Circuitos Integrados (cont.)

- Sistemas microprocesadores
 - Microprocesador y componentes adicionales (componentes estándar)
 - SoC: System on Chip (ASIC)
 - SoPC: System on Programmable Chip (FPGA)



Tecnologías digitales

- Las puertas lógicas son circuitos electrónicos
- El nivel lógico (0 o 1) se representa mediante un nivel de tensión
- Generalmente se utiliza “lógica positiva”
 - Tensión alta (5V, 3.3V, 2.5 V, etc) → 1
 - Tensión baja (0V) → 0
- Existen muchas tecnologías, según la forma en que se realizan las puertas lógicas y las características que se obtienen



Características de las tecnologías digitales

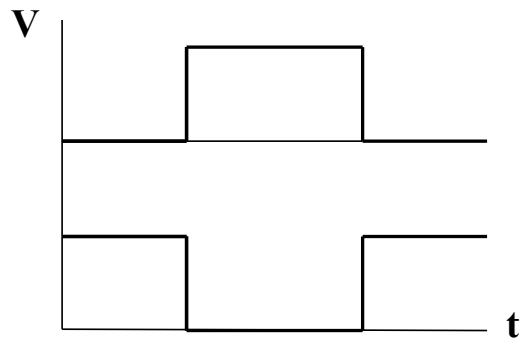
- Principales características:
 - Margen de temperaturas de operación
 - Tensión de alimentación
 - Margen de ruido (intervalos de tensiones que se asocian a un nivel lógico determinado)
 - Retardo de conmutación
 - Consumo
 - Otros
- Cada tecnología o subfamilia presenta valores diferentes respecto a estos parámetros



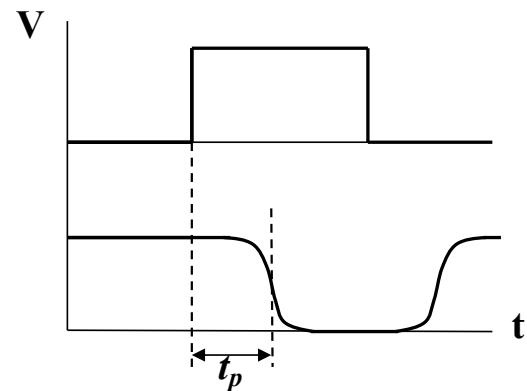
Retardos

- Las puertas lógicas no comutan instantáneamente

Inversor ideal



Inversor real



- El retardo limita la velocidad de operación del circuito



Consumo

- Las puertas lógicas consumen energía:
 - Estática: la que se consume por tener alimentada la puerta lógica, sin cambiar los valores lógicos
 - Dinámica: la que se consume al comutar
- En la tecnología CMOS (la más utilizada actualmente), el consumo estático es muy pequeño. Sin embargo,
 - Los circuitos modernos pueden llegar a tener más de 10^8 puertas lógicas!
 - El consumo dinámico es proporcional a la frecuencia de comutación
- El consumo es un problema importante:
 - La energía consumida se transforma en calor, que hay que disipar. Si el circuito consume mucho, puede ser difícil disipar el calor
 - En dispositivos portátiles, el tamaño y el peso de la batería es limitado

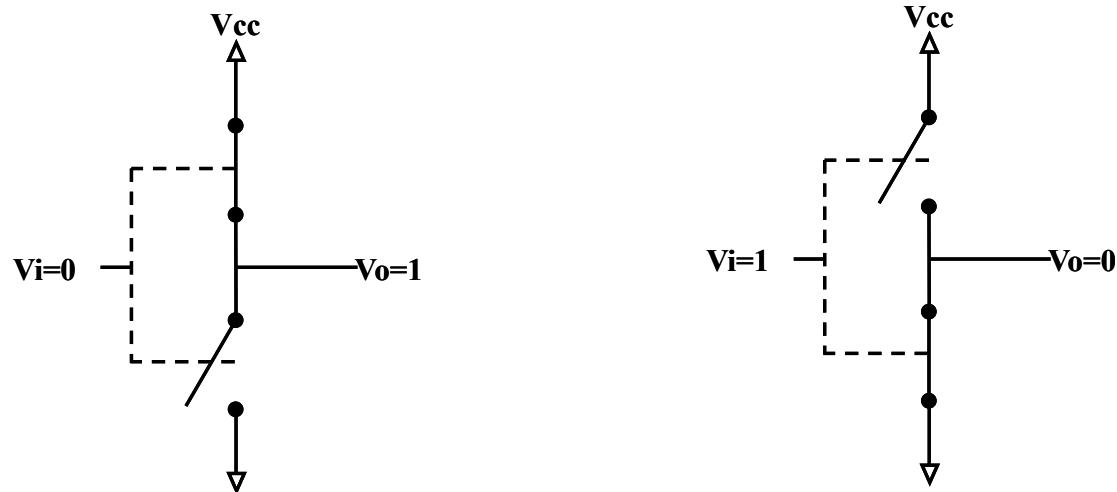


Tecnología CMOS

- La tecnología CMOS (Complementary Metal Oxide Semiconductor) es la tecnología más utilizada en la actualidad
- Basada en:
 - Transistores MOS: interruptores controlados por tensión
 - Complementarios: cada transistor o interruptor tiene su complementario, de manera que si un interruptor está abierto su complementario está cerrado y viceversa



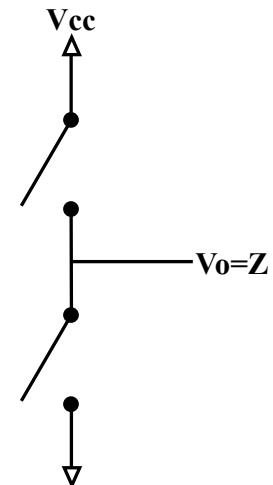
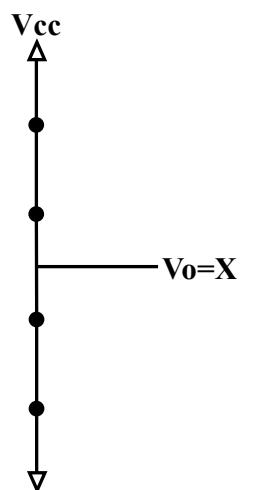
Inversor CMOS





Valores metalógicos

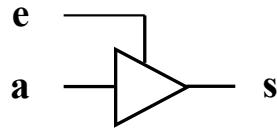
- Hay situaciones que no se corresponden con valores lógicos
 - Cortocircuito (X)
 - Alta impedancia o triestado (Z)





Buffer triestado

- Un tipo especial de puerta lógica que puede poner su salida en alta impedancia

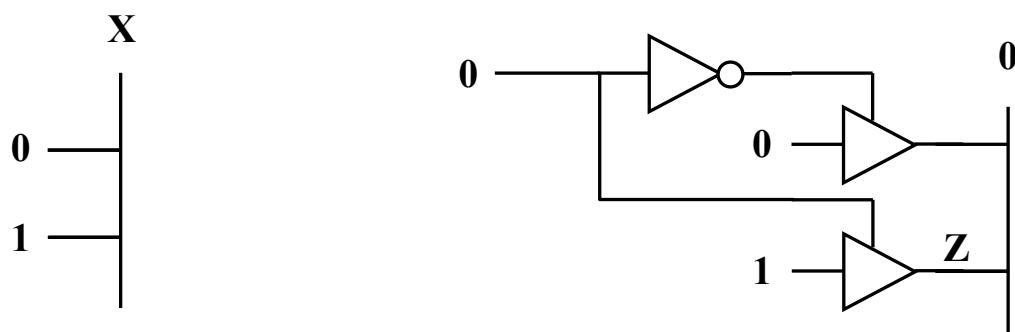


e	a	s
0	0	Z
0	1	Z
1	0	0
1	1	1



Buffer triestado

- Los buffers triestados son útiles para permitir múltiples conexiones a un mismo punto evitando cortocircuitos



Cortocircuito!



Familias lógicas

- El conjunto de componentes digitales básicos, tales como puertas lógicas y otros que estudiaremos a lo largo del curso, se conoce popularmente como *Serie* o *Familia 74*
- Existen numerosas subfamilias:
 - Segundo el rango de temperaturas de operación:
 - Serie 74: 0º a 70º
 - Serie 54: -55º a 125º
 - Segundo la tecnología utilizada:
 - LS
 - ALS
 - F
 - HC
 - AHC
 - G
 -



Familias lógicas

- Designación de componentes:
 - <Serie><Subfamilia><Componente>
- Ejemplo: 74HC00
 - Serie 74: rango de temperaturas convencional
 - Subfamilia HC (High speed CMOS)
 - Componente 00: 4 puertas NAND de 2 entradas
- Importante: las subfamilias no son compatibles entre sí
 - No se deben mezclar componentes de distintas subfamilias en un circuito



ASICs

- Los ASICs son circuitos integrados de aplicación específica
- Una vez diseñado un circuito cualquiera, se puede obtener su “layout” que consiste en un conjunto de máscaras necesarias para la fabricación del circuito



ASICs

- Tamaño: pequeño
- Velocidad: muy alta
 - Tarjeta: $f < 100$ MHz
 - FPGA: 500 MHz
 - ASIC: $f < 3$ GHz
- Coste: depende del número de unidades que se fabriquen
 - Coste inicial: diseño y prototipado (100.000€)
 - Coste por pieza: 1-200€
 - Rentables para grandes tiradas (>10.000 piezas/año)
- Fiabilidad: alta; más inmunes al ruido
- Consumo: bajo



Dispositivos programables

- Circuitos programables (PLD, Programmable Logic Devices)
 - Simples
 - PROM: Programmable Read Only Memory
 - PLA: Programmable Logic Array
 - PAL: Programmable Array Logic
 - GAL: Generic Array Logic
 - Complejos
 - CPLD: Complex Programmable Logic Device
 - FPGA: Field Programmable Gate Array



Tecnologías de circuitos programables

- Transistor MOS de puerta flotante (EPROM-FLASH)
 - Transistores que, al aplicarles sobretensión, pueden mantener su tensión de puerta (conexiones programables)
- Memoria RAM estática (SRAM)
 - La memoria permite implementar funciones lógicas
 - Se usan LUTs (Look-Up Tables) de 4 o 5 entradas
- Antifusibles
 - Al fundirse un antifusible se produce un cortocircuito
 - Los cortocircuitos tienen menor resistencia que los diodos-fusibles, proporcionando mayores prestaciones



Circuitos programables simples

PLDs (Programmable Logic Devices)

Entradas + Inversores

Matriz
AND

Matriz
OR

Biestables (opcional)

Inversores + Salidas



Circuitos programables complejos

- CPLD:
Complex Programmable Logic Devices
- FPGA:
Field Programmable Gate Array
- Diferencias con los PLDs simples
 - Arquitectura
 - Cantidad de recursos lógicos
- Fabricantes de CPLDs/FPGAs
 - Xilinx
 - Altera
 - Actel
 - Atmel
 - Lattice
 - Cypress



CPLD: resumen de características

- Estructura de PAL con registros y lógica de interconexión
- Capacidad media (hasta 25000 puertas)
- Velocidad media/alta
- Consumo alto
- Tecnología EPROM (reprogramable, no volátil)
- Precio bajo
- La matriz de interconexión global limita el tamaño
- ISP (In-System Programming). JTAG.



FPGAs

- Field Programmable Gate Arrays (Matrices de puertas programables en campo)
- Superan las limitaciones en tamaño de las CPLDs, mediante arquitecturas avanzadas
- Ofrecen mayor variedad de recursos lógicos
 - Lógica combinacional
 - Lógica secuencial
 - Memoria RAM
 - Conformadores de reloj
 - Señales globales
 - Multiplicadores

Fabricantes

- Xilinx
- Altera
- Actel
- Atmel



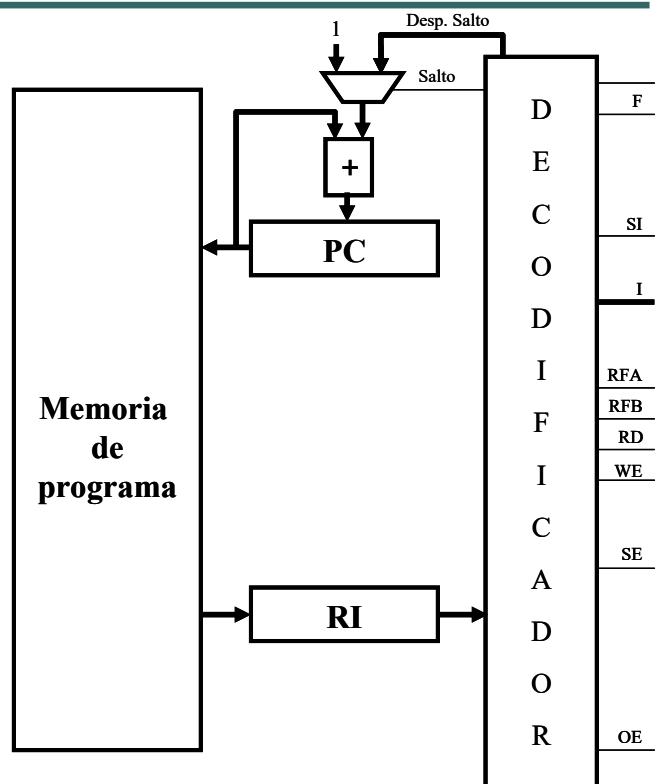
Microprocesadores

- Microprocesador
 - Orientado a realizar una amplia variedad de algoritmos
- Componentes:
 - Unidad de Proceso o Unidad Central de Proceso (CPU), que se encarga de realizar las operaciones necesarias
 - Ruta de datos general
 - Unidad de control programable (para poder realizar múltiples algoritmos)
 - Memoria, para el almacenamiento de información
 - Unidades de Entrada/Salida, para comunicarse con el exterior



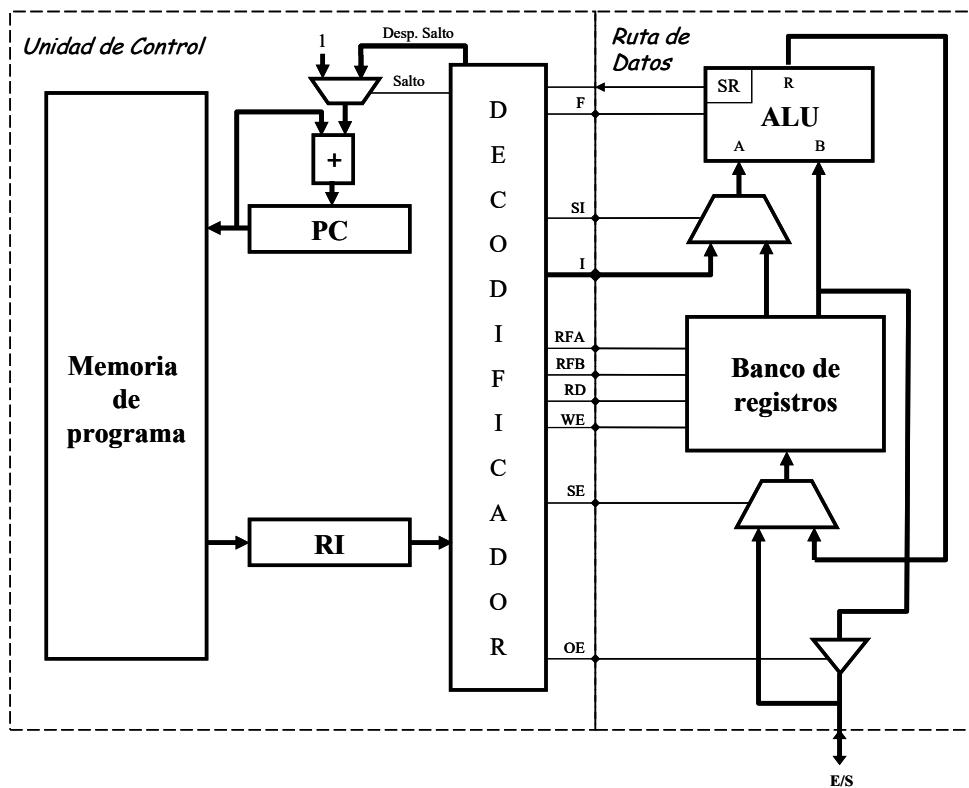
Unidad de control programable

- Las palabras de control se almacenan en una memoria
 - Para ahorrar espacio, se codifican las palabras de control
 - Las palabras de control codificadas las llamamos *instrucciones*
- En cada ciclo de reloj se lee a una instrucción (RI) y se decodifica
- La instrucción correspondiente a cada paso se determina con un contador (PC).
 - Los saltos en la secuencia se realizan con instrucciones, que pueden ser condicionados dinámicamente por SR



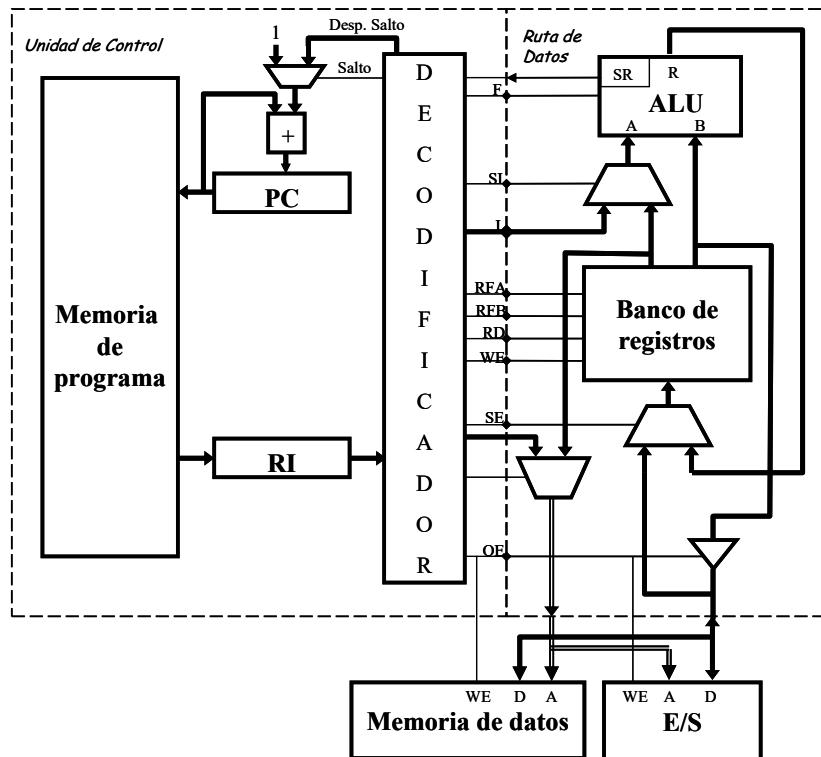


Microprocesador



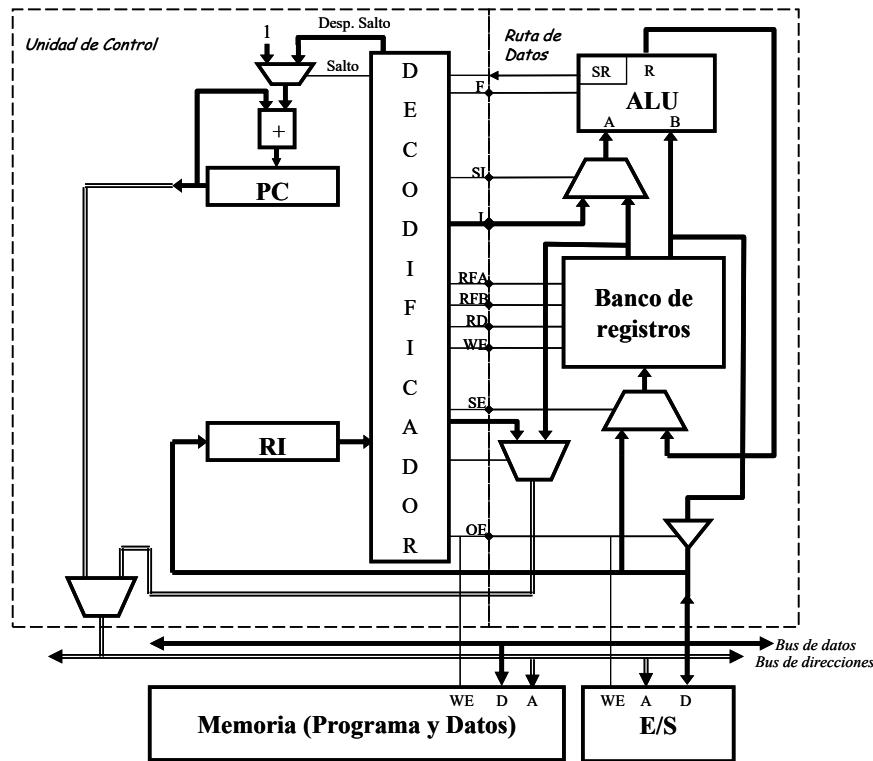


Computador básico (Arquitectura Harvard)





Computador básico (Arquitectura von Neuman)





Funcionamiento del computador elemental. Instrucciones

- ¿Cómo se ejecutan las instrucciones?
 - Ciclo de instrucción
- ¿Qué tipos de instrucciones hay?
- ¿Cómo se codifica una instrucción?
 - Formato de instrucción
 - Modos de direccionamiento



Ciclo de instrucción

- Cada instrucción se procesa típicamente en 2 fases:
 - Búsqueda de la instrucción: carga la instrucción en el IR
 - Ejecución de la instrucción: decodifica la instrucción y configura la ruta de datos para realizar la operación
- Cada una de estas fases puede realizarse en 1 ciclo de reloj o en varios, dependiendo de la complejidad del microprocesador



Tipos de instrucciones

- Instrucciones de transferencia de datos
 - Transferencia de datos entre registros, entre un registro y la memoria, o entre un registro y un interfaz de E/S
- Instrucciones aritmético-lógicas
 - Realizan operaciones con la ALU: sumas, restas, desplazamientos, etc.
- Instrucciones de salto y bifurcación
 - Permiten realizar cambios en la secuencia de ejecución de las instrucciones
 - Modifican el PC
 - Ejemplos: saltos, llamadas a subrutina, etc.



Formato de instrucción

- Organización de la información de las instrucciones.
Las instrucciones se codifican habitualmente por campos:
 - Código de operación (Opcode): indica la operación que se debe realizar
 - Operandos: indican los datos sobre los que se debe operar
- El número de operandos y el tamaño de los campos puede ser variable
- El tamaño de las instrucciones debe ser un múltiplo del ancho de palabra de la memoria



Modos de direccionamiento

- Los operandos pueden indicarse de diversas formas, conocidas como modos de direccionamiento
- Algunos modos usuales:
 - *Inmediato*: el valor del operando se indica en la instrucción
 - *Directo por registro*: la instrucción indica un registro que contiene el operando
 - *Directo a memoria*: la instrucción indica una posición de memoria para el operando
 - *Indirecto*: la instrucción indica un registro que contiene la posición de memoria para el operando



Ejemplo de formato de instrucción

- Para la arquitectura del ejemplo anterior, con los siguientes parámetros:
 - Código de operación: 5 bits
 - Banco de 8 registros (direcciónamiento directo con 3 bits)
- Formato de instrucción

Opcode	OpA	OpB	OpD	Otros
--------	-----	-----	-----	-------



Ejemplos de instrucciones

Operación	Nemónico	Código de operación
Carga dato de memoria	LD	00000
Almacena dato en memoria	ST	00001
Suma	ADD	01000
Resta	SUB	01001
NOT	NOT	01100
AND	AND	01101
OR	OR	01110
Desplazamiento	SHL, SHR	01111
Salto incondicional	JMP	10000
Llamada a subrutina	CALL	10100
Retorno de subrutina	RET	10101
...



Ejemplos de instrucciones

- Instrucción de una palabra

Opcode	RFA	RFB	RD	Otros
01000	001	010	011	00

R3 = R1 + R2

- Instrucción de dos palabras

Opcode	RFA	RFB	RD	Otros
00000	000	000	011	00
Direccion				
1010 1011 1100 1101				

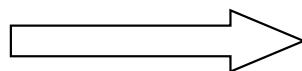
**Carga en R3 el dato
contenido en la posición de
memoria ABCDh**



Lenguaje ensamblador

- Los códigos de instrucción resultan muy poco manejables
- *Lenguaje ensamblador*: las instrucciones se especifican mediante nemónicos y los operandos mediante nombres simbólicos
- Programa ensamblador: traducen las instrucciones de un programa a su código correspondiente
- Ejemplo:

ADD R1, R2, R3



0100000101001100



Bibliografía

- Webs de fabricantes:
 - Xilinx: www.xilinx.com
 - Altera: www.altera.com
 - Actel: www.actel.com
 - Lattice: www.latticesemi.com
- “Principios de Diseño Digital”. D. Gajski. Ed. Prentice Hall
- “Fundamentos de Sistemas Digitales”. Thomas L. Floyd. Pearson Prentice Hall
- “Sistemas digitales: principios y aplicaciones”, Tocci, Ronald J. Pearson Prentice Hall
- “Dispositivos lógicos programables (PLD): diseño práctico de aplicaciones”. García Iglesias, José Manuel. RaMa