Colección Manuales Docentes

Lógica digital y tecnología de computadores

Un enfoque práctico mediante simulación con Logisim

Jesús Salido Tercero

Ediciones de la Universidad de Castilla-La Mancha

Lógica Digital y Tecnología de Computadores

Un enfoque práctico mediante simulación con Logisim

Jesús Salido Tercero

Ediciones de la Universidad de Castilla-La Mancha

UXT (Thema)

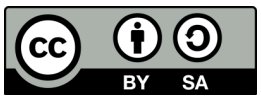
Edita: Ediciones de la Universidad de Castilla-La Mancha.

Colección Manuales docentes n.º 26.

Esta editorial es miembro de la UNE, lo que garantiza la difusión y comercialización de sus publicaciones a nivel nacional e internacional.

I.S.B.N.: 978-84-9044-621-8 (Edición electrónica)
D.O.I.: https://doi.org/10.18239/manuales_2023.26.00
ISNI: 0000000506819532 (Ediciones UCLM)

Hecho en España (U.E.) – *Made in Spain (E.U.)*



A Dorita,
JuanPa y Jacobo

Lógica Digital y Tecnología de Computadores
Un enfoque práctico mediante simulación con Logisim

Jesús Salido Tercero

Resumen

Esta obra está concebida como manual docente para la asignatura de primer curso *Tecnología de Computadores* impartida en el grado en Ingeniería Informática de la Universidad de Castilla-La Mancha, aunque puede ser empleado en otras titulaciones para materias relacionadas con el Diseño y Análisis de Sistemas Digitales. El texto se divide en tres partes:

I.- **Representación de información y lógica booleana** (Caps. 1 a 3),
II.- **Sistemas Combinacionales** (Caps. 4 a 6), y
III.- **Sistemas Secuenciales** (Caps. 7 a 9).

El rasgo diferenciador de esta obra reside en su enfoque práctico mediante utilización intensiva de simulación lógica de circuitos digitales. De este modo se facilita la experimentación inmediata e intuitiva de los circuitos digitales, independientemente de su complejidad. En una primera aproximación, la simulación lógica de los circuitos permite obviar los aspectos prácticos de su realización física, otorgando más relevancia a los aspectos formales de su análisis y diseño lógico. La herramienta de simulación *opensource* elegida en esta obra ha sido Logisim [1] (www.cburch.com/logisim), un programa multiplataforma de software libre.

En los nueve capítulos de esta obra se incluyen 122 ejemplos prácticos explicados que cubren tanto el análisis como diseño de sistemas lógicos, y alrededor de 170 circuitos lógicos elaborados con Logisim, que se pueden recrear fácilmente. Cada capítulo finaliza con una sección de problemas propuestos, hasta un total de 174, cuya resolución se aborda en una obra complementaria —en preparación—.

❧ ❧

Agradecimientos

Esta obra ve la luz gracias a muchas personas que directa o indirectamente han facilitado algún aspecto de su producción. Por dicho motivo esta sección puede ser injusta con aquellas que no son mencionadas expresamente, pero a quienes igualmente estoy muy agradecido. Por su relación con el contexto académico de esta obra, agradezco a mis compañeros, los profesores Antonio Adán e Inocente Sánchez, el material preparado durante su dilatada experiencia como docentes en la asignatura de *Tecnología de Computadores* impartida en la ESI-UCLM. Del mismo modo, siento una enorme gratitud hacia mis estudiantes —sin excepción— porque son la motivación para cuestionar siempre mis métodos docentes facilitando en este camino mi aprendizaje continuo. También doy las gracias al personal —PDI y PAS– de la ESI-UCLM, del Departamento de Ingeniería Eléctrica, Electrónica, Automática y Comunicaciones (IEEAC), y de la UCLM que hacen posible mi labor como docente e investigador, encontrando siempre apoyo en tan excelentes profesionales.

En gran parte esta obra es posible gracias a la labor del Dr. Carl Burch creador de Logisim. La composición tipográfica de esta obra ha sido factible gracias a la utilización de LaTeX empleando TeXstudio como entorno integrado de desarrollo. Por ello, deseo expresar mi sincero agradecimiento a todas las personas (creadores, contribuidores, mantenedores, etc.) que han proporcionado desinteresadamente a la comunidad, las herramientas software empleadas en esta obra.

No puedo olvidar a mi círculo próximo constituido por familiares —en especial a mis padres, Mari y Sagra[†]— y amistades: personas que con su paciencia y benevolencia soportan muchas de mis limitaciones y hacen posible, por innumerables razones, que pueda aportar mi humilde grano de arena a la sociedad.

Por último, agradezco a quien encuentre de utilidad esta modesta obra, el envío por correo electrónico de erratas (asunto: `TeCo-contrib`), omisiones y en general cualquier sugerencia destinada a mejorar el resultado final.

`jesus.salido@uclm.es`
Ciudad Real, 2023

Prefacio

La *Tecnología de Computadores* es una materia obligatoria en los estudios universitarios de *Ingeniería Informática*. Sus contenidos, tratados también en otras titulaciones técnicas relacionadas, se centran en el estudio de los módulos constituyentes de los sistemas digitales para abordar su análisis y diseño.

Los aspectos del funcionamiento electrónico y lógico de los sistemas digitales han sido cubiertos por excelentes obras de referencia en el entorno académico, como las preparadas por el profesor Thomas L. Floyd [2] y el profesor Morris Mano [4], entre otras. En este contexto el rasgo diferenciador de la modesta obra que aquí se presenta, reside en su enfoque práctico mediante el uso intensivo de la simulación lógica de circuitos digitales. Su propósito es facilitar la experimentación inmediata e intuitiva de los circuitos lógicos digitales, independientemente de su complejidad.

La simulación lógica de circuitos permite obviar los aspectos prácticos de implementación física que son menos importantes en un primer curso de Ingeniería Informática, en el que adquiere mayor relevancia aprender los métodos de análisis y diseño lógico de sistemas. La herramienta de simulación lógica elegida para la preparación de esta obra ha sido el programa informático *Logisim* [1] (www.cburch.com/logisim). Esta elección se basa en las características siguientes que lo describen como:

- *Intuitivo*: Se requiere poco esfuerzo para aprender, en cuestión de minutos, los conceptos básicos que permiten la simulación lógica del primer circuito.
- *Multiplataforma*. Al estar programado en Java, se puede ejecutar de igual modo en cualquiera de los sistemas operativos más habituales como Windows, MacOS o Linux.
- *Gratuito*. Está disponible en Internet sin coste ni limitaciones de uso. Además, debido a los escasos recursos necesarios, se puede usar en equipos informáticos muy modestos.
- *Abierto*. Su código fuente es público,[1] lo que facilita su estudio y mejora continua.

Aunque en la actualidad Logisim se emplea en más de 150 centros académicos de todo el mundo, existe carencia de manuales docentes en los que se aborde su uso en materias relacionadas con el estudio de la *Lógica Digital* y la *Tecnología de Computadores*. Este hecho justifica el esfuerzo para preparar este manual en el que se proporcionan circuitos de ejemplo con Logisim para facilitar su análisis y simulación lógica, cubriendo en su totalidad los contenidos típicos de la asignatura de *Tecnología de Computadores*.[2]

[1] Disponible en: `https://sourceforge.net/projects/circuit/`.

[2] Asignatura semestral (6 créditos ECTS) impartida en la UCLM en el primer curso del grado en Ingeniería Informática.

El texto de esta obra se divide en tres partes:

I.- **Representación de información y lógica booleana** (Caps. 1 a 3).
Comienza con un análisis breve de la naturaleza de la información y su representación en los sistemas digitales. El Capítulo 2 se centra en el manejo de información numérica mediante los sistemas de numeración ponderados para motivar la existencia de distintos códigos binarios y las operaciones aritméticas elementales efectuadas con ellos. Esta parte finaliza con un capítulo amplio destinado a describir el alcance del *álgebra de Boole* a través de sus resultados y herramientas de aplicación a los circuitos digitales construidos con puertas lógicas elementales.

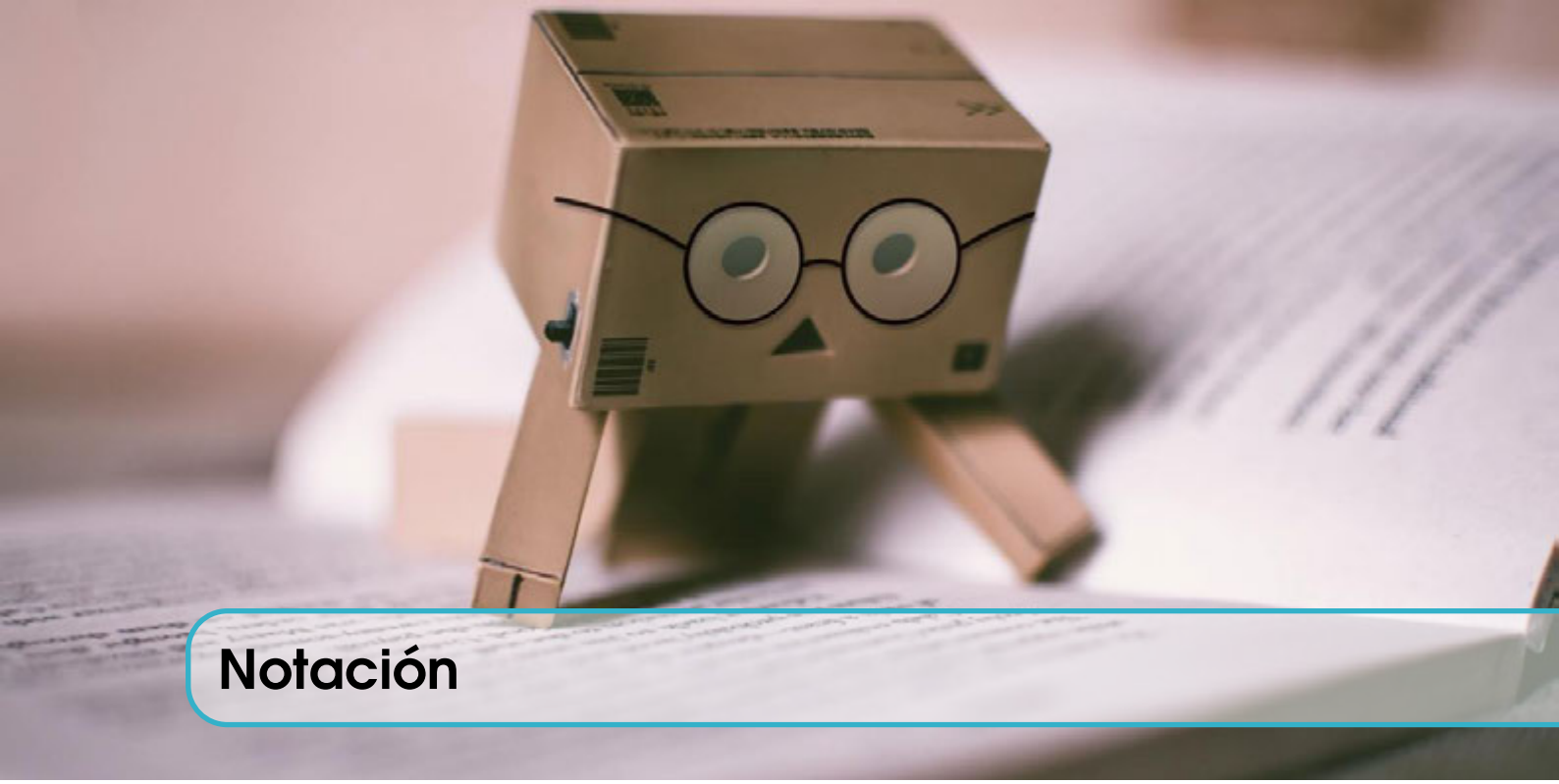
II.- **Sistemas Combinacionales** (Caps. 4 a 6).
La segunda parte de esta obra está dedicada al análisis y diseño de sistemas digitales combinacionales. En ella se describen los principales módulos combinacionales empleados en el *diseño jerárquico* de sistemas digitales que cubren la funcionalidad de transmisión de datos, conversión de códigos y operaciones aritmético-lógicas.

III.- **Sistemas Secuenciales** (Caps. 7 a 9).
Contempla la descripción del funcionamiento de los distintos tipos de *biestables* que constituyen las celdas elementales de almacenamiento del estado en los sistemas digitales. Además, se describen los dos tipos de módulos más importantes en los sistemas digitales secuenciales: *registros* y *contadores*. Esta parte finaliza con un capítulo destinado al análisis y las técnicas de diseño de *sistemas secuenciales síncronos* mediante el empleo de los *autómatas finitos*.

A lo largo de nueve capítulos se incluyen 122 ejemplos prácticos explicados que ilustran los contenidos y procedimientos expuestos, además de sugerir la realización de ejercicios complementarios. La obra contiene alrededor de 170 circuitos lógicos que se pueden recrear fácilmente con Logisim. Cada capítulo finaliza con una sección de problemas propuestos, hasta un total de 174, cuya resolución se abordará en una obra complementaria —en preparación—.

Esta obra ha sido concebida como una guía de estudio personal en la materia que puede ser empleada también como un manual de referencia para el profesorado. Se distribuye de modo libre y gratuito para fomentar su accesibilidad y promover su uso sin limitaciones, esperando que pueda ser de utilidad a cualquier estudiante con interés en sus contenidos, e incluso para el personal docente en la preparación de clases de teoría y práctica. El autor agradece a cualquier persona que utilice esta obra, la comunicación de erratas, omisiones y cualquier sugerencia que contribuya a mejorar su aprovechamiento.

Jesús Salido Tercero
(jesus.salido@uclm.es)
Ciudad Real, 2023



Notación

$A, B, C, D, \ldots$:	Variables lógicas. Se empleará preferentemente letra mayúscula.
$f(A,B,C)$:	Función lógica f dependiente de las variables A, B y C.
$f(A,B,0)$:	Valor de la función $f(A,B,C)$ si $C = 0$.
$f(A,B,1)$:	Valor de la función $f(A,B,C)$ si $C = 1$.
X/y :	Entrada/Salida de sistema secuencial.
$X = [ABC\ldots] \equiv [X_{n-1}\ldots X_i\ldots X_1 X_0]$:	Variable de entrada multibit definida como agrupación de n bits.
$y = [fgh\ldots] \equiv [y_{n-1}\ldots y_i\ldots y_1 y_0]$:	Función de salida multibit definida como agrupación de n bits.
$Q_i = [q_{n-1}\ldots q_j\ldots q_1 q_0]$:	Estado de sistema secuencial codificado como agrupación del estado binario individual de n biestables.
$[s_0, s_1, \ldots, s_{n-1}]$:	Secuencia de n valores, habitualmente binarios multibit.
$A' \equiv \overline{A}$:	Operador NOT (negación).
$AB \equiv A \cdot B$:	Operación lógica AND o producto lógico.
$(AB)' \equiv (A \cdot B)' \equiv \overline{A \cdot B}$:	Operación lógica NAND (producto negado).
$(A+B)' \equiv \overline{A+B}$:	Operación lógica NOR (suma negada).
$A \oplus B = AB' + A'B$:	Operación lógica XOR (OR exclusivo).
$A \odot B \equiv (A \oplus B)' = AB + A'B'$:	Operación lógica XNOR (OR exclusivo negado).
m_i :	Minitérmino o producto canónico, con valor binario i.
M_j :	Maxitérmino o suma canónica, con valor binario j.
$\sum m(3,4) \equiv m_3 + m_4$:	Suma abreviada de minitérminos.
$\prod M(3,4) \equiv M_3 \cdot M_4$:	Producto abreviado de maxitérminos.
$*$:	Valor comodín (0/1, A' o A) de variable, empleado en la expansión de términos no canónicos.
x :	Empleado para representar las condiciones libres de una función lógica o un valor flotante en simulación con Logisim.

Índice general

Representación de información y lógica booleana

1. Introducción a los sistemas digitales

Para entender los principios de funcionamiento de los sistemas digitales es esencial comprender la naturaleza y representación de la información con la que estos operan. En este capítulo se analiza las características de las señales digitales y la información contenida en ellas.

1.1 Señales digitales vs. señales analógicas

Atendiendo a la naturaleza de su origen, la información procesada en los sistemas digitales se puede considerar de dos tipos:

Continua. Se asocia habitualmente con magnitudes físicas cuya evolución temporal admite una representación matemática mediante una función continua, ya que puede adoptar cualquier valor dentro de un rango. Además, dicho valor puede cambiar en cualquier instante de tiempo. Por ejemplo: temperatura, presión, velocidad, aceleración, etcétera. Para estas magnitudes físicas también se cumple que a un intervalo infinitesimal de tiempo corresponde una variación infinitesimal de la magnitud.[1]

Discreta. Si toma solo valores determinados. Esto es, cambia a «saltos». Se puede asociar a acciones puntuales de tipo TODO/NADA (encendido/apagado) y a información elaborada artificialmente, como los símbolos empleados para nombrar y contar cosas o atributos asociados a una persona. Por ejemplo: número de hijos, estado civil, DNI, estado de un interruptor, día de la semana, etcétera.

Dependiendo del mecanismo empleado para la representación de información, esta se puede clasificar como:

Analógica. Si la representación admite infinitos valores dentro de un rango determinado.

Digital. Cuando la representación solo permite un número finito de valores contenidos en un rango. Como los sistemas digitales emplean una representación basada en el código binario, compuesto por concatenación de dígitos '1' y '0', se asume que los sistemas digitales son aquellos que emplean el código binario para la representación de información con una precisión fijada por el número de dígitos binarios (*bits*) empleado y el rango de valores representados.

[1] En lenguaje matemático esta propiedad se conoce como *diferenciabilidad*, equivalente a afirmar que la función tiene derivada finita.

La figura 1.1 muestra distintos tipos de dispositivos de representación analógica (termómetro de mercurio y reloj de agujas) y digital (display de temperatura y despertador digital).

(a) Analógico (b) Digital

Figura 1.1: Dispositivos de representación analógicos y digitales.

> Recuerda que «digital» y «continuo» no son términos antagónicos, ya que tanto si la información original es continua como discreta, admitirá una representación digital mediante códigos construidos por concatenación de bits (‘0’ y ‘1’). Eso sí, en cualquier método de representación de información se emplea un conjunto finito de símbolos determinado por la *resolución* requerida.

1.2 El sistema binario

El sistema binario emplea dos símbolos o bits[2] $\{0, 1\}$ asociados a los dos únicos dígitos posibles para representar la información digital. La elección del sistema binario se debe a la facilidad con que es posible reproducir, mediante dispositivos electrónicos, los dos estados o niveles lógicos representados por un bit. La interpretación de estos dos estados puede ser muy diversa. Por ejemplo: 0/1, *abierto/cerrado*, *apagado/encendido*, *verdadero/falso*, etcétera.

Aunque es habitual asociar el nivel lógico ‘1’ con un valor mayor de tensión (p. ej., 5 V) que el nivel lógico ‘0’ (p. ej., 0 V), en la práctica dicha asociación es dependiente del sistema digital concreto y la tecnología microelectrónica empleada para su fabricación. En realidad, los estados lógicos de una señal están asociados a rangos excluyentes de valores de tensión, dependiente de la tecnología empleada. En la sección 1.4 (pág. 9) se amplía información sobre este aspecto.

Los estados binarios de un sistema digital se obtienen habitualmente como salida de un componente electrónico denominado *transistor*. El funcionamiento simplificado de un transistor se asemeja al de un interruptor microscópico, sin partes móviles, cuyo estado (abierto/cerrado) está controlado por una señal eléctrica de pequeña potencia. Mientras que un interruptor mecánico cambia de estado por aplicación de una fuerza externa, un transistor es gobernado por una señal eléctrica. Así ocurre con los transistores MOSFET, cuya tensión V_{GS} gobierna el estado de conducción del transistor. A su vez, las señales de gobierno se pueden generar por otros transistores, de modo, que la concatenación de acciones muy simples, permite construir sistemas digitales flexibles con funciones lógicas muy complejas.

La capacidad de cómputo y procesamiento de información en los sistemas digitales actuales deriva del gran número de transistores que son capaces de albergar sus unidades de procesamiento —en la actualidad un único circuito integrado llega a albergar decenas de miles de millones de transistores— y su velocidad para cambiar de estado —del orden de mil millones de cambios por segundo—.

La figura 1.2 muestra el esquema de un transistor MOSFET *(Metal oxide semiconductor Field-effect)* de efecto de campo tipo nMOS. La aplicación de una tensión (V_G=5 V) en el terminal G de puerta *(gate)* de dicho transistor provoca la formación de un canal con portadores de carga negativa en el semiconductor que lo compone, de modo que se permite el paso de corriente eléctrica entre los terminales S *(source)* y D *(drain)*. En este caso el transistor se comporta como un interruptor cerrado que permite el paso de corriente entre los terminales mencionados. Se dice que el transistor está en estado de *conducción*. Por el contrario, cuando V_G=0 V, el transistor no conduce y su comportamiento es análogo al de un interruptor abierto que impide el paso de corriente entre sus terminales. En este caso también se dice que el transistor está en estado de *corte*.

[2] Del inglés ***bi**nary dig**its***

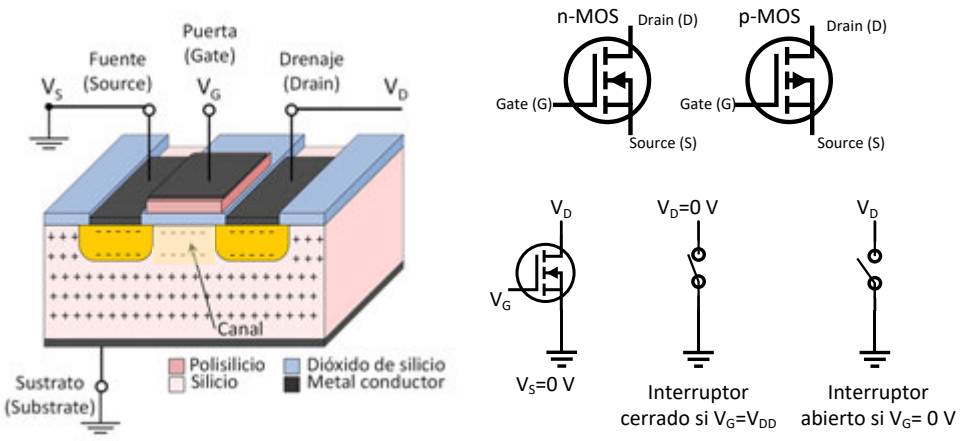


Figura 1.2: Gráfico de la composición de un transistor MOSFET de *tipo npn*, símbolos y diagramas eléctricos equivalentes.

Mediante diseño jerárquico, un sistema digital se construye a partir de módulos funcionales sencillos cuya agrupación permite crear nuevos módulos para aumentar progresivamente la complejidad del sistema final. Como se indica en la figura 1.3, el nivel más sencillo de un sistema digital está constituido por transistores. A partir de ellos se construyen las puertas lógicas elementales (NOT, AND, OR, etc.), cuya agrupación permite diseñar módulos combinacionales básicos (*decodificador*, *multiplexor*, *codificador*, etc.). Mediante la asociación de sucesivos niveles de complejidad en el sistema digital se obtienen diseños con mayor funcionalidad (ver figura 1.3) y sucesivos niveles de abstracción.

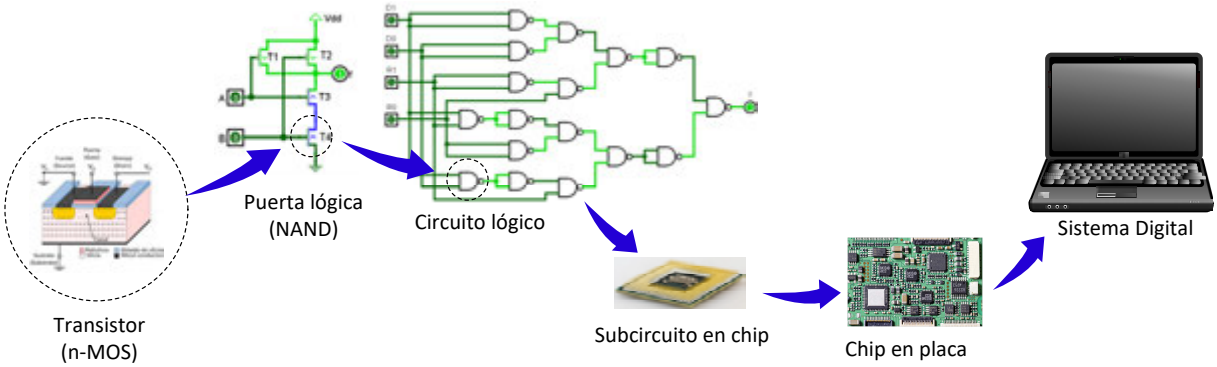


Figura 1.3: Esquema jerárquico de estructura de un sistema digital

> Ten en cuenta que en un sistema digital los ceros y unos no existen como tales. En realidad, se trata de dos estados posibles a los que se representa —por conveniencia— con los símbolos '0' y '1' para desarrollar la teoría de los sistemas digitales de información basada en el *Álgebra de Boole*. Los dos estados lógicos en los que se basan los sistemas digitales pueden ser de distinta naturaleza (óptica, magnética, mecánica, etc.) pero en la actualidad su manipulación en un computador se realiza mediante su conversión a señales de tipo eléctrico.

1.3 Codificación digital

La codificación digital de la información emplea un código binario para su representación. Las codificaciones posibles se basan en concatenar un número determinado de bits {0, 1} de modo que cada combinación concreta de bits represente únicamente un valor. Independientemente del sistema de codificación empleado, en los sistemas digitales las agrupaciones de bits más habituales son:

Nibble. Agrupación de 4 bits. Por ejemplo, 0011.
Byte. Agrupación de 8 bits (2 nibbles). Por ejemplo, 0011 1010.
Word (palabra). Agrupación de 16 bits (2 bytes). Por ejemplo, 0011 1010 1011 1110.
Double word (palabra doble). Agrupación de 32 bytes (2 palabras).

Con la codificación digital se puede representar tanto información numérica como simbólica:

Numérica. Representa cualquier número entero o real asociado a una magnitud cuantitativa, teniendo en cuenta que la resolución máxima que se consigue depende tanto del rango de valores a representar como del número de bits utilizados en la representación. Por ejemplo, el número 5 se representa en *binario natural* como 0101 con un *nibble*. Este tipo de representación también se emplea para codificar enumeraciones de valores ordenados o desordenados. Por ejemplo, días de la semana (a los que asociamos un código de 7 valores posibles), meses del año y enumeraciones sin orden implícito, como colores, nombres, etcétera.

Simbólica. Emplea la agrupación de un conjunto finito de símbolos a los que se asigna una codificación individual binaria de longitud fija. Por ejemplo, códigos ASCII (ver figura 1.4).

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
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	{	|	}	~	DEL

Figura 1.4: Tabla de códigos ASCII.

En la tabla ASCII de la figura 1.4, el código binario correspondiente a cada símbolo se obtiene mediante su equivalente hexadecimal obtenido de combinar fila y columna. Por ejemplo, a la letra 'A' le corresponde el código hexadecimal 41 (fila 4, columna 1). Por tanto, su código binario ASCII asociado es $0100\,0001 \equiv 41_{(16}$. En la sección 2.4.3 (ver pag. 21) se explica cómo pasar de código hexadecimal a binario y viceversa.

1.3.1 Conversión Analógica-Digital

Cuando los sistemas digitales tienen que procesar información continua adquirida del mundo real (p. ej., temperatura, velocidad, etc.), deben convertir dicha información en una representación digital que utilice una codificación binaria. A dicha conversión se le denomina conversión A-D (analógica a digital). Si la información digital se debe volver a representar en el mundo real mediante una señal continua, el proceso se realiza mediante la conversión en sentido inverso, denominada conversión D-A (digital a analógica). Los dispositivos encargados de la conversión se conocen como convertidores ADC *(Analog to Digital Converter)* y DAC *(Digital to Analog Converter)*. Un ejemplo del proceso descrito es el que tiene lugar en la grabación de sonido en formato digital (conversión A-D) y la posterior reproducción (conversión D-A).

En la conversión A-D, primero se lleva a cabo un proceso de muestreo *(sampling)* en el que se obtienen los valores de la señal analógica en instantes separados por un intervalo regular de tiempo. Es decir, el tiempo transcurrido entre la toma de dos muestras consecutivas es fijo y se denomina *periodo de muestreo* (T). Para cada valor de la señal original obtenido en un instante de muestreo, se calcula el código binario correspondiente mediante un proceso denominado *cuantización*. En dicho proceso se asigna el código del valor digital más próximo en la codificación empleada para el valor numérico de la muestra. La figura 1.5 ilustra este proceso de conversión en dos etapas: (1) *muestreo* y (2) *cuantización*.

El código digital asociado a cada muestra depende del número de bits empleado en la conversión del valor y del rango de valores a convertir. Así, la *resolución* o error máximo de la conversión que se comete al emplear n bits es:

$$\text{Resolución} = \frac{\text{Rango}}{2^n} \tag{1.1}$$

Cuanto mayor es el número de bits empleado en la conversión de un rango de valores, menor es el error máximo que se comete en la conversión. Por tanto, cuanto más bits se emplean, mayor es la precisión

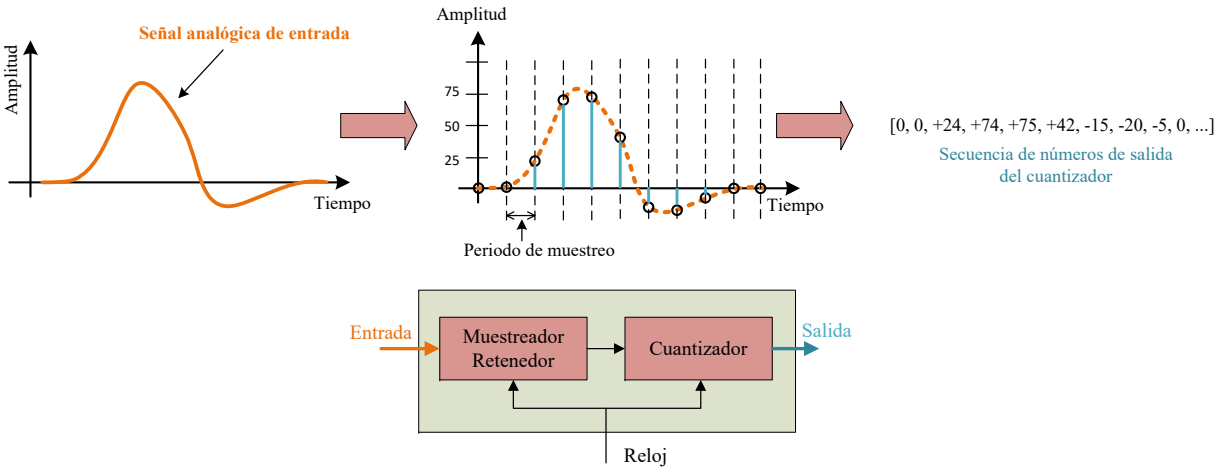


Figura 1.5: Ejemplo de conversión A-D.

alcanzada en la conversión y menor el intervalo entre dos códigos consecutivos. En la figura 1.6 se observa la aproximación de la representación digital a la señal analógica original en función del número de bits empleado en la conversión.

Para calcular el código digital que corresponde a un valor de la señal analógica que se convierte a digital, se debe tener en cuenta la relación siguiente:

$$\frac{\text{Código digital de conversión}}{\text{Valor analógico entrada (V)}} = \frac{\text{Código digital máximo}}{\text{Valor máximo señal analógica (V)}}$$

Teniendo en cuenta que el código digital máximo expresado con n bits es:

$$\text{Código digital máximo} = 2^n - 1$$

se tiene que:

$$\text{Código de conversión} = \frac{2^n - 1}{\text{Valor máximo analógico (V)}} \times \text{Valor de entrada (V)} \tag{1.2}$$

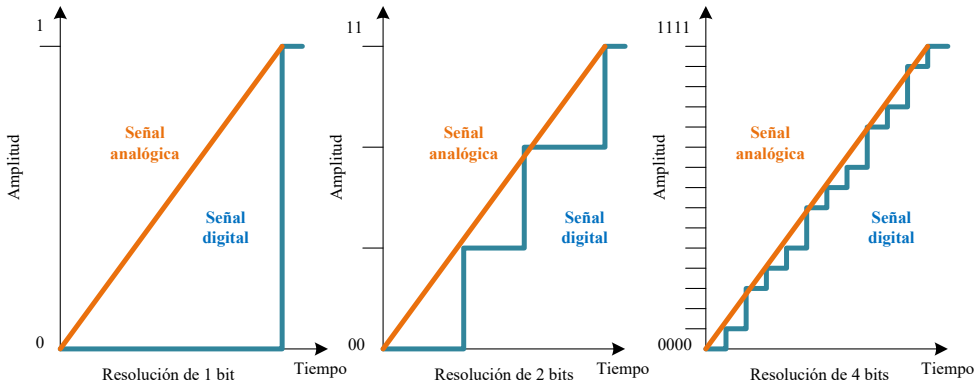


Figura 1.6: Información perdida entre escalones consecutivos de discretización en el proceso de conversión A-D dependiendo del número de bits empleado.

Ejemplo 1.1 — Conversión de una magnitud analógica a su valor digital. Si un sistema digital posee un conversor A-D de 10 bits, con una tensión máxima de entrada de 5 V y la entrada al conversor es de 1.72 V ¿qué código digital de conversión le corresponderá?

Solución:

Aplicando la relación expresa en la ec. 1.2, se tiene que:

$$x = \frac{(2^{10}-1)}{5V} \cdot 1.72V = \frac{1\,023}{5} \cdot 1.72 = 351.91$$

Por tanto se toma el valor entero más próximo: 352. Dicho valor expresado en binario con 10 bits es: 01 0110 0000. ■

Junto a la resolución, la elección apropiada del periodo de muestreo (T) de la señal original determina la fidelidad de la conversión A-D. Por ejemplo, si T es muy grande, las muestras están muy separadas en el tiempo. En este caso es probable que se pierdan detalles de la señal original. Por el contrario, si T es muy pequeño, las muestras están muy próximas y la información recogida en muestras sucesivas puede ser redundante si la señal cambia lentamente de valor. La figura 1.7 ilustra el caso de un periodo de muestreo T demasiado grande con el que se pierden los cambios de la señal original entre muestras sucesivas adquiridas para la conversión digital.

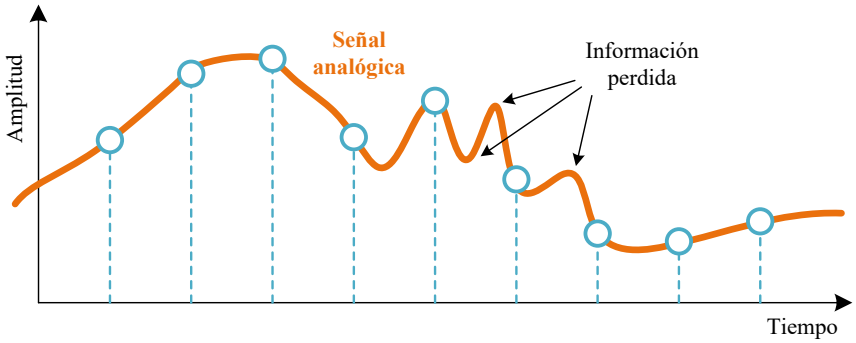


Figura 1.7: Información perdida entre muestras sucesivas durante la conversión A-D en una señal con cambios de valor muy rápidos (alta frecuencia) con respecto al periodo de muestreo.

El *Teorema del Muestreo*[3] proporciona un modo eficiente para elegir el periodo de muestreo óptimo para convertir en digital una señal analógica dada. Dicho teorema demuestra que para retener toda la información relevante de la señal a convertir, la frecuencia mínima (periodo máximo) de muestreo debe ser al menos el doble de la frecuencia máxima presente —derivada de su análisis de Fourier— en la señal a convertir.

1.3.2 Ventajas de la codificación digital

Las señales digitales presentan grandes ventajas frente a las señales continuas de las que proceden, a saber:

- Velocidad y eficiencia de comunicación. Las señales digitales aprovechan mejor los medios de transmisión, ya que permiten utilizar un mayor número de canales sobre el mismo medio compartido. Para conseguir esta eficiencia se emplean técnicas de multiplexación y soportes físicos de transmisión, como la fibra óptica, con atenuación muy reducida.
- Inmunidad de las señales al «ruido» eléctrico. En una señal digital es más importante el nivel lógico de la misma que el valor concreto de la magnitud eléctrica que transmite. El nivel lógico de la señal se preserva, siendo menos afectado por las perturbaciones mientras el valor concreto de la magnitud eléctrica esté dentro del rango asociado al nivel lógico correcto. Además, es posible aplicar técnicas efectivas de procesamiento de la señal, tanto para detección como corrección de errores.
- Almacenamiento eficiente de grandes volúmenes de información. En la actualidad es posible almacenar grandes volúmenes de información digital a un coste muy reducido (p. ej., memorias de

[3] Este teorema fue formulado por Harry Nyquist en 1928 (Certain topics in telegraph transmission theory), y fue demostrado formalmente por Claude E. Shannon en 1949 (Communication in the presence of noise).

estado sólido). Asimismo, es sencillo aplicar algoritmos de compresión que reducen el espacio de almacenamiento requerido.

- Aplicabilidad de algoritmos de procesamiento digital. Se han desarrollado gran cantidad de algoritmos que aplicados a las señales digitales obtienen resultados convenientes como: filtrado, compresión, corrección de errores, codificación, multiplexación, etcétera. Aunque estos algoritmos requieren el cálculo de un número elevado de operaciones matemáticas, se han desarrollado procesadores de señal específicos o DSP *(Digital Signal Processor)* que facilitan el procesamiento mencionado.
- Facilidad de diseño e integración en sistemas. Todas las ventajas anteriores derivan de la facilidad para implementar los algoritmos necesarios en circuitos digitales integrados que pueden formar parte de sistemas más complejos (p. ej., computadores personales, servidores, controladores industriales, electrodomésticos, equipos multimedia, automóviles, etc.). La lista de dispositivos en los que se procesa información digital es tan extensa que aborda numerosas parcelas de la actividad humana y hace imposible imaginar el desarrollo tecnológico alcanzado en la actualidad sin la intervención de los sistemas digitales.

1.4 Características de las señales digitales

Las señales digitales suelen ser de naturaleza eléctrica. Por tanto, su nivel lógico puede depender tanto de su valor de tensión (V), como de su intensidad de corriente (I). En esta obra la característica más importante de las señales digitales es su nivel lógico. En consecuencia, el valor concreto de tensión o intensidad de las señales no será relevante salvo indicación en otro sentido.

Al decidir las magnitudes eléctricas apropiadas para representar los niveles lógicos de una señal digital se recurre a:

- Señales de corriente continua (CC o DC) frente a corriente alterna (AC), pues la electrónica requerida en los sistemas es más simple. Además, tanto el almacenamiento como la generación de corriente continua son sencillos mediante el uso baterías.
- Voltajes e intensidades de valor reducido. Dado que el valor concreto de la magnitud eléctrica es menos relevante que su nivel, es más eficiente escoger valores reducidos de tensión e intensidad para representar los valores lógicos de los sistemas digitales. De este modo los sistemas son más sencillos y se requiere menos energía para su funcionamiento. Algunos de los rangos típicos de las señales eléctricas empleadas en los sistemas digitales son: 0-5 V, 0-3.3 V y 4-20 mA.

Los niveles lógicos utilizados en los sistemas digitales dependen de las tecnologías utilizadas. La tabla 1.1 muestra los valores de las tensiones correspondientes a los niveles lógicos 0 y 1 cuando se emplean tecnologías TTL (*Transistor-transistor Logic*) o CMOS (*Complementary metal–oxide–semiconductor*).

Tabla 1.1: Rangos de tensión asociados a niveles lógicos para las familias tecnológicas TTL y CMOS.

Tecnología	0 lógico (V)	1 lógico (V)
TTL	0–0.8	2.0–5
CMOS	0–1.35	3.15–5

> Recuerda que adicionalmente al convenio respecto de los valores de la señal eléctrica empleada, es posible establecer un criterio para su interpretación como un valor lógico. Habitualmente se asume que un nivel alto tiene asociado el valor lógico '1' y el nivel bajo se asocia al '0' lógico. Pero es posible la interpretación opuesta de los niveles lógicos y en algunos casos ocurre de este modo.

1.4.1 Forma de onda de una señal

En la figura 1.8 se observa la forma de onda aproximada de una señal digital cuyas transiciones se han exagerado a afectos didácticos. En dicha figura se señalan los parámetros que definen la forma de onda de una señal digital:

Flanco de subida. *(Rising edge)* Transición de la señal desde nivel bajo a nivel alto. También se denomina *flanco positivo*.

Flanco de bajada. *(Falling edge)* Transición de la señal desde nivel alto a nivel bajo. También se denomina *flanco negativo*.

Tiempo de subida. *(Rising time)* Es la duración de un flanco de subida de la señal. Para acotar los instantes concretos desde los que realizar la medida, se suele acotar el cambio de la señal en el rango 10-90 % o 5-95 %, considerando este porcentaje sobre el nivel máximo de la señal.

Tiempo de bajada. *(Falling time)* Duración del flanco de bajada de la señal.

Retardo de propagación. *(Propagation delay)* Es el tiempo transcurrido entre los valores del 50 % de una señal de salida de un sistema y la señal de entrada que la provoca. Este tiempo se puede medir tanto en las transiciones de subida de las señales como en las de bajada.

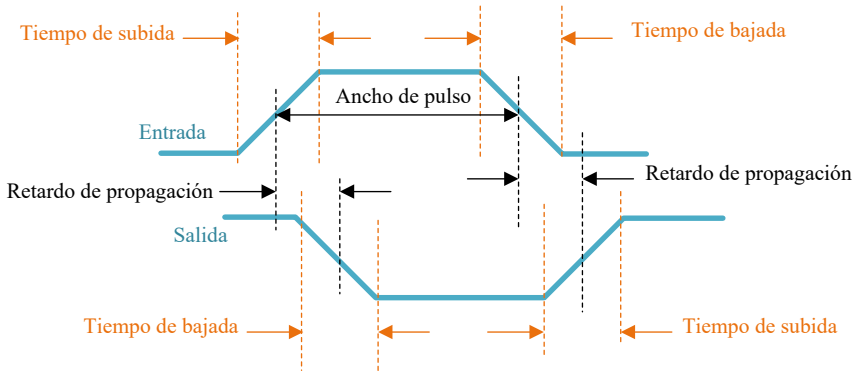


Figura 1.8: Parámetros de la forma de onda correspondiente a la entrada y salida de un sistema digital constituido por un inversor.

Se denomina *pulso* a la porción de onda que transcurre entre dos flancos consecutivos de la señal, suficientemente próximos. Dependiendo del nivel lógico del pulso entre dos flancos consecutivos de la señal, se denomina *pulso positivo* (con nivel alto entre flancos) o *pulso negativo* (con nivel bajo entre flancos). Al tiempo transcurrido entre los flancos de un pulso se le conoce como *duración de pulso* o *ancho del pulso (pulse width)* (ver Figs. 1.8 y 1.9).

Los parámetros mencionados son importantes en el diseño de sistemas reales, pero dependiendo del propósito de su estudio, el valor de algunos de estos parámetros se puede idealizar. Así, es habitual asumir el cambio instantáneo de las señales.[4] Esto es, se consideran tiempos de subida y bajada nulos. Por esta razón, las formas de onda digitales se representan con transiciones abruptas representadas por las líneas verticales de la forma de onda (ver Fig. 1.9). Otra idealización relevante considera despreciable el retardo de propagación de las señales.[5]

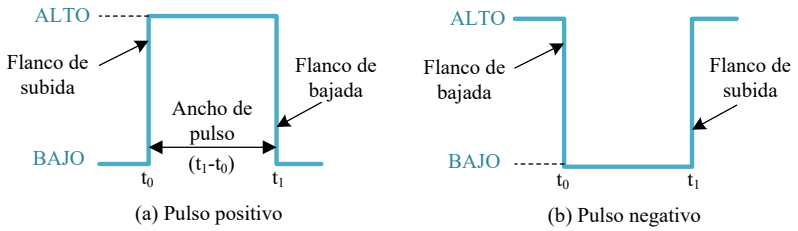


Figura 1.9: Diferentes tipos de pulso de una señal digital.

Una simplificación adicional común en el estudio de los sistemas digitales, supone que las señales están «limpias» de ruido *(noise)*. En los sistemas reales el ruido está presente siempre debido a causas muy diversas (p. ej., ruido electromagnético, ruido térmico, etc.). El ruido es responsable del perfil de sierra irregular que se superpone al valor ideal de una forma de onda (ver ejemplo en Fig. 1.10).

[4] En realidad el cambio se produce en un breve intervalo de tiempo, ya que el cambio instantáneo es imposible por exigir una transferencia de energía infinita hacia el sistema.

[5] Aunque estas idealizaciones se consideran oportunas para la validación a nivel funcional de los sistemas estudiados en esta obra, no son apropiadas para la validación temporal requerida en su implementación física real.

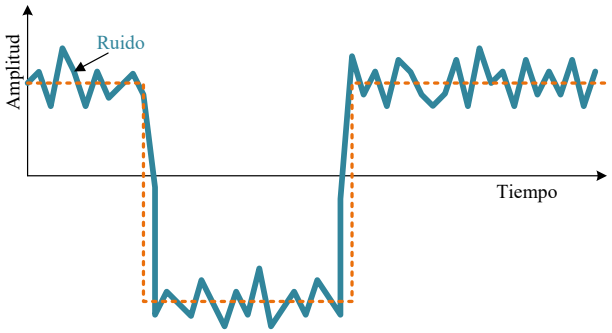


Figura 1.10: Forma de onda de una señal digital perturbada superpuesta a su valor ideal.

En las señales analógicas, la presencia de ruido falsea la información contenida en la señal original. Por ello, se diseñan sistemas con filtros que minimicen dicho ruido. En el caso de las señales digitales, el margen de interpretación de los niveles de tensión actúa como un primer filtro para las perturbaciones. Por tanto, es habitual que el ruido no perturbe el nivel lógico de las señales digitales, siempre que no se sobrepasen los umbrales de tensión asociados a un nivel lógico dado (ver Fig. 1.11 dcha.). Por este motivo las señales digitales se consideran más inmunes al ruido que las señales analógicas. En la figura 1.11 se compara el efecto de ruido en una señal analógica y en una señal digital. Se dice que la señal digital es inmune al ruido, ya que este no afecta al nivel lógico asociado a la señal.

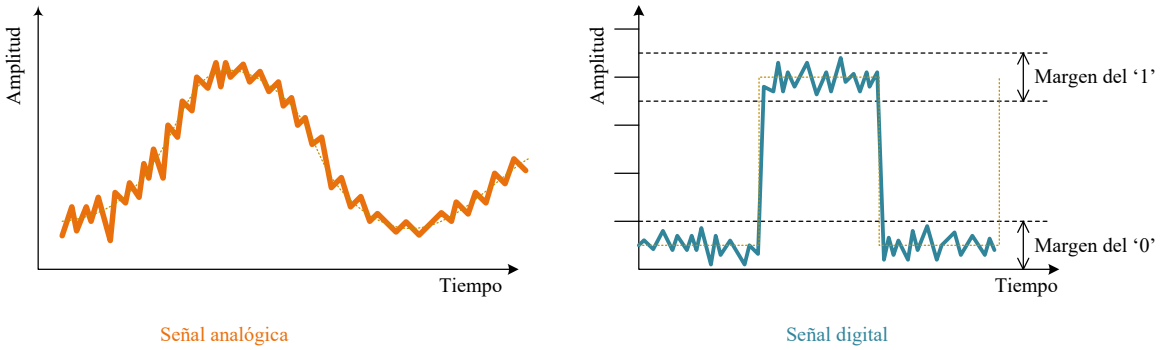


Figura 1.11: Diferencia entre los efectos de perturbación por ruido en una señal analógica (izda.) y en una digital (dcha.).

1.4.2 Periodo y frecuencia de señales periódicas

Cuando se habla de señales que varían en el tiempo, estas se pueden clasificar como:

Señales periódicas. Si su forma de onda sigue un patrón repetido en el tiempo (ver Fig. 1.12 izda.). Se caracterizan por el tiempo que transcurre entre la repetición de la onda o duración del patrón. Dicho tiempo se denomina *periodo* (T) y se mide en segundos. El inverso del periodo se denomina *frecuencia* (f o fr) y expresa el número de veces que se repite la onda por unidad de tiempo. Su valor se mide en ciclos por segundo o hercios (Hz).

Señales aperiódicas. Su forma de onda no sigue un patrón que se repita en el tiempo (ver Fig. 1.12 dcha.).

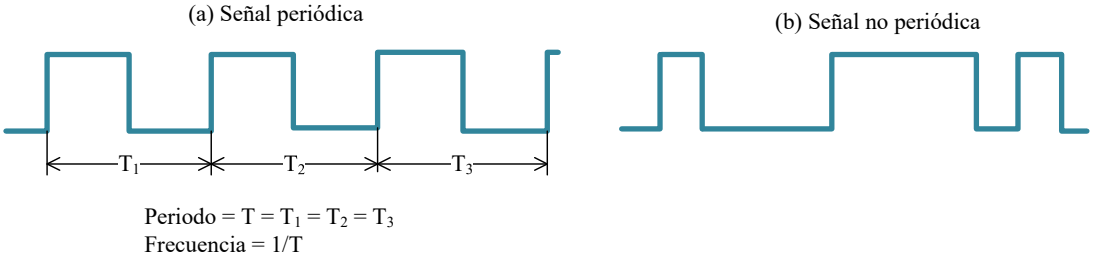


Figura 1.12: Señales digitales periódicas (a) y no periódicas (b).

La *señal de reloj* (*clock* o *CLK*) es una señal periódica importante en los sistemas digitales, ya que permite sincronizar el funcionamiento de distintos módulos del sistema. Esta señal marca el ritmo al que se procesa la información de un sistema digital. Por tanto, la frecuencia del reloj de un sistema condiciona la velocidad máxima a la que se procesa la información. La figura 1.13 muestra una señal de reloj y una señal digital sincronizada para extraer la información binaria asociada.

Las señales digitales periódicas no siempre son simétricas como una señal de reloj. Esto es, el nivel alto y el bajo tienen distinta duración. En la figura 1.14 se muestra una señal de este tipo. Los parámetros que la definen son: el periodo (T) y la anchura del pulso positivo (t_w). En este tipo de señales, el ciclo de trabajo *(duty cycle)* es una forma alternativa de ofrecer el valor del ancho de pulso como un porcentaje del periodo total de la señal. Por tanto, se tiene que:

$$\text{Ciclo de trabajo (duty cycle)} = \frac{t_w}{T} \cdot 100 \tag{1.3}$$

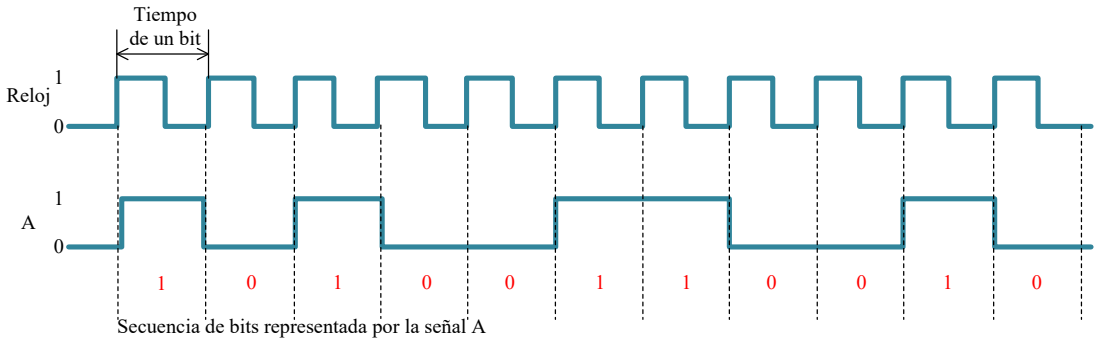


Figura 1.13: Señal de reloj de un sistema digital e información digital asociada a una señal sincronizada con el reloj.

Por ejemplo, una señal con un ciclo de trabajo del 10 % posee un ancho de pulso que es la décima parte del periodo total de la señal (ver Fig. 1.14). Las señales de reloj siempre presentan un ciclo de trabajo del 50 %.

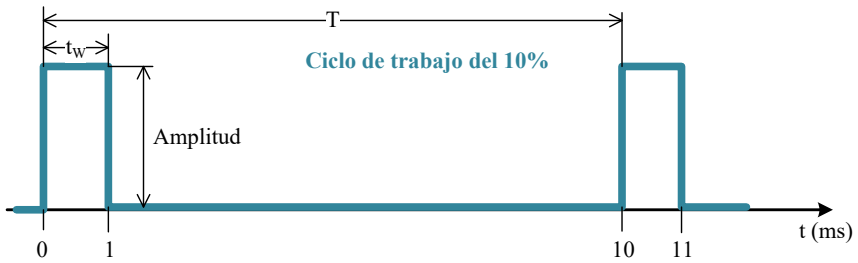


Figura 1.14: Señal digital periódica PWM con un ciclo de trabajo del 10 %.

Este tipo de señales asimétricas permite la codificación de una señal analógica (con múltiples valores) directamente en una señal digital (con dos niveles) utilizando el ancho del pulso como valor asociado a la información analógica. De este modo, el ancho del pulso de la señal resultante sería proporcional a la magnitud del valor a codificar. Este tipo de codificación particular de señales analógicas se denomina *codificación por anchura de pulso* o PWM *(Pulse Wide Modulation)* y dos de sus usos más populares son el control de energía transferida a un sistema (p. ej., iluminación, velocidad, etc.) y codificación de la posición de un servomotor.

Ejemplo 1.2 — Aplicación de PWM. Un caso simple de aplicación de PWM es el control de la intensidad luminosa emitida por un diodo emisor de luz o LED *(light-emitting diode)*. Al alimentar un led con una señal PWM que conmuta entre su valor máximo (V_{DD}) y 0 V con una frecuencia inferior a 40 Hz y un ciclo de trabajo del 50 %, se observa el parpadeo regular del led. Esto es debido a que el

ojo humano sano detecta frecuencias de 40 Hz e inferiores. En esta situación una variación del ciclo de trabajo de la señal se percibe como un parpadeo irregular entre los instantes que el led se mantiene encendido y apagado. Con un ciclo de trabajo superior al 50 % el led estaría más tiempo encendido que apagado y a la inversa en caso de reducir el ciclo de trabajo.

¿Qué sucede si la frecuencia se incrementa por encima de los 40 Hz hasta 50 Hz?

En este caso el ojo humano no es capaz de percibir los instantes en que el led está apagado porque la retina retiene el estímulo luminoso. Por tanto, el led se percibe como si se mantuviese encendido permanentemente. Por este motivo no debe extrañarnos que las luminarias tradicionales se perciban con un brillo constante al ser alimentadas con la tensión alterna de la red de suministro eléctrico a 50 Hz.

¿Cómo afecta en este caso un cambio en el ciclo de trabajo de la señal por encima o por debajo del 50 %?

En esta situación, el cambio de ciclo de trabajo se percibe como un aumento o disminución del brillo. En consecuencia, la alimentación de un led con una señal PWM permite controlar su iluminación mediante el control del ciclo de trabajo de la señal. ■

1.4.3 Cronograma de una señal digital

Para conocer el funcionamiento de un sistema digital real se realiza el análisis de la evolución temporal de las señales digitales implicadas. El *diagrama temporal* o *cronograma (timing diagram)* de una señal digital es una representación de su evolución temporal. La figura 1.15 muestra el cronograma correspondiente a las señales de entrada (A, B) y salida (f) de un circuito digital que efectúa la función lógica OR de las señales de entrada (puerta lógica). En un circuito real, el cronograma se obtiene conectando, las señales eléctricas bajo estudio, a un analizador lógico que muestra en un display la evolución temporal de dicha señales.

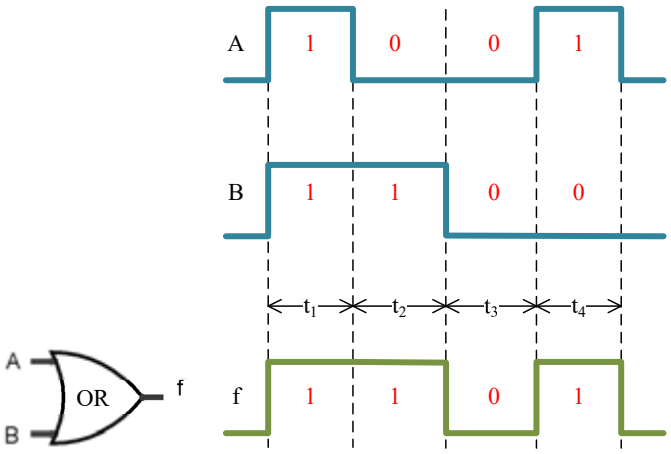


Figura 1.15: Cronograma ideal correspondiente a las señales de entrada y salida de un circuito que calcula la función lógica OR.

> Recuerda que la *tabla de verdad* de la señal de salida de un sistema digital se construye con los valores posibles de todas sus entradas y la salida correspondiente a cada una de ellas. Aunque los valores obtenidos del cronograma de las señales de entrada y salidas de un sistema digital se pueden resumir en su tabla de verdad, en dicha tabla las posibles entradas aparecen ordenadas por su valor binario mientras que en cronograma se representa una secuencia determinada de valores de entrada y las salidas correspondientes.

Los cronogramas son muy útiles en la fase de diseño de sistemas digitales reales pues permiten analizar los efectos derivados de retardos temporales y sus causas. En la figura 1.16 se muestra el aspecto de un cronograma de una señal multibit en el que se muestran los valores lógicos de las señales obtenidas mediante un analizador lógico. Las señales multibit se muestran con los dos estados posibles y en la parte intermedia se especifica el valor binario de la señal —por comodidad se suele emplear su valor

hexadecimal (entre paréntesis)—. Las transiciones de la señal se señalan con líneas oblicuas y con rallado continuo oblicuo en ambas direcciones se indica una situación de indeterminación en la que es imposible conocer el valor exacto de la señal.

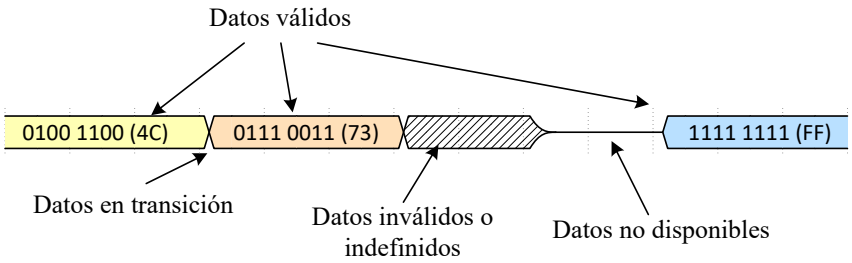


Figura 1.16: Interpretación de un cronograma.

En esta obra los cronogramas se emplearán solo como un recurso alternativo al análisis idealizado del comportamiento de los sistemas digitales sin tener en cuenta los retardos. De este modo los diagramas mostrarán únicamente el valor lógico de las señales.

Conceptos clave

- ☑ Diferencia entre magnitudes continuas y discretas.
- ☑ Representación digital de magnitudes continuas e influencia del número de bits empleado.
- ☑ Relación entre los valores de las señales digitales y los niveles lógicos asociados.
- ☑ Periodo y frecuencia de señales periódicas.
- ☑ Cronograma de una señal digital.

Problemas propuestos

Problema 1.1 ¿De qué tipo es una variable cuyo valor puede ser cualquier día de la semana expresado por su primera letra? Esto es: L, M, X, J, V, S y D.

a) Continua
b) Discreta
c) Digital
d) Analógica

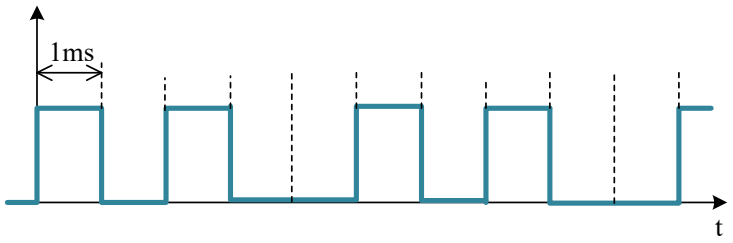
Problema 1.2 ¿Qué resolución tiene la representación digital de una señal de temperatura en el rango [+5 °C, +20 °C] cuando se emplean 8 bits?

Problema 1.3 ¿Cuál es la frecuencia de una señal cuyo periodo (T) tiene una duración de 5 ms?

Problema 1.4 ¿Qué tiempo transcurre entre la repetición cíclica de una señal cuya frecuencia es 1.5 kHz?

Problema 1.5 ¿Cuál es el periodo T de una señal cuyo ciclo de trabajo es del 20 % y duración de pulso de 3 ms? ¿Cuánto tiempo estará dicha señal a nivel bajo en el intervalo de un periodo de la señal? Representa gráficamente dicha señal.

Problema 1.6 Dado el cronograma de la figura adjunta para una señal variable en el tiempo, responde razonadamente. ¿Es dicha señal periódica o aperiódica? ¿Se puede calcular su periodo y su frecuencia?



Problema 1.7 Dibuja el cronograma de una señal PWM con una frecuencia de 200 Hz y un ciclo de trabajo del 40 %. ¿Cuál debe ser su periodo (T) y la anchura del pulso (t_w)?

2. Sistemas de numeración

En los sistemas digitales es necesario representar tanto información numérica como simbólica. Los computadores fueron concebidos inicialmente como máquinas de cálculo numérico y esta sigue siendo una de sus funciones más importantes. Por ello, es relevante estudiar cómo representar la información numérica en los sistemas digitales. En este capítulo se introducen los sistemas de numeración más importantes que utilizan códigos binarios creados por concatenación de los bits '1' y '0'.

2.1 Sistema decimal

En nuestra vida diaria empleamos el sistema decimal de base 10 quizá porque la forma más primitiva de contar empleaba los dedos de las manos. De ahí el origen del término *dígito* (del latín *digitus* ≡ dedo). El sistema decimal consta de 10 dígitos diferentes, del '0' al '9'. Es un sistema posicional aditivo en el que los dígitos o cifras de un número tienen un valor que está relacionado con su posición.[1] Cada dígito representa el factor que multiplica a la potencia de la base numérica (en este caso 10) cuyo exponente es señalado por la posición contada de derecha a izquierda empezando con la posición 0, para las unidades. Dicho de un modo más simple, las posiciones de los dígitos tomados de derecha a izquierda indican el número de unidades (10^0), decenas (10^1), centenas (10^2), millares (10^3) y así sucesivamente, que se van sumando a la magnitud representada. En este sistema es muy importante el uso del 0, pues permite expresar una adición nula en cualquiera de las posiciones.

> Ten en cuenta que en este texto en la representación de números se adopta como convención el punto '.' para separar la parte entera de la parte fraccionaria. Para la separación de miles se empleará un espacio en blanco de anchura inferior al espacio habitual. De este modo se evita la ambigüedad que se podría derivar del uso de la coma ',' empleada para separación de la parte fraccionaria o de los millares en el sistema anglosajón.

Cuando se habla de números con parte fraccionaria la extensión es inmediata. De este modo, se consideran potencias de la base con exponente negativo y decreciente de izquierda a derecha. En consecuencia, las distintas cifras indican el número de décimas (10^{-1}), centésimas (10^{-2}), milésimas (10^{-3}), etcétera.

Ejemplo 2.1 — Expansión numérica de un número en base decimal. Dado el número decimal 203.625, realiza su expansión para calcular su valor numérico.

[1] Por el contrario, un sistema de numeración como el romano no es posicional, ya que los símbolos mantienen su valor independientemente de su posición en la magnitud representada.

Solución:

$$\begin{aligned} 203.625 &= \mathbf{2}\times 10^2 + \cancel{\mathbf{0}\times 10^1} + \mathbf{3}\times 10^0 + \mathbf{6}\times 10^{-1} + \mathbf{2}\times 10^{-2} + \mathbf{5}\times 10^{-3} \\ &= \mathbf{2}\times 100 + \mathbf{3} + \mathbf{6}\times 0.1 + \mathbf{2}\times 0.01 + \mathbf{5}\times 0.001 \end{aligned}$$

Como en los sistemas posicionales el valor aumenta de derecha a izquierda para la parte entera, se denomina *dígito más significativo* (el de mayor valor) al que está más a la izquierda y *dígito menos significativo* (el de menor valor) al que está más a la derecha.[2] En cualquier sistema la adición de un 0 a la izquierda en la parte entera o a la derecha, en la parte fraccionaria, no modifican el valor del número. Cuando se emplea codificación binaria se habla respectivamente del bit más y menos significativo (MSB y LSB) (*Most/Least Significant Bit*).

2.2 Sistema binario natural

El sistema de numeración *binario natural* o simplemente binario sigue el mismo método de cálculo del valor posicional que el sistema decimal, pero en este caso la base es 2 y los únicos dígitos (cifras) posibles son el '0' y el '1'. Los valores de cada una de las posiciones de un número se calculan multiplicando la potencia correspondiente de la base numérica, que en este caso es 2. En la tabla 2.1 se muestran los valores posicionales de los dígitos de un número tanto en su parte entera como fraccionaria.

Tabla 2.1: Valores decimales asociados a cada una de las posiciones del sistema binario.

Potencias positivas de 2 (parte entera)									**Potencias negativas** de 2 (parte fraccionaria)				
2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}
256	128	64	32	16	8	4	2	1	1/2	1/4	1/8	1/16	1/32
									0.5	0.25	0.125	0.0625	0.03125

Ejemplo 2.2 — Conversión a decimal de un número expresado en binario natural. Calcula el valor decimal del número binario 0111.101. Emplea la tabla 2.1 para expandir el valor numérico asociado.

Solución:

$$\begin{aligned} 10111.101_{(2} &= 1\times 2^4 + \cancel{0\times 2^3} + 1\times 2^2 + 1\times 2^1 + 1\times 2^0 + 1\times 2^{-1} + \cancel{0\times 2^{-2}} + 1\times 2^{-3} \\ &= 16+4+2+1+0.5+0.125 \\ &= 23.625_{(10} \end{aligned}$$

El método de expansión mostrado en el ejemplo 2.2 permite hacer la conversión hacia el sistema decimal desde un sistema numérico que emplee cualquier base de representación (p. ej., base 2). Como se muestra en dicho ejemplo, para evitar ambigüedad sobre la base numérica empleada, esta se señala mediante un subíndice a la izquierda para indicar la base. También es habitual que para evitar errores tipográficos, la base se separe del resto mediante un paréntesis abierto como se muestra en los ejemplos: $1011_{(10}$, $1101_{(2}$ y $110_{(8}$.

La tabla 2.2 muestra la equivalencia entre el sistema binario y el decimal para agrupaciones de 4 bits (*nibble*) que es un tipo de agrupación muy empleada en los sistemas digitales (ver Sec. 1.3).

[2] Recuerda que en la parte fraccionaria el valor decrece hacia la derecha. Esto es, cambia la posición del MSB y LSB.

Tabla 2.2: Equivalencia entre los sistemas decimal y binario de 4 bits.

Dec.	Binario			
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
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1

2.3 Rango de representación y otras bases numéricas

Para calcular la cantidad de números o códigos que se pueden representar en un sistema numérico mediante n dígitos, se debe tener en cuenta que el número disponible de símbolos es igual a la base del sistema numérico elegido. La cantidad de números o códigos que se pueden expresar en una base determinada empleando una concatenación de n dígitos, viene dado por:

$$\text{Total de números representables} = (\text{base})^n \tag{2.1}$$

Ejemplo 2.3 — Rango de representación en base n. ¿Cuál es la cantidad de números que se pueden representar en el sistema decimal con 4 cifras? ¿Y en el sistema binario?

Solución:

Como en base decimal el número de dígitos distintos es 10 (del 0 al 9). El total de números que se puede expresar con 4 dígitos es: $10^4 = 10\,000$. En el caso del sistema binario dicha cantidad es: $2^4 = 16$. ■

Hay que tener cuidado, pues si el 0 está incluido entre los números representados, esto limita la magnitud máxima representable. De este modo, para obtener la magnitud máxima representable en un sistema de numeración que contiene el 0, es preciso restar 1 a la cantidad total de números representables. Por esta razón las magnitudes mayores representables con 4 dígitos en el sistema decimal y el sistema binario son respectivamente: $9999_{(10} = 10\,000 - 1$ y $1111_{(2} = 15_{(10} = 16 - 1$. De forma general, el *rango de representación* en un sistema de numeración con una base dada y n dígitos es: $[0, (\text{base})^{n-1}]$.

Como se ha comentado anteriormente, la codificación de cualquier tipo de información en los sistemas digitales se realiza considerando dos posibles estados (0 y 1). Este tipo de codificación requiere de un gran número de elementos individuales (bits) para representar números cada vez más grandes. Por ejemplo, el número $123\,456\,789_{(10} = 111\,0101\,1011\,1100\,1101\,0001\,0101_{(2}$. Para aumentar el rango de representación con un número de dígitos dado, en los sistemas digitales se recurre al empleo de sistemas de numeración de bases mayores a 2. Los sistemas de numeración más habituales en los sistemas digitales emplean bases que son potencias de 2. De este modo, los sistemas de codificación más usuales son: *octal* (base 8), *hexadecimal* (base 16), etcétera. Los sistemas de numeración con base potencia de 2 son convenientes, ya que el paso de uno a otro es sencillo. Estos sistemas de representación permiten reducir el tamaño de la representación de información en los sistemas digitales, de modo que siga siendo sencillo obtener el valor binario asociado.

Los sistemas de numeración con base 10 o inferior emplean los dígitos desde el '0' a (base−1) (véase tabla 2.3). En los sistemas de base mayor que 10, para los símbolos que corresponden a cifras mayores que 9 se emplean letras. Por este motivo en hexadecimal (base 16) se emplean: A (≡ 10), B (≡ 11), C (≡ 12), D (≡ 13), E (≡ 14) y F (≡ 15).

Tabla 2.3: Dígitos válidos en sistemas de numeración con distinta base.

Base	Rango de dígitos
2	{0,1}
4	{0,1,2,3}
8	{0,1,2,3,4,5,6,7}
10	{0,1,2,3,4,5,6,7,8,9}
16	{0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F}

2.4 Conversión entre sistemas de base diferente

A continuación se describen los métodos empleados para obtener la representación de una magnitud en distintas bases de numeración.

2.4.1 Conversión a base decimal

Esta conversión se realiza teniendo en cuenta la expansión de valores posicionales del número que se desea convertir desde cualquier base de partida. Por tanto, se calcula la suma del producto del valor de cada dígito por la potencia de la base elevada a la posición ocupada por dicho dígito. Recuerda que la posición ocupada por las unidades es la 0 y a partir de ella las posiciones son positivas (1, 2, 3,...) hacia la izquierda en la parte entera y negativas (−1, −2, −3,...) hacia la derecha, en la parte fraccionaria.

Ejemplo 2.4 — Conversión desde base n a decimal. Expresa los números $10\,1100.11_{(2}$ y $BC92_{(16}$ en base decimal.

Solución:

Realizando la expansión de cada número teniendo en cuenta su base, se tiene:

$$\begin{aligned} 10\,1100.11_{(2} &= 1\times 2^5+1\times 2^3+1\times 2^2+1\times 2^{-1}+1\times 1\times 2^{-2} \\ &= 32+2+4+16+4+2+1+0.5+0.5+0.25 = 44.75_{(10} \\ BC92_{(16} &= 11\times 16^3+12\times 16^2+9\times 16^1+2\times 16^0 \\ &= 45\,056+3\,072+144+2 = 48\,274_{(10} \end{aligned}$$

■

2.4.2 Conversión desde decimal a cualquier otra base numérica

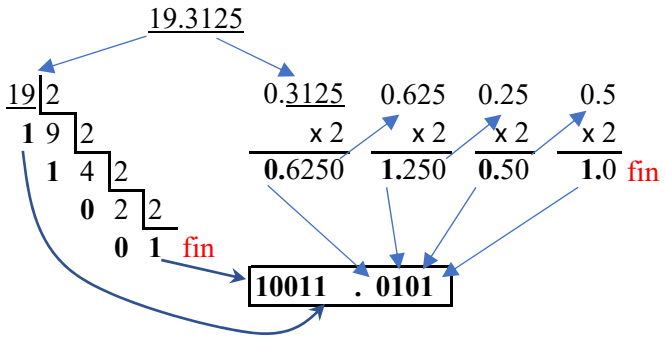
El método de conversión es distinto para la parte entera y la parte fraccionaria:

- Conversión de la parte entera mediante divisiones sucesivas. Se comienza dividendo el número por la base y se continúa dividiendo los cocientes sucesivamente por la base. Los restos obtenidos en cada división son los dígitos en orden cada vez más significativo. Por tanto, el último cociente no divisible por la base proporciona el dígito más significativo de la conversión. La parte entera del número en la nueva base se obtiene tomando el último cociente como cifra más significativa y de ahí se van tomando los restos en orden inverso a su obtención.
- Conversión de la parte fraccionaria mediante multiplicaciones sucesivas. La parte fraccionaria del número a convertir se multiplica sucesivamente por la base y la parte entera obtenida en cada multiplicación va ofreciendo el dígito del resultado desde la posición más significativa hacia la menos significativa. El proceso se detiene cuando la parte fraccionaria resultante se hace 0 o se obtiene el número de dígitos deseado.

Ejemplo 2.5 — Conversión de números decimales a binario. Calcula la representación en binario del número $19.3125_{(10}$.

Solución:

Para la conversión de la parte entera se realizan divisiones sucesivas entre 2. La parte fraccionaria se convierte por multiplicación sucesiva por 2. La figura siguiente ilustra el proceso en ambos casos.



Para conversiones a otras bases, las operaciones de multiplicación y división sucesiva se hacen por la base destino de la conversión. ■

2.4.3 Conversiones entre bases numéricas potencias de 2

Un caso muy habitual en los sistemas digitales es el empleo de bases numéricas que son potencias de 2. Los más comunes son: base 2 (binario), base 8 (octal) y base 16 (hexadecimal). En estos casos la conversión entre ellas es muy sencilla si se tiene en cuenta que un número expresado en una base superior a 2 se puede convertir a binario mediante una agrupación de bits de tamaño n, siendo n el exponente que satisface: $2^n = \text{base}$.

Para hacer los cambios entre las bases numéricas mencionadas anteriormente los bits se agrupan en conjuntos de n bits de derecha a izquierda en la parte entera y de izquierda a derecha en la parte fraccionaria. Si es preciso, la agrupación se completa con 0 a la izquierda en la parte entera y 0 a la derecha en la parte fraccionaria, ya que estos no alteran el valor numérico. Cada agrupación se convierte al valor numérico en la base de destino. Por ejemplo, para pasar a octal se hacen agrupaciones de 3 bits (puesto que $2^3 = 8$) y cada agrupación se convierte en su valor octal. En el caso de hexadecimal se consideran agrupaciones de 4 bits (puesto que $2^4 = 16$).

Ejemplo 2.6 — Conversiones entre binario y base 2^n. Convierte el número binario $10\,1100.11_{(2}$ en octal y hexadecimal.

Solución:

Para realizar la conversión se realizan agrupaciones de 3 bits (octal) y 4 bits (hexadecimal). Los bits añadidos para completar las agrupaciones y que no añaden valor a la magnitud resultante se destacan en color.

$$10\,1100.11_{(2} = \overbrace{101}^{5}\ \overbrace{100}^{4}\ .\ \overbrace{110}^{6} \rightleftarrows 54.6_{(8}$$

$$10\,1100.11_{(2} = \overbrace{0010}^{2}\ \overbrace{1100}^{C}\ .\ \overbrace{1100}^{C} \rightleftarrows 2C.C_{(16}$$

■

Este tipo de conversión se aplica también en sentido inverso. De este modo se deduce que el cambio entre bases que son potencias de 2 es muy sencillo si se emplea el sistema binario como paso intermedio.

> Ten en cuenta que un método derivado para pasar rápidamente un número decimal a binario consiste en el empleo de divisiones/multiplicaciones sucesivas por 8 o 4 para pasar a las bases correspondientes y de estas a binario por el procedimiento ya descrito. De este modo se precisa menor número de divisiones/multiplicaciones.

2.5 Otros códigos binarios

Los sistemas de numeración mencionados en los apartados anteriores —*binario natural*, *octal* y *hexadecimal*— se denominan *códigos de palabra*, ya que codifican su valor como un todo. Por el contrario los *códigos BCD* (*Binary Coded Digit*) codifican de modo independiente cada dígito de la magnitud.

Los bits de un código binario se suelen agrupar para facilitar su lectura. En el tema anterior (ver sección 1.3) se introdujeron las agrupaciones siguientes: *nibble*, *byte*, *word* y *double word*. Para facilitar la lectura de dichas agrupaciones, es habitual escribirlas en grupos de 4 dígitos separados por un espacio en blanco. Para simplificar la escritura de cada uno de dichos grupos, se emplea usualmente su equivalente hexadecimal.

Ejemplo 2.7 — Conversión de binario natural a hexadecimal. Representar en hexadecimal los códigos binarios que se indica a continuación: 0011, 00111010 y 0011101010111110.

Solución:

Para obtener la conversión en hexadecimal se realizan agrupaciones de cuatro bits obteniendo la conversión hexadecimal de cada agrupación individualmente. De este modo se obtiene:

$$\overbrace{0011}^{3} = 3_{(16}$$

$$\overbrace{0011}^{3}\ \overbrace{1010}^{A} = 3A_{(16}$$

$$\overbrace{0011}^{3}\ \overbrace{1010}^{A}\ \overbrace{1011}^{B}\ \overbrace{1110}^{E} = 3ABE_{(16}$$

■

2.5.1 Características de los códigos binarios

Cada combinación de bits de un código se denomina *palabra* y su longitud indica el número de bits del código. El *peso* de una palabra del código es el número de unos que posee la palabra. Por ejemplo, la palabra 1011 0111 tiene longitud 8 y peso 6.

> La *palabra nula* y la *palabra unidad* son aquellas que están constituidas respectivamente solo por ceros y unos. Son palabras que a ser posible se deben evitar en un código generado por medios mecánicos o eléctricos, ya que pueden ser fuente de ambigüedad al no poder distinguir si su origen es la consecuencia de un error.

Si todas las palabras de un código tienen la misma longitud se dice que es un *código uniforme* y si el número de códigos es 2^n siendo n la longitud de la palabra entonces se trata de un *código completo* o sin redundancia.

Se habla de un *código ponderado* cuando cada posición del código tiene un peso asociado. Dicha ponderación debe cumplir una serie de reglas:

- El peso de la posición i-ésima no puede superar en más de 1 a la suma de los pesos de las posiciones previas. Por ejemplo, un código de pesos 9 4 2 1 tendrá huecos, ya que 9 supera en 2 a la suma de pesos previos, puesto que: $4+2+1=7$.
- Si el código tiene cuatro bits la suma de todos los pesos debe ser al menos 9. Por ejemplo, 2 4 2 1.
- Un peso solo puede aparecer una vez en la distribución de pesos para evitar ambigüedad. Por ejemplo, el código con pesos 2 4 2 1 presenta ambigüedad en la representación del número 6, ya que este se puede representar por los códigos 1100 y 0110. Para resolver dichas situaciones es preciso asignar de modo unívoco un valor a cada código.

2.5.2 Código decimal binario o BCD

Este sistema tiene como propósito facilitar la representación en binario de los números decimales. No es un código de palabra, sino que emplea un número de 4 bits para codificar cada una de las cifras de

un número decimal. También se denomina *BCD puro* o código 8 4 2 1 por la ponderación que se otorga a cada uno de los bits del código. La tabla 2.4 muestra la equivalencia entre los dígitos decimales y su equivalente BCD.

Tabla 2.4: Codificación en BCD de dígitos decimales.

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

Ejemplo 2.8 — Representación de números decimales en BCD. Representa en BCD los números expresados en sistema decimal 35, 98, 170 y 2 469.

Solución:

Representando cada cifra decimal por su agrupación de cuatro bit correspondiente se obtiene:

$$35 = \overbrace{0011}^{3}\ \overbrace{0101}^{5}$$

$$98 = \overbrace{1001}^{9}\ \overbrace{1000}^{8}$$

$$170 = \overbrace{0001}^{1}\ \overbrace{0111}^{7}\ \overbrace{0000}^{0}$$

$$2\,469 = \overbrace{0010}^{2}\ \overbrace{0100}^{4}\ \overbrace{0110}^{6}\ \overbrace{1001}^{9}$$

La codificación mediante BCD es sencilla, pero solo emplea 10 de los 16 códigos posibles que se pueden construir con 4 bits, ya que en decimal solo existen los 10 dígitos: $\{0,1,2,3,4,5,6,7,8,9\}$. Los códigos no utilizados son: $1010 \equiv 10_{(10} = A_{(16}$, $1011 \equiv 11_{(10} = B_{(16}$, $1100 \equiv 12_{(10} = C_{(16}$, $1101 \equiv 13_{(10} = D_{(16}$, $1110 \equiv 14_{(10} = E_{(16}$ y $1111 \equiv 15_{(10} = F_{(16}$. Por tanto, se dice que BCD es un sistema de codificación incompleto.

2.5.3 Otros códigos numéricos

Entre los más utilizados están:

- Código Aiken o 2 4 2 1. Es un código BCD ponderado que para evitar ambigüedad emplea una codificación complementaria: 0000 (0), 0001 (1), 0010 (2), 0011 (3), 0100 (4), 1011 (5), 1100 (6), 1101 (7), 1110 (8), 1111 (9).
- BCD-XS3 (exceso a 3) de Stibitz. Es un código no ponderado que se obtiene sumando 3 al código BCD puro.

2.5.4 Códigos cíclicos no numéricos

Existen múltiples codificaciones binarias que no tienen como propósito codificar valores numéricos (p. ej., posiciones) pero son cíclicos en el sentido de que su procedimiento de generación lleva desde el último código de nuevo al primero de ellos. A continuación se describen tres de ellos.

Código Gray

Junto con el código binario natural o ponderado, es el código de palabra más empleado en los sistemas digitales. Es un código binario con la particularidad de que entre un código y el siguiente solo cambia un bit. Por esta razón también se denomina *código de paso simple*. En este caso, también se habla de códigos adyacentes. Esta codificación se emplea por motivos prácticos en la simplificación de sistemas digitales. También se utiliza cuando en un sistema digital automatizado se trata de codificar valores consecutivos de una magnitud física. En estos casos, cuando cambian simultáneamente varios bits entre dos instantes

consecutivos, es posible inferir la ocurrencia de un error o lectura falsa en los sistemas encargados de generar dichos códigos.

El *código Gray* se relaciona directamente con el binario natural mediante las expresiones, para códigos de 3 bits, que se indican en la tabla 2.5, teniendo en cuenta que el símbolo $\oplus$ corresponde a la operación lógica OR-exclusivo (operación cuyo resultado es 0 solo si todos los operandos tienen el mismo valor o el número de los que valen 1, es par).

Tabla 2.5: Ecuaciones de conversión entre códigos Gray $\rightleftarrows$ Binario de tres bits.

Binario $\rightarrow$ Gray	Gray $\rightarrow$ Binario
$G_2 = B_2$	$B_2 = G_2$
$G_1 = B_2 \oplus B_1$	$B_1 = B_2 \oplus G_1$
$G_0 = B_1 \oplus B_0$	$B_0 = B_1 \oplus G_0$

Ejemplo 2.9 — Paso de binario a código Gray y viceversa. Empleando las ecuaciones lógicas de la tabla 2.5 se obtienen las conversiones entre los códigos Gray y binario que se indican a continuación:

Binario	000	001	010	011	100	101	110	111
Gray	000	001	011	010	110	111	101	100

■

Codificación *one-hot* y *one-cold*

Es una codificación que emplea un número total de códigos igual al número de bits empleado. Si los códigos solo tienen un 1 se denomina *one-hot bit encoding* y si solo tiene un 0 se trata de *one-cold bit encoding*. Ambas codificaciones son complementarias y se emplean en el control de sistemas cuando se debe activar o desactivar de modo cíclico un subsistema concreto. Para obtener cualquiera de ellas se emplea un *contador en anillo* (ver Sec. 8.2.4, pág. 222) que produce una salida decodificada. Esto es, la posición del bit singular señala el valor decodificado del código generado según se muestra en el ejemplo siguiente.

Ejemplo 2.10 — Codificación binaria no natural. A continuación se muestran los códigos correspondientes a codificación *one-hot/cold* con 4 bits. La posición del 1 (*one-hot*) o el 0 (*one-cold*) indica el orden del código en la secuencia cíclica que lo genera.

$$\text{One-hot encoding:} 1 \rightarrow 000\mathbf{1},\ 2 \rightarrow 00\mathbf{1}0,\ 3 \rightarrow 0\mathbf{1}00,\ 4 \rightarrow \mathbf{1}000$$
$$\text{One-cold encoding:} 1 \rightarrow 111\mathbf{0},\ 2 \rightarrow 11\mathbf{0}1,\ 3 \rightarrow 1\mathbf{0}11,\ 4 \rightarrow \mathbf{0}111$$

■

Código Johnson

Es una codificación cíclica que se obtiene mediante el empleo de un *contador Johnson* (ver Sec. 8.2.4, pág. 223). Para 4 bits está compuesto por la secuencia:

$$[0000, 0001, 0011, 0111, 1111, 1110, 1100, 1000]$$

2.5.5 Codificación ASCII

El código estándar estadounidense para el intercambio de información o ASCII (*American Standard Code for Information Interchange*) se creó en el año 1963 para facilitar el intercambio de información entre computadores. Por tanto, además de representar información numérica, permite codificar cualquier otro tipo de información manejada por un computador, como símbolos e instrucciones de control de la comunicación.

La codificación original constaba de 128 símbolos a los que se asigna un código de 1 byte (8 bits) de los que se empleaban únicamente 7. Su ampliación a los 8 bits disponibles permitió su extensión hasta

256 códigos. En esta codificación los dígitos del 0 al 9 quedan representados por los códigos 48 a 57. Para representar los códigos se emplea codificación hexadecimal, puesto que el manejo del código binario asociado a cada símbolo es engorroso, ya que requiere 8 bits. De este modo, solo se emplean dos dígitos hexadecimales (2 grupos de 4 bits) para representar cualquiera de los 128 códigos de la tabla ASCII. En la tabla de la figura 1.4 se muestra la tabla de caracteres ASCII con el código correspondiente expresado en base decimal y hexadecimal. Los caracteres de 0 al 31 más el 127 no tienen representación gráfica, ya que corresponden a caracteres de control.

Con el tiempo esta codificación se ha mostrado insuficiente por lo que se han creado extensiones compatibles para suplir sus limitaciones. En la actualidad los métodos universales de codificación de información se basan en el código Unicode.

2.6 Aritmética en el sistema binario

Las operaciones aritméticas básicas (suma, resta, multiplicación y división) son muy simples de realizar en binario. En las secciones siguientes se profundiza en esta cuestión y en la representación de números enteros, con signo.

2.6.1 Suma binaria

La tabla 2.6 muestra el resultado de la suma binaria de dos bits (a_i y b_i) cuando no hay bit de *acarreo* de entrada o *llevada* (*carry*). La llevada o acarreo en el resultado se produce cuando los dos bits que se suman son 1. En este caso el resultado de la suma (s_o) es 0 y el acarreo (c_o) es 1. El acarreo se traslada al resultado como una unidad en la posición más significativa del resultado cuando este excede el valor representable en la posición actual.

Tabla 2.6: Tabla de sumar en binario sin acarreo de entrada.

Entrada				Salida	
a_i	+	b_i	=	c_o	s_o
0	+	0	=	0	0
0	+	1	=	0	1
1	+	0	=	0	1
1	+	1	=	1	0

Ejemplo 2.11 — Suma en binario natural. A continuación se muestran varios ejemplos de sumas binarias y sus equivalentes en decimal. En azul se indica el valor del acarreo.

(a)
$$\begin{array}{rr} & 1\ 1 \\ +\ & 1\ 1 \\ \hline 1 & 1\ 0 \end{array} \qquad \begin{array}{r} 3 \\ +\ 3 \\ \hline 6 \end{array}$$

(b)
$$\begin{array}{rrr} & 1 & 0\ 0 \\ +\ & 1 & 0 \\ \hline & 1 & 1\ 0 \end{array} \qquad \begin{array}{r} 4 \\ +\ 2 \\ \hline 6 \end{array}$$

(c)
$$\begin{array}{rrrr} & 1 & 1 & 1 \\ +\ & & 1 & 1 \\ \hline 1 & 0 & 1 & 0 \end{array} \qquad \begin{array}{r} 7 \\ +\ 3 \\ \hline 10 \end{array}$$

La tabla de la suma binaria se puede completar si se considera un acarreo de entrada. Así se obtiene la tabla 2.7 completa.

Tabla 2.7: Tabla de sumar en binario con acarreo de entrada.

Entrada						Salida	
c_i	+	a_i	+	b_i	=	c_o	s_o
1	+	0	+	0	=	0	1
1	+	0	+	1	=	1	0
1	+	1	+	0	=	1	0
1	+	1	+	1	=	1	1

2.6.2 Resta binaria

Las reglas de la resta en binario son similares a las del sistema decimal. Se debe tener en cuenta que en la resta binaria el resultado de $0-1$ es 1 con llevada o acarreo negativo de 1 que se suma al sustraendo en la posición más significativa.

Ejemplo 2.12 — Resta en binario. A continuación se muestran varios ejemplos de restas binarias y sus equivalentes en decimal. En el caso (c) se muestra una situación de llevada negativa.

(a)

$$\begin{array}{r} 1\ 1 \\ -\ 0\ 1 \\ \hline 1\ 0 \end{array} \qquad \begin{array}{r} 3 \\ -\ 1 \\ \hline 2 \end{array}$$

(b)

$$\begin{array}{r} 1\ 1 \\ -\ 1\ 0 \\ \hline 0\ 1 \end{array} \qquad \begin{array}{r} 3 \\ -\ 2 \\ \hline 1 \end{array}$$

(c)

$$\begin{array}{r} 1\ 0\ 1 \\ -\ \boxed{0}\ 1\ 1 \\ \hline 0\ 1\ 0 \end{array} \qquad \begin{array}{r} 5 \\ -\ 3 \\ \hline 2 \end{array}$$

■

2.6.3 Multiplicación y división en binario

La multiplicación en el sistema binario es trivial, ya que cuando alguno de los factores es 0 el resultado es 0. El resultado es 1 solo si ambos factores son 1. Como se puede ver en el ejemplo siguiente, la operación de multiplicación de valores binarios con más de un bit se convierte en operaciones de suma desplazada de los productos parciales.

Ejemplo 2.13 — Multiplicación en binario. A continuación se muestran varios ejemplos de multiplicaciones binarias y sus equivalentes en decimal.

(a)

$$\begin{array}{r} 1\ 1 \\ \times\ 1\ 1 \\ \hline 1\ 1 \\ +\ 1\ 1 \\ \hline 1\ 0\ 0\ 1 \end{array} \qquad \begin{array}{r} 3 \\ \times\ 3 \\ \hline 9 \end{array}$$

(b)

$$\begin{array}{r} 1\ 1\ 1 \\ \times\ 1\ 0\ 1 \\ \hline 1\ 1\ 1 \\ 0\ 0\ 0 \\ +\ 1\ 1\ 1 \\ \hline 1\ 0\ 0\ 0\ 1\ 1 \end{array} \qquad \begin{array}{r} 7 \\ \times\ 5 \\ \hline 35 \end{array}$$

■

La división se calcula mediante restas sucesivas del divisor al dividendo hasta que el resto es 0 o inferior al divisor. El número de veces que el divisor está contenido en el dividendo es el cociente de la división.

Una conclusión importante respecto a la aritmética en el sistema binario es que las cuatro operaciones básicas se pueden reducir a la operación de suma, ya que la resta se puede obtener sumando al minuendo el sustraendo con signo negativo. En las secciones siguientes se explican los métodos de representación de números enteros negativos en binario.

2.6.4 Representación de números enteros en signo magnitud

En la representación de binario natural no se tiene en cuenta el signo de los números. Es decir, en binario natural se representan los números naturales. Como se ha explicado previamente, para expresar un número en binario natural se codifica su magnitud mediante un sistema posicional ponderado que emplea base 2. Cuando se desea codificar números negativos es preciso codificar la información de signo y añadirla a la representación de la magnitud del número. En este caso se estarían codificando los números enteros (números naturales con signo).

El signo de un número entero puede ser positivo o negativo. Por tanto, estas dos alternativas se pueden codificar con un bit añadido en la posición más significativa (MSB). Con este sistema de codificación de signo, es habitual considerar el '0' para el signo positivo y el '1' para el signo negativo.[3] Esta codificación se denomina *signo magnitud*, en adelante SM para abreviar.

[3] Se ha marcado en color rojo el bit de signo a efectos de favorecer la comprensión.

Ejemplo 2.14 — Representación de números enteros en SM. ¿Cuál es la codificación SM del número entero (-23) empleando 1 byte?

Solución:

De los 8 bits empleados, el MSB (bit más significativo) se utiliza como bit de signo y por tanto se pone a 1. En el resto de los 7 bits se codifica la magnitud, que es 23. Teniendo en cuenta que:

$$23 = \cancel{0\times 2^6} + \cancel{0\times 2^5} + 1\times 2^4 + \cancel{0\times 2^3} + 1\times 2^2 + 1\times 2^1 + 1\times 2^0 = 16+4+2+1$$

Gráficamente:

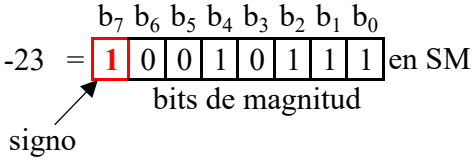


Tabla 2.8: Comparación de codificación con 3 bits en SM y binario natural (en gris se marca la representación del 0).

Código	SM	Binario natural
000	0	0
001	1	1
010	2	2
011	3	3
100	−0	4
101	−1	5
110	−2	6
111	−3	7

Dado un número binario con codificación SM de n bits, solo $n-1$ de ellos se emplean para codificar la magnitud y el restante para el signo. Por tanto, el rango de representación en SM es:

$$[-(2^{n-1}-1),\ +(2^{n-1}-1)]$$

Como se observa en la expresión del rango en SM, tanto en la parte negativa como en la positiva se resta 1 porque la codificación comienza en el 0. Aunque el 0 carece de signo, en SM obtiene una codificación doble: una en la parte positiva y otra en la negativa (ver tabla 2.8). Es importante notar que el número total de códigos posibles distintos solo difiere en 1 respecto a la codificación en binario natural con n bits. Dicha diferencia se debe a la doble codificación del 0 que hace perder un código respecto a la codificación en binario natural.

Para obtener el valor decimal de un código binario dado en SM, se pasan a decimal los bits de magnitud —sin tener en cuenta el bit de signo— y se añade el signo indicado por el bit de signo.

Aritmética en SM

En SM se opera empleando la suma y la resta binaria sobre la magnitud de los operandos dependiendo del signo de estos. El bit de signo se manipula aparte para decidir el signo del resultado. Es decir, se hacen dos operaciones una para calcular el valor de la magnitud del resultado y otra para calcular su signo.

✓ Caso 1 - Los operandos son del mismo signo. El resultado se calcula sumando la magnitud de los operandos. El signo del resultado coincide con el de los operandos.

✓ Caso 2 - Los operandos son de signo distinto. El resultado se calcula restando la magnitud mayor a la menor de los operandos. El signo del resultado coincide con el signo del operando de mayor magnitud.

> La resta se trata como una suma con inversión de signo para el sustraendo. En este texto se emplea el color rojo para destacar el bit de signo en las representaciones binarias.

Ejemplo 2.15 — Aritmética en SM. Cuando se suman dos números de distinto signo, la magnitud del resultado es la resta de las magnitudes y su signo el del sumando de mayor magnitud.

$$(-3)+(+4) = +(4-3) = 1 \Rightarrow \overbrace{0}^{+}100 - \overbrace{1}^{-}011 = \overbrace{0}^{+}001 \equiv (+1)$$

Cuando se suman dos números negativos (con igual signo). Las magnitudes se suman para calcular la magnitud del resultado. El signo del resultado coincide con el de los operandos.

$$(-2)+(-3) = -(2+3) = -5 \Rightarrow \overbrace{1}^{-}010 + \overbrace{1}^{-}011 = \overbrace{1}^{-}101 \equiv (-5)$$

■

Cuando el resultado de una operación excede el rango de valores representables con el número de bits disponible, se produce un error denominado *desbordamiento* (*overflow*). La determinación de una condición de error por desbordamiento requiere un tratamiento especial en sistemas con codificación SM.

Ejemplo 2.16 — Operaciones aritméticas con desbordamiento. En la operación matemática que se muestra a continuación se emplean 3 bits que son insuficientes para almacenar la magnitud del resultado. En este caso se produce desbordamiento porque el resultado no es representable con los bits disponibles.

$$(+5)+(+4) = +(5+4) = 9 \Rightarrow 0101+0100 = 0001 \equiv (+1) \qquad \text{¡ERROR!}$$

La solución a este tipo de error es añadir un bit adicional a la codificación para ampliar el rango de representación. Si en el caso planteado se emplean 4 bits para la operación, el problema se resuelve. En los sistemas digitales es preciso detectar las condiciones de desbordamiento para activar las medidas correctoras que realicen el tratamiento de los posibles errores.

$$(+5)+(+4) = +(5+4) = +9 \Rightarrow 0\,0101+0\,0100 = 0\,1001 \equiv (+9)$$

■

La multiplicación o producto de números se calcula mediante la suma desplazada de los productos parciales de las magnitudes de los operandos. El signo de la operación es positivo si los operandos tienen el mismo signo. Si los operandos poseen distinto signo, el signo del resultado es negativo. En la multiplicación no es preciso comprobar condición de desbordamiento, ya que lo normal es disponer de un número de bits suficiente para representar el resultado que es la suma de bits de magnitud de ambos operandos.

2.6.5 Representación de números enteros en complemento a 1

La representación en SM es simple, pero tiene como inconveniente principal que su aritmética depende del signo de los operandos. Los cálculos con números de distinto signo requieren la operación de resta binaria de la magnitud menor a la mayor. La representación en complemento persigue el uso de la suma independientemente del signo de los operandos.

En *complemento a 1* o C1 un número positivo se representa igual que en SM, el bit de signo es 0 y la magnitud se representa en binario natural. Sin embargo, los números negativos se codifican mediante el

operador de complemento aplicado a todos los bits del número positivo correspondiente. El operador de complemento transforma un 1 en un 0 y viceversa: $(1)' = 0$ y $(0)' = 1$. La tabla 2.9 muestra el equivalente decimal de la codificación de números enteros en C1 con 3 bits. Al igual que en SM, el 0 tiene doble representación en C1.

Tabla 2.9: Comparación de equivalente decimal al emplear codificación con 3 bits en C1, SM y binario natural (en gris se marca la representación del 0).

Código	C1	SM	Binario
000	0	0	0
001	1	1	1
010	2	2	2
011	3	3	3
100	−3	−0	4
101	−2	−1	5
110	−1	−2	6
111	−0	−3	7

> Recuerda que el operador de complemento admite varias representaciones. Las más habituales son la barra superior y el símbolo prima, pero en esta obra se utilizará preferentemente el símbolo prima para facilitar su escritura y evitar ambigüedades en las expresiones complejas. Por tanto:
>
> $$\text{complemento}(A) \equiv A' \equiv \overline{A}$$

Ejemplo 2.17 — Representación de números enteros en C1. ¿Cómo se representan con 4 bits los números enteros $+3$ y -3 en C1?

Solución:

$$+3 = 0011$$

$$-3 = \overbrace{(0011)}^{+3}{}' = 1100_{\text{C1}}$$

La aplicación repetida del operador complemento (cambio de signo) dos veces deja inalterado el número binario sobre el que se aplica. El rango de representación con codificación C1 es igual que en SM. Esto es: $[-(2^{n-1}-1),\ +(2^{n-1}-1)]$

Para pasar un número positivo de C1 a decimal se opera como si estuviera expresado en binario natural. Si es negativo se hace previamente el complemento del número sin olvidar que el número resultante es negativo.

Ejemplo 2.18 — Conversión a decimal de números en C1. ¿Cuál es el equivalente decimal de los números enteros representados en C1 como 0010_{C1} y 1010_{C1}?

Solución:

El número 0010_{C1} es positivo y por tanto, su equivalente decimal se obtiene por la expansión numérica que permite obtener el valor de un número expresado en binario natural.

$$0010_{\text{C1}} = 0010_{(2} = 1 \times 2^1 = +2$$

En cambio para pasar el número negativo 1010_{C1} a decimal, se complementa solo la magnitud del número dejando inalterado el bit que corresponde al signo y calculando el equivalente digital de la magnitud.

$$1010_{\text{C1}} = 1(010)' = 1101 = (-)(1 \times 2^2 + 1 \times 2^0) = (-)(4+1) = -5$$

Este proceso también se puede llevar a cabo complementando primero todo el número y convirtiendo el resultado a decimal como binario natural manteniendo el signo negativo original.

$$1010_{C1} = (-)(1010)' = (-)0101 = (-)(1\times 2^2 + 1\times 2^0) = (-)(4+1) = -5$$

Un modo alternativo, en un solo paso —algo más complejo— para pasar los números negativos de C1 a decimal, consiste en hacer la expansión numérica dando valor negativo a la posición más significativa (bit de signo) y sumando 1 al resultado final.

$$1010_{C1} = -\mathbf{1}\times 2^3 + \cancel{0\times 2^2} + 1\times 2^1 + \cancel{0\times 2^0} + \mathbf{1} = -8+2+1 = -5$$

La dificultad de este segundo método reside en recordar que la expansión resta el término más significativo y suma 1 al resultado final. ■

Aritmética en C1

Cuando se trabaja con representación en C1 solo es preciso tener en cuenta la suma, ya que la resta se puede considerar como la suma del sustraendo con signo negativo. El bit de signo se suma del mismo modo que el resto de bits. La operación de suma tiene en cuenta el bit de acarreo final que siempre se suma al resultado obtenido. En este caso se habla de recirculación del acarreo.

Ejemplo 2.19 En este ejemplo se realiza la operación de suma de números enteros con resultado correcto. El bit de signo se indica con color rojo y el de acarreo con color azul.

(a)

	0	1	0	0	(+4)
+	1	1	0	0	(-3)
1	0	0	0	0	
+				1	
	0	0	0	1	(+1)

(b)

	0	0	1	0	(+2)
+	1	0	1	0	(-5)
0	1	1	0	0	
+				0	
	1	1	0	0	(-3)

■

Antes de calcular el resultado de la operación aritmética es posible saber el signo correcto del resultado, ya que este debe coincidir con el signo del operando de mayor magnitud. Por tanto, un método de detección de desbordamiento consiste en comprobar la validez del bit de signo cuando los operandos tienen el mismo signo. En caso de desbordamiento se puede disponer un bit adicional que permita representar la magnitud del resultado evitando desbordamiento.

Ejemplo 2.20 — Aritmética en C1. En este ejemplo se llevan a cabo operaciones en las que se produce desbordamiento. Para resolver esta situación es preciso representar los números con un bit adicional para que el resultado final esté incluido en el rango de representación admitido.

(a)

	0	1	0	0	(+4)
+	0	1	0	1	(+5)
0	1	0	0	1	
+				0	
	1	0	0	1	(-6)

¡Desbordamiento!

	0	0	1	0	0 (+4)
+	0	0	1	0	1 (+5)
0	0	1	0	0	1
+					0
	0	1	0	0	1 (+9)

Correcto

(b)

	1	0	1	1	(-4)
+	1	0	1	0	(-5)
1	0	1	0	1	
+				1	
	0	1	1	0	(+6)

¡Desbordamiento!

	1	1	0	1	1 (-4)
+	1	1	0	1	0 (-5)
1	1	0	1	0	1
+					1
	1	0	1	1	0 (-9)

Correcto

■

2.6.6 Representación de números enteros en complemento a 2

La representación de números enteros en *complemento a 2* (C2) o *complemento a la base*, es aquella representación que sumada al número positivo correspondiente produce como resultado 0. Los números positivos tienen idéntica representación que en binario natural, SM y C1. Por el contrario, la representación

en C2 de números negativos se obtiene sumado 1 a su representación en C1. De este modo, se tiene que:

$$A_{C2} = A_{C1} + 1$$

Como sucede con C1, al repetir la operación de cambio de signo mediante C2 se obtiene el número de partida.

Ejemplo 2.21 — Representación de números enteros en C1 y C2. ¿Cuál es la representación con 4 bits de los números enteros $+3$ y -3 en C1 y C2?

Solución:

Como la representación de números positivos es idéntica en binario natural, C1 y C2 se tiene:

$$+3 = 0011_{(2} = 0011_{C1} = 0011_{C2}$$

En el caso de los números negativos hay que tener en cuenta la operación de complemento:

$$-3 = (\overbrace{0011}^{+3})' = 1100_{C1} = 1100_{C1} + 1 = 1101_{C2}$$

Un modo alternativo simple de obtener la representación en C2 de un número negativo partiendo del número positivo de igual magnitud, consiste en copiar todos los bits de derecha a izquierda incluido el primer 1 y a partir de este, copiar el complemento del resto de bits. Aplicando este método se tiene que:

$$-3 = (001)'1 = 1101_{C2}$$

Donde se ha copiado el 1 más a la dcha. y el resto de bits se complementan. ■

El rango de representación en C2 es: $[-2^{n-1},\ +(2^{n-1}-1)]$ Respecto a la representación en C1, el rango se amplía en 1 para los números negativos, ya que en C2 el 0 no tiene doble representación. La tabla 2.10 muestra el equivalente digital de las distintas representaciones de números enteros. En dicha tabla se comprueba que el número total de códigos es el mismo, pero cambia la interpretación y el rango de valores representados.

Tabla 2.10: Comparación del equivalente decimal para representaciones con 3 bits en C2, C1, SM y binario natural (en gris se marca la representación del 0).

Código	C2	C1	SM	Binario
000	0	0	0	0
001	1	1	1	1
010	2	2	2	2
011	3	3	3	3
100	−4	−3	−0	4
101	−3	−2	−1	5
110	−2	−1	−2	6
111	−1	−0	−3	7

Para convertir la representación de un entero negativo en C2 a su equivalente decimal se repite la operación de C2 manteniendo el signo negativo. También se puede recurrir a la expansión numérica dando valor negativo a la posición más significativa correspondiente al bit de signo.

Ejemplo 2.22 — Conversión a decimal de números en C2. ¿Cuáles son los números enteros representados por: 1101_{C2} y 1010_{C2}?

Solución:

Según se señala la representación empleada es C2. Por tanto, el bit MSB corresponde al signo y como en ambos casos dicho bit es 1, se trata de números negativos. Para encontrar su valor en decimal se emplearán varios métodos alternativos.

1. Se repite la operación de cambio de signo en C2 consistente en complementar y sumar 1.

$$1101_{C2} \equiv (-)(1101)' + 1 = (-)0010 + 1 = (-)0011 = -3$$

$$1010_{C2} \equiv (-)(1010)' + 1 = (-)0101 + 1 = (-)0110 = -6$$

2. Se repite la operación de cambio de signo en C2 consistente en la copia de bits desde la derecha hacia la izquierda hasta el primer 1 inclusive y el complemento del resto de bits.

$$1101_{C2} \equiv (-)(110)'1 = (-)0011 = (-)0011 = -3$$

$$1010_{C2} \equiv (-)(10)'10 = (-)0110 = (-)0110 = -6$$

3. Se realiza la expansión numérica directa considerando como negativo el valor asociado al bit más significativo.

$$1101_{C2} = -1 \times 2^3 + 1 \times 2^2 + \cancel{0 \times 2^1} + 1 \times 2^0 = -8 + 4 + 1 = -3$$

$$1010_{C2} = -1 \times 2^3 + \cancel{0 \times 2^2} + 1 \times 2^1 + \cancel{0 \times 2^0} = -8 + 2 = -6$$

La tabla 2.11 resume los rangos de representación empleando distintos tipos de codificación binaria para un número n de bits dado. En dicha tabla se observa como el rango de representación en C2 aumenta en un código con respecto a C1 y SM, ya que estos últimos emplean una representación doble para el 0.

Tabla 2.11: Comparación de rangos de representación con distintos sistemas de codificación binaria.

Bits	Tot. códigos	Rango binario	Rango SM	Rango C1	Rango C2
4	$2^4 = 16$	$[0,\ 15]$	$[-7,\ +7]$	$[-7,\ +7]$	$[-8,\ +7]$
8	$2^8 = 256$	$[0,\ 255]$	$[-127,\ +127]$	$[-127,\ +127]$	$[-128,\ +127]$
16	$2^{16} = 65\,536$	$[0,\ 65\,535]$	$[-32\,767,\ +32\,767]$	$[-32\,767,\ +32\,767]$	$[-32\,768,\ +32\,767]$
n	2^n	$[0,\ 2^n - 1]$	$[-(2^{n-1} - 1),\ +(2^{n-1} - 1)]$	$[-(2^{n-1} - 1),\ +(2^{n-1} - 1)]$	$[-2^{n-1},\ +(2^{n-1} - 1)]$

> En ocasiones se comete un abuso del lenguaje utilizando el nombre de la representación (C1, C2) en lugar de la operación de cambio de signo en dicha representación. De este modo, cuando se pide calcular el C2 de un número positivo, en realidad se solicita la operación de cambio de signo para dicho número. Esto es, el número negativo correspondiente. Así, se dice «calcular el C2 de A», en vez de, «calcular la representación de $(-)A$ en C2»; o de modo equivalente, «obtener el número resultante de cambiar el signo a A en C2». Por tanto, es muy importante distinguir si se desea la representación del número o el resultado de la operación de cambio de signo.

Aritmética en C2

Si se emplea representación en C2, todas las operaciones aritméticas se realizan mediante sumas en las que el bit de signo se trata como un bit convencional. En la aritmética en C2 no es preciso sumar el acarreo final al resultado, como se hace en C1. Por tanto, el acarreo final se descarta. El desbordamiento del resultado de una operación en C2 se determina comprobando la validez del bit de signo de dicho resultado. En la sección 6.4.3 (pág. 163) se realiza un análisis detallado de la condición bajo la que se produce desbordamiento.

Ejemplo 2.23 — Aritmética en C2. En este ejemplo se llevan a cabo operaciones en C2 en las que se produce desbordamiento. Para resolver esta situación es preciso representar los números con un bit adicional para que el resultado final esté incluido en el rango de representación admitido. En las operaciones se indica el descarte del bit de acarreo y la adición de un bit para corregir el error de desbordamiento.

(a)

	0	1	0	0	(+4)
+	1	1	0	1	(-3)
~~1~~	0	0	0	1	(+1)

Correcto

(b)

	0	0	1	0	(+2)
+	1	0	1	1	(-5)
~~0~~	1	1	0	1	(-3)

Correcto

(c)

	1	1	0	0	(-4)
+	1	0	1	1	(-5)
~~1~~	0	1	1	1	(+7)

¡Desbordamiento!

	1	1	1	0	0	(-4)
+	1	1	0	1	1	(-5)
~~1~~	1	0	1	1	1	(-9)

Correcto

2.6.7 Representación de números enteros con signo en Logisim

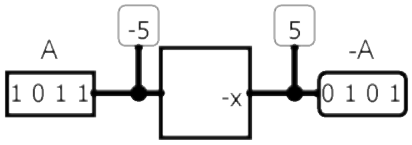
En Logisim los números con signo emplean representación en C2. En consecuencia, cuando se emplea el módulo de cambio de signo (*negator*) de la librería aritmética (*Arithmetic*) se aplica la operación de cambio de signo en C2.

El elemento sonda de prueba (*probe*) de la librería *Wiring* permite visualizar el valor correspondiente en la línea donde se conecta. La configuración de dicho elemento permite visualizar el valor de la señal en diferentes sistemas de representación (en inglés *radix*): binario, octal, decimal con signo (considerando representación binaria en C2), decimal sin signo y hexadecimal.

Ejemplo 2.24 — Visualización de enteros con signo en Logisim. Construye un circuito en Logisim para cambiar el signo de un número de 4 bits y mostrar tanto el valor de entrada como el de salida como un número decimal con signo.

Solución:

El circuito es muy simple y consta de sendos pines de entrada y salida de cuatro bits (A, $-A$) conectados a través del módulo de cambio de signo (ver figura adjunta).



2.6.8 Suma en BCD

BCD no es un código de palabra, ya que no codifica un valor total sino los valores parciales de cada uno de los dígitos de un número decimal. Cada dígito se codifica en binario natural con un bloque de 4 bits. En este caso la adición de números codificados en BCD sigue las siguientes reglas:

a) Cada bloque de 4 bits se suma aplicando las reglas de la suma en binario.

b) Si la suma de un bloque es menor o igual que 9, dicha suma es válida.

c) Si la suma de un bloque es superior a 9, se le debe sumar $0110_{(C2} = 6_{(10}$ para obtener un valor válido. La razón es que en BCD solo se codifica del 0 al 9. Por tanto, existen 6 códigos no utilizados y al sumar 6 se obtiene un código válido.

d) El acarreo generado en la suma de un bloque se debe sumar al bloque siguiente.

Ejemplo 2.25 — Suma en BCD. Suma en BCD con acarreo entre bloques.

	1001		9
	+ 0100		+4
	1101	Número BCD no válido (>9)	13
	+ 0110	Sumar 6	
0001	**0011**	Número BCD válido	
↓	↓		
1	3		

Conceptos clave

- ☑ Representación de números en bases posicionales.
- ☑ Conversión de números fraccionarios de base 10 a base 2.
- ☑ Conversión entre bases 2^n: binario ($n = 1$), octal ($n = 3$) y hexadecimal ($n = 4$).
- ☑ Representación de números negativos en C1 y C2.
- ☑ Aritmética en C1 y C2.
- ☑ Desbordamiento en las operaciones aritméticas.

Problemas propuestos

Problema 2.1 Convierte el número $1011\ 0101.11_{(2}$ a hexadecimal, octal y decimal.

Problema 2.2 Convierte el número $55.81_{(8}$ a decimal.

Problema 2.3 Convierte el número decimal $226.875_{(10}$ a binario, octal y hexadecimal.

Problema 2.4 Convierte el número decimal $58.75_{(10}$ a base 5, octal, hexadecimal y binario.

Problema 2.5 Convierte los números $589_{(10}$ y $475_{(10}$ a binario, hexadecimal y octal.

Problema 2.6 Convierte el número binario $11\ 0001.0010\ 11_{(2}$ a hexadecimal.

Problema 2.7 Demuestra si se cumple la igualdad siguiente: $100\ 1001_{C2} = 100\ 1010_{C1}$

Problema 2.8 ¿Cuál es el rango de representación con 4 dígitos en binario natural, octal, decimal y hexadecimal?

Problema 2.9 ¿Cuál es el rango de representación en binario natural con 6 bits? ¿Y en signo magnitud (SM)? ¿Cuántos códigos diferentes son posibles en ambos casos?

Problema 2.10 En una aplicación informática se debe codificar la diferencia en los husos horarios entre varias ciudades cuyo valor se encuentra en el rango $[-8,\ +8]$. ¿Cuántos bits son necesarios para codificar dicha diferencia?

Problema 2.11 Se desea representar con una precisión de décima de grado (0.1°C) la temperatura de un proceso químico que oscila entre –20°C y +50°C. ¿Cuántos bits se necesitan como mínimo para representar la T^{a} en C2? ¿Qué códigos binarios quedan sin utilizar?

Problema 2.12 Dados los números $A = 0010$ y $B = 1010$ expresados en C1, calcula el resultado de la operación $A - B$.

Problema 2.13 ¿Cómo se expresa el número $+6$ con 5 bits en C1? ¿Y en C2?

Problema 2.14 ¿Cómo se expresa el número -6 con 5 bits en C1? ¿Y en C2?

Problema 2.15 Dado el número $1011\ 0101_{C1}$ ¿cuál es su representación en C2?

Problema 2.16 ¿Cuál es el resultado de cambiar el signo al número $+127_{(8}$ expresando el resultado en C2 con 8 bits?

Problema 2.17 Realiza las siguientes operaciones en binario empleando codificación C1 y C2 con 8 bits. Comprueba el resultado con el obtenido en sistema decimal:

A) $47 + 18$
B) $-24 - 8$
C) $25 - 45$

3. Puertas lógicas y álgebra de Boole

El álgebra de Boole permite la formalización y estudio de las propiedades de los circuitos eléctricos basados en dispositivos de conmutación capaces de realizar operaciones lógicas. Junto a esta herramienta matemática, el avance continuo en el proceso de miniaturización y velocidad de conmutación de los transistores, constituyen la base fundamental del desarrollo tecnológico actual de los sistemas digitales.

Este capítulo aborda el estudio de las puertas lógicas, unos circuitos electrónicos básicos compuestos de unos pocos transistores, capaces de realizar las operaciones lógicas elementales. Dicho estudio permite tanto el análisis del comportamiento de los sistemas digitales como su diseño óptimo.

3.1 Valor lógico asociado a las señales digitales

El elemento fundamental de los sistemas digitales es el transistor (ver sección 1.2, 4). Su funcionamiento es intuitivamente análogo al de un interruptor microscópico gobernado por una señal eléctrica. De este modo, los circuitos con transistores se pueden configurar para realizar una función lógica concreta: AND, OR, NOT, etcétera. El funcionamiento de los transistores no es objeto de estudio en esta obra, en la que solo se abordan los aspectos de comportamiento lógico de los circuitos. Por el contrario, aquí el interés se centra en el análisis del valor lógico de las señales, sin importar las magnitudes eléctricas de los circuitos electrónicos que implementan las funciones lógicas.

El transistor electrónico es el elemento básico empleado en el diseño jerárquico de los sistemas digitales basado en módulos que aumentan progresivamente su complejidad. A partir del transistor, el siguiente nivel de la jerarquía está constituido por las puertas lógicas que realizan las operaciones lógicas básicas. El diseño jerárquico utiliza módulos funcionales sofisticados sin necesidad de especificar la composición del sistema en un nivel inferior. Así, en esta obra, el análisis y diseño de sistemas se realiza a escala de puertas lógicas y los módulos combinacionales básicos construidos con ellas, sin necesidad de llevar el estudio al nivel del transistor, salvo a título ilustrativo.

Definición 3.1 — Señales, variables y funciones lógicas. Los sistemas lógicos digitales son circuitos electrónicos diseñados para producir unas señales de salida a partir de unas señales de entrada. Todas las señales implicadas en el proceso son de naturaleza eléctrica, a cuyo nivel se asocia un valor lógico representado por los símbolos ‘0’ y ‘1’. Por dicho motivo, a cada señal corresponde una variable matemática de tipo lógico (booleana). Como el valor lógico de las variables de salida de un sistema es el resultado de aplicar una función lógica —implementada por un circuito electrónico digital— a sus

variables de entrada, se dice que las variables de salida son funciones —de salida— dependientes de las variables lógicas de entrada. ■

En este texto para nombrar a las variables lógicas se emplean letras mayúsculas tales como: A, B, C, D, etcétera. En cambio para las funciones lógicas se prefiere la utilización de letras minúsculas: f, g, h,... y así sucesivamente. Cuando una variable lógica es de tipo multibit, esto es un valor numérico codificado en binario por varios bits, se prefiere el uso de subíndices. Por ejemplo, para variables lógicas de E/S con 4 bits, la denominación sería: $A = [A_3 A_2 A_1 A_0]$, $f = [f_3 f_2 f_1 f_0]$, etcétera. Por comodidad, en ocasiones las variables multibit se construyen por agrupación de variables binarias (p. ej., $[ABCD]$ para entradas y $[pqrs]$ para salidas).[1]

Definición 3.2 — Tabla de verdad de una función lógica. La *tabla de verdad* de una función lógica es un arreglo en el que se ordena por orden binario cada una de las combinaciones de valores lógicos de las señales de entrada de las que depende la función junto al valor lógico correspondiente que adquiere la señal de salida para dicha combinación. ■

Tabla 3.1: Tabla de verdad genérica de una función lógica dependiente de dos variables.

Id.	A	B	$f(A,B)$
0	0	0	$f(0,0)$
1	0	1	$f(0,1)$
2	1	0	$f(1,0)$
3	1	1	$f(1,1)$

Como se muestra en la tabla 3.1, las combinaciones de valores de las variables de entrada se ordenan en la tabla siguiendo un orden binario natural creciente de arriba hacia abajo. Este orden asigna una ponderación o valor de significancia a las variables de entrada en la tabla de verdad. La variable más significativa será la ubicada más a la izquierda en la tabla (p. ej., A en la tabla 3.1). De este modo las filas de la tabla de verdad se identifican de modo unívoco mediante el código decimal (Id.) correspondiente al código binario constituido por los valores que las variables de entrada tienen asociado en dicha fila —suprimido habitualmente en la tabla de verdad—.

Ejemplo 3.1 — Interpretación de tabla de verdad. ¿Qué valor lógico tienen las variables de entrada A y B en la fila 3 de la tabla de verdad de una función lógica genérica $f(A,B)$?

Solución:

En la tabla de verdad de una función $f(A,B)$ dependiente de las variables A y B, la fila 3 corresponde al valor binario 11. Es decir, a la combinación de valores lógicos de entrada $A = 1$ y $B = 1$. Recuerda no confundir la fila con Id.=3 con la 3ª fila (Id.=2), ya que el Id. de las filas de la tabla de verdad comienza con el Id.=0. ■

[1] Observa que a diferencia de la agrupación de elementos de una secuencia, en las variables multibit los bits representados entre corchetes no están separados por comas (‘,’).

3.2 Simulación de sistemas digitales con Logisim

Logisim[2] es un programa informático para facilitar el aprendizaje de los métodos de análisis y diseño de sistemas digitales. Este programa permite simular, de un modo sencillo e intuitivo, el comportamiento lógico de los sistemas digitales. La simulación lógica es más simple que la simulación electrónica, ya que esta última tiene en cuenta además de los aspectos lógicos, los niveles de diseño físico como tecnología, distribución física, retardos, etcétera. Sin embargo, la sencillez de la simulación lógica y su capacidad para ilustrar gran parte de los conceptos abordados durante esta obra facilitan un enfoque práctico de la materia apoyado en el uso intensivo de este programa informático.

La figura 3.1 muestra el aspecto de la interfaz gráfica de la ventana principal de Logisim donde se distinguen las distintas zonas que presenta al usuario.

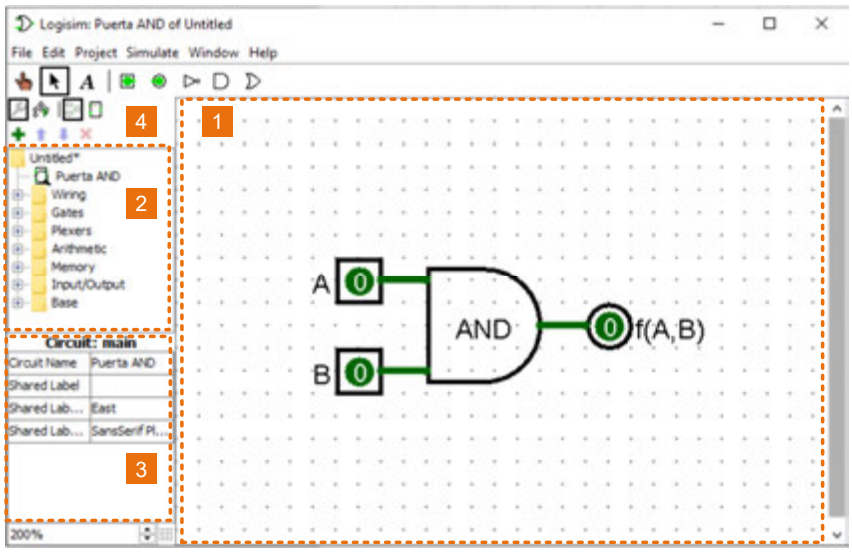


Figura 3.1: Ventana principal de Logisim: 1) área de trabajo, 2) panel de componentes de librería, 3) panel de atributos y propiedades del componente seleccionado, y 4) controles.

> Logisim es un software desarrollado por el Dr. Carl Burch, un ingeniero informático que durante algunos años se dedicó a la formación en materias relacionadas con la organización de computadores y el desarrollo de algoritmos. Durante sus años como profesor desarrolló Logisim, que es un programa escrito en Java con el propósito de simular sistemas lógicos digitales. Este programa *opensource* gratuito, se distribuye bajo licencia GNU GPL v2. Se puede ejecutar en sistemas Windows, MacOS y Linux, con ayuda traducida a idioma español, aunque el manual disponible en Internet para este idioma (mayo, 2008) no cubre todas las características de la última versión disponible (marzo, 2011).

A diferencia de otros programas más complejos y costosos, Logisim es un programa de simulación lógica que no tiene en cuenta parámetros eléctricos. Por tanto, no permite el diseño de sistemas «reales». El valor lógico de las señales es la única característica que se tiene en cuenta en las simulaciones y análisis realizados con este programa. Los sistemas digitales en Logisim se representan intuitivamente de forma gráfica por su diagrama lógico con elementos fácilmente reconocibles por sus símbolos que se puede ajustar al estándar DIN 40700. Dispone de varias librerías con numerosos elementos constructivos que permiten el diseño lógico jerárquico de sistemas digitales de cierta complejidad como microcontroladores y microprocesadores. Las librerías nativas incorporadas son:

- *Wiring*. Proporciona elementos de conexionado entre objetos de un diagrama (*splitter* o separador, pines de E/S, *probe* o sonda de prueba, túneles, valor constante, extensor de bits,...).
- *Gates*. Colección de distintos tipos de puertas lógicas como NOT, AND, OR, NAND, NOR, etc.
- *Plexers*. Módulos combinacionales como multiplexor, decodificador, codificador, etc.
- *Arithmetic*. Módulos aritméticos como sumador, comparador, etc.
- *Memory*. Colección de elementos de memoria como biestables, registro, contador, etc.
- *Input/Output*. Proporciona elementos de E/S como pulsador, LED, displays, etc.

[2]Disponible en Internet, URL: `http://www.cburch.com/logisim/`.

- *Base*. Utilidades y controles para crear y editar los diagramas.

Logisim permite realizar el diseño jerárquico extensible, de modo que cualquier sistema construido se puede importar y emplear como un módulo o subcircuito en un nuevo diseño. Además, incorpora funcionalidades para simplificar los sistemas mediante el uso de señales multibit (buses) y documentar los diseños con la utilización de etiquetas de texto. Las *etiquetas* son nombres asociados a las señales del circuito, pero también se pueden utilizar como comentarios. Están formadas por caracteres alfanuméricos y el carácter '_' (subrayado).

> Recuerda que con Logisim se pueden simular tanto sistemas combinacionales como secuenciales con elementos de memoria. Para la simulación de sistemas secuenciales se proporciona un componente reloj con frecuencia de operación basada en una señal periódica interna (*tick*). La frecuencia de esta última se puede modificar desde 0.25 Hz a 4.1 kHz. La frecuencia máxima del bloque reloj de usuario es la mitad de la frecuencia elegida para el tick interno empleado en la simulación. De este modo, si se desea emplear una señal de reloj con un periodo de un segundo ($T = 1$ s) se debe emplear una frecuencia de tick de simulación de 2 Hz.

Todos los sistemas digitales de ejemplo que se proporcionan en este texto emplean preferentemente la convención que se indica a continuación.

- Para señales de entrada se emplean las etiquetas A, B, C, etc., también conocidas como variables lógicas asociadas a las señales. Tomando siempre este mismo orden decreciente de significancia para la combinación binaria de sus valores lógicos asociados. Este es el orden en que se mostrarán en las tablas de verdad de las señales de salida dependientes de ellas.
- Para señales de salida (funciones lógicas) se emplearán las etiquetas (variables): f, g, h, p, q, r, s, t, u, v, x, y, z.
- Para señales de E/S de tipo multibit (buses) se emplean las etiquetas (variables) similares a las anteriores: $A = [A_{n-1}\,A_{n-2}\ldots A_0]$, $f = [f_{n-1}\,f_{n-2}\ldots f_0]$, siendo n el número de bits de la señal. El bit menos significativo es el señalado por el subíndice menor.

> Existen numerosos proyectos derivados (*forks*) de Logisim cuyo propósito es la incorporación de nueva funcionalidad al proyecto original. Entre los proyectos derivados sobresalen por su mayor actividad y estabilidad: Logisim ITA[3] y Logisim Evolution.[4] Sin embargo, todos los ejemplos que se muestran en este texto han sido probados solo en la versión original del proyecto, aunque su adaptación a las versiones mencionadas es muy sencilla.

3.3 Puertas lógicas

Como ya se ha explicado en la sección 1.2 (pag. 4) y al inicio de este capítulo, el transistor es el elemento clave de los sistemas digitales, ya que permite dotar de un significado lógico a las señales que intervienen en su funcionamiento. De modo intuitivo un transistor se comporta como un interruptor microscópico controlado por una señal eléctrica capaz de cambiar de estado muy rápidamente y con un consumo reducido. En la actualidad, los circuitos integrados más complejos puede estar constituidos por miles de millones de estos diminutos transistores permitiendo la implementación de sistemas muy complejos. Para llegar a diseñar sistemas complejos se recurre al diseño jerárquico donde el transistor da paso a bloques constructivos sucesivamente más sofisticados desde los que parte el diseño en un nivel superior de la jerarquía. Así pues, en el nivel más bajo está el transistor y en el inmediato superior se sitúan las *puertas lógicas*.

Definición 3.3 — Puerta lógica. Una *puerta lógica* es un circuito compuesto de varios transistores con diversas señales de entrada y una salida que es una función lógica elemental de las señales de entrada. Dichas funciones son: NOT, AND, OR, XOR, NAND, NOR y XNOR. ■

A modo de ejemplo en la figura 3.2 se muestran los diagramas lógicos de las puertas NAND y NOR en Logisim. En la parte superior de la figura se han usado directamente transistores *nMOS/pMOS* (CMOS),

[3] Disponible en Internet, URL: https://github.com/Logisim-Ita/Logisim.
[4] Disponible en Internet, URL: https://github.com/logisim-evolution/logisim-evolution.

mientras que en la inferior se muestran las mismas funciones elementales con sus módulos equivalentes empleados habitualmente en los circuitos. Con esta última representación se evita un nivel de detalle innecesario para llevar a cabo el análisis y diseño lógico de los sistemas digitales.

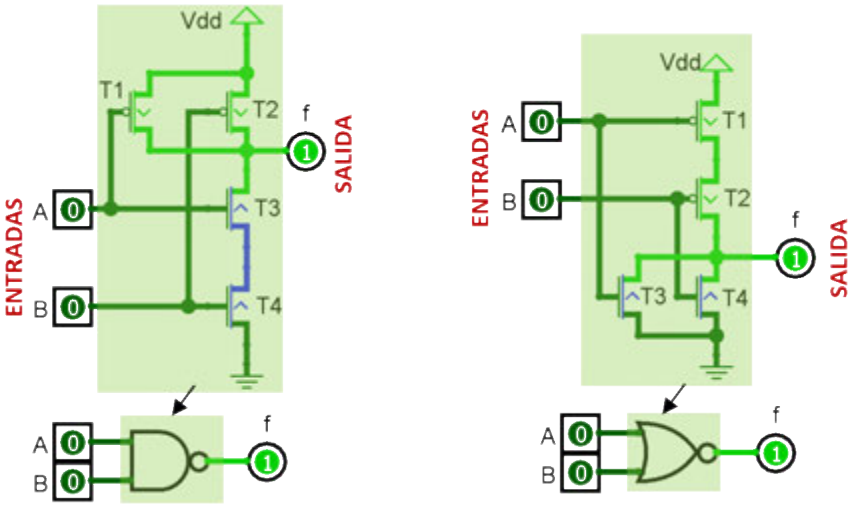


Figura 3.2: Diagramas lógicos en Logisim de circuitos con transistores nMOS/pMOS (CMOS) para puertas NAND (izda.) y NOR (dcha.) junto a sus módulos equivalentes.

Los circuitos lógicos correspondientes a las puertas lógicas (NAND y NOR) de la figura 3.2 demuestran como el cambio lógico de las señales de entrada A y B afecta al valor de la señal de salida f. La tabla 3.2 muestra los distintos valores que se obtienen para la salida para cada una de las combinaciones de valores lógicos de las señales de entrada.

Tabla 3.2: Tabla de verdad de las puertas lógicas NAND y NOR de 2 entradas.

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

> Los transistores nMOS y pMOS tienen un comportamiento complementario en cuanto a los niveles de tensión requeridos para que se comporten como un interruptor cerrado. El nMOS se activa con nivel alto (1) y el pMOS con nivel bajo (0). Cuando en la implementación de puertas lógicas intervienen ambos tipos de transistores complementarios se dice que el sistema emplea tecnología CMOS. Con esta se consigue reducir considerablemente el consumo eléctrico del conjunto. Recuerda que en este texto los circuitos lógicos —en su nivel más sencillo— emplean puertas lógicas, por lo que a partir de este punto no será necesario recurrir a una comprensión de los circuitos a nivel del transistor.

■ **Ejercicio 3.1** Comprobar con Logisim que los circuitos lógicos implementados con transistores y puertas lógicas que se muestran en la figura 3.2 son equivalentes y que las tablas de verdad de las funciones lógicas NAND y NOR son las indicadas en la tabla 3.2. ■

En las secciones siguientes se explica cada una de las puertas elementales describiendo sus características, propiedades y principales aplicaciones.

3.3.1 Puerta NOT o inversor

La puerta NOT o *inversor* es un *operador* que actúa sobre una única señal (entrada única), a diferencia de las puertas lógicas que actúan sobre dos o más señales. Su salida es el complemento o negación lógica de la señal de entrada. La figura 3.3 muestra el esquema correspondiente al uso de la puerta NOT.

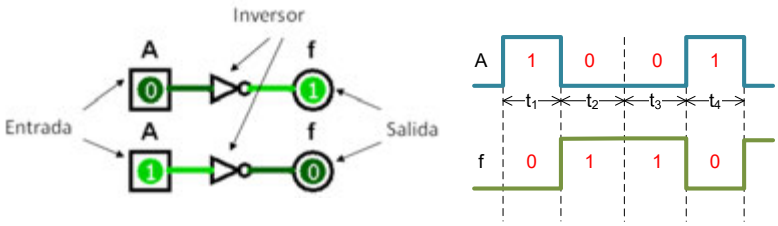


Figura 3.3: Circuito lógico con la puerta NOT en Logisim y cronograma ideal correspondiente.

> El símbolo del inversor consiste en un triángulo seguido con una «bolita» o burbuja en la punta (○). Esta burbuja es el rasgo distintivo del inversor, ya que señala la operación de inversión. Si se prescinde de la burbuja, el valor lógico de la señal permanece inalterado. En este caso el triángulo sin burbuja corresponde a un elemento denominado *buffer* cuyo propósito es aumentar el nivel de potencia de la señal sin alterar su nivel lógico. En este último caso se suele aumentar el tamaño del triángulo respecto al empleado para el inversor.

Para completar un circuito lógico de aplicación de la puerta NOT en Logisim, la puerta se debe conectar a una señal de entrada y a una señal de salida (como en los circuitos de la figura 3.3). En dichos circuitos la señal de entrada emplea un símbolo cuadrado (pin de entrada) en cuyo interior se ve sobreimpreso el valor lógico de la señal. Las señales de salida se conectan a través de un símbolo circular (pin de salida) con el valor lógico de la señal de la salida sobreimpreso en su interior.

En la simulación de cualquier circuito, el valor de las señales de entrada se puede modificar a voluntad mediante pulsación sucesiva del puntero sobre el pin de entrada correspondiente a la señal. El valor de las señales de salida se obtiene como resultado de las funciones lógicas aplicadas a las señales de entrada mediante los elementos lógicos del circuito que conectan las entradas con las salidas.

En los circuitos Logisim todas las señales de E/S deben tener asociado un pin de entrada o salida al que se asigna una etiqueta que coincide con el nombre de una variable o función lógica correspondiente a la señal (p. ej., A, B, f, etc.). En este texto se seguirá el criterio de nombrado de señales que se ha mencionado anteriormente (ver sección 3.2, pág. 39). En Logisim también es posible fijar valores constantes para las señales de un circuito. Se trata de valores lógicos de entradas que no se modifican durante la simulación del circuito. El elemento constante se proporciona en la librería *Wiring*. Es importante distinguir estos valores de las señales cuyo valor se puede modificar durante la simulación (variables).

Para representar la función lógica asociada a la puerta NOT se emplea el símbolo matemático de la negación lógica con una línea superior sobre el símbolo de la señal (variable) sobre la que se aplica $\overline{A}$ con la alternativa menos habitual $\neg A$ que se evitará en este texto.

Aunque en muchas obras de referencia se emplee la notación $\overline{A}$ para representar la operación de negación lógica, para evitar ambigüedades y facilitar su escritura, en este texto se usará preferentemente el símbolo prima para la negación lógica A'. Así pues, se tiene que: $A' \equiv \overline{A} \equiv \neg A$

> En Logisim para expresar la negación son válidas todas las expresiones lógicas siguientes:
>
> $$A' \equiv \sim A \equiv !A \equiv \text{NOT } A$$
>
> , recuerda que todas ellas son equivalentes a $\overline{A}$.

Para describir el resultado de aplicar la puerta NOT a una señal lógica se emplea su tabla de verdad, asociada al operador lógico del mismo nombre. De modo general, la tabla de verdad de una puerta lógica es la correspondiente a la función lógica implementada por dicha puerta (ver definición 3.2).

Tabla 3.3: Tabla de verdad de la puerta NOT (inversor o negación lógica).

A	A'
0	1
1	0

A partir de la tabla de verdad de una puerta lógica es sencillo realizar el cronograma correspondiente a la salida para una señal de entrada dada, ya que los valores de la señal de salida son los indicados en la tabla de verdad para cada uno de los valores de la señal de entrada. De este modo se obtiene el cronograma mostrado en la figura 3.3.

> Recuerda que el cronograma de una señal es la representación gráfica de su evolución temporal. Es un modo extendido de representar la relación lógica existente entre las señales de entrada y salida. A menos que se indique lo contrario, los cronogramas empleados en esta obra son ideales. Por tanto, en ellos no se reflejan ni los retardos de propagación de las señales, ni los tiempos de transición. En cierta medida las simulaciones con Logisim pueden tener en cuenta los retardos de propagación, pero en este texto no los tendremos en cuenta.

Propiedades de la puerta NOT

La principal propiedad de la puerta NOT se denomina *involución* por doble aplicación. Esta propiedad indica que el valor lógico de una señal no cambia al aplicarle dos operaciones de inversión consecutivas. Expresado de modo matemático:

$$(A')' = A'' = A$$

■ **Ejercicio 3.2** Comprueba con ayuda de Logisim que se cumple la propiedad de involución al conectar dos inversores seguidos a una señal de entrada. ■

Aplicaciones de la puerta NOT

Ya se ha explicado que la puerta NOT implementa la negación lógica u operador lógico NOT. Un ejemplo práctico de su utilidad es la implementación de la operación de cálculo del complemento a 1 (C1) de un número binario. En la figura 3.4 se muestra el esquema de un circuito en Logisim para calcular el C1 de una señal que representa un número binario de 1 byte. En la misma figura se muestra el empleo de señales multibit para simplificar la construcción con idéntica funcionalidad.

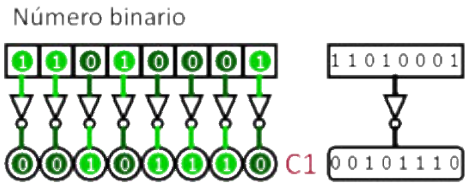


Figura 3.4: Simulación de circuito lógico del cálculo el complemento a 1 (C1) de un número de 8 bits. A la derecha la misma funcionalidad implementada con una señal de tipo multibit.

> Logisim permite el manejo de señales multibit y su conexión mediante buses. Las operaciones lógicas se aplican sobre dichas señales bit a bit. De este modo se consigue simplificar la elaboración de los diagramas correspondientes a circuitos complejos. Como se muestra en la figura 3.4, los pines de E/S se pueden definir de modo multibit a través del menú de atributos (atributo *Data bits*) o bien mediante el atajo de teclado Alt + n, donde *n* indica el número de bits de datos para el elemento seleccionado (p. ej., puerta lógica, pin de E/S, etc.).

3.3.2 Puerta AND

La puerta AND implementa la función lógica Y o producto lógico que admite varios símbolos para su expresión booleana.

Definición 3.4 — Puerta AND. La puerta AND es aquella cuya salida es 0 si alguna de sus entradas es igual a 0. Basta que una de sus entradas sea 0 para determinar que la salida también lo es. ■

Es una puerta cuyo valor de salida es 1 si y solo si todas sus señales de entrada tiene valor lógico 1. La figura 3.5 muestra un circuito lógico en Logisim con una puerta AND de dos entradas, en el que se muestra su símbolo gráfico estandarizado. La puerta AND tiene como mínimo dos entradas, pero es posible que tenga más de dos entradas.

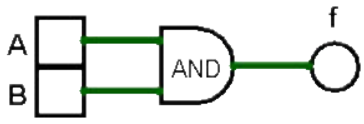


Figura 3.5: Circuito lógico Logisim para la puerta AND de dos entradas.

Aunque en Logisim es posible incluir puertas lógicas con multitud de entradas, los circuitos electrónicos reales de puertas lógicas con más de cuatro entradas no son habituales por las dificultades para conseguir un comportamiento adecuado. En Logisim se puede modificar el número de entradas de una puerta lógica seleccionada mediante pulsación de una tecla numérica [n] (siendo n el número de entradas deseado).

El símbolo matemático para la función lógica AND o producto lógico es el punto '$\cdot$' y '$\wedge$'. Sin embargo, dichos símbolos no se emplearán en este texto para evitar ambigüedad y facilitar la representación de la función AND. Se evitará especialmente el uso del símbolo '$\wedge$' por su confusión con el símbolo '^' (acento circunflejo) que en Logisim se emplea para la función lógica XOR (OR-exclusivo). En las expresiones lógicas de este texto no emplearemos ningún símbolo específico para la función AND, de modo que el producto lógico se representará separando las variables lógicas por un pequeño espacio en blanco. Así pues, las expresiones siguientes son equivalentes aunque se evitará el empleo de las dos últimas (señaladas en color rojo):

$$AB = A \cdot B = A \text{ AND } B = A \wedge B$$

En Logisim para expresar la función AND son válidas todas las expresiones lógicas siguientes:

$$AB \equiv A \text{ AND } B \equiv A\&B \equiv A\&\&B$$

Recuerda que en el producto es habitual omitir el punto '$\cdot$'.

Puesto que la salida de la puerta AND solo es 1 cuando todas sus entradas son 1, se tiene la tabla de verdad que se muestra a continuación (tabla 3.4). En dicha tabla se observa que su resultado coincide con el resultado del producto aritmético binario.

Tabla 3.4: Tabla de verdad de la puerta AND de dos entradas.

A	B	AB
0	0	0
0	1	0
1	0	0
1	1	1

■ **Ejercicio 3.3** Comprobar con ayuda de Logisim que el valor de salida de la puerta AND para los distintos valores de las entradas A y B son los indicados en la tabla 3.4. ■

Propiedades de la puerta AND

A partir del estudio de la tabla de verdad correspondiente a la puerta AND se pueden describir un conjunto de propiedades y la expresión matemática con variables lógicas que se deriva de cada propiedad:

- **Propiedad conmutativa**. Se pueden intercambiar las señales de entrada de una puerta AND de dos entradas sin afectar a la salida de la misma.
  $AB = BA$
- **Propiedad asociativa**. Al encadenar dos puertas AND, el orden en que se asocian estas no afecta al resultado. Esta propiedad proporciona un modo en que se puede implementar una puerta AND con más de dos entradas mediante concatenación de puertas de dos entradas.
  $A(BC) = (AB)C \equiv ABC$

- **Propiedad del elemento nulo**. Si una de las entradas de una puerta AND se conecta a una constante de valor 0, la salida siempre será 0 independientemente del valor de la segunda entrada. Es decir, basta con que una de las entradas de la puerta sea 0 para determinar que el valor de la salida es 0.
  $A \cdot 0 = 0$
- **Propiedad del elemento neutro o unidad**. Si una de las entradas de una puerta AND se conecta a una constante de valor 1, la salida depende del valor de la segunda entrada. Es decir, las entradas que valen 1 tienen efecto neutro en la salida. De este modo, es posible emplear una puerta AND con más entradas de las necesarias si las entradas sobrantes se conectan a un valor constante 1.
  $A \cdot 1 = A$
- **Propiedad de idempotencia**. Si todas las entradas de una puerta AND se conectan a la misma señal, la salida tiene el mismo valor que la entrada.
  $A \cdot A = A$
- **Propiedad del complemento**. Si en las entradas de una puerta AND se conecta tanto una señal como su complemento, la salida de la puerta AND siempre es 0 (elemento nulo para el producto lógico).
  $A \cdot A' = 0$

La tabla 3.5 resume las propiedades de la puerta AND. En dicha tabla se han marcado aquellas que conviene recordar por su utilidad en la simplificación de circuitos.

Tabla 3.5: Propiedades de la puerta AND.

Expresión algebraica	Denominación
$A \cdot B = B \cdot A$	*Conmutativa*
$A \cdot (B \cdot C) = (A \cdot B) \cdot C \equiv A \cdot B \cdot C$	*Asociativa*
$A \cdot 0 = 0$	*Elemento nulo*
$A \cdot 1 = A$	*Elemento neutro*
$A \cdot A = A$	*Idempotencia*
$A \cdot A' = 0$	*Complemento*

Entre las propiedades de la puerta AND se destacan la propiedad del elemento nulo y del complemento porque permiten simplificar los circuitos con puertas AND. Si una de las entradas de una puerta AND es 0, o entre sus entradas aparecen tanto una señal como su complementaria, la puerta se puede sustituir por una señal constante de valor 0.

Recuerda que la propiedad asociativa es interesante porque permite la implementación de un producto lógico de varias variables sin necesidad de crear una puerta con un gran número de entradas. Esto último puede ser imposible en la práctica por limitaciones técnicas, pero es factible llegar a una solución equivalente mediante la concatenación de puertas lógicas más sencillas. La figura 3.6 muestra los circuitos equivalentes de puertas AND de dos entradas y la puerta AND de tres entradas equivalente a ellos.

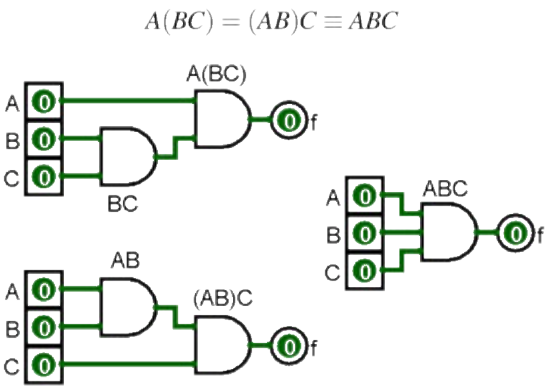


Figura 3.6: Circuitos lógicos equivalentes con puertas AND.

■ **Ejercicio 3.4** Con ayuda de Logisim demuestra la validez de las propiedades de la puerta lógica AND de dos entradas y su extensión a puertas con más entradas. ■

Aplicaciones de la puerta AND

La función lógica AND se emplea frecuentemente en los sistemas digitales para comprobar si varias condiciones se satisfacen simultáneamente. De este modo, por ejemplo en la figura 3.7 se muestra el esquema de un posible circuito digital que activa su salida si el conductor de un vehículo lo arranca sin haberse abrochado el cinturón de seguridad. Esto es, la condición: *«Si el vehículo se ha puesto en marcha Y el cinturón NO está abrochado ENTONCES generar alerta»*. En este circuito la puerta AND es la encargada de generar la señal de alerta cuando se cumple simultáneamente que el vehículo está en marcha y el cinturón no está abrochado. Las señales de entrada al circuito serían las procedentes de sensores en el sistema de arranque del vehículo y anclaje del cinturón de seguridad. Las salidas podrían activar señales acústicas y luminosas en el salpicadero que avisarían al conductor de la situación de alerta.

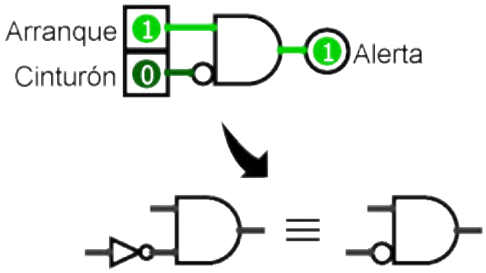


Figura 3.7: Circuito lógico Logisim de aplicación de puerta AND que comprueba dos condiciones simultáneamente para activar la alerta de un vehículo.

El circuito de la figura 3.7 muestra como las entradas de una puerta AND pueden estar precedidas de un inversor para negar una condición. En este caso particular que el cinturón NO esté abrochado es una de las condiciones para que el valor de salida de la puerta AND sea 1. Esta condición activada cuando una variable lógica tiene valor 0, se indica mediante un inversor aplicado a dicha señal de entrada.

> Recuerda que en un circuito Logisim la conexión de un inversor precediendo la entrada de una puerta lógica es equivalente a negar la entrada correspondiente de dicha puerta, indicado mediante una burbuja (○) en la entrada señalada. La negación de las entradas se indica en el panel de atributos de la puerta lógica.

La puerta AND también se utiliza en los sistemas digitales para habilitar/inhibir la señal de salida. La habilitación/inhibición (*enable/disable*) de una señal lógica es el proceso que permite que dicha señal evolucione o no hacia la salida del sistema. En el circuito Logisim de la figura 3.8-izda. la puerta AND deja pasar la señal de entrada hacia la salida cuando la señal *Enable* es 1. Ya que si esta es 0, la salida será 0 independientemente del valor de la entrada (ver propiedad de elemento nulo). También es posible que la señal de habilitación sea activa a nivel bajo (ver figura 3.8-dcha.). En este caso la señal va afectada por una negación antes de entrar en la puerta AND.

Figura 3.8: Ejemplo de habilitación a nivel alto (izda.) y bajo (dach.) de una señal variable (reloj) mediante una puerta AND.

> Para facilitar la simulación en Logisim del circuito de la figura 3.8, la señal de *Enable* se genera mediante un pulsador activado por los clics del puntero sobre él. Recuerda que el elemento pulsador (*Button*) está disponible desde la librería *Input/Output* de Logisim. El elemento *Clock* está disponible en la librería *Wiring*.

■ **Ejercicio 3.5** Reproduce en Logisim los circuitos de la figura 3.8 y simula su comportamiento. Añade en la salida un elemento LED de la librería *Input/Output* para visualizar mejor el cambio de estado en la salida y una frecuencia de simulación reducida (p. ej., 2 Hz). ■

3.3.3 Puerta OR

La puerta OR implementa la función lógica del mismo nombre denominada también suma lógica que admite varios símbolos para su expresión booleana.

> **Definición 3.5 — Puerta OR.** La puerta OR es aquella cuya salida es 1 si alguna de sus entradas es igual a 1. Basta que una de sus entradas sea 1 para determinar que la salida es 1. ■

Es una puerta cuyo valor de salida es 0 si y solo si todas sus señales de entrada tiene valor lógico 0. La figura 3.9 muestra un circuito lógico en Logisim con una puerta OR de dos entradas, en el que se muestra su símbolo gráfico estandarizado. La puerta OR tiene como mínimo dos entradas, pero es posible que tenga más de dos entradas.

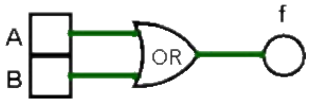


Figura 3.9: Circuito lógico Logisim para la puerta OR de dos entradas.

Los símbolos matemáticos para la función lógica OR o suma lógica son '$\vee$' y preferentemente '$+$', aunque este último no debe confundirse con la suma aritmética.

> En Logisim para expresar la función OR son válidas todas las expresiones lógicas siguientes:
>
> $$A+B \equiv A \text{ OR } B \equiv A|B \equiv A||B$$

Puesto que la salida de la puerta OR solo es 0 cuando todas sus entradas son 0, se tiene la tabla de verdad que se muestra a continuación (tabla 3.6). En dicha tabla se observa que su resultado no coincide con el resultado de la suma aritmética binaria.

Tabla 3.6: Tabla de verdad de la puerta OR de dos entradas.

A	B	$A+B$
0	0	0
0	1	1
1	0	1
1	1	1

■ **Ejercicio 3.6** Comprobar con ayuda de Logisim que el valor de salida de la puerta OR para los distintos valores de las entradas A y B son los indicados en la tabla 3.6. ■

Propiedades de la puerta OR

A continuación se describen las propiedades de la puerta OR y la expresión matemática con variables lógicas que se deriva de cada una:

- **Propiedad conmutativa**. Se pueden intercambiar las señales de entrada de una puerta OR de dos entradas sin afectar a la salida de la misma.
  $A+B=B+A$
- **Propiedad asociativa**. Al encadenar dos puertas OR consecutivas, el orden en que se asocian estas no afecta al resultado. Esta propiedad proporciona un modo en que se puede implementar una puerta OR con más de dos entradas mediante concatenación de puertas de dos entradas.
  $A+(B+C)=(A+B)+C \equiv A+B+C$

- **Propiedad del elemento nulo**. Si una de las entradas de una puerta OR se conecta a una constante de valor 1, la salida siempre será 1 independientemente del valor de la segunda entrada. Es decir, basta con que una de las entradas de la puerta sea 1 para determinar que el valor de la salida es 1. Observa que a diferencia de la operación AND, en la suma lógica (OR) el 1 es el elemento nulo.
  $A+1=1$
- **Propiedad del elemento neutro o unidad**. Si una de las entradas de una puerta OR se conecta a una constante de valor 0, la salida depende del valor de la segunda entrada. Es decir, las entradas que valen 0 tienen efecto neutro en la salida. Observa que a diferencia de la operación AND, en la suma lógica (OR) el 0 es el elemento neutro o unidad. Así pues, es posible emplear una puerta OR con más entradas de las necesarias si las entradas sobrantes se conectan a un valor constante 0.
  $A+0=A$
- **Propiedad de idempotencia**. Si todas las entradas de una puerta OR se conectan a la misma señal, el valor de la señal de salida coincide con el de entrada.
  $A+A=A$
- **Propiedad del complemento**. Si en las entradas de una puerta OR están conectadas una señal y su complemento, la salida de la puerta OR siempre es 1 (elemento nulo para la suma lógica).
  $A+A'=1$

La tabla 3.7 resume las propiedades de la puerta OR. En dicha tabla se han marcado aquellas que conviene recordar por su utilidad en la simplificación de circuitos.

Tabla 3.7: Propiedades de la puerta OR.

Expresión algebraica	Denominación
$A+B=B+A$	*Conmutativa*
$A+(B+C)=(A+B)+C\equiv A+B+C$	*Asociativa*
$A+1=1$	*Elemento nulo*
$A+0=A$	*Elemento neutro*
$A+A=A$	*Idempotencia*
$A+A'=1$	*Complemento*

Entre las propiedades de la puerta OR se destacan la propiedad del elemento nulo y del complemento porque permiten simplificar los circuitos con puertas OR. Si una de las entradas de una puerta OR es 1, o entre sus entradas aparecen tanto una señal como su complementaria, la puerta se puede sustituir por una señal constante de valor 1.

■ **Ejercicio 3.7** Con ayuda de Logisim demuestra la validez de las propiedades de la puerta lógica OR de dos entradas y su extensión a puertas con más entradas. ■

Aplicaciones de la puerta OR

La función lógica OR se emplea frecuentemente en los sistemas digitales para comprobar si se cumple alguna condición entre varias. De este modo, por ejemplo en la figura 3.10 se muestra el esquema de un posible circuito digital que activa su salida si alguna de las condiciones de intrusión en el hogar se activa. Esto es, la condición: *«SI se abre una ventana O se abre la puerta principal O se detecta presencia en el pasillo ENTONCES se genera una alarma»*. En este circuito la puerta OR es la encargada de generar la señal de alerta cuando se cumple alguna de las condiciones de intrusión. Las señales de entrada al circuito serían las procedentes de los distintos sensores de intrusión. Las salidas podrían activar señales acústicas y enviar un mensaje a la central de alarmas de la empresa encargada de la seguridad doméstica.

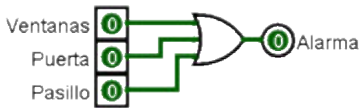


Figura 3.10: Circuito lógico Logisim de aplicación de puerta OR que comprueba alguna de las condiciones de alarma de intrusión en un hogar.

3.3.4 Puerta NAND

La puerta NAND implementa la función lógica AND negada denominada también producto lógico negado. Es decir, equivale a una puerta AND seguida de un inversor.

> **Definición 3.6 — Puerta NAND.** La puerta NAND es aquella cuya salida es 1 si alguna de sus entradas es igual a 0. Basta que una de sus entradas sea 0 para determinar que la salida es 1. ■

Es una puerta cuyo valor de salida es 0 si y solo si todas sus señales de entrada tiene valor lógico 1. La figura 3.11 muestra un circuito lógico en Logisim con una puerta NAND de dos entradas, en el que se muestra su símbolo gráfico estandarizado. La puerta NAND tiene como mínimo dos entradas, pero es posible que tenga más de dos entradas.

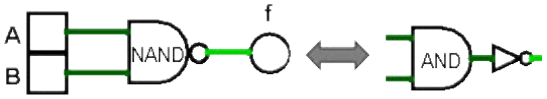


Figura 3.11: Circuito lógico Logisim para la puerta NAND de dos entradas.

La función lógica NAND se expresa matemáticamente mediante la negación de la operación AND: $(AB)' \equiv \overline{AB}$

Puesto que la salida de la puerta NAND es la negación de la función AND, se tiene la tabla de verdad que se muestra a continuación.

Tabla 3.8: Tabla de verdad de la puerta NAND de dos entradas.

A	B	$(AB)'$
0	0	1
0	1	1
1	0	1
1	1	0

■ **Ejercicio 3.8** Comprobar con ayuda de Logisim que el valor de salida de la puerta NAND para los distintos valores de las entradas A y B son los indicados en la tabla 3.8. ■

Propiedades de la puerta NAND

De forma similar a la puerta lógica AND se pueden establecer las propiedades de la puerta NAND, resumidas en la tabla 3.9.

Tabla 3.9: Propiedades de la puerta NAND.

Expresión algebraica	Denominación
$(A \cdot B)' = (B \cdot A)'$	*Conmutativa*
$(A \cdot 0)' = 1$	*Elemento nulo*
$(A \cdot 1)' = A'$	*Elemento neutro*
$(A \cdot A)' = A'$	*Idempotencia*
$(A \cdot A')' = 1$	*Complemento*

La puerta NAND no satisface la propiedad asociativa. Es decir, no es posible construir una puerta NAND de múltiples entradas concatenando puertas de dos entradas:

$$((AB)'C)' \neq (A(BC)')' \neq (ABC)'$$

■ **Ejercicio 3.9** Con ayuda de Logisim demuestra la validez de las propiedades de la puerta lógica NAND de dos entradas. Comprueba la invalidez de la propiedad asociativa. ■

3.3.5 Puerta NOR

La puerta NOR implementa la función lógica OR negada denominada también suma lógica negada. Es decir, equivale a una puerta OR seguida de un inversor.

> **Definición 3.7 — Puerta NOR.** La puerta OR es aquella cuya salida es 0 si alguna de sus entradas es igual a 1. Basta que una de sus entradas sea 1 para determinar que la salida es 0. ■

Es una puerta cuyo valor de salida es 1 si y solo si todas sus señales de entrada tiene valor lógico 0. La figura 3.12 muestra un circuito lógico en Logisim con una puerta NOR de dos entradas, en el que se muestra su símbolo gráfico estandarizado. La puerta NOR tiene como mínimo dos entradas, pero es posible que tenga más de dos entradas.

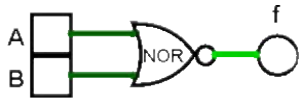


Figura 3.12: Circuito lógico Logisim para la puerta NOR de dos entradas.

La función lógica NOR se expresa matemáticamente mediante la negación de la operación OR: $(A+B)' \equiv \overline{A+B}$

Puesto que la salida de la puerta NOR es la negación de la función OR, se tiene la tabla de verdad que se muestra a continuación.

Tabla 3.10: Tabla de verdad de la puerta NOR de dos entradas.

A	B	$(A+B)'$
0	0	1
0	1	0
1	0	0
1	1	0

■ **Ejercicio 3.10** Comprobar con ayuda de Logisim que el valor de salida de la puerta NOR para los distintos valores de las entradas A y B son los indicados en la tabla 3.10. ■

Propiedades de la puerta NOR

La tabla 3.11 resume las propiedades de la puerta NOR.

Tabla 3.11: Propiedades de la puerta NOR.

Expresión algebraica	Denominación
$(A+B)' = (B+A)'$	*Conmutativa*
$(A+1)' = 0$	*Elemento nulo*
$(A+0)' = A'$	*Elemento neutro*
$(A+A)' = A'$	*Idempotencia*
$(A+A')' = 0$	*Complemento*

La puerta NOR no satisface la propiedad asociativa. Es decir, una puerta NOR de tres entradas no equivale a la contatenación de dos puertas NOR:

$$((A+B)'+C)' \neq (A+(B+C)')' \neq (A+B+C)'$$

■ **Ejercicio 3.11** Con ayuda de Logisim demuestra la validez de las propiedades de la puerta lógica NOR de dos entradas. Comprueba la invalidez de la propiedad asociativa. ■

3.3.6 Puertas equivalentes a puertas NAND y NOR

Al reflexionar sobre las definiciones dadas de las puertas lógicas NAND y NOR se deduce la relación que existe entre dichas puertas lógicas y las puertas OR y AND, respectivamente. De este modo, la definición de la puerta NAND coincide con proporcionada para la puerta OR cuando el estado de las entradas es 0 en lugar de 1. De igual modo, se puede reflexionar con las puertas NOR y AND. Por tanto, se concluye la equivalencia de las puertas NAND y NOR que se indica a continuación y se muestra de modo gráfico en la figura 3.13.

Definición 3.8 — Equivalencias para las puertas NAND y NOR. Una puerta NAND equivale a una puerta OR con todas sus entradas negadas y una puerta NOR equivale a una puerta AND con sus entradas negadas. ■

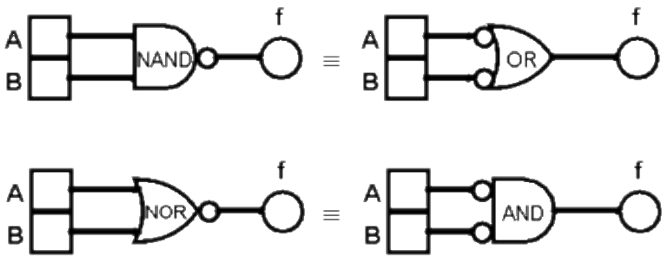


Figura 3.13: Circuitos lógicos equivalentes para la puerta NAND y NOR.

Recurriendo a las expresiones matemáticas de las puertas elementales NAND, OR, NOR y AND, se tiene que:

1. $(AB)' \equiv A' + B'$
2. $(A + B)' \equiv A'B'$

Estas dos equivalencias, conocidas como leyes de De Morgan son muy útiles para manipular las expresiones algebraicas de las funciones lógicas cuando un operador de negación afecta a un término suma o producto, ya que la equivalencia se cumple independientemente del número de variables que aparecen en el término suma o producto.

■ **Ejercicio 3.12** Con ayuda de Logisim demuestra la validez de la equivalencia de las puertas NAND y NOR, con las puertas OR y AND con sus entradas negadas. Realiza la comprobación para puertas con más de dos entradas. ■

3.3.7 Puerta XOR

La puerta OR-exclusivo o XOR se define del modo que se indica a continuación.

Definición 3.9 — Puerta OR-exclusivo o XOR. La puerta OR-exclusivo o XOR de dos entradas es aquella cuya salida es 1 si alguna de sus entradas es igual a 1, pero no ambas. Es una puerta cuya salida es 1 si sus entradas son distintas y 0 si sus entradas son iguales. ■

La figura 3.14 muestra un circuito lógico en Logisim con una puerta XOR de dos entradas, en el que se muestra su símbolo gráfico estandarizado. La puerta XOR tiene como mínimo dos entradas, pero es posible que tenga más de dos entradas. El símbolo matemático para la función lógica XOR es '$\oplus$'.

Figura 3.14: Circuito lógico Logisim para la puerta XOR de dos entradas.

En Logisim para expresar la función XOR no se emplea el símbolo $\oplus$. En su lugar se emplean las expresiones equivalentes:

$$A\hat{}B \equiv A'B + AB' = (A+B)(A'+B')$$

Tabla 3.12: Tabla de verdad de la puerta XOR de dos entradas.

A	B	$A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

■ **Ejercicio 3.13** Comprobar con ayuda de Logisim que el valor de salida de la puerta XOR para los distintos valores de las entradas A y B son los indicados en la tabla 3.12. ■

Propiedades de la puerta XOR

La tabla 3.13 resume las propiedades de la puerta XOR.

Tabla 3.13: Propiedades de la puerta XOR.

Expresión algebraica	Denominación
$A \oplus B = B \oplus A$	*Conmutativa*
$A \oplus (B \oplus C) = (A \oplus B) \oplus C \equiv A \oplus B \oplus C$	*Asociativa*
$A \oplus 0 = A$	*Elemento neutro*
$A \oplus 1 = A'$	*Inversión*
$A \oplus A = 0$	—
$A \oplus A' = 1$	—
$(A' \oplus B') = A \oplus B$	*Invarianza a la inversión**

Como se ha señalado la puerta XOR satisface la propiedad asociativa. Esto proporciona un método para la implementación práctica de puertas de más de dos entradas como contatenación de puertas XOR de dos entradas. La puerta XOR de múltiples entradas resultante quedaría definida como *una puerta cuya salida proporciona la paridad impar de las entradas*. Esto es, su salida vale 1 si el número de entradas que vale 1, es un número impar. No obstante, una puerta XOR con más de dos entradas pueden adoptar una definición alternativa proporcionada por el estándar IEEE91, pero no compatible con la propiedad asociativa. En dicha definición una puerta XOR de múltiples entradas *es aquella cuya salida es 1 si y solo si, una de sus entradas vale 1*. En las puertas de dos entradas, las dos definiciones de puerta XOR son compatibles y solo difieren para puertas de tres o más entradas.

En la lista de propiedades de la tabla 3.13 se ha señalado que la puerta XOR es invariante a la inversión de sus entradas. Sin embargo, esto es cierto solo si el número de entradas es par y la definición del XOR es compatible con la propiedad asociativa. De este modo se tiene que:

$$A' \oplus B' = A \oplus B \text{ , pero } A' \oplus B' \oplus C' \neq A \oplus B \oplus C$$

Recuerda que las puertas XOR de múltiples entradas tienen dos posibles definiciones válidas. La que considera su salida como el valor de paridad impar de las entradas (en congruencia con la propiedad asociativa) o aquella que indica que la salida es 1, solo si una de las entradas es 1. Esta última es la definición intuitiva utilizada por defecto en Logisim, aunque es posible seleccionar cualquiera de las dos definiciones mediante los atributos de la puerta.

■ **Ejercicio 3.14** Con ayuda de Logisim demuestra la validez de las propiedades de la puerta lógica XOR de dos entradas. Comprueba la validez o invalidez de la propiedad asociativa dependiendo de la definición empleada para las puertas de múltiples entradas. ■

Aplicaciones de la puerta XOR

A pesar de no implementar una función lógica elemental, la puerta XOR tiene aplicaciones interesantes:

1. Comparación entre bits. La puerta XOR es 1 si los bits comparados son distintos.
2. Suma aritmética. La salida de la puerta XOR es el resultado de la suma aritmética de dos bits sin tener en cuenta el acarreo final.
3. Inversor controlado. Cuando una entrada de la puerta XOR es igual a 1, se comporta como un inversor para la otra entrada. Este hecho se emplea en los módulos aritméticos para calcular el complemento a dos de un número binario. En el ejemplo de la figura 3.15 se ilustra el cálculo del C2 de un número de 4 bits a partir del valor de la magnitud y el bit de signo.

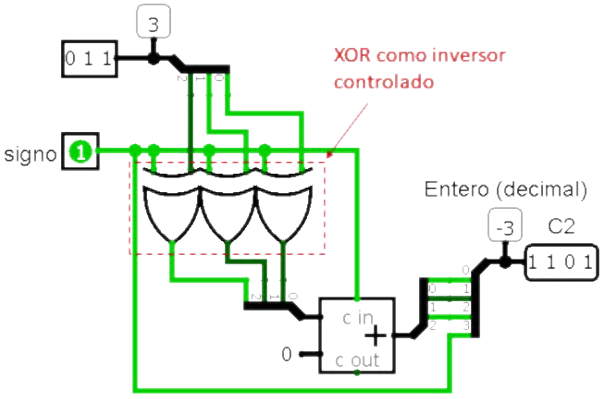


Figura 3.15: Circuito lógico Logisim de aplicación de puerta XOR como inversor controlado para cálculo del complemento a 2 de un número binario.

3.3.8 Puerta XNOR

La puerta XNOR de dos entradas se define del modo que se indica a continuación.

Definición 3.10 — Puerta OR-exclusivo negada o XNOR. La puerta OR-exclusivo negada o XNOR de dos entradas es aquella cuya salida es 1 si ambas entradas son iguales. ■

La figura 3.16 muestra un circuito lógico en Logisim con una puerta XNOR de dos entradas en el que se muestra su símbolo gráfico estandarizado. La puerta XNOR tiene como mínimo dos entradas, pero es posible que tenga más. El símbolo matemático para la función lógica XNOR es '$\odot$'. Así se tiene:

$$(A \oplus B)' \equiv \overline{A \oplus B} \equiv A \odot B$$

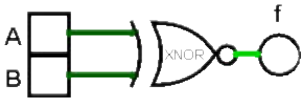


Figura 3.16: Circuito lógico Logisim para la puerta XNOR de dos entradas.

En Logisim para expresar la función XNOR no se emplea el símbolo $\odot$. En su lugar se emplean las expresiones equivalentes:

$$(A \hat{} B)' \equiv AB + A'B' = (A + B')(A' + B)$$

Tabla 3.14: Tabla de verdad de la puerta XNOR de dos entradas.

A	B	$(A \oplus B)'$
0	0	1
0	1	0
1	0	0
1	1	1

Propiedades de la puerta XNOR

A continuación, la tabla 3.15 resume las propiedades de la puerta XNOR. En dicha tabla se han marcado aquellas que conviene recordar por su utilidad en la simplificación de circuitos.

Tabla 3.15: Propiedades de la puerta XNOR.

Expresión algebraica	Denominación
$(A \oplus B)' = (B \oplus A)'$	*Conmutativa*
$(A \oplus (B \oplus C)')' = ((A \oplus B)' \oplus C)' \equiv (A \oplus B \oplus C)'$	*Asociativa*
$(A \oplus 0)' = A'$	*Inversión*
$(A \oplus 1)' = A$	*Elemento neutro*
$(A \oplus A)' = 1$	—
$(A \oplus A')' = 0$	—
$(A \oplus B)' = A' \oplus B = A \oplus B'$	—

3.3.9 Universalidad de puertas NAND y NOR

Las puertas NOT, AND y OR se denominan *puertas lógicas elementales* porque combinándolas en un circuito lógico se puede implementar cualquier función lógica. Las puertas XOR y XNOR proporcionan operaciones secundarias, de utilidad para implementar funciones relacionadas con los circuitos comparadores y aritméticos. Sin embargo, se debe recordar que dichas funciones también se pueden implementar con las puertas elementales. Basta con tener en cuenta las expresiones algebraicas desarrolladas de las puertas XOR y XNOR:

$$A \oplus B = A'B + AB' = (A+B)(A'+B')$$
$$(A \oplus B)' = AB + A'B' = (A'+B)(A+B')$$

La figura 3.17 muestra los circuitos lógicos equivalentes para las puertas XOR y XNOR realizados con puertas elementales. La negación a la entrada de las puertas AND y OR se indica mediante una burbuja (○), equivalente a la presencia de una puerta inversora.

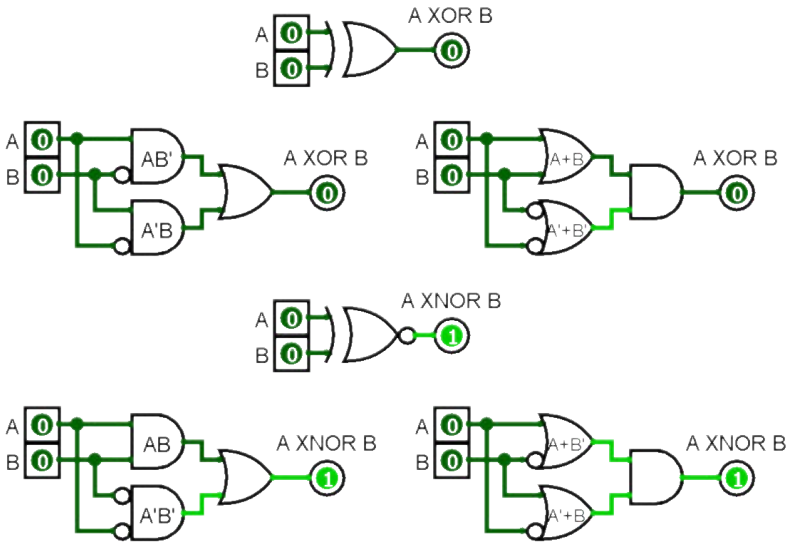


Figura 3.17: Equivalencia de las puertas XOR y XNOR implementadas con puertas elementales.

> **Definición 3.11 — Puerta lógica universal.** Una *puerta lógica universal* es aquella con la que se puede implementar cualquier función lógica. Para considerar que una puerta es universal basta comprobar que permite expresar las tres operaciones lógicas elementales: NOT, AND y OR. ■

Las puertas NAND y NOR son puertas universales, ya que solo con una de ellas es posible expresar cualquier función lógica que imaginemos. La figura 3.18 muestra como cualquiera de las puertas elementales (NOT, AND y OR) se puede implementar con las puertas NAND y NOR, lo cual es suficiente para demostrar su condición de universalidad.

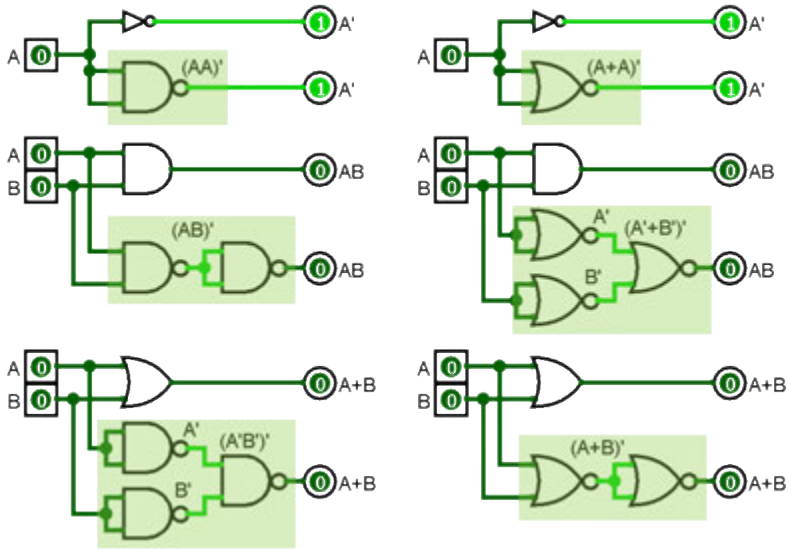


Figura 3.18: Circuitos equivalentes a puertas elementales NOT, AND y OR obtenidos mediante puertas universales NAND (izda.) y NOR (dcha.).

Las puertas universales tienen ante todo interés práctico, ya que al construir los circuitos electrónicos que implementan las funciones lógicas es más sencillo fabricar un circuito con un único tipo de puerta lógica (NAND o NOR, ver en Fig. 3.2 de la página 41, el esquema del circuito electrónico de su implementación con tecnología CMOS). La figura 3.19 es un resumen de los tipos de puertas lógicas existentes.

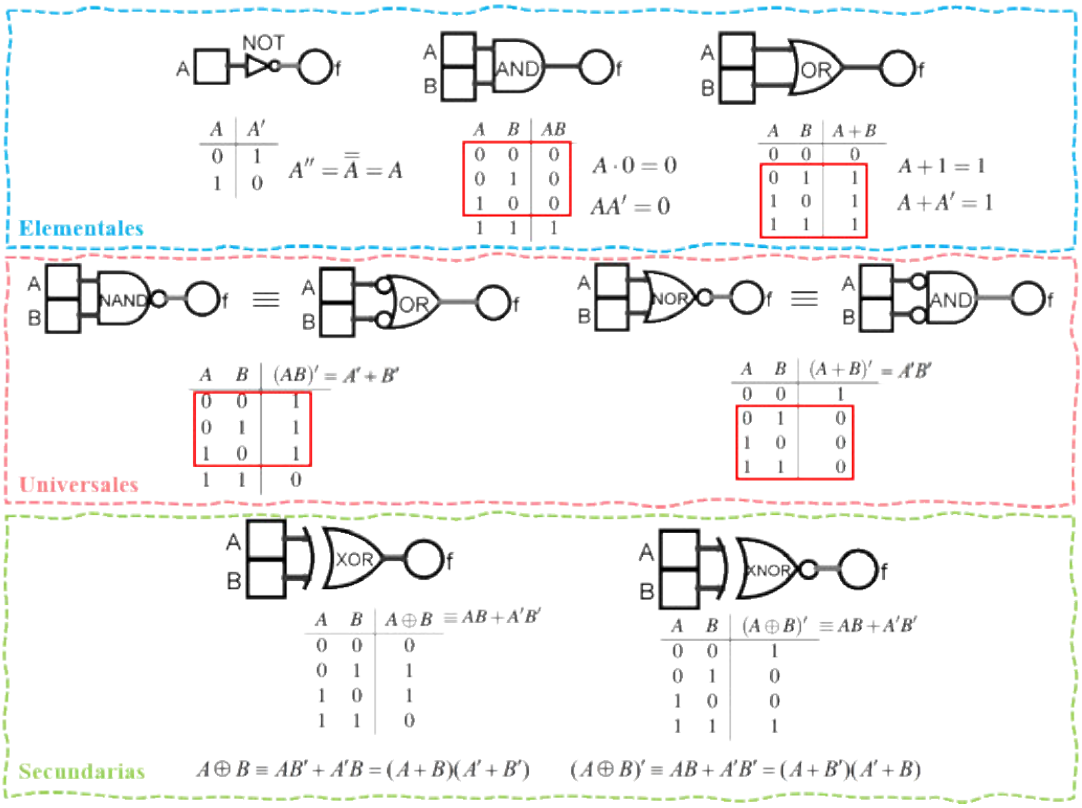


Figura 3.19: Resumen de todas las puertas lógicas con sus símbolos y tablas de verdad.

3.4 Álgebra de Boole aplicada a los sistemas digitales

George Boole fue un matemático inglés que a mediados del siglo XIX formuló un álgebra para tratar con las proposiciones lógicas, a las que había dedicado años de estudio dentro de la *lógica proposicional*.

> **Definición 3.12 — Álgebra de Boole.** Es una estructura matemática definida sobre un conjunto $\mathbb{Z}_2$ de dos elementos, sobre el que se definen tres operaciones básicas, negación $\neg$ ($'$), conjunción $\wedge$ o producto lógico ($\cdot$), y disyunción $\vee$ o suma lógica ($+$), junto a unas propiedades satisfechas por dichas operaciones. Su propósito inicial fue proporcionar herramientas matemáticas para el cálculo proposicional. ∎

El matemático americano Edward V. Huntington (1874-1952) perfeccionó el planteamiento del álgebra de Boole enunciando los postulados siguientes:

1. El conjunto $\mathbb{Z}_2 = \{0, 1\}$ es cerrado respecto a las operaciones lógicas de producto y suma.
2. El elemento identidad del producto lógico es 1 y el de la suma es el 0. Esto es:

$$A \cdot 1 = A$$
$$A + 0 = A$$

3. Tanto el producto lógico como la suma lógica cumplen la propiedad conmutativa:

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

4. Se cumple la propiedad distributiva del producto respecto de la suma y de la suma respecto del producto —aunque esta última no es válida en el caso del álgebra de los números reales—:

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

5. Se define un operador de complemento o negación ($'$, NOT) de modo que se cumple $0' = 1$ y $1' = 0$. Este operador también cumple las propiedades siguientes para el producto y la suma lógicos:

$$A \cdot A' = 0$$
$$A + A' = 1$$

6. Existen al menos dos elementos pertenecientes al conjunto $\mathbb{Z}_2$ y ambos son distintos:

$$\exists x, y \in \mathbb{Z}_2 : x \neq y$$

Para interpretar correctamente las expresiones algebraicas lógicas se debe tener en cuenta la precedencia de las tres operaciones básicas. El orden de precedencia es: NOT, AND y OR. Para alterar el orden de precedencia de dichas operaciones se emplean paréntesis. De este modo, en la expresión $A \cdot B'$, la negación afecta solo a B. Si se desea negar todo el producto es preciso utilizar un paréntesis para indicar que la negación afecta a todo el producto: $(A \cdot B)'$.

En 1937, el ingeniero americano Claude Shannon propuso en su tesis de máster la aplicación del álgebra de Boole[5] a los circuitos lógicos eléctricos —denominados puertas lógicas— implementados con relés u otros dispositivos de conmutación. De este modo demostró que cualquier función lógica se puede implementar mediante circuitos electrónicos digitales. En 1948, C. Shannon publicó[6] un trabajo titulado *«Una teoría matemática de la comunicación» («A Mathematical Theory of Communication»)* con

[5] *The Mathematical Analysis of Logic, Being an Essay towards a Calculus of Deductive Reasoning*. London, England: Macmillan, Barclay, & Macmillan, 1847.

[6] *A Mathematical Theory of Communication*. The Bell System Technical Journal, 27 (3), July 1948. DOI: 10.1002/j.1538-7305.1948.tb01338.x.

el que se convirtió en fundador de la *Teoría de la Comunicación* y, por tanto, también de las *Ciencias de la Computación*.

Figura 3.20: (De izda. a dcha.) George Boole (1815-1864), Augustus De Morgan (1806-1871), y Claude Shannon (1916-2001). Fuente: G. Boole y A. De Morgan bajo Dominio Público; C. Shannon de Konrad Jacobs, Erlagen, CC-BY-SA 2.0.

Para C. Shannon no pasó desapercibido que cada una de las operaciones lógicas básicas del álgebra de Boole ($'$, $\cdot$ y $+$) se pueden realizar con circuitos eléctricos basados en dispositivos de conmutación conocidos como puertas lógicas NOT, AND y OR. El álgebra de Boole permite formalizar el análisis y diseño de sistemas digitales en los que se asocian dos valores lógicos a dos niveles distintos de una magnitud eléctrica. De este modo, cualquier expresión lógica se puede materializar mediante un circuito digital y a la inversa, la función de cualquier circuito digital se puede expresar como una relación algebraica booleana entre variables lógicas.

> Recuerda que una de las conclusiones más importantes del trabajo de C. Shannon es que cualquier resultado obtenido por aplicación del álgebra de Boole se puede reproducir mediante el empleo de los circuitos lógicos equivalentes a las expresiones booleanas implicadas.

3.4.1 Principio de dualidad

En el álgebra de Boole se cumple el *Principio de Dualidad*. Dicho principio permite aplicar el operador de complemento a los dos términos de una igualdad —obtenida a partir de una propiedad o teorema— para llegar a una expresión dual que es tan válida como la igualdad original.

La aplicación de la negación en ambos términos de una igualdad algebraica permite obtener expresiones duales mediante: intercambio de las operaciones de producto y suma, negación de variables e intercambio de las constantes 0 y 1. Desde un punto de vista práctico la aplicación de este principio permite deducir propiedades y teoremas a partir de otros.

Ejemplo 3.2 — Aplicación del principio de dualidad. Aplica el principio de dualidad a la igualdad: $A+1=1$.

Solución:

Realizando la negación en ambos términos de la igualdad, se obtiene:

$$(A+1)' = 1' \Rightarrow A \cdot 0 = 0$$

Esta nueva igualdad también es válida. Observa que se podría pensar que en esta segunda igualdad A debería estar negada, pero al tratarse de una variable muda (el resultado no depende de su valor), se puede escribir A sin alterar la validez de la igualdad. ■

3.4.2 Axiomas, propiedades, leyes y teoremas del álgebra de Boole

Los axiomas, propiedades y leyes del álgebra de Boole (ver tablas 3.16 y 3.17) son enunciados válidos que no es preciso demostrar. Es sencillo relacionar dichos enunciados con las propiedades de las puertas lógicas que se exponen en las secciones precedentes.

Tabla 3.16: Axiomas del álgebra de Boole.

	Axioma		Dual
A.1:	$B = 0$ si $B \neq 1$	A.1′:	$B = 1$ si $B \neq 0$
A.2:	$0' = 1$	A.2′:	$1' = 0$
A.3:	$0 \cdot 0 = 0$	A.3′:	$1 + 1 = 1$
A.4:	$1 \cdot 1 = 1$	A.4′:	$0 + 0 = 0$
A.5:	$0 \cdot 1 = 1 \cdot 0 = 0$	A.5′:	$1 + 0 = 0 + 1 = 1$

Tabla 3.17: Propiedades y leyes del álgebra de Boole.

	Expresión		Dual	Propiedad/Ley
P.1:	$A \cdot B = B \cdot A$	P.1′:	$A + B = B + A$	*Propiedad conmutativa*
P.2:	$A \cdot (B + C) = A \cdot B + A \cdot C$	P.2′:	$A + B \cdot C = (A + B) \cdot (A + C)$	*Propiedad distributiva*
P.3:	$(A \cdot B)' = A' + B'$	P.3′:	$(A + B)' = A' \cdot B'$	*Leyes de De Morgan*

Se podría pensar erróneamente que la propiedad asociativa debería estar incluida en la tabla 3.17, ya que tanto el producto lógico como la suma satisfacen dicha propiedad. Sin embargo, no es un requisito del álgebra de Boole. Además, las puertas lógicas NAND y NOR no satisfacen dicha propiedad.

A diferencia de los axiomas, propiedades y leyes, los teoremas son enunciados derivados de los anteriores que admiten una demostración. En la tabla 3.18 aparece el listado de los teoremas del álgebra de Boole con su expresión algebraica correspondiente y la denominación con la que son conocidos.

Tabla 3.18: Teoremas del álgebra de Boole.

	Expresión		Dual	Teorema
T.1:	$A \cdot 1 = A$	T.1′:	$A + 0 = A$	*Elemento identidad*
T.2:	$A \cdot 0 = 0$	T.2′:	$A + 1 = 1$	*Elemento nulo*
T.3:	$A \cdot A = A$	T.3′:	$A + A = A$	*Idempotencia*
T.4:	$A'' = A$			*Involución*
T.5:	$A \cdot A' = 0$	T.5′:	$A + A' = 1$	*Complemento*
T.6:	$A + A \cdot B = A$	T.6′:	$A \cdot (A + B) = A$	*Absorción (I)*
T.7:	$A + A' \cdot B = A + B$	T.7′:	$A \cdot (A' + B) = A \cdot B$	*Absorción (II)*
T.8:	$A \cdot B + A \cdot B' = A$	T.8′:	$(A + B) \cdot (A + B') = A$	*Consenso*

En la tabla anterior se indica para cada teorema su dual. Por aplicación del principio de dualidad, basta con validar uno de ellos para asumir como cierto el dual.

> Es interesante recordar los teoremas del álgebra de Boole para simplificar las expresiones lógicas, mediante su aplicación. Ten en cuenta, que los teoremas también se pueden demostrar por prueba exhaustiva comprobando su validez para todos los valores posibles de las variables presentes en las expresiones lógicas.

Ejemplo 3.3 — Demostración de teoremas. Demuestra la validez del teorema de consenso (T8): $AB + AB' = A$.

Solución:

Aplicando la propiedad distributiva (P2) seguida del teorema del complemento para la suma (T.5′) se tiene:

$$AB + AB' = A\overbrace{(B + B')}^{=1} = A$$

■

■ **Ejercicio 3.15** Comprueba la validez de los axiomas y teoremas del álgebra de Boole mediante circuitos realizados con Logisim. ■

3.5 Expresiones algebraicas equivalentes de una función lógica

La definición 3.1 describe una función lógica como la aplicada por un circuito electrónico a las señales de entrada al mismo para producir una señal de salida. Aquí se ofrece una definición puramente matemática.

Definición 3.13 — Función lógica. Una *función lógica* expresa matemáticamente la relación entre variables lógicas, de modo que a cada combinación de dichas variables corresponde un valor lógico de la función. Toda función lógica es también una variable lógica. ■

Ejemplo 3.4 — Expresión de funciones lógicas. La siguiente expresión booleana define una función lógica f dependiente de tres variables lógicas A, B y C. Si las variables independientes se omiten en la declaración de la función, se entiende que dichas variables son las que aparecen en la definición de su expresión.

$$f(A,B,C) = AC + B'(A + C)' \Longleftrightarrow f = AC + B'(A + C)'$$

■

Una pregunta que surge en este contexto es si cualquier función lógica admite una única expresión algebraica dependiente de sus variables. Para contestar esta pregunta se introduce una nueva definición.

Definición 3.14 — Expresiones y funciones lógicas equivalentes. Se dice que dos expresiones lógicas son equivalentes, si ambas tienen el mismo valor lógico. Es decir, si en ambas expresiones se obtiene el mismo valor lógico asignando a sus variables idéntico valor lógico. Si dos funciones tienen expresiones lógicas equivalentes, en realidad se trata de la misma función. Las expresiones equivalentes de una función lógica son formas alternativas de expresar dicha función. ■

La determinación de equivalencia entre dos funciones se puede abordar siguiendo dos enfoques: transformado una expresión en la otra o comparando las tablas de verdad de ambas expresiones. Si a priori se sabe que las expresiones son equivalentes, es posible buscar el modo de transformar una en otra mediante la aplicación de propiedades y teoremas del álgebra de Boole. Sin embargo, este método no es apropiado si se desea probar la no equivalencia. Por el contrario, la comparación de las tablas de verdad asociadas a ambas expresiones permite demostrar si estamos ante una única función o dos funciones distintas.

> Recuerda que siempre es posible transformar una expresión lógica en otra equivalente mediante aplicación de las propiedades y teoremas del álgebra de Boole, de modo que no se altere el valor lógico de la expresión resultante (p. ej., sumando una expresión de valor lógico 0 o multiplicando por una expresión de valor lógico 1).

Ejemplo 3.5 — Determinación de equivalencia entre expresiones lógicas. Dadas las funciones lógicas f y g, demuestra que f y g son equivalentes:

$$f(A,B,C) = A + BC; \quad g(A,B,C) = AB + AB' + BC$$

Solución:

Si se aplica el teorema del consenso (T8) a la expresión de g se tiene que:

$$g = \overbrace{AB + AB'}^{=A} + BC = A + BC = f$$

De modo alternativo, si se calcula la tabla de verdad para f y g, se comprueba que para cada combinación de las variables independientes se obtiene idéntico valor para f y g:

A	B	C	f	g
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	1	1
1	0	0	1	1
1	0	1	1	1
1	1	0	1	1
1	1	1	1	1

Si se recuerda que en virtud del teorema del elemento identidad cualquier término producto permanece inalterado al multiplicarlo por una expresión cuyo valor sea 1 (T1) y que una suma no cambia al sumarle 0 (T1′) se puede llegar al siguiente resultado.

Corolario 3.5.1 — Número de expresiones equivalente de una función lógica. Toda función lógica posee un número infinito de expresiones algebraicas equivalentes.

Si se aplica este resultado a los circuitos de puertas lógicas se puede enunciar la conclusión siguiente.

Corolario 3.5.2 — Número de circuitos lógicos equivalentes. Dado un circuito de puertas lógicas, es posible construir un número infinito de circuitos equivalentes al mismo.

Con los resultados anteriores en mente se plantea la pregunta:

> *Entre todas las expresiones equivalentes que pueden definir a una función lógica dada: ¿existe alguna expresión de especial interés?*

La respuesta inmediata a la pregunta planteada es: *«aquella expresión que se pueda implementar por el circuito digital más simple.»* En las secciones siguientes se explicará cómo encontrar de modo sistemático la expresión equivalente más sencilla posible de una función lógica dada.

3.5.1 Expresiones SOP y POS de una función lógica

Dado que las operaciones lógicas básicas del álgebra de Boole son la suma y el producto, cualquier función lógica se puede expresar de dos modos alternativos:

- Forma SOP o *suma de productos* (*Sum Of Products*)[7] y
- Forma POS o *producto de sumas* (*Product Of Sums*).[8]

[7] Conocida también como *forma disyuntiva*.
[8] Denominada también como *forma conjuntiva*.

Ejemplo 3.6 — Expresiones en forma SOP y POS. A continuación, se presentan varios ejemplos de formas SOP y POS.

- Formas SOP (con términos producto señalados):

$$\underbrace{AB}_{p_1}+\underbrace{CD}_{p_2};\quad \underbrace{CA'D}_{p_1}+\underbrace{BC}_{p_2};\quad \underbrace{A}_{p_1}+\underbrace{B}_{p_2}+\underbrace{C}_{p_3}$$

  La tercera expresión es una suma de tres términos producto que poseen un solo factor cada uno.
- Formas POS (con términos suma señalados):

$$\underbrace{(A+B)}_{s_1}\underbrace{(C+D)}_{s_2};\quad \underbrace{(C+A'+D)}_{s_1}\underbrace{(B'+C)}_{s_2};\quad \underbrace{A+B+C}_{s_1}$$

Observa que la expresión $A+B+C$ se puede interpretar como una forma POS de un único término suma o una forma SOP con tres términos producto con un solo factor en cada uno. ■

Si la expresión dada de una función no está en forma SOP o POS, basta con aplicar la propiedad distributiva y las leyes de De Morgan para obtener una forma SOP o POS.

Ejemplo 3.7 — Obtención de forma SOP. Dada la función f, modifícala para expresarla en forma SOP:

$$f(A,B,C)=AC+B'(A+C)'$$

Solución:

Para pasar una expresión lógica a forma SOP y POS basta con aplicar las propiedades distributivas y las leyes de De Morgan. Para la expresión anterior, por simple aplicación de ley de De Morgan para la negación de una suma (P3′) se obtiene:

$$(A+C)'=A'C' \quad \text{por tanto} \quad f(A,B,C)=AC+B'A'C' \quad \text{(forma SOP)}$$

■

Una expresión SOP es una suma lógica en la que se cumple el teorema del elemento nulo para la suma (T2′). Es decir, una expresión SOP vale 1 si alguno de los términos de la suma lógica vale 1. Como cada término de la suma es un producto (AND), dicho término solo puede ser 1 si todos los factores del producto valen 1 (revisar definición de operación AND).

Ejemplo 3.8 — Condición para que un término producto valga 1. Indica cuál debe ser el valor de las variables lógicas A, B y C para que el producto $AB'C$ sea 1.

Solución:

$$AB'C=1 \leftrightarrow A=1,\ B=0,\ C=1$$

El producto es 1 cuando todos los factores son 1. Por esta razón las variables que están negadas deben valer 0. ■

De modo dual, una expresión POS es un producto lógico en el que se cumple el teorema del elemento nulo (T2). Es decir, una expresión POS vale 0 si alguno de los términos del producto vale 0. Como cada término del producto es una suma (OR), dicho término solo puede ser 0 si todas las variables del mismo valen 0 (revisar definición de operación OR).

Ejemplo 3.9 — Condición para que un término suma valga 0. Indica cuál debe ser el valor de las variables lógicas A, B y C para que la suma $A+B'+C$ sea 0.

Solución:

$$A + B' + C = 0 \leftrightarrow A = 0,\ B = 1,\ C = 0$$

La suma es 0 cuando todos los sumandos son 0. Por esta razón las variables que están negadas deben valer 1. ■

3.6 Formas canónicas de una función lógica

Entre las posibles formas SOP y POS equivalentes de una función lógica, las dos más importantes se denominan *formas canónicas* o *normales* de la función.

Definición 3.15 — Expresión canónica o normal de una función lógica. Una *forma canónica* o *normal* de una función lógica es aquella constituida exclusivamente por términos producto —en el caso de la forma SOP— o suma —en el caso de la forma POS— en los que aparecen todas las variables de la función. Los términos suma o producto de una forma canónica se denominan *términos canónicos*. ■

Ejemplo 3.10 — Condición para que una expresión lógica sea canónica. Dadas las funciones lógicas f, g, y h dependientes de A, B y C. Determina si sus expresiones están en forma canónica.

- $f(A,B,C) = ABC + ABC'$
- $g(A,B,C) = A + BC$
- $h(A,B,C) = A + B' + C$

Solución:

- $f(A,B,C) = ABC + ABC'$, **SI es una forma canónica** SOP con dos productos canónicos. Los dos términos producto son canónicos porque contienen las tres variables de la función.
- $g(A,B,C) = A + BC$, esta en forma SOP, pero **NO es una forma canónica** porque en el primer término producto no aparecen las variables B y C, mientras que en el segundo no aparece la variable A.
- $h(A,B,C) = A + B' + C$, *¡cuidado!* aunque se puede interpretar como suma de tres productos no canónicos, también se puede ver como una forma POS con un único término suma con las tres variables. Por tanto, **SI es una forma canónica** POS con un único término suma.

■

Para determinar si la expresión de una función lógica está en forma normal o canónica, primero se identifica si se trata de una forma SOP o POS, y a continuación si cada uno de los términos es un término canónico —con todas las variables de la función—.

Definición 3.16 — Minitérminos y maxitérminos de una forma canónica. A los productos canónicos de una forma canónica SOP se les denomina *minitérminos*. A los términos suma de una forma canónica POS se les denomina *maxitérminos*. ■

Por tanto, se puede afirmar que la forma canónica SOP es una suma de minitérminos y la forma canónica POS es un producto de maxitérminos.

Corolario 3.6.1 — Unicidad de las formas canónicas. Toda función lógica se puede expresar mediante una forma canónica disyuntiva SOP y una forma canónica conjuntiva POS que son únicas.

Corolario 3.6.2 — Equivalencia de expresiones lógicas. Dos expresiones lógicas son equivalentes si sus formas canónicas coinciden.

Con base en los colorarios anteriores se puede asegurar que cualquier función lógica queda definida de forma unívoca mediante sus formas canónicas. Más adelante comprobaremos que basta con determinar una de ellas para que la otra quede perfectamente determinada.

Definición 3.17 — Primera forma canónica (SOP) de una función lógica. La primera forma canónica (SOP) o disyuntiva de una función lógica es aquella que tiene valor lógico 1 cuando cualquiera de sus productos canónicos (minitérminos) vale 1. ■

Definición 3.18 — Segunda forma canónica (SOP) de una función lógica. La segunda forma canónica (POS) o conjuntiva de una función lógica es aquella que tiene valor lógico 0 cuando cualquiera de sus sumas canónicas (maxitérminos) vale 0. ■

Ejemplo 3.11 — Primera y segunda forma canónica de una función. Dada la función f por sus dos formas canónicas, comprueba que ambas expresiones son equivalentes y, por tanto, definen la misma función.

- 1ª FC: $f(A,B,C) = A'B'C' + AB'C + ABC$
- 2ª FC: $f(A,B,C) = (A+B+C')(A+B'+C)(A+B'+C')(A'+B+C)(A'+B'+C)$

Solución:

Como ya se ha indicado anteriormente, para comprobar si dos expresiones lógicas son equivalentes, se recurre a comparar sus tablas de verdad asociadas. También es posible verificar para cada forma canónica cuál es la combinación de valores de las variables para que la 1ª FC valga 1 y de modo similar las que obtienen un valor 0 para la 2ª FC.

Una vez obtenida la tabla de verdad indicada más abajo, se puede comprobar que los valores de las variables que hacen $f = 1$ se corresponden con los que también producen un 1 para los términos de la 1ª FC de f y los que hacen $f = 0$, son los que producen un 0 para los términos de su 2ª FC.

Id.	A	B	C	f	
0	0	0	0	1	$\rightarrow f(0,0,0)$
1	0	0	1	0	$\rightarrow f(0,0,1)$
2	0	1	0	0	$\rightarrow f(0,1,0)$
3	0	1	1	0	$\rightarrow f(0,1,1)$
4	1	0	0	0	$\rightarrow f(1,0,0)$
5	1	0	1	1	$\rightarrow f(1,0,1)$
6	1	1	0	0	$\rightarrow f(1,1,0)$
7	1	1	1	1	$\rightarrow f(1,1,1)$

Por tanto, ambas expresiones corresponden a la misma función. ■

El número de términos canónicos (productos o sumas) depende del número de variables disponibles, ya que las variables pueden aparecer negadas o sin negar.

Corolario 3.6.3 — Número de términos canónicos. Dada una función lógica de n variables, el número de posibles términos canónicos (productos o minitérminos y sumas o maxitérminos) de la misma, se calcula mediante el número 2^n de combinaciones posibles de las variables negadas y sin negar.

Ejemplo 3.12 — Número de términos canónicos posible de una función. ¿Cuál es el número posible de términos canónicos de una función $f(A,B)$?

Solución:

El número posible de productos (minitérminos) y sumas canónicas (maxitérminos) de la función f es $2^2 = 4$. Si construimos dichos términos, se tiene:

- Minitérminos $\rightarrow A'B'$, $A'B$, AB', AB
- Maxitérminos $\rightarrow (A+B)$, $(A+B')$, $(A'+B)$, $(A'+B')$

Las combinaciones posibles de valores de las variables A y B $\{00,01,10,11\}$ son las que hace que los minitérminos valgan 1 y los maxitérminos 0. ■

3.6.1 Tabla de verdad y formas canónicas

La tabla de verdad (TV) de una función lógica expresa el valor de la función para cada una de las combinaciones de las variables de las que depende dicha función. En la TV los valores posibles de las variables se escriben, de arriba a abajo en la tabla, en orden creciente de su valor en binario natural (ver tabla 3.19).

Tabla 3.19: Términos canónicos asociados a una función genérica $f(A,B,C)$ de tres variables.

Id.	A	B	C	*minitérmino*	*maxitérmino*
0	0	0	0	$m_0 = A'B'C'$	$M_0 = A+B+C$
1	0	0	1	$m_1 = A'B'C$	$M_1 = A+B+C'$
2	0	1	0	$m_2 = A'B\,C'$	$M_2 = A+B'+C$
3	0	1	1	$m_3 = A'B\,C$	$M_3 = A+B'+C'$
4	1	0	0	$m_4 = A\,B'C'$	$M_4 = A'+B+C$
5	1	0	1	$m_5 = A\,B'C$	$M_5 = A'+B+C'$
6	1	1	0	$m_6 = A\,B\,C'$	$M_6 = A'+B'+C$
7	1	1	1	$m_7 = A\,B\,C$	$M_7 = A'+B'+C'$

Si se consideran las posibles combinaciones de valores de las variables presentes en la TV de una función, se puede asociar a cada una de dichas combinaciones un minitérmino (producto canónico) y un maxitérmino (suma canónica). El minitérmino asociado sería aquel que vale 1 para la combinación de los valores de las variables y el maxitérmino aquel que vale 0. La tabla 3.19 muestra los posibles términos canónicos (minitérminos y maxitéminos) asociados a una función lógica genérica $f(A,B,C)$ dependiente de tres variables.

Ejemplo 3.13 — Obtención de tabla de verdad desde expresión SOP. Dada la función $f(A,B,C) = AC + A'B'C'$ por su forma SOP, calcula su tabla de verdad.

Solución:

Es posible construir la TV teniendo en cuenta que $f = 1$ si alguno de los términos suma de la expresión SOP vale 1. Esto solo se produce en las combinaciones donde $A = C = 1$ o $A = B = C = 0$. Así, en la tabla de verdad a las combinaciones que cumplen dichas condiciones se asocia el valor 1:

$$f(1,*,1) = 1, \quad f(0,0,0) = 1$$

al resto de combinaciones (filas) se asocia valor 0. De este modo se obtiene:

Id.	A	B	C	f	
0	0	0	0	1	←
1	0	0	1	0	
2	0	1	0	0	
3	0	1	1	0	
4	1	0	0	0	
5	1	0	1	1	←
6	1	1	0	0	
7	1	1	1	1	←

NOTA: Ten en cuenta que si las variables se toman en orden diferente, en la tabla resultante los unos y ceros de la función cambian de posición. ■

Definición 3.19 — Valor binario de un término canónico. El valor binario de un término canónico es el equivalente decimal (Id.) de la combinación de valores asociados a las variables que hacen valer 1 un minitérmino y 0 un maxitérmino. ■

Para obtener el valor binario de un término canónico, en los minitérminos se sustituyen las variables negadas por 0 y las no negadas por 1. En el caso de los maxitérminos, se sustituye las variables negadas por 1 y las no negadas por 0. Cada término canónico se identifica unívocamente por la letra m para minitérmino y M para maxitérmino, con un subíndice que indica el valor binario del mismo (ver tabla 3.19).

Ejemplo 3.14 — Cálculo de valor binario de términos canónicos. Dado el minitérmino $A'BC'$ y el maxitérmino $A'+B+C'$ de una función lógica de tres variables, indica el valor binario de los mismos y su identificación simplificada.

Solución:

El valor binario de un minitérmino se obtiene como la combinación binaria de los valores de las variables que hacen 1 el producto. En consecuencia, se tiene que:

$$A'BC' \rightarrow 010_{(2} = 2_{(10}$$

Por esta razón a este producto también se le denomina minitérmino 2 o simplemente m_2.

El valor binario de un maxitérmino se obtiene como la combinación binaria de los valores de las variables que hace 0 la suma. De este modo se tiene:

$$A'+B+C' \rightarrow 101_{(2} = 5_{(10}$$

A esta suma canónica se la denomina maxitérmino 5 o M_5.

En resumen: Para obtener el valor binario de un minitérmino se sustituye en cada posición la variable negada por 0 y la no negada por 1. En el caso de los maxitérminos el cambio es el opuesto. La variable negada se sustituye por 1 y la no negada por 0. ■

Es importante tener presente que el valor de los términos canónicos de una función depende del orden en que se tomen las variables, ya que dicho orden afecta a la ponderación o valor significativo de cada variable. Una vez elegido un orden, este se debe mantener para todos los términos canónicos de la función. De este modo cada término canónico se puede identificar unívocamente mediante su valor binario.

Ejemplo 3.15 — Identificación de términos canónicos por su valor binario. Dada una función $f(A,B,C,D)$, indica la expresión algebraica correspondiente a los términos canónicos m_7 y M_2.

Solución:

- $m_7 = A'BCD$, ya que $A=0,B=1,C=1,D=1 \Rightarrow \underbrace{A'BCD}_{0111\ (7)} = 1$.
- $M_2 = A+B+C'+D$, ya que $A=0,B=0,C=1,D=0 \Rightarrow \underbrace{A+B+C'+D}_{0010\ (2)} = 0$.

En resumen: Para obtener el producto de variables correspondiente a un minitérmino dado, se sustituye en cada posición del código binario del minitérmino, los 0 por la variable negada y los 1 por la variable sin negar. En el caso de los maxitérminos el cambio es el opuesto. El 1 se cambia por la variable negada y el 0 por la variable sin negar. ■

Corolario 3.6.4 — Equivalencia entre formas canónicas y tabla de verdad. Las formas canónicas de una función representan un modo alternativo de definir su TV. La 1ª FC (SOP) indica qué combinaciones de las variables hacen que la función valga 1. Es decir, cuáles son las filas de la tabla de verdad con valor de la función igual a 1. Por el contrario, la 2ª FC (POS) indica qué combinaciones de

variables hacen que la función valga 0. Es decir, cuáles son las filas de la tabla de verdad con valor de la función igual a 0. Basta con conocer una de las dos formas canónicas de la función para determinar la otra, ya que las filas de la TV que no son 1 serán 0 y viceversa.

Todo lo expuesto permite resumir la importancia de las formas canónicas en dos conclusiones:

1. Las formas canónicas son únicas para una función dada.
2. Cualquiera de las funciones lógicas dependientes de las mismas variables se obtiene por combinación de los términos canónicos posibles para dichas variables.

> Recuerda que existen dos funciones especiales denominada función identidad y nula cuyos valores son siempre 1 y 0, respectivamente. Son las únicas funciones que se expresan mediante una única forma canónica. La función identidad solo tendría minitérminos —todas las filas de la TV igual a 1— y la función nula sería la que solo tiene maxitérminos —todas las filas de la TV igual a 0—.

Se puede afirmar que los términos canónicos son los elementos constituyentes de cualquier función lógica dependiente de un número dado de variables. Por tanto, el número de funciones lógicas que se pueden definir con n variables no es infinito, sino que queda determinado por el número de combinaciones de los 2^n términos canónicos que pueden formar parte de la función. Teniendo en cuenta el enunciado del corolario 3.6.4, el número de funciones lógicas que se pueden describir con n variables está determinado por las combinaciones de unos y ceros en la tabla de verdad para el número de variables dado. Dicho número es $2^{(2^n)}$. Aunque dicho número no es infinito, se trata de un valor que crece mucho más rápido con n que el número de términos canónicos, según se indica en la tabla 3.20.

Tabla 3.20: Número de funciones lógicas posibles dependientes de n variables.

# variables	# m/M-términos	# funciones
$n = 2 \rightarrow$	$2^2 = 4$	$2^{(2^2)} = 16$
$n = 3 \rightarrow$	$2^3 = 8$	$2^{(2^3)} = 256$
$n = 4 \rightarrow$	$2^4 = 16$	$2^{(2^4)} = 65\,536$
$n = 5 \rightarrow$	$2^5 = 32$	$2^{(2^5)} = 4\,294\,967\,296$
$n = 6 \rightarrow$	$2^6 = 64$	$2^{(2^6)} = 18\,446\,744\,073\,709\,551\,616$

Teniendo en cuenta la relación entre las expresiones canónicas de toda función lógica f y su tabla de verdad, se tiene que:

- 1ª forma canónica (SOP, suma de minitérminos)

$$f = \sum m_i : i \in [0, 2^n - 1] \tag{3.1}$$

- 2ª forma canónica (POS, producto de maxitérminos)

$$f = \prod M_{j \neq i} : i, j \in [0, 2^n - 1] \tag{3.2}$$

En dichas expresiones los índices de minitérminos y maxitérminos son distintos para completar el número total de índices dado por $2^n - 1$, siendo n el número de variables de la función. Como en la TV de toda función las filas son 1 o 0, las primeras se asocian a sus minitérminos y las segundas a sus maxitérminos.

Ejemplo 3.16 — Expresión abreviada de una función lógica. Dada la función lógica f cuya tabla de verdad se muestra en el ejemplo 3.13, exprésala de modo abreviado mediante sus minitérminos y maxitérminos.

Solución:

Como los 1/0 en la TV corresponden a los mini/maxi-términos de la función, esta se puede expresar como:

- 1ª forma canónica (SOP):

$$\begin{aligned} f =& \underbrace{A'B'C'}_{000\ (0)} + \underbrace{AB'C}_{101\ (5)} + \underbrace{ABC}_{111\ (7)} \\ =& m_0 + m_5 + m_7 = \sum(m_0, m_5, m_7) \equiv \sum m(0,5,7) \end{aligned}$$

- 2ª forma canónica (POS):

$$\begin{aligned} f =& \underbrace{(A+B+C')}_{001\ (1)} \underbrace{(A+B'+C)}_{010\ (2)} \underbrace{(A+B'+C')}_{011\ (3)} \underbrace{(A'+B+C)}_{100\ (4)} \underbrace{(A'+B'+C)}_{110\ (6)} \\ =& M_1 \cdot M_2 \cdot M_3 \cdot M_4 \cdot M_6 = \prod(M_1, M_2, M_3, M_4, M_6) \equiv \prod M(1,2,3,4,6) \end{aligned}$$

Debajo de los términos canónicos se señala la combinación de variables que hacen que valga 1 o 0 dependiendo de que se trate de un producto o suma, respectivamente. Observa que de los posibles 8 términos canónicos (con índices de 0 a 7), los que no son minitérminos, son maxitérminos. ■

Basta con tener una de las expresiones canónicas para poder expresar la otra con los términos canónicos cuyos valores (subíndices) no aparecen en la expresión de partida.

Ejemplo 3.17 — Obtención de forma canónica complementaria. Dada la expresión canónica POS de la función lógica $f = \prod M(3,5,6)$, determina la forma canónica SOP de dicha función.

Solución:

La forma canónica SOP está formada por los minitérminos cuyos valores binarios (subíndices) están ausentes en la expresión POS. Por tanto, se tiene:

$$f = \sum m(0,1,2,4,7)$$

Observa que al sumar el número de mini/maxitérminos de f, el total debe ser 8. ■

Analizando la relación entre mini/maxi-términos de una función, se obtiene:

$$m_i' = M_i \Leftrightarrow m_i = M_i' \tag{3.3}$$

Corolario 3.6.5 — Negación de términos canónicos. La negación de un minitérmino es el maxitérmino del mismo valor binario. De modo dual, la negación de un maxitérmino es el minitérmino del mismo valor binario.

Corolario 3.6.6 — Formas canónicas de una función y su negada. Dada una función f en forma canónica, su función negada f' se obtiene intercambiando minitérminos y maxitérminos en las expresiones canónicas de f.

Ejemplo 3.18 — Formas canónicas de funciones complementarias. Dada la función f por sus expresiones canónicas, determina las formas canónicas de f':

$$f = \sum m(0,5,7) = \prod M(1,2,3,4,6)$$

Solución:

Intercambiando minitérminos y maxitérminos se tiene:

$$f' = \sum m(1,2,3,4,6) = \prod M(0,5,7)$$

■

3.6.2 Teorema de expansión de Shannon

En las secciones anteriores la obtención de las expresiones canónicas de una función lógica se ha aceptado sin demostración. Sin embargo, es posible llegar a una demostración de estas expresiones mediante la aplicación del *Teorema de expansión o descomposición de Shannon*, también conocido como *teorema de expansión de Boole*.

Teorema 3.1 Toda función lógica f se puede expresar como suma del producto de una de sus variables negada por el valor de la función cuando dicha variable es 0, más el producto de la variable sin negar por el valor de la función cuando dicha variable vale 1.

Aplicado a una función f de dos variables, se tiene:

$$f(A,B) = A'f(0,B) + Af(1,B) = B'f(A,0) + Bf(A,1) \tag{3.4}$$

Aplicando el teorema sobre A y después sobre B se obtiene la expresión canónica SOP de f como suma de sus productos canónicos (minitérminos):

$$\begin{aligned} f(A,B) &= A'f(0,B) + Af(1,B) \\ &= A'[B'f(0,0) + Bf(0,1)] + A[B'f(1,0) + Bf(1,1)] \\ &= A'B'f(0,0) + A'Bf(0,1) + AB'f(1,0) + ABf(1,1) \end{aligned}$$

En la expresión final se obtiene todos los productos canónicos posibles multiplicados por el valor de la función para los valores de las variables asociados a cada producto.

El teorema se puede demostrar de modo sencillo comprobando que la tabla de verdad para ambos miembros de la igualdad son coincidentes. De este modo se tiene:

A	B	f	$A'B'f(0,0) + A'Bf(0,1) + AB'f(1,0) + ABf(1,1)$
0	0	$f(0,0)$	$f(0,0)$
0	1	$f(0,1)$	$f(0,1)$
1	0	$f(1,0)$	$f(1,0)$
1	1	$f(1,1)$	$f(1,1)$

Al aplicar recursivamente el teorema de Shannon a una función de varias variables, se obtiene una expresión en la que cada término canónico aparece multiplicado por el valor de la función para la combinación de valores de las variables de entrada asociados al término canónico. Los valores que se mantienen son aquellos para los cuales la función vale 1. De este modo se obtiene la expresión de la función original como suma de minitérminos. Aplicando dualidad se deriva la expresión de la segunda forma canónica de la función.

Corolario 3.6.7 — Aplicación del teorema de expansión de Shannon. El teorema de expansión de Shannon proporciona un método para expresar cualquier función lógica como una suma de productos canónicos de cualesquiera variables de las que dependa la función. Cada producto canónico irá afectado por un coeficiente dado por la evaluación de la función para la combinación de valores de las variables asociado al producto canónico.

Este resultado será relevante para poder expresar funciones lógicas con módulos combinacionales generadores de productos canónicos como multiplexores y decodificadores.

3.6.3 Obtención de formas canónicas

En las secciones previas se ha expuesto la relación estrecha entre la tabla de verdad y las formas canónicas de una función lógica. De hecho se pueden contemplar como alternativas para definir cualquier función lógica, ya que el paso de una a otra es inmediato. Por tanto, obtener las formas canónicas es equivalente a obtener la tabla de verdad de la función o los índices de los minitérminos y maxitérminos de la función.

El método para obtener las formas canónicas de una función a partir de su expresión algebraica se resume en los pasos siguientes:

1. Expresar la función en forma SOP o POS.
2. Expandir los términos no canónicos. Si alguno de los términos producto de la expresión SOP —o suma en el caso de la forma POS— no es canónico, este se expande en 2^m términos, siendo m el número de variables ausentes en el término canónico. Cada variable ausente produce dos términos canónicos uno con la variable negada y otro con la variable sin negar.
3. Eliminar los términos canónicos repetidos al expandir los términos no canónicos en el paso anterior.

Ejemplo 3.19 — Expansión de forma SOP no canónica a forma canónica. Dada la función $f(A,B,C) = AB + A'C$, determina sus formas canónicas.

Solución:

Como f depende de A, B y C, la expresión dada no es canónica aunque aparece en forma SOP. Para obtener la 1ª FC (SOP) habría que expandir los términos no canónicos. Dicha expansión se puede hacer a partir de los valores binarios asociados a cada producto canónico (la variable ausente se sustituye por un comodín '$*$') que puede adoptar los valores '0' y '1':

$$AB* \to 11* \Rightarrow 110\ (6),\ 111\ (7) \to ABC',\ ABC$$
$$A'*C \to 0*1 \Rightarrow 001\ (1), 011\ (3) \to A'B'C,\ A'BC$$

Por tanto, la 1ª FC (SOP) que se obtiene es:

$$f(A,B,C) = \underbrace{ABC'}_{110\ (6)} + \underbrace{ABC}_{111\ (7)} + \underbrace{A'B'C}_{001\ (1)} + \underbrace{A'BC}_{011\ (3)} = m_6 + m_7 + m_1 + m_3 = \sum m(1,3,6,7)$$

La 2ª FC (POS) se obtiene como producto de maxitérminos cuyos índices no aparecen en la suma de minitérminos:

$$f = \prod M(0,2,4,5)$$

■

> Observa que en este texto se emplean varios recursos no estandarizados para facilitar la comprensión de las explicaciones, como: colores, variable ausente sustituida por un comodín '$*$', llaves inferiores para señalar el valor binario de términos canónicos y llaves superiores para indicar valores equivalentes de expresiones booleanas.

Las conclusiones principales para recordar respecto de las funciones lógicas y su representación canónica son:

- ✓ La tabla de verdad se puede interpretar como una representación de los términos canónicos de la función lógica dada, ya que cada fila de la tabla de verdad tiene asociado un producto canónico y una suma canónica.
- ✓ La forma SOP o 1ª forma canónica de una función lógica se expresa mediante la suma de los productos canónicos asociados a las filas de la tabla de verdad en que la función vale 1.
- ✓ La forma POS o 2ª forma canónica de una función lógica se expresa mediante el producto de las sumas canónicas asociadas a las filas de la tabla de verdad en que la función vale 0.
- ✓ La representación canónica de una función lógica es única en cualquiera de sus dos formas SOP (1ª FC) o POS (2ª FC), ya que en cada fila de la tabla de verdad la función vale 1 o 0. Por tanto, el término canónico correspondiente solo puede ser de un tipo.

3.7 Simplificación de funciones lógicas

Si las formas canónicas de una función tienen distinto número de términos, aquella con menor número será considerada la más sencilla. A partir de la tabla de verdad de la función, es fácil determinar cuál de las dos formas canónicas en la más simple pues dependerá de la cantidad de unos y ceros presentes en la TV.

Como ya se ha visto (ver ejemplo 3.19), hay expresiones algebraicas de una función lógica más sencillas que las formas canónicas. De un modo intuitivo, estas expresiones se obtienen por un proceso de simplificación inverso al de expansión de términos no canónicos. En este proceso se persigue encontrar aquellas variables comodín que es posible eliminar en los términos de la expresión canónica de la función. Dicha simplificación se deriva del teorema de consenso del álgebra de Boole (T8 y T8′) que permite eliminar una variable que aparece negada y sin negar en dos términos canónicos con el resto de variables iguales.

La simplificación de una función lógica persigue la obtención de sus expresiones SOP y POS equivalentes más sencillas, que no coinciden habitualmente con las expresiones canónicas de dicha función. La simplificación mencionada se puede conseguir mediante dos métodos:

1. *Simplificación algebraica*. Se basa en la manipulación de las expresiones algebraicas aplicando los teoremas del álgebra de Boole. Es un método rápido en los casos de expresiones sencillas, pero inviable en el caso de expresiones complejas. No es sistemático y su éxito depende de la destreza de quien lo aplica.
2. *Simplificación algorítmica*. Emplea un algoritmo sistemático que permite su automatización. Se aplica a expresiones de complejidad arbitraria e incluso facilita la simplificación simultánea de varias funciones.

Ejemplo 3.20 — Simplificación algebraica de funciones lógicas. Simplifica la función lógica:

$$f(A,B,C,D) = ((A+B)'(CD)')'$$

Solución:

Aplicando las leyes de De Morgan (P3) y el teorema de involución (T4) se tiene:

$$\begin{aligned} f(A,B,C,D) &= ((A+B)'(CD)')' \\ &= (A+B)'' + (CD)'' \quad \leftarrow (\text{P3}) \\ &= A+B+CD \quad \leftarrow (\text{T4}) \end{aligned}$$

■

Ejemplo 3.21 — Simplificación algebraica de funciones lógicas. Simplifica la función lógica:

$$f(A,B,C,D) = (((A+B)'C)' + ((CD)'+B)')'$$

Solución:

Aplicando las leyes de De Morgan y los teoremas del álgebra de Boole se tiene:

$$\begin{aligned} f(A,B,C,D) &= (((A+B)'C)' + ((CD)'+B)')' \\ &= ((A+B)'C)''(CD)'+B)'' \quad \leftarrow (\text{P3}') \\ &= (A+B)'C((CD)'+B) \quad \leftarrow (\text{T4}) \\ &= A'B'C(C'+D'+B)\overbrace{A'B'CC'}^{=0} + A'B'CD' + \overbrace{AB'CB}^{=0} \quad \leftarrow (\text{T5}) \\ &= A'B'CD' \end{aligned}$$

■

Ejemplo 3.22 — Simplificación algebraica de funciones lógicas. Simplifica la función lógica:

$$f = (A' + B)' + BC' + ((A' + B)'(C + D))'$$

Solución:

Aplicando las leyes de De Morgan y los teoremas del álgebra de Boole se tiene:

$$\begin{aligned}
f &= (A' + B)' + BC' + ((A' + B)'(C + D))' \\
&= \underbrace{AB'}_{\sim\sim} + BC' + A' + \underbrace{B}_{\sim} + C'D' \quad \leftarrow (\text{P3}', \text{T4}) \\
&= \overbrace{AB' + B}^{=A+B} + BC' + A' + C'D' \quad \leftarrow (\text{T7}) \\
&= \underbrace{A}_{\sim} + B + BC' + \underbrace{A'}_{\sim} + C'D' \quad \leftarrow (\text{P1}') \\
&= \overbrace{A + A'}^{=1} + B + BC'C'D' \quad \leftarrow (\text{T5}') \\
&= 1
\end{aligned}$$

■

3.7.1 Mapas de Karnaugh-Veitch

Para sistematizar la simplificación de las funciones lógicas, Maurice Karnaugh —un físico y matemático norteamericano— publicó[9] en 1953 un método basado en unos diagramas denominados en su honor *mapas de Karnaugh* (*K-maps*).[10] Los K-maps se presentaron como un método refinado frente a los diagramas que Edward Veitch había publicado en 1952 retomando los diagramas lógicos que Allan Marquand había propuesto en 1881, pero en este caso con especial énfasis en los circuitos lógicos construidos con elementos de conmutación.

Los K-maps proporcionan un método de reordenación de la tabla de verdad de una función lógica para facilitar la identificación sistemática de los términos canónicos que se pueden simplificar en la función. Los K-maps se emplean con funciones de hasta 6 variables aunque en esta obra los ejemplos más complejos que se muestran son de 5 variables.

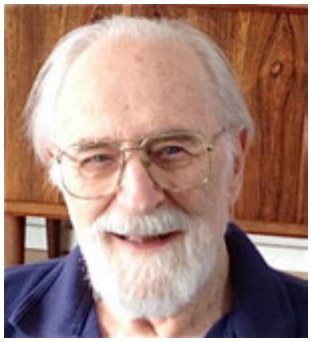


Figura 3.21: Maurice Karnaugh (1924-). Fuente: IT History Society (ithistory.org).

Construcción de K-maps

Un K-map es una reordenación de la tabla de verdad de una función lógica dada, de modo que a cada celda del K-map se asocian dos valores: el valor binario de las variables y el valor correspondiente de la función. La ordenación de celdas en el K-map persigue que entre celdas adyacentes solo cambie el valor de una de las variables asociadas a dichas celdas.

[9] *The map method for synthesis of combinational logic circuits*. Transactions of the American Institute of Electrical Engineers, Part I: Communication and Electronics. 72 (5):593–599, Nov. 1953. DOI: 10.1109/TCE.1953.6371932.

[10] También conocidos como *mapas de Karnaugh-Veitch*.

Dada una celda del K-map, el cambio de valor de una única variable nos lleva en horizontal o vertical, a una celda adyacente a la celda de partida. En los bordes del K-map las celdas adyacentes se corresponde con las del borde opuesto.

La figura 3.22 muestra la estructura de los K-maps para 2, 3 y 4 variables. En cada celda se incluye (en color rojo) el índice del mini/maxi-término asociado —suprimido habitualmente en la práctica—. En cada K-map se indica mediante flechas las celdas adyacentes a las coloreadas (en rojo y verde). En los K-maps las combinaciones de cada pareja de variables en filas y columnas siguen el código Gray (00, 01, 11 y 10) en lugar de la ordenación en binario natural. De este modo se asegura la adyacencia, en filas y columnas, de los códigos asociados a las celdas.

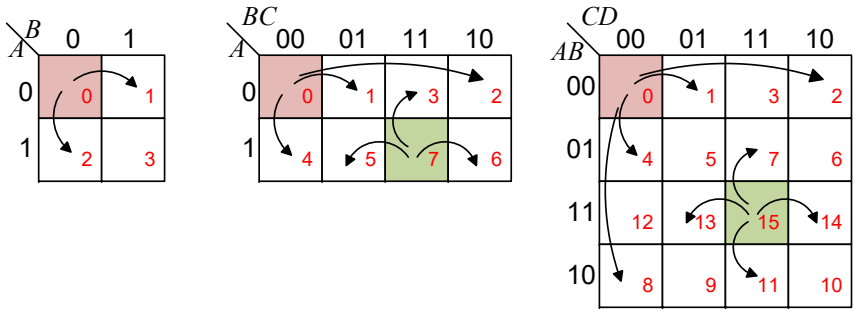


Figura 3.22: K-maps de 2, 3 y 4 variables mostrando el índice de la combinación de valores de las variables asociado a la celda y la adyacencia entre celdas (señaladas con flechas).

> Los K-maps se pueden construir colocando las variables en un orden arbitrario, pero con la condición de respetar la adyacencia de celdas en el mismo. En este texto se utilizará siempre la ordenación de variables que se muestra en la figura 3.22. Esta ordenación coincide con la ordenación empleada en Logisim y en la publicación original de M. Karnaugh. Sin embargo, en otros libros de texto se pueden encontrar ordenaciones alternativas e incluso la sustitución de los valores de las variables por líneas que señalan las celdas con valor 1 para cada variable.

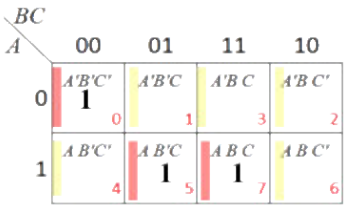
La obtención del los K-map de una función lógica es equivalente a obtener las expresiones canónicas de la función. En el K-map se marcarán con un 1 las celdas asociadas a los minitérminos de la función. El resto de celdas estará asociado a los maxitérminos o valor 0 de la función.

Ejemplo 3.23 — Obtención de K-map a partir de tabla de verdad. Dada la tabla de verdad de la función lógica $f(A,B,C)$, determina el K-map para dicha función.

Id.	A	B	C	f
0	0	0	0	1
1	0	0	1	0
2	0	1	0	0
3	0	1	1	0
4	1	0	0	0
5	1	0	1	1
6	1	1	0	0
7	1	1	1	1

Solución:

En la tabla de verdad de f se han identificado en color rosado las filas asociadas a minitérminos y en amarillo las asociadas a maxitérminos. El K-map se obtiene señalando los minitérminos en la reordenación que ofrece el K-map.



Ejemplo 3.24 — Obtención de K-map a partir de la forma canónica. Dada la forma canónica SOP de la función lógica $f(A,B,C)$, determina el K-map para dicha función:

$$f = A'B'C' + AB'C' + ABC' + ABC$$

Solución:

Recuerda que en un producto canónico las variables negadas se sustituyen por 0 y las no negadas por 1. Si se calculan los valores binarios o índices de los minitérminos (productos canónicos) de f, se tiene:

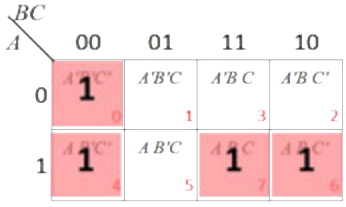
$$f = \underbrace{A'B'C'}_{000\ (0)} + \underbrace{AB'C'}_{100\ (4)} + \underbrace{ABC'}_{110\ (6)} + \underbrace{ABC}_{111\ (7)}$$

Por tanto, expresando f de modo abreviado queda: $f = \sum m(0,4,6,7)$

A continuación se muestra la tabla de verdad de f señalando sus minitérminos (rosado) y maxitérminos (amarillo):

Id.	A	B	C	f
0	0	0	0	1
1	0	0	1	0
2	0	1	0	0
3	0	1	1	0
4	1	0	0	1
5	1	0	1	0
6	1	1	0	1
7	1	1	1	1

A partir de la tabla de verdad se marcan los minitérminos en el K-map.



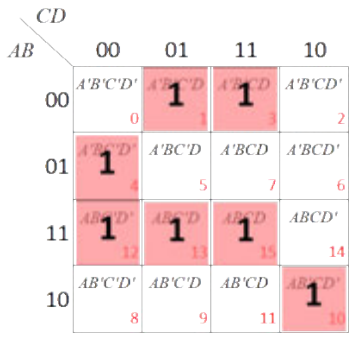
Ejemplo 3.25 — Obtención de K-map a partir de forma canónica. Dada la forma canónica SOP de la función lógica $f(A,B,C,D)$, determina el K-map para dicha función.

$$f = A'B'CD + A'BC'D' + ABC'D + ABCD + ABC'D' + A'B'C'D + AB'CD'$$

Solución:

Calculando el valor binario de cada término y pasándolo al K-map se tiene:

$$f = \underbrace{A'B'CD}_{0011\ (3)} + \underbrace{A'BC'D'}_{0100\ (4)} + \underbrace{ABC'D}_{1101\ (13)} + \underbrace{ABCD}_{1111\ (15)} + \underbrace{ABC'D'}_{1100\ (12)} + \underbrace{A'B'C'D}_{0001\ (1)} + \underbrace{AB'CD'}_{1010\ (10)}$$
$$= \sum m(1,3,4,10,12,13,15)$$



Aunque una función no se exprese en forma canónica, es posible completar el K-map asociado siguiendo alguno de los dos métodos que se indican a continuación:

1. Expansión canónica de cada término no canónico (ver sección 3.6.3 y ejemplos). Para la expansión de cada término producto en una expresión SOP, se obtiene el valor binario de cada término sustituyendo las variables negadas por 0 y las no negadas por 1. Las variables ausentes son sustituidas por el símbolo comodín '$*$'. Los valores comodín se sustituyen por todas las combinaciones posibles de 0 y 1 para obtener todos los términos canónicos posibles. En las formas POS el proceso es dual, teniendo en cuenta que para los términos suma, las variables negadas se sustituyen por 1 y las no negadas por 0.
2. Marcado de las celdas que satisfacen el criterio contemplado en cada término de la expresión lógica. Cada término con n variables ausentes (comodines) permitirá marcar 2^n celdas, teniendo en cuenta que algunas celdas pueden ser marcadas de modo redundante por la expansión de varios términos no canónicos. En realidad este método es equivalente al anterior pero haciendo la expansión en un único paso. Para aplicar este método es de gran ayuda marcar con una línea, en el exterior del K-map, las celdas correspondientes al valor 1 para cada variable.

Ejemplo 3.26 — Obtención de K-map a partir de expresiones no canónicas. Dadas las funciones f y g por sus expresiones SOP:

$$\begin{aligned} f(A,B,C) &= A' + AB' + ABC' \\ g(A,B,C,D) &= B'C' + AB + A'BC + AB'CD' \end{aligned}$$

Construye los K-maps asociados a f y g.

Solución:

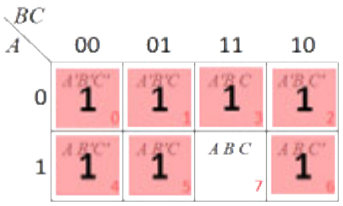
1er. Método: Expansión de términos no canónicos. Cada término no canónico da lugar a 2^n términos canónicos, siendo n el número de variables ausentes en el término no canónico. Aplicado a f y g, se tiene:

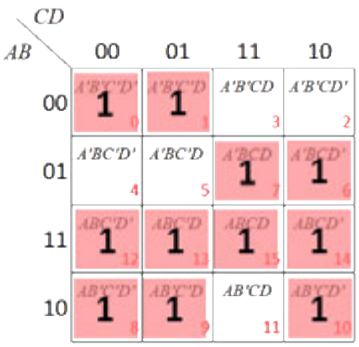
$$\begin{aligned} A'** &\to 0** \Rightarrow 000\ (0), 001\ (1), 010\ (2), 011\ (3) \\ AB'* &\to 10* \Rightarrow 100\ (4), 101\ (5) \\ ABC' &\to 110\ (6) \end{aligned}$$

Por tanto: $f = \sum m(0,1,2,3,4,5,6) = \prod M(7)$

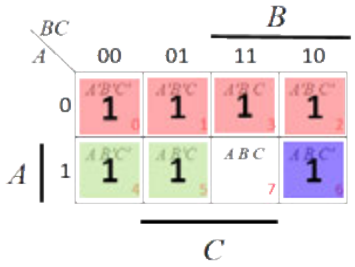
$$\begin{aligned} *B'C'* &\to *00* \Rightarrow 0000\ (0), 0001\ (1), 1000\ (8), 1001\ (9) \\ AB** &\to 11** \Rightarrow 1100\ (12), 1101\ (13), 1110\ (14), 1111\ (15) \\ A'BC* &\to 011* \Rightarrow 0110\ (6), 0111\ (7) \\ AB'CD' &\to 1010\ (10) \end{aligned}$$

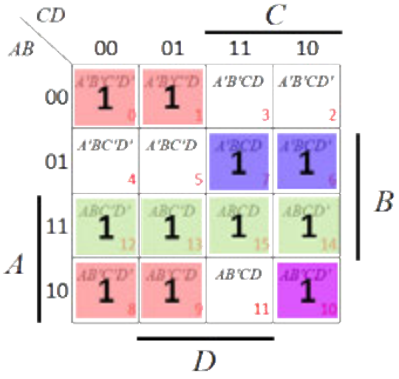
Por tanto: $g = \sum m(0,1,6,7,8,9,10,12,13,14,15) = \prod M(2,3,4,5,11)$, y los K-maps correspondientes quedan:





2º Método: Consiste en marcar directamente en el K-map las celdas asociadas a cada término de la expresión algebraica de la función. En este caso la colocación de líneas en el K-map para indicar las celdas asociadas al valor de cada variable ayuda en el proceso. En Logisim-evolution es posible escoger esta representación alternativa en los K-maps. A continuación se marcan en el mismo color los términos de cada expresión y las celdas asociadas a cada término.





$f = A' + AB' + ABC'$ $g = B'C' + AB + A'BC + AB'CD'$

Por supuesto, con ambos métodos se obtienen idénticos K-maps para cada función. ■

Simplificación de funciones a forma SOP con K-maps

El propósito de los K-maps es identificar de un modo sencillo los términos canónicos en celdas adyacentes. Entre dichas celdas se puede aplicar la simplificación de las variables que cambian de valor (aparecen negadas y sin negar). De este modo se dispone de un método sistemático de simplificación de expresiones booleanas para obtener una expresión equivalente más sencilla.

El método para obtener una forma SOP simplificada se puede resumir en los pasos siguientes:

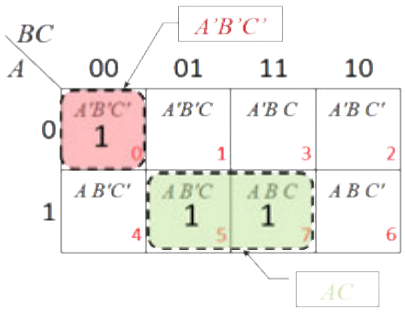
1. Se agrupan las celdas adyacentes del K-map que están marcadas con 1 (minitérminos) en grupos de tamaño 2^n (2, 4, 8, 16, etc.), siendo n el mayor entero posible. Una celda se puede agrupar varias veces si en cada una de las agrupaciones hay celdas nuevas no agrupadas previamente.
2. Si alguna celda marcada con 1 no puede agruparse con celdas adyacentes, el término canónico asociado no se puede simplificar.
3. Cada agrupación genera un producto de variables omitiendo las n variables que cambian de valor en dicha agrupación. Así, las relaciones entre el tamaño de la agrupación y las variables eliminadas son: $2 \rightarrow 1,\ 4 \rightarrow 2,\ 8 \rightarrow 3,\ 16 \rightarrow 4$, etcétera.
4. La función lógica equivalente simplificada será la suma lógica de los términos producto obtenidos en el paso previo.

Ejemplo 3.27 — Simplificación de forma canónica SOP mediante K-maps. Dada la función f, simplifícala a una forma SOP empleando K-maps.

$$f(A,B,C) = A'B'C' + AB'C + ABC = \sum m(0,5,7)$$

Solución:

En el K-map se marcan los minitérminos de la f y se crean las agrupaciones que se indican en la figura adjunta. La única posibilidad de agrupación es la que contiene a los minitérminos 5 y 7 que permite la eliminación de una variable (B). La función simplificada queda:



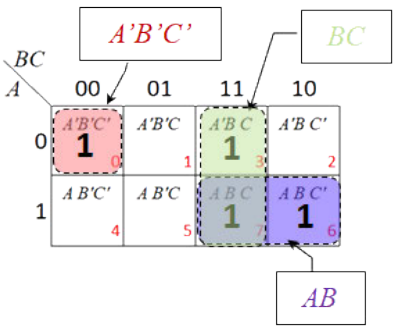
$$f = A'B'C' + AC$$

Ejemplo 3.28 — Simplificación de formas canónicas SOP mediante K-maps. Dadas las funciones f y g, simplifícalas a sus formas SOP mediante los K-maps correspondientes.

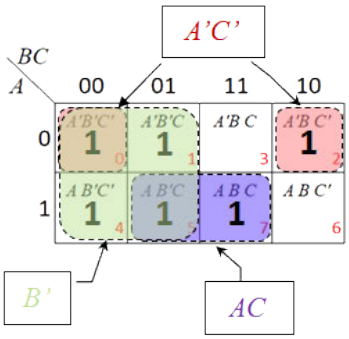
- $f(A,B,C) = \sum m(0,3,4,6)$
- $g(A,B,C) = \prod M(3,6)$

Solución:

Para g se proporciona la forma canónica con maxitéminos (forma POS), pero es sencillo obtener la forma canónica de minitérminos. A partir de las formas canónicas SOP de f y g, es posible construir sus K-maps y aplicar el método de simplificación a forma SOP para obtener las expresiones que se muestran a continuación:



$$f = A'B'C' + BC + AB$$



$$g = A'C' + B' + AC$$

La simplificación mediante K-maps no es única (ver expresiones de f y g en la figura 3.23). Por tanto, a partir de la versión simplificada de dos expresiones lógicas, no es posible deducir su equivalencia. Aunque entre dos expresiones todos los términos sean diferentes, se puede tratar de expresiones equivalentes de la misma función (véase el caso de la función g en la figura 3.23). El método más sencillo de comprobar si dos expresiones algebraicas son equivalentes es la comparación de sus tablas de verdad.

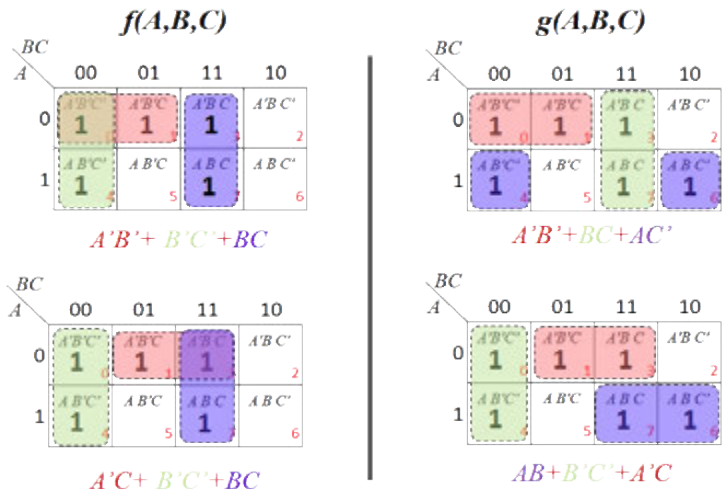


Figura 3.23: Expresiones SOP simplificadas alternativas a partir de K-maps para funciones de 3 variables.

Simplificación de funciones a forma POS con K-maps

La simplificación a forma POS es dual a la estudiada para las formas SOP agrupando los ceros del K-map y siguiendo reglas idénticas a las explicadas. Sin embargo, al obtener el término suma resultante de una agrupación, se tiene en cuenta que si el valor de la variable es 0, la misma aparece sin negar en la suma. Mientras que si el valor de la variable es 1, esta debe aparecer negada en el término suma asociado. La expresión POS final es el producto de los términos suma obtenidos en cada agrupación de ceros.

Ejemplo 3.29 — Simplificación de formas canónicas POS mediante K-maps. Dada la función f por la expresión:

$$(B+C+D)(A+B+C'+D)(A'+B+C+D')(A+B'+C+D)(A'+B'+C+D)$$

Determina la forma POS simplificada mediante el K-map correspondiente.

Solución:

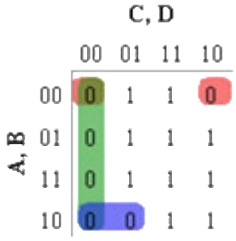
Para la expresión algebraica de f se obtienen los códigos binarios de los maxitérminos de la expresión canónica. En el proceso las sumas no canónicas se deben expandir en sus términos canónicos equivalentes.

$$\underbrace{(B+C+D)}_{\substack{0\\1}000\ (0,8)}\underbrace{(A+B+C'+D)}_{0010\ (2)}\underbrace{(A'+B+C+D')}_{1001\ (9)}\underbrace{(A+B'+C+D)}_{0100\ (4)}\underbrace{(A'+B'+C+D)}_{1100\ (12)}$$

Por tanto, f se puede expresar de forma abreviada como:

$$f=\prod M(0,2,4,8,9,12)=\sum m(1,3,5,6,7,10,11,13,14,15)$$

Trasladando estos valores al K-map y agrupando los ceros se tiene:



$$f=(A+B+D)(C+D)(A'+B+C)$$

La simplificación a forma SOP y POS puede dar lugar a resultados de distinto nivel de complejidad. Se podría pensar que cuanto mayor es el número de términos canónicos de la expresión de partida más compleja será la expresión simplificada, pero esto no siempre es así.

Ejemplo 3.30 — Comparación de simplificación SOP y POS. Dada la función f, por la expresión:

$$f = (A'+B'+C+D)(A+B'+C+D)(A+B+C+D')(A+B+C'+D')(A'+B+C+D')(A+B+C'+D)$$

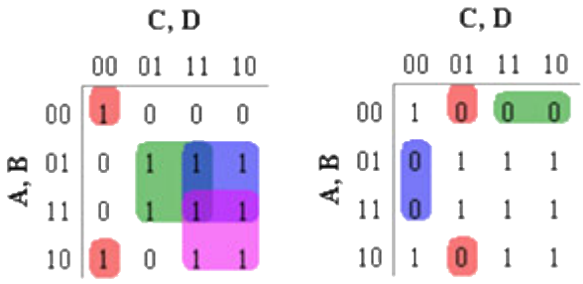
Determina las expresiones simplificadas de f mediante el empleo de K-maps.

Solución:

Para facilitar la construcción del K-maps de f se obtiene su expresión abreviada mediante mini/maxi-términos.

$$f = \underbrace{(A'+B'+C+D)}_{1100\,(12)}\underbrace{(A+B'+C+D)}_{0100\,(4)}\underbrace{(A+B+C+D')}_{0001\,(1)}\underbrace{(A+B+C'+D')}_{0011\,(3)}\underbrace{(A'+B+C+D')}_{1001\,(9)}\underbrace{(A+B+C'+D)}_{0010\,(2)}$$

Por tanto, se tiene: $f = \prod M(1,2,3,4,9,12) = \sum m(0,5,6,7,8,10,11,13,14,15)$, estos valores se trasladan al K-map para obtener las expresiones simplificadas SOP y POS:



$$f = B'C'D' + BD + BC + AC = (B+C+D')(A+B+C')(B'+C+D)$$

En este caso se observa como la expresión POS es más sencilla (con 3 términos) que la forma SOP (con 4 términos). ■

Ejemplo 3.31 — Comparación de simplificación SOP y POS. Dada la función

$$f = A' + AB' + ABC'$$

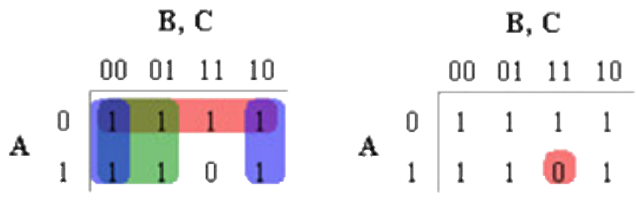
Realiza su simplificación a formas SOP y POS para comparar los resultados.

Solución:

Como f no está expresada en forma canónica, se hace la expansión de sus términos no canónicos:

$$\begin{aligned} A' &\rightarrow 0** \Rightarrow 000\ (0),\ 001\ (1),\ 010\ (2),\ 011\ (3) \\ AB' &\rightarrow 10* \Rightarrow 100\ (4),\ 101\ (5) \\ ABC' &\rightarrow 110\ (6) \end{aligned}$$

Por tanto, se tiene: $f = \sum m(0,1,2,3,4,5,6) = \prod M(7)$, que se puede trasladar al K-map para realizar la simplificación a formas SOP y POS para obtener:



$$f = A' + B' + C' = A' + B' + C'$$

Las formas simplificadas SOP y POS coinciden. Además, si es considerada una forma POS, también coincide con la forma canónica, pero no cuando se contempla como forma SOP. ■

Recuerda que cuando una función lógica tiene una expresión algebraica consistente solo en un término suma o producto:

- ✓ Las expresiones simplificadas SOP y POS de la función coinciden con el término dado. Por tanto, no es preciso realizar la simplificación.
- ✓ La expresión de la función se puede considerar canónica (SOP o POS, pero no ambas).

Simplificación de funciones con condiciones libres

Para algunas funciones lógicas existen valores de las variables que se denominan *condiciones libres*, *indiferentes* o *no importa* (*don't care*), ya que para dichos valores de las variables, el valor que tome la función es indiferente. La razón es que los valores mencionados para las variables no se producen en una situación real. Por tanto, el valor que se obtenga para la función es irrelevante en esos casos. Por ejemplo, si las 4 variables de las que depende una función lógica representan valores codificados en BCD, los valores de la función para códigos superiores a 9 corresponden a condiciones libres, ya que el valor de la función para dichos códigos es indiferente.[11]

Las condiciones indiferentes se representan en la tabla de verdad de la función con el símbolo 'x'. Dicho símbolo indica que el valor es indiferente. Por tanto, podría valer 0 o 1. Este mismo símbolo se empleará en la realización de los K-maps, pero en la simplificación se tomarán como 0 o 1 de modo que se consiga la mayor simplificación posible. Por supuesto, una vez realizada la simplificación, la función resultante otorga un valor determinado a las condiciones indiferentes, siendo este el que facilita obtener la expresión simplificada más simple.

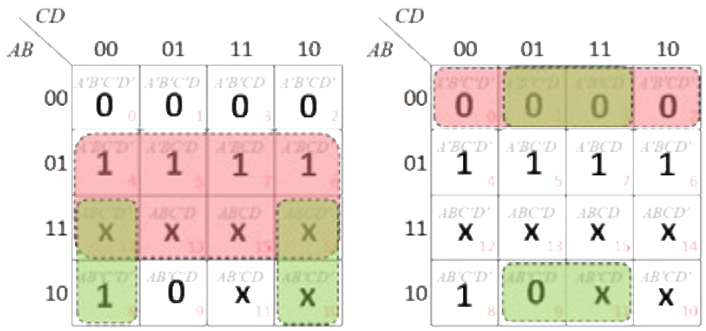
Ejemplo 3.32 — Simplificación de funciones con condiciones libres. Dada la función lógica por su tabla de verdad que se indica a continuación, determina las formas SOP y POS simplificadas.

Id.	A	B	C	D	f
0	0	0	0	0	0
1	0	0	0	1	0
2	0	0	1	0	0
3	0	0	1	1	0
4	0	1	0	0	1
5	0	1	0	1	1
6	0	1	1	0	1
7	0	1	1	1	1
8	1	0	0	0	1
9	1	0	0	1	0
10	1	0	1	0	x
11	1	0	1	1	x
12	1	1	0	0	x
13	1	1	0	1	x
14	1	1	1	0	x
15	1	1	1	1	x

Solución:

Dada la tabla de verdad, es posible construir el K-map correspondiente. En este caso las condiciones indiferentes se consideran 0 o 1 de modo que la simplificación sea lo más sencilla posible. De este modo se obtienen las expresiones SOP y POS simplificadas siguientes:

[11] Recuerda que los códigos BCD representan cifras del 0 al 9. Por tanto, los códigos de 4 bits superiores a 1001 (9) no se emplean.



$$f = B + AD' = (A + B)(B + D')$$

En este ejemplo además se muestra que no todos los valores de las condiciones indiferentes tiene que ser el mismo. En el ejemplo, casi todos se toman como 1, pero la celda 11 ($1011_{(2)}$) se considera 0 en la simplificación de la forma POS. ■

3.7.2 Tablas de verdad y K-maps con Logisim

Logisim incorpora una herramienta de análisis de funciones lógicas instructiva y sencilla de utilizar. Con dicha herramienta las funciones lógicas se pueden introducir en cuatro formatos diferentes:

1. *Tabla de verdad*. Introduciendo manualmente los valores para todas las filas de la tabla.
2. *Expresión algebraica*. En la que se emplean los operadores lógicos básicos permitidos: negación ($'$), producto, suma ($+$) y OR-exclusivo (^).
3. *K-map*. Los valores de las celdas del K-map se introducen manualmente. En este caso la expresión simplificada SOP y POS se calcula de modo interactivo.
4. *Diagrama del circuito de puertas lógicas*. Se crea el diagrama de un circuito válido que tendrá asociada una función lógica a su salida dependiente de las variables conectadas a las entradas del circuito. En este caso las etiquetas asociadas a los pines de entrada y salida se asignarán a los nombres de las variables de entrada y funciones lógicas dependientes, respectivamente.

Los tres primeros formatos se introducen desde la herramienta de análisis combinacional que acompaña a Logisim, accesible desde el menú (ver figura 3.24): Project » Analyze Circuit
Estos tres modos de definición de funciones están sincronizados, de modo que al introducir la función en uno de ellos, el resto queda determinado automáticamente.

> Aunque en las preferencias de Logisim se puede seleccionar el idioma español para la interfaz de usuario, en este texto se hace referencia siempre a la interfaz en inglés para evitar interpretaciones erróneas, ya que la traducción a español presenta algunas erratas.

Antes de introducir la función es preciso definir las variables de entrada (*inputs*) y las variables de salida o funciones (*outputs*). El convenio seguido en este texto es el empleo de letras mayúsculas para las entradas (A, B, C, D, E, etc.) y minúscula para las funciones (f, g, h, etc.). El orden en el que se definen las variables determina su posición en la tabla de verdad y, por tanto, su ponderación posicional en los códigos binarios construidos con sus valores. El orden de las variables se puede modificar en el panel de definición sin necesidad de volverlas a definir. Una vez definidas las entradas y salidas del circuito, es posible emplear alguno de los formatos mencionados anteriormente para definir las expresiones deseadas para las funciones lógicas (salidas):

- *Tabla de verdad*. Los valores de las variables se pueden ajustar mediante sucesivos clics de ratón o bien mediante pulsación de teclas [1], [0] y [x] (valor por defecto). Este método es apropiado cuando se conocen las expresiones canónicas de la función. Cuando hay varias funciones definidas permite comprobar si se trata de funciones equivalentes comprobando la coincidencia en las tablas de verdad asociadas.

- *Expresión algebraica*. Se puede emplear cuando se dispone de una expresión algebraica no canónica de las funciones. Es útil para comprobar la equivalencia de expresiones algebraicas en el panel correspondiente a las tablas de verdad asociadas.
- *K-map*. Permite obtener las formas SOP y POS simplificadas de las funciones. Es muy útil para comprobar la simplificación de funciones. Sobre el K-map es posible cambiar mediante clics de ratón los valores de las celdas. Logisim no permite la visualización de K-maps de 5 o más variables, aunque si es posible obtener la simplificación de funciones en dichos casos. También es importante tener presente que cuando se trabaja con condiciones indiferentes en las funciones, Logisim realiza las simplificaciones a forma SOP y POS de modo independiente. Es decir, se obtienen las expresiones SOP y POS óptimas, pero no tiene en cuenta la compatibilidad entre ellas que se debe forzar de modo manual.

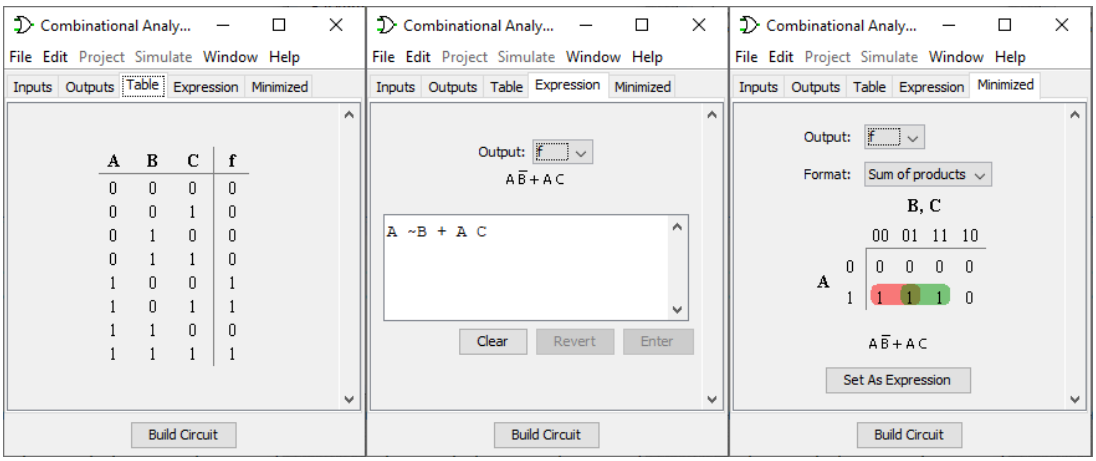


Figura 3.24: Panel de definición de una función lógica en Logisim (de izda. a dcha.): tabla de verdad, expresión booleana y K-map.

■ **Ejercicio 3.16 — Equivalencia de funciones lógicas con Logisim.** Comprueba con ayuda de Logisim la equivalencia de las funciones f y g del ejercicio 3.5 introduciendo manualmente las expresiones algebraicas de dichas funciones y comprobando los valores de sus tablas de verdad. ■

■ **Ejercicio 3.17 — Simplificación de funciones lógicas con Logisim.** Comprueba la simplificación de las funciones definidas en los ejemplos de este capítulo empleado los K-maps que proporciona Logisim. ■

3.7.3 Funciones lógicas de 5 o más variables

La simplificación mediante K-maps se puede aplicar a funciones que dependen de 5 variables o más. Sin embargo, en estos casos su elaboración manual es tediosa y propensa a errores. En el caso de 5 variables se pueden definir los K-maps siguiendo dos disposiciones:

1. Doble matriz 4×4. Cada matriz 4×4 corresponde al valor 0/1 de la variable más significativa y los valores de 4 variables. Para la agrupación de celdas del K-map se tiene en cuenta la adyacencia por superposición de las dos matrices obtenidas. Es un método conveniente para aplicación manual, pues el índice asociado a las celdas se obtiene de modo simple.
2. Matriz 4×8 con dos variables en las filas y las tres restantes en las columnas. En este caso, tanto en filas como en columnas, el valor de las variables sigue una numeración correspondiente al código Gray. La línea vertical central del mapa marca una línea de adyacencia especular en las celdas del mapa.[12]

En estos casos es muy interesante añadir el índice asociado al término canónico correspondiente a cada celda, ya que de este modo es muy sencillo trasladar la expresión canónica de la función al K-map.

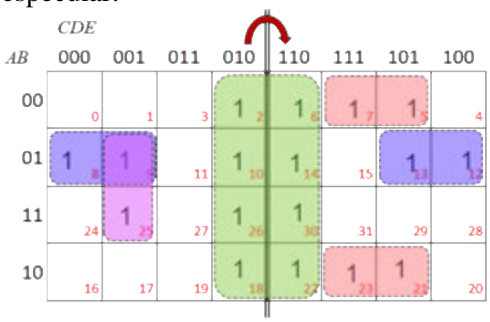
[12] Este es el tipo de K-maps empleado en Logisim-evolution. Logisim clásico solo representa gráficamente hasta K-maps de 4 variables.

Ejemplo 3.33 — K-maps de 5 variables. Simplifica la función $f(A,B,C,D,E)$ a su forma SOP mediante K-maps.

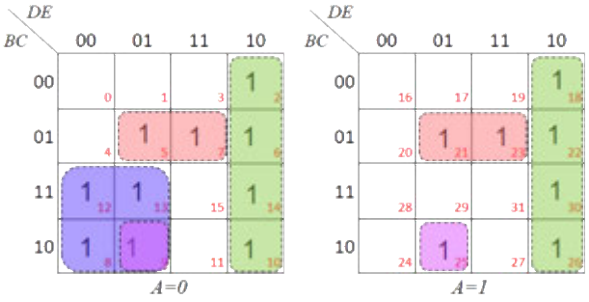
$$f = \sum m(2,5,6,7,8,9,10,12,13,14,18,21,22,23,25,26,30)$$

Solución:

Se construirán los dos tipos de K-maps descritos para funciones de 5 variables con adyacencia especular:



K-map con adyacencia por superposición:



La combinación de valores de las variables asociados a cada celda depende del tipo de K-map construido. Aunque la simplificación con K-maps no siempre es única, para este ejemplo se obtiene la misma forma SOP simplificada con los dos procedimientos descritos:

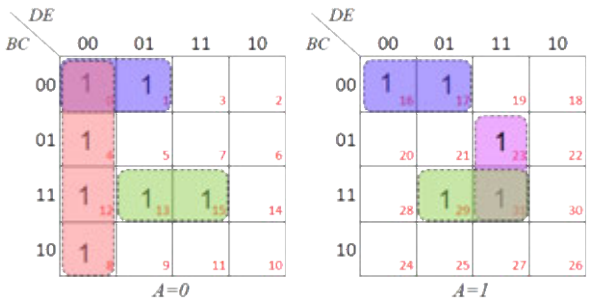
$$f = B'CE + DE' + A'BD' + BC'D'E$$

■

Ejemplo 3.34 — Simplificación SOP de funciones de 5 variables. Simplifica la función $f(A,B,C,D,E)$ a su forma SOP mediante K-maps: $f = \sum m(0,1,4,8,12,13,15,16,17,23,29,31)$

Solución:

Construyendo el K-maps con adyacencia de superposición se tiene:



$$f = A'D'E' + B'C'D' + BCE + ACDE$$

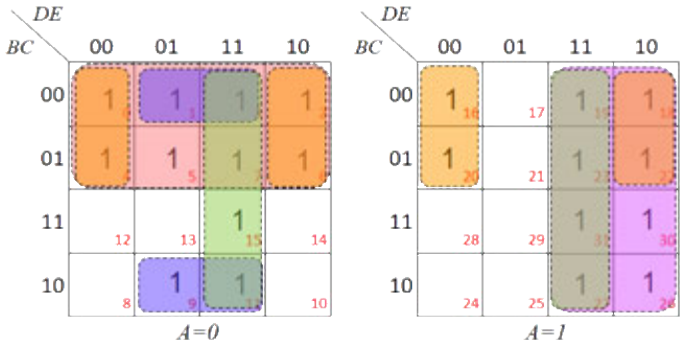
■

Ejemplo 3.35 — Simplificación SOP y POS de funciones de 5 variables. Compara la simplificación a forma SOP y POS mediante K-maps para la función $f(A,B,C,D,E)$ definida como:

$$f = \sum m(0,1,2,3,4,5,6,7,9,11,15,16,18,19,20,22,23,26,27,30,31)$$

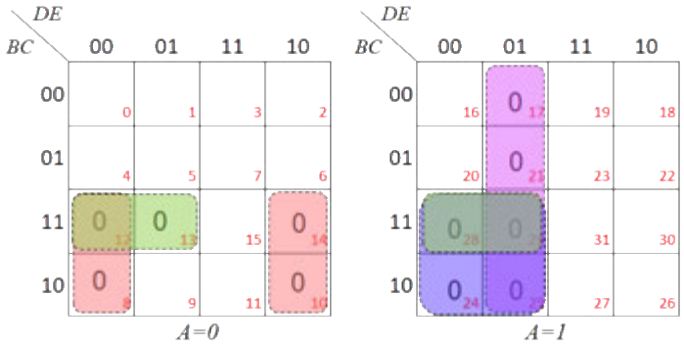
Solución:

Construyendo el K-maps con adyacencia de superposición se tiene para la forma SOP:



$$f = A'B' + DE + A'C'E + AD + B'E'$$

La simplificación de la forma POS $f = \prod M(8,10,12,13,14,17,21,24,25,28,29)$ queda:



$$f = (A + B' + E)(B' + C' + D)(A' + B' + D)(A' + D + E')$$

La introducción de funciones de 5 variables o más mediante su tabla de verdad es tedioso debido a su tamaño. Aunque Logisim clásico permite trabajar con estas funciones e incluso obtener las expresiones minimizadas, no ofrece las agrupaciones en los K-maps. En cambio, Logisim-evolution proporciona dichas agrupaciones para funciones de 6 variables, empleando adyacencia especular horizontal, a la que se añade adyacencia vertical. Para estos casos, una herramienta alternativa a Logisim es *Karnaugh Map Solver*[13] de Charlie Coleman que emplea también K-maps con adyacencia especular. Esta herramienta web es muy útil para comprobar las simplificaciones de K-maps, especialmente si las funciones dependen de 5 variables o más. Con ella la introducción de las funciones lógicas se facilita indicando solo sus minitérminos, sin necesidad de definir la tabla de verdad completa.

La simplificación de funciones lógicas con K-maps admite una formulación más apropiada para su implementación en computador mediante el *algoritmo de tabulación de Quine-McCluskey* —desarrollado en 1952 por Willard V. Quine y extendido posteriormente en 1956 por Edward J. McCluskey—. Dicho algoritmo se basa en la búsqueda del conjunto mínimo de *implicantes primos* que cubren la función lógica a minimizar. Este algoritmo se aplica con éxito a funciones para las que el empleo de K-maps es complicado. En 1982, Robert K. Brayton y sus colaboradores en IBM desarrollaron el *minimizador lógico*

[13] Disponible en Internet, URL: https://www.charlie-coleman.com/experiments/kmap/.

heurístico ESPRESSO[14] cuyas variantes se emplean en los programas actuales de diseño de circuitos digitales.

■ **Ejercicio 3.18 — Simplificación de funciones de 5 variables.** Comprueba los resultados de simplificación de funciones de 5 variables de los ejercicios precedentes mediante la herramienta Karnaugh Map Solver de Charlie Coleman. Intenta obtener los resultados mediante Logisim. ■

Conceptos clave

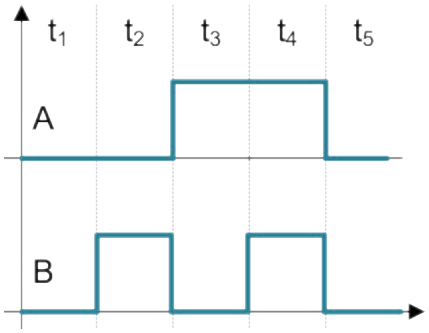
- ☑ Puertas lógicas elementales, secundarias y universales.
- ☑ Teoremas, propiedades y leyes del álgebra de Boole.
- ☑ Tabla de verdad de una función booleana.
- ☑ Equivalencia de función booleanas.
- ☑ Función booleana de salida de un circuito lógico.
- ☑ Formas canónicas de una función booleana, minitérminos y maxitérminos.
- ☑ Simplificación de funciones lógicas mediante K-maps.

[14] Disponible en Internet, URL: https://is.gd/tFUuM7 como parte del software *Logic Friday* de la Universidad de California.

Problemas propuestos

Problema 3.1 Dados los cronogramas de las señales A y B, completalos con las señales de salida para las funciones lógicas que se indica a continuación.

- NOT: A',
- AND: $A \cdot B$,
- OR: $A + B$, y
- XOR: $A \oplus B$.



Problema 3.2 Comprueba la validez de las leyes de De Morgan con ayuda de Logisim. Para ello, verifica la equivalencia de los circuitos correspondientes a una puerta NAND y una puerta NOR.

Problema 3.3 Dada la tabla de verdad adjunta, encuentra su relación con la tabla correspondiente a una puerta AND. Propón la modificación de un circuito con puerta AND para que responda a la tabla de verdad dada y emplea Logisim para hacer la comprobación. Intenta un razonamiento similar empleando la puerta NOR.

A	B	f
0	0	0
0	1	1
1	0	0
1	1	0

Problema 3.4 Dada la tabla de verdad proporcionada, encuentra su relación con la tabla correspondiente a una puerta OR. Propón la modificación de un circuito con puerta OR para que responda a la tabla de verdad dada y emplea Logisim para hacer la comprobación. Intenta un razonamiento similar empleando la puerta NAND.

A	B	f
0	0	1
0	1	0
1	0	1
1	1	1

Problema 3.5 Comprueba —con ayuda de Logisim— la equivalencia entre la puerta lógica XOR con dos entradas (A, B), y los circuitos lógicos que implementan las funciones f y g:

- $f(A,B) = A'B + AB'$
- $g(A,B) = (A+B)(A'+B')$

Problema 3.6 Comprueba —con ayuda de Logisim— la validez de las propiedades distributivas del álgebra de Boole. Esto es, la equivalencia de las expresiones lógicas indicadas mediante circuitos lógicos equivalentes:

- $A(B+C) = AB + AC$
- $A + BC = (A+B)(A+C)$

Problema 3.7 Demuestra que las funciones NAND y NOR no cumplen la propiedad asociativa.

Problema 3.8 Demuestra mediante manipulación algebraica que las funciones XOR y XNOR cumplen la propiedad asociativa.

Problema 3.9 Demuestra mediante manipulación algebraica la igualdad: $A \oplus B \oplus A' = B'$

Problema 3.10 Si $A \oplus B = C$, ¿cuál es el valor de la expresión $A \oplus B \oplus C$?

Problema 3.11 Demuestra mediante manipulación algebraica la igualdad:
$A' \oplus B = (A \oplus B)' \oplus 0$

Problema 3.12 Aplica los axiomas y teoremas del álgebra de Boole para obtener una expresión minimizada de la función $f(A,B,C) = ((AB)' + A)'(ABC' + C)'$

Problema 3.13 ¿Cuál es el resultado de la simplificación algebraica de la función f?
$f(A,B,C,D) = (A' + B)' + BC' + ((A' + B)'CD)'$

Problema 3.14 Indica cuál de las expresiones siguientes es equivalente a $(A+B)(A'+C)(B+C)$:

a) $(A+B)(A'+C)$
b) $A+C$
c) $(B+C)(A'+C)$
d) $B+C$

Problema 3.15 Demuestra la validez o falsedad de la igualdad: $A' + A(A'B + B'C)' = A'$

Problema 3.16 Dada la función $f(A,B,C) = AC + A'B'C'$ indica si está expresada en forma canónica. En caso contrario, calcula sus formas canónicas SOP y POS.

Problema 3.17 Si una función lógica $f(A,B,C)$ tiene una expresión formada por 8 productos canónicos, ¿de qué función se trata? ¿Cuántas sumas canónicas tendrá la forma canónica alternativa?

Problema 3.18 Dada la función $f(A,B) = AB$ ¿es una expresión canónica? ¿Por qué? En caso afirmativo encuentra la expresión canónica alternativa.

Problema 3.19 Expresa la función $f = A' + BC + AC$ en sus dos formas canónicas. A partir de la expresión proporcionada, intenta encontrar mediante manipulación algebraica, la expresión equivalente más sencilla.

Problema 3.20 Calcula las dos formas canónicas de la función f:
$f(A,B,C,D) = (B+D)(A'+C+D)$

Problema 3.21 Calcula las formas canónicas de la función f', dada la forma canónica conjuntiva (forma POS) de f:
$f(A,B,C) = \prod M(1,3,4)$

Problema 3.22 Aplica las leyes de De Morgan para simplificar la función lógica f y expresarla en sus formas canónicas:
$f(A,B,C,D) = [(AB)(C + D')']'$

Problema 3.23 Aplica los axiomas y teoremas del álgebra de Boole para simplificar la expresión de la función f. ¿Qué se puede afirmar respecto de las expresiones canónicas de f y sus versiones minimizadas? Comprueba el resultado con Logisim.
$f(A,B) = (A+B)'(A \oplus B) + (AB)'(A \odot B)$

Problema 3.24 Simplifica la función $f(A,B,C,D) = \sum m(0,1,2,3,6,7,8,10,12,14)$ empleando mapas de Karnaugh.

Problema 3.25 Dada la función $f'(A,B,C,D) = \sum m(0,1,2,3,9,11)$, determina las formas canónicas y expresiones simplificadas de $f(A,B,C,D)$.

Problema 3.26 Dada la función $f(C,B,A) = m_2 + M_6$, calcula sus formas canónicas y las expresiones SOP y POS minimizadas. *(Nota: Ten en cuenta el orden proporcionado para las variables.)*

Problema 3.27 Dada la función $f(A,B,C) = (C'B + ACB')(B + A')$, determina sus dos formas canónicas y sus expresiones SOP y POS simplificadas. Si se considera C como la variable más significativa, ¿cambian sus formas canónicas?

II

Sistemas Combinacionales

4. Circuitos digitales combinacionales

En el capítulo anterior se explica el álgebra de Boole, que proporciona las herramientas para el análisis y diseño de circuitos digitales implementados con puertas lógicas. Los circuitos digitales denominados *combinacionales* son aquellos cuya salida depende únicamente del valor instantáneo de sus entradas. En este capítulo se resumen las técnicas empleadas para el análisis y diseño de los circuitos combinacionales utilizando los resultados derivados del álgebra de Boole. El análisis permite determinar la funcionalidad del circuito para su posible optimización, mientras que el diseño tiene como objetivo implementar un circuito que satisfaga la especificación funcional proporcionada.

4.1 Circuitos combinacionales y funciones lógicas

Los circuitos combinacionales son circuitos lógicos construidos con puertas lógicas que pueden tener varias entradas y salidas, como el que muestra la figura siguiente.

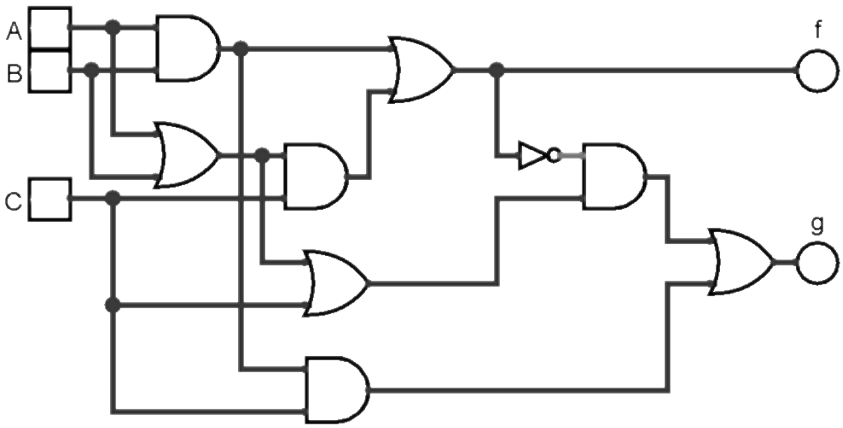


Figura 4.1: Circuito combinacional con 3 entradas (A, B, C) y 2 salidas (f y g).

Definición 4.1 — Circuito combinacional. Se denomina *combinacional* al circuito digital cuyas señales de salida solo dependen del valor instantáneo de sus señales de entrada. Las salidas solo son afectadas por el valor de las entradas. ■

El estudio del álgebra de Boole y sus resultados tienen aplicación directa en los sistemas digitales implementados con circuitos electrónicos basados en puertas lógicas. Todo circuito lógico digital implementa una o varias funciones lógicas aplicadas a las señales de entrada del circuito. En los circuitos combinacionales, cada función de salida depende solo del valor de las señales de entrada al circuito. Este tipo de circuitos nunca presentan realimentación desde su salida hacia su entrada, ya que esto indicaría la dependencia de la salida de valores previos en la misma.

> A lo largo de esta obra nos referimos a un *circuito lógico* tanto para denominar al *circuito electrónico digital* como al *diagrama lógico* que representa a dicho circuito.

En el capítulo anterior se explica cómo se describe una función lógica mediante una expresión algebraica booleana o su tabla de verdad. En este punto se introduce la definición de una función lógica mediante su circuito lógico equivalente.

Corolario 4.1.1 — Función lógica implementada por un circuito lógico. La función lógica implementada por un circuito de puertas lógicas se obtiene componiendo las operaciones lógicas asociadas a cada puerta lógica desde las entradas del circuito hacia su salida.

Ejemplo 4.1 — Función lógica implementada por un circuito lógico. Dado el circuito lógico de la figura 4.1, determina las funciones lógicas que implementa. Expresa dichas funciones en alguna de sus formas canónicas SOP o POS.

Solución:

Si se componen las operaciones elementales que realiza cada puerta lógica del circuito desde las entradas hacia las salidas se obtienen las funciones:

$$
\begin{aligned}
f(A,B,C) &= AB+(A+B)C = AB+AC+BC \quad \text{(forma SOP-no canónica)}\\
g(A,B,C) &= (AB+(A+B)C))'(A+B+C)+ABC = (AB+AC+BC)'(A+B+C)+ABC \quad \leftarrow \text{(P.2)}\\
&= (A'+B')(A'+C')(B'+C')(A+B+C)+ABC \quad \leftarrow \text{(P.3)}\\
&= (A'+B'C')(B'+C')(A+B+C)+ABC \quad \leftarrow \text{(P.2}')\\
&= (A'B'+B'C'B'+A'C'+B'C'C')(A+B+C)+ABC \quad \leftarrow \text{(P.2)}\\
&= (A'B'+B'C'+A'C')(A+B+C)+ABC \quad \leftarrow \text{(T.3)}\\
&= \cancel{A'B'A}+B'C'A+\cancel{A'C'A}+\cancel{A'B'B}+\cancel{B'C'B}+A'C'B+A'B'C+\cancel{B'C'C}+\cancel{A'C'C}+ABC\\
&= \underbrace{AB'C'}_{100\ (4)}+\underbrace{A'BC'}_{010\ (2)}+\underbrace{A'B'C}_{001\ (1)}+\underbrace{ABC}_{111\ (7)} = m_4+m_2+m_1+m_7 \quad \text{(forma SOP-canónica)}
\end{aligned}
$$

Haciendo la expansión de términos no canónicos de f, se tiene:

$$
\begin{aligned}
&AB* \rightarrow 11* \Rightarrow 110\ (6), 111\ (7) \rightarrow ABC', ABC\\
&A*C \rightarrow 1*1 \Rightarrow 101\ (5), \cancel{111\ (7)} \rightarrow AB'C\\
&*BC \rightarrow *11 \Rightarrow 011\ (3), \cancel{111\ (7)} \rightarrow A'BC
\end{aligned}
$$

Por tanto: $f(A,B,C) = \sum m(3,5,6,7) = \prod M(0,1,2,4)$

Como g ya está en forma canónica, se tiene: $g(A,B,C) = \sum m(1,2,4,7) = \prod M(0,3,5,6)$ ■

Corolario 4.1.2 — Circuitos lógicos equivalentes a una función lógica. Una vez obtenida la función lógica asociada a un circuito, cualquier función lógica equivalente a ella tiene un circuito de puertas lógicas que es equivalente al circuito de partida.

4.1.1 Circuitos equivalentes a expresiones SOP y POS

Toda función lógica en forma SOP (suma de productos) tiene asociado un circuito en dos niveles que es una red de puertas AND/OR. Las puertas del nivel de entrada AND llevan a cabo las operaciones producto

y la puerta OR del nivel de salida realiza la suma (ver ejemplo de la figura 4.2).

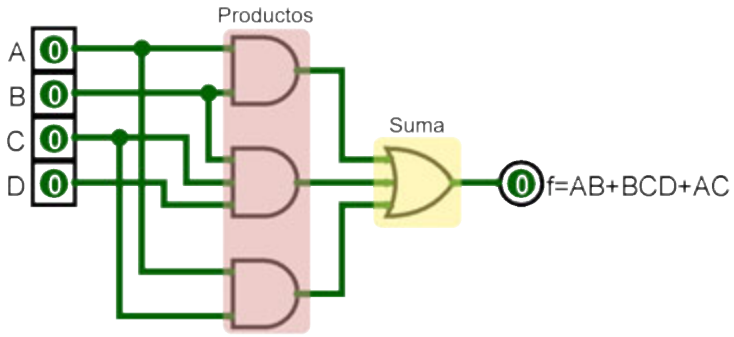


Figura 4.2: Ejemplo de red AND/OR de dos niveles para implementar función en forma SOP.

En una red AND/OR de dos niveles, una puerta AND implementa un producto canónico si está conectada a todas las entradas del circuito. Por ejemplo, en el circuito de la figura 4.2 las puertas AND no implementan productos canónicos, ya que ninguna puerta está conectada a las 4 entradas del circuito.

Toda función lógica en forma POS (producto de sumas) tiene asociado un circuito en dos niveles que es una red de puertas OR/AND. Las puertas del nivel de entrada OR llevan a cabo las operaciones de suma y la puerta AND del nivel de salida realiza el producto (ver figura 4.3).

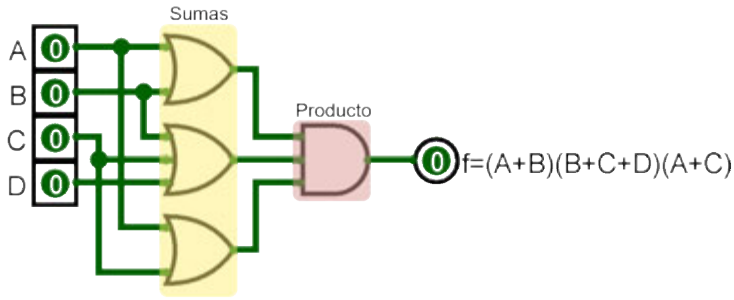


Figura 4.3: Ejemplo de red OR/AND de dos niveles para implementar función en forma POS.

En una red OR/AND de dos niveles, cada puerta OR implementa una suma canónica si está conectada a todas las entradas del circuito. Por ejemplo, en el circuito de la figura 4.3 las puertas OR no implementan sumas canónicas, ya que ninguna puerta está conectada a las 4 entradas del circuito.

■ **Ejercicio 4.1** Calcula las expresiones canónicas de los circuitos lógicos que se muestran en las figuras 4.2 y 4.3. Comprueba el resultado mediante Logisim obteniendo la TV asociada a cada uno de los circuitos. Recuerda que en dicha tabla aparece un 1 en las filas asociadas a los minitérminos y un 0 en las filas asociadas a los maxitérminos de la función lógica. ■

4.1.2 Obtención de circuito equivalente desde tabla de verdad de la función

Cuando una función lógica se define mediante su TV, es inmediato expresar dicha función en sus formas canónicas. Las filas de la TV en que la función vale 1, definen los minitérminos de la misma y su suma constituye la 1ª FC o forma canónica SOP. Por el contrario, las filas en que la función es 0, determinan los maxitérminos de la función y su producto constituye la 2ª FC o forma canónica POS.

Puesto que las formas canónicas son respectivamente formas SOP y POS, su implementación en un circuito lógico se obtiene con sendas redes en dos niveles AND/OR y OR/AND, respectivamente.

Ejemplo 4.2 — Circuito lógico equivalente a una función dada por su TV. Dada la TV de la función lógica $f(A,B,C)$, determina los circuitos lógicos equivalentes a las formas canónicas SOP y POS para implementar dicha función.

Id.	A	B	C	f
0	0	0	0	1 ←
1	0	0	1	0
2	0	1	0	0
3	0	1	1	0
4	1	0	0	0
5	1	0	1	1 ←
6	1	1	0	0
7	1	1	1	1 ←

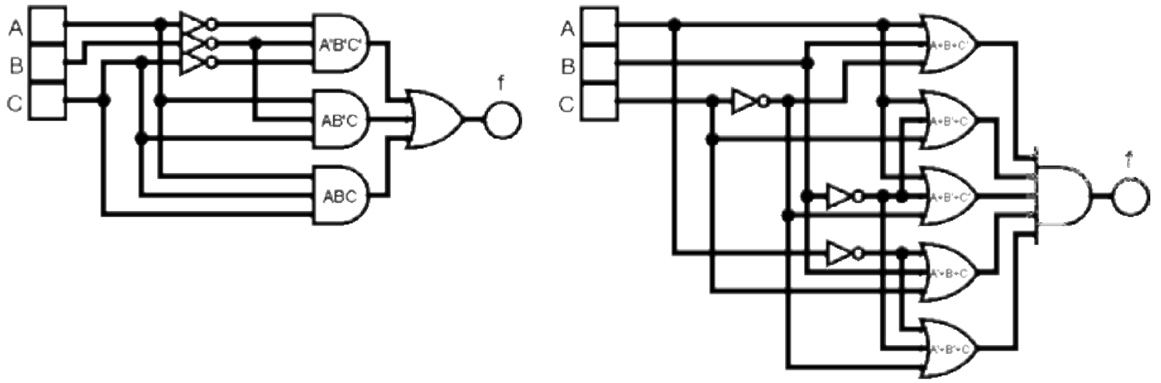
Solución:

Los minitérmnos (productos canónicos) de f están asociados a las filas de la TV en que $f = 1$ (marcadas con una flecha). De modo complementario, los maxitérminos (sumas canónicas) de f están asociados a las filas de la TV en que $f = 0$. En consecuencia, se tiene:

- 1ª FC (forma SOP): $f = \sum m(0,5,7) = \underbrace{A'B'C'}_{000\ (0)} + \underbrace{AB'C}_{101\ (5)} + \underbrace{ABC}_{111\ (7)}$
- 2ª FC (forma POS):

$$f = \prod M(1,2,3,4,6) = \underbrace{(A+B+C')}_{001\ (1)}\underbrace{(A+B'+C)}_{010\ (2)}\underbrace{(A+B'+C')}_{011\ (3)}\underbrace{(A'+B+C)}_{100\ (4)}\underbrace{(A'+B'+C)}_{110\ (6)}$$

Teniendo en cuenta que una forma SOP se implementa con una red AND/OR y la forma POS mediante una red OR/AND, las formas canónicas se pueden implementar mediante los circuitos de la figura adjunta.



4.1.3 Aspectos prácticos de la simulación de circuitos con Logisim

Durante la simulación con Logisim el color de las conexiones señala las condiciones de funcionamiento que se indican a continuación:

- **Verde oscuro**. Indica que el valor lógico de la señal es 0.
- **Verde claro**. Indica que el valor lógico de la señal es 1.
- **Azul**. Indica que la conexión no tiene asignado ningún valor lógico. Se trata de una conexión flotante (desconectada) de las entradas. Se produce cuando hay partes del circuito sin conectar o cuando las conexiones son incorrectas (p. ej., en el caso de entradas de puertas lógicas sin conectar y de pines de entradas conectados incorrectamente). Este estado es conocido también como de *alta impedancia* y se suele marcar con un valor '**x**', que no se debe confundir con el valor dado a las condiciones libres. Tampoco se debe interpretar como un error, pues dependiendo del circuito puede ser un estado provocado deliberadamente cuando se desea inhibir la conexión a un bus compartido.
- **Rojo**. Indica error ('**E**') y se produce cuando Logisim no es capaz de asignar un valor lógico a la conexión.

La figura 4.4 muestra dos de los errores usuales cometidos al implementar circuitos con Logisim. Recuerda que en un circuito Logisim se puede inspeccionar el valor lógico en cualquier conexión (incluso si es de tipo multibit) mediante la pulsación sobre la misma con la herramienta de cambio de valor de pines de entrada. Para visualizar permanentemente el valor de una señal se emplea el elemento *sonda de prueba* (*probe*) conectado a la conexión a monitorear.

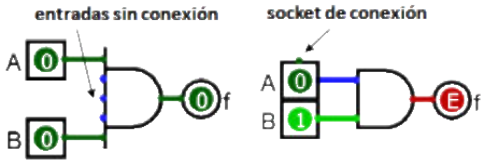


Figura 4.4: Errores habituales en Logism (de izda. a dcha.): entradas de puertas sin conexión y conexión incorrecta de pines de entrada.

Para evitar errores de simulación en Logisim se debe tener presente:

- Puertas con entradas no conectadas. Al colocar una nueva puerta en el área de trabajo, esta se crea con un número de entradas que puede exceder el requerido. En esta situación, un error común es dejar desconectadas las entradas sobrantes (en color azul). Para evitarlo, el atributo de número de entradas de cada elemento del circuito se debe fijar en el valor preciso. Un modo rápido de fijar el número de entradas de una puerta seleccionada sobre el área de trabajo, consiste en pulsar la tecla de número coincidente con el de entradas deseadas.
- Pines de E/S conectados incorrectamente. Los pines de E/S poseen un extremo para la conexión que en ocasiones es difícil de distinguir y es propenso a confusión. Para evitarlo es conveniente utilizar un nivel de zoom que permita ver con nitidez los puntos de conexión.

En los sistemas digitales es muy habitual el uso de buses de conexión compartidos para intercambio de información (datos, direcciones, etc.) entre subsistemas (ver figura 4.5).

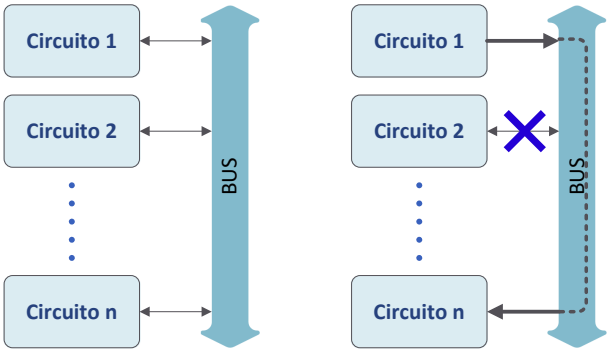


Figura 4.5: Diagrama de conexión de subcircuitos a bus común y estado flotante para permitir la interconexión entre ellos.

Definición 4.2 — Bus de interconexión. Un *bus* es una conexión, compartida por distintos subsistemas que simplifica el modo de conexión entre ellos evitando su interconexión punto a punto. Dependiendo del ancho del bus, esto es, si el bus transmite un bit o varios en un instante dado, se denominan respectivamente *bus serie* y *bus paralelo*. ■

Para poder arbitrar el acceso de distintos dispositivos a un bus compartido, es preciso disponer de elementos que permitan la desconexión eléctrica de circuitos al bus. Esta «desconexión» se consigue mediante el estado denominado de *alta impedancia* o *estado flotante* a la salida de un circuito lógico (ver figura 4.5). Cuando la salida de un circuito queda en alta impedancia, su nivel lógico queda flotante para ser determinado por otros dispositivos conectados en dicha conexión. Mediante este estado flotante es posible conseguir que los dispositivos oportunos estén conectados al bus compartido.

Uno de los dispositivos empleados para conseguir un estado flotante a la salida de un circuito es el *buffer triestado*. El buffer triestado tiene dos entradas, una de datos y otra de control. La entrada lógica de datos se transmite a la salida cuando la entrada de control está activada. Si la entrada de control está desactivada, la salida del buffer queda en estado flotante. Este comportamiento se resume en la tabla de verdad 4.1.

Tabla 4.1: Tabla de verdad del buffer triestado.

Entrada	Control	Salida
0/1	0	x (flotante)
0	1	0
1	1	1

La figura 4.6 muestra el funcionamiento del buffer triestado y su empleo para conectar dos entradas de datos a un bus de interconexión. En dicha figura se observa como la señales de control a los buffers se deben arbitrar para que en un instante dado haya conectada solo una entrada y una salida de datos. En este ejemplo el arbitraje mencionado se consigue con el empleo de un inversor. Cuando la señal de control está desactivada, el buffer de la entrada de datos inferior (2) está activado y el superior desactivado (1). Por el contrario, cuando la señal de control está activada el buffer habilitado es el correspondiente a la entrada de datos superior (1) que se transmiten por el bus. En situaciones con un mayor número de sistemas conectados al bus se emplean decodificadores para generar las señales de control que permiten arbitrar el acceso al bus compartido.

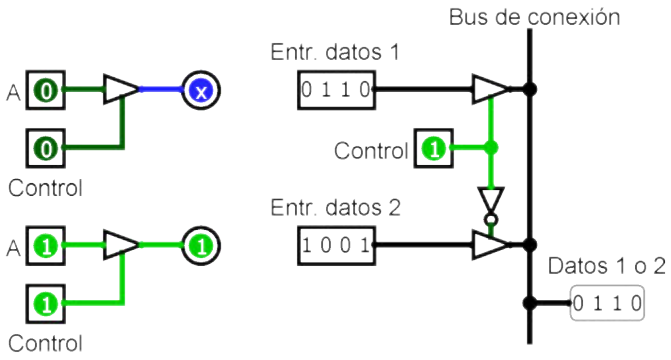


Figura 4.6: Diagrama lógico con buffer triestado y ejemplo de aplicación de conexión a bus.

El *inversor controlado triestado* tiene un comportamiento similar al buffer controlado, pero su salida invierte la señal de entrada. En el símbolo del buffer controlado se añade una burbuja de negación (○) a la salida del mismo. Además del buffer triestado y el inversor controlado, Logisim proporciona otros elementos en los que es posible simular la aparición de un estado flotante en su salida. Estos elementos comprenden los transistores en estado de corte y aquellos módulos en los que es posible definir su salida en estado flotante cuando están inhibidos (con señal de *enable* desactivada).

> En Logisim cuando un módulo esté inhabilitado (p. ej., decodificador, multiplexor, etc.), se recomienda configurar su salida a 0, en lugar de flotante. De este modo se evita que la conexión de módulos en cascada provoque errores de simulación. Es un valor que se modifica en los atributos del elemento y que por defecto está ajustado en valor flotante.

En Logisim es posible definir señales de entrada y salida de tipo multibit definiendo de este modo sus pines correspondientes. Esto permite la conexión de elementos de un circuito mediante buses multibit. Las señales multibit son necesarias en algunos de los elementos disponibles. Por ejemplo, para las entradas de selección en decodificadores y multiplexores. Algunos elementos en Logisim (como las puertas lógicas) tienen un atributo para definir el número de bits de datos del elemento. Dicho atributo se puede ajustar de modo rápido mediante el atajo de teclado Alt + n, donde *n* indica el número de bits de datos para el

elemento. Cuando se emplean elementos multibit en operaciones lógicas simuladas con Logisim, estás se realizan bit a bit (ver figura 4.7).

Las señales multibit de Logisim permiten simplificar los diagramas lógicos de los circuitos al agrupar varias señales en una única línea de conexión. La figura 4.7 muestra una operación AND bit a bit entre señales multibit de 4 bits (izda.) y una operación de inversión (dcha.). En la dicha figura se hace uso del elemento *sonda lógica de prueba* (*probe*), disponible en la librería *Wiring* como el elemento *'ver'*. La sonda de prueba permite obtener, independientemente del número de bits, el valor en cualquier línea de conexión para mostrarla en el sistema numérico de representación (*radix*) elegido: binario, octal, decimal con/sin signo y hexadecimal.[1]

Para poder acceder a cada uno de los bits de una señal multibit en Logisim se proporciona un *elemento separador* de bits (*splitter*). Este elemento identifica con un número, el orden de cada uno de los bits de la conexión. La figura 4.7 (dcha.) muestra el uso de un *splitter* conectado a la entrada y otro a la salida de un bus. En dicha figura se observa como la señales multibit se pueden interpretar como agrupación de señales independientes $[A\,B\,C\,D]$ o bien como los bits componentes de una una variable multibit $[A_3\,A_2\,A_1\,A_0]$.

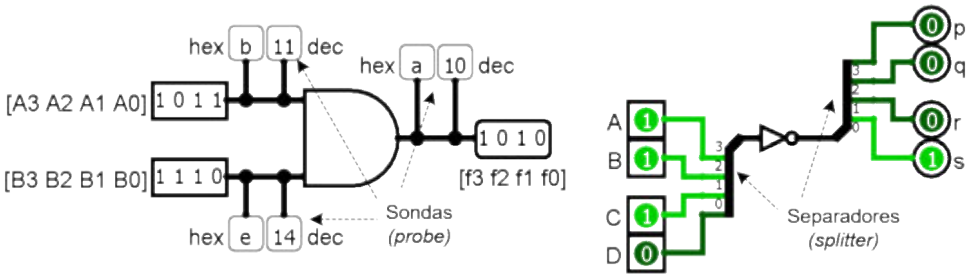


Figura 4.7: Esquema de conexionado en Logisim con splitter, bus y E/S multibit.

Al igual que otros componentes de Logisim, el elemento *splitter* posee unos atributos que permiten modificar sus características, incluido su aspecto visual y la ordenación de cada una de las ramas asociadas a cada bit. Haciendo clic derecho se puede invertir el orden ascendente (por defecto) de los bits, a un orden descendente.

Uno de los errores habituales en la elaboración de circuitos lógicos con Logisim es la conexión de elementos multibit con número de bits (o anchura) incompatibles. En estos casos un error informa de dicha circunstancia (ver figura 4.8).

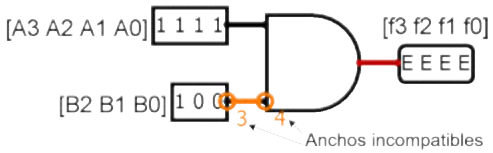


Figura 4.8: Error provocado por la conexión de señales multibit con un número de bits incompatible.

Aunque el diseño de sistemas con señales de E/S multibit simplifica los diagramas lógicos realizados con Logisim, su utilización tiene una limitación importante, ya que no permite el análisis automatizado de los circuitos que las emplean para conectar las entradas y/o las salidas. Para superar esta limitación las E/S multibit se deben separar en cada uno de sus bits componentes mediante el elemento *splitter*.

4.2 Optimización de circuitos lógicos

Partiendo de un circuito lógico, es posible obtener la función lógica asociada al mismo mediante una red de puertas en dos niveles. Para las funciones en forma SOP, la red es de tipo AND/OR y para las funciones en forma POS la red es de tipo OR/AND. Por tanto, si se proporciona una función en sus

[1] En Logisim, los valores con signo se interpretan por defecto con codificación en complemento a dos.

formas canónicas, aquella con menor número de términos produce una implementación más sencilla. Esto es, con menor número de puertas lógicas o con menos entradas en ellas. Sin embargo, las formas canónicas de una función no son las más simples para expresar la función. En consecuencia, es posible implementar frecuentemente un circuito equivalente más sencillo a las formas canónicas, como las expresiones simplificadas SOP y POS obtenidas mediante K-maps u otros métodos de optimización como el algoritmo de Quine-McCluskey.

Además de la simplificación de los circuitos mediante técnicas de minimización algebraica de funciones lógicas, resulta muy conveniente la implementación de los circuitos con un único tipo de puerta lógica. Para conseguirlo se recurre a las puertas universales NAND y NOR. La tabla 4.2 resume como las operaciones NAND y NOR permiten obtener cada una de las tres operaciones elementales: NOT ($'$), AND ($\cdot$) y OR ($+$).

Tabla 4.2: Obtención de las operaciones booleanas elementales con las puertas universales NAND y NOR.

	Inversion ($'$)	**Producto** ($\cdot$)	**Suma** ($+$)
NAND	$A' = (AA)' = (A \cdot 1)'$	$AB = ((AB)')'$	$A+B = (A'B')'$
NOR	$A' = (A+A)' = (A+0)'$	$AB = (A'+B')'$	$A+B = ((A+B)')'$

Cuando se plantea el uso de puertas universales para implementar funciones lógicas mediante sus circuitos equivalentes SOP y POS surge la pregunta:

> *¿Cómo cambian los circuitos en dos niveles (redes AND/OR y OR/AND) cuando empleamos puertas universales?*

Corolario 4.2.1 — Implementación de redes AND/OR y OR/AND con puertas universales. Una red de puertas AND/OR admite una implementación equivalente en la que todas las puertas se sustituyen por puertas NAND. De modo dual, una red de puertas OR/AND admite una implementación equivalente en la que todas las puertas se sustituyen por puertas NOR.

Este resultado se muestra intuitivamente en la figura 4.9. Dicha figura expone como las puertas AND se pueden sustituir por puertas NAND si a continuación su salida se niega (entrada negada en puertas OR). Este cambio produce que la puerta OR tenga todas sus entradas negadas, de modo que se puede sustituir por una puerta NAND. En consecuencia, la red AND/OR original es equivalente a una red en la que todas las puertas se sustituyen por puertas NAND. Un razonamiento similar se emplea para la equivalencia entre una red OR/AND y una red con puertas NOR. Si se parte de expresiones algebraicas SOP o POS y se aplican las propiedades del álgebra de Boole se puede llegar a un resultado equivalente.

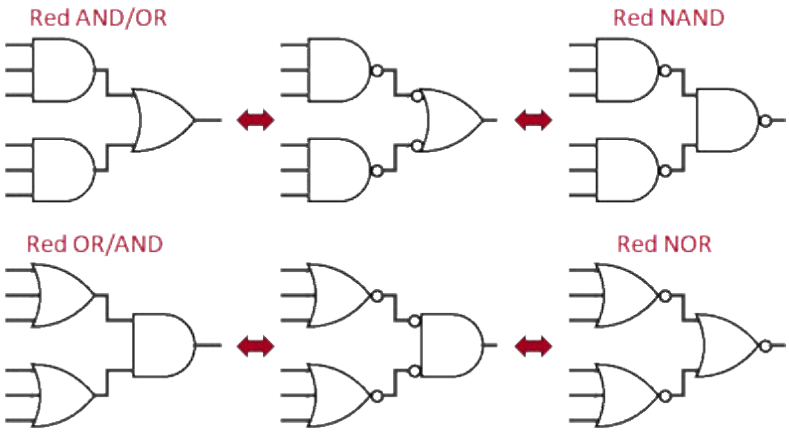


Figura 4.9: Equivalencia entre redes AND/OR con redes NAND y OR/AND con redes NOR.

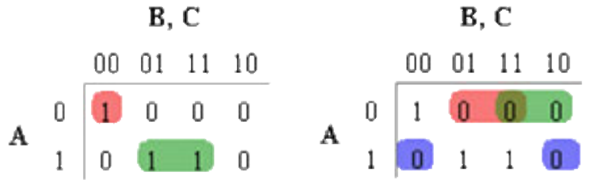
Corolario 4.2.2 — Expresión de formas SOP y POS mediante operaciones NAND y NOR. Cualquier forma SOP se puede transformar empleando las propiedades y teoremas del álgebra de Boole para contener solo operaciones de tipo NAND. De modo dual, cualquier forma POS se puede transformar para contener solo operaciones de tipo NOR.

Para realizar las transformaciones mencionadas basta con aplicar teorema de involución (doble negación) y posteriormente aplicar las leyes de De Morgan, para convertir las sumas en productos negados (NAND) y los productos en sumas negadas (NOR).

Ejemplo 4.3 — Implementación de funciones mediante puertas lógicas universales. Dada la función $f(A,B,C) = \sum m(0,5,7)$, determina sus formas simplificadas SOP y POS empleando K-maps. Emplea los resultados para obtener las expresiones y circuitos equivalentes solo con operaciones NAND y NOR.

Solución:

Puesto que se proporcionan los minitérminos de la función, es posible construir de forma inmediata el K-map y obtener las formas simplificadas SOP y POS.



$$f = A'B'C' + AC \quad \text{(forma SOP)}$$
$$f = (A+C')(A+B')(A'+C) \quad \text{(forma POS)}$$

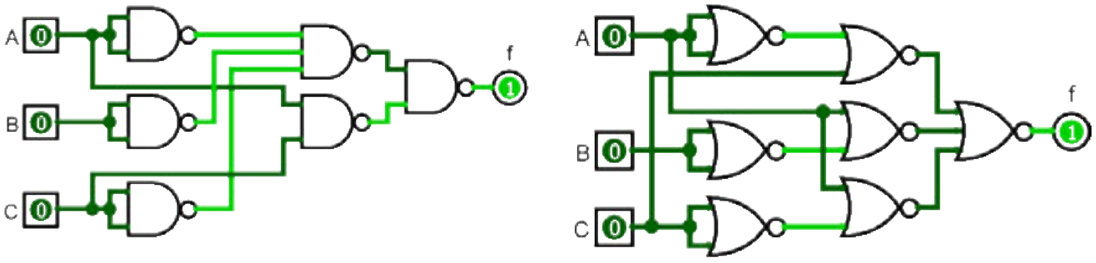
La forma SOP se transforma para expresarla solo con operaciones NAND:

$$\begin{aligned} f &= A'B'C' + AC \\ &= (A'B'C' + AC)'' \quad \leftarrow (\text{T4}) \\ &= ((A'B'C')'(AC)')' \quad \leftarrow (\text{P3}) \end{aligned}$$

La forma POS se transforma para expresarla solo con operaciones NOR:

$$\begin{aligned} f &= (A+C')(A+B')(A'+C) \\ &= ((A+C')(A+B')(A'+C))'' \quad \leftarrow (\text{T4}) \\ &= ((A+C')' + (A+B')' + (A'+C)')' \quad \leftarrow (\text{P3}') \end{aligned}$$

Los circuitos lógicos asociados a las formas SOP y POS empleando solo puertas universales NAND (forma SOP) y NOR (forma POS) quedan:

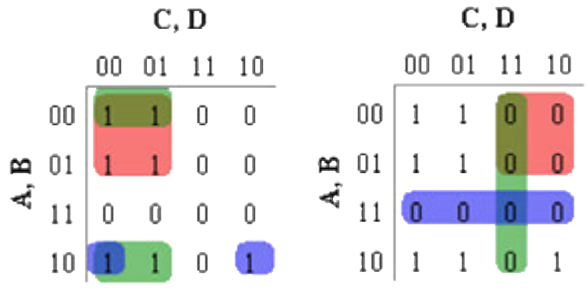


Ejemplo 4.4 — Implementación de funciones mediante puertas lógicas universales. Dada la función $f(A,B,C) = \prod M(2,3,6,7,11,12,13,14,15)$, determina sus formas simplificadas SOP y POS empleando K-maps. Emplea los resultados para obtener las expresiones y circuitos equivalentes solo

con operaciones NAND y NOR.

Solución:

Puesto que se proporcionan los maxitérminos de la función, es posible construir de forma inmediata el K-map y obtener las formas simplificadas SOP y POS.



$$f = A'C' + B'C' + AB'D' \quad \text{(forma SOP)}$$
$$f = (A + C')(C' + D')(A' + B') \quad \text{(forma POS)}$$

Expresadas solo con operaciones NAND y NOR quedan:

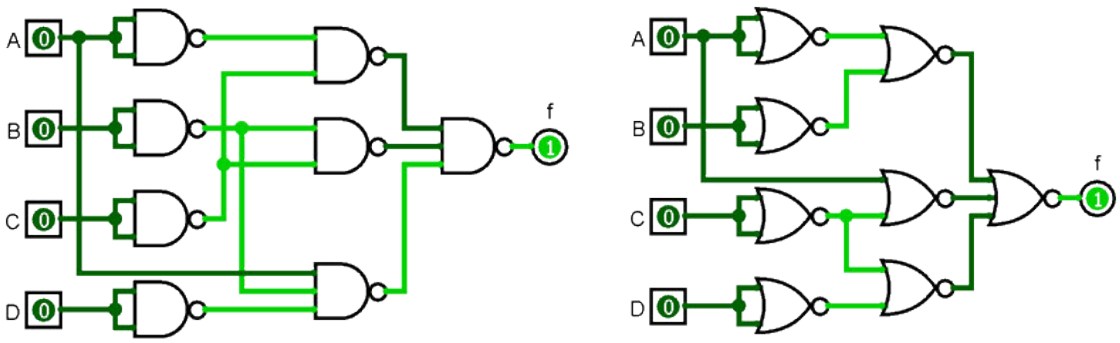
- Forma SOP:

$$\begin{aligned} f &= A'C' + B'C' + AB'D' \\ &= (A'C' + B'C' + AB'D')'' \\ &= (A'C')'(B'C')'(AB'D')')' \end{aligned}$$

- Forma POS:

$$\begin{aligned} f &= (A + C')(C' + D')(A' + B') \\ &= ((A + C')(C' + D')(A' + B'))'' \\ &= ((A + C')'(C' + D')'(A' + B')')' \end{aligned}$$

Las implementaciones de los circuitos equivalentes en los que solo se emplean puertas universales (incluso los inversores):



En la práctica es habitual el requisito de limitar el número de entradas de las puertas (*fan-in*) que utiliza un circuito digital, ya que el uso de puertas con un número elevado de entradas produce circuitos más lentos. Este requisito no supone ninguna dificultad cuando el circuito está constituido por redes AND/OR y OR/AND, puesto que ambos tipos de puertas lógicas satisfacen la propiedad asociativa y, en consecuencia, la concatenación de puertas AND/OR es equivalente a una única puerta AND/OR con las entradas empleadas. Sin embargo, no sucede lo mismo en los circuitos con puertas NAND y NOR, ya que estas puertas no satisfacen la propiedad asociativa. Por tanto, no es posible la concatenación de puertas NAND/NOR para obtener puertas equivalentes de múltiples entradas.

Ejemplo 4.5 — Incumplimiento de propiedad asociativa en puertas NAND y NOR. Demuestra que no es posible obtener un circuito equivalente a una puerta NAND/NOR de tres entradas mediante concatenación de puertas NAND/NOR de dos entradas. Encuentra el modo de obtener la funcionalidad de una puerta NAND/NOR de tres entradas con puertas NAND/NOR de dos entradas.

Solución:

Deseamos validar la equivalencia entre la expresión algebraica derivada de una puerta NAND/NOR de tres entradas con la concatenación de 2 puertas NAND/NOR. Esto es:

$$(ABC)' \equiv ((AB)'C)' \quad (A+B+C)' \equiv ((A+B)'+C)'$$

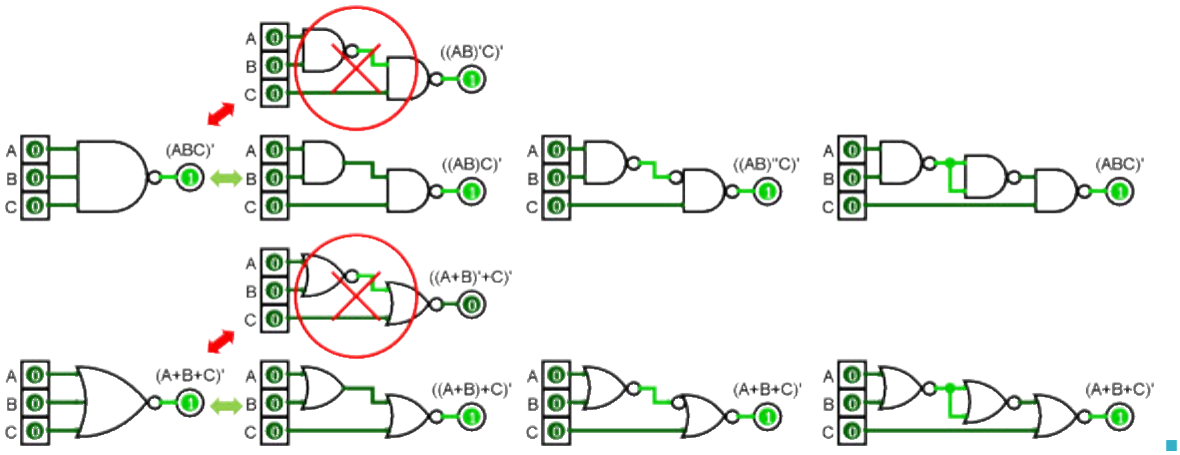
Aplicando las leyes de De Morgan en las expresiones se tiene:

$$(ABC)' = A'+B'+C'$$
$$((AB)'C)' = (A'+B')'+C' = AB+C' \Rightarrow (ABC)' \neq ((AB)'C)'$$

$$(A+B+C)' = A'B'C'$$
$$((A+B)'+C)' = (A'B')'C' = (A+B)C' \Rightarrow (A+B+C)' \neq ((A+B)'+C)'$$

Con lo que quedaría demostrado que la propiedad asociativa no es válida ni para las puertas NAND ni para las puertas NOR.

Para obtener la funcionalidad de una puerta NAND/NOR de tres entradas con puertas NAND/NOR de dos entradas, se puede partir de la concatenación de una puerta AND/OR seguida de una NAND/NOR y despues aplicar doble negación en la salida de la entrada AND/OR para convertirla en NAND/NOR y añadir un inversor en la entrada a la siguiente puerta NAND/NOR. De modo gráfico:

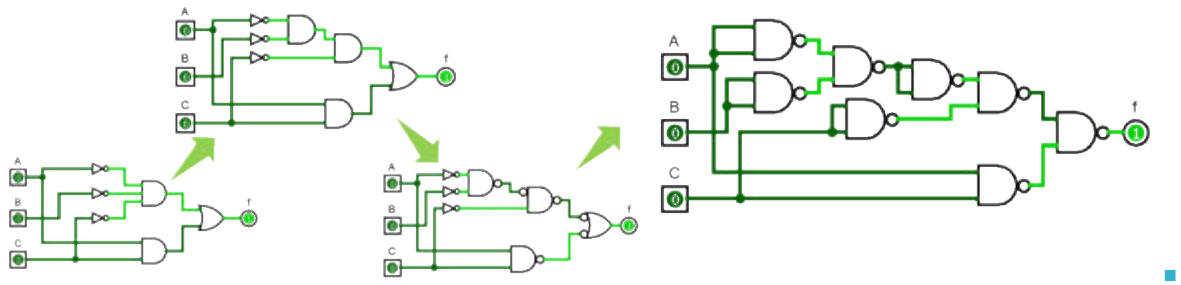


Ejemplo 4.6 — Implementación de formas SOP con puertas NAND de 2 entradas. Implementa la forma simplificada SOP de la función $f(A,B,C)$ obtenida en el ejemplo 4.3 mediante puertas NAND de dos entradas.

$$f(A,B,C) = A'B'C'+AC$$

Solución:

La implementación de la forma simplificada SOP de f es inmediata mediante puertas NAND con un número arbitrario de entradas. En este caso se emplean puertas NAND de dos entradas y una de tres para el producto de tres variables de entrada. Para implementar la función solo con puertas NAND de dos entradas, el producto de tres factores se implementa concatenando dos puertas AND de dos entradas. Este circuito es sencillo de transformar en una concatenación de puertas NAND con la entrada negada, correspondiente a la salida de una NAND previa. La figura ilustra este proceso.



4.3 Análisis de circuitos combinacionales

El análisis de circuitos combinacionales parte de un circuito constituido por puertas lógicas con múltiples entradas y salidas. El objetivo del análisis es determinar las funciones lógicas equivalentes implementadas por dicho circuito y su posible simplificación.

Todo el proceso de análisis de circuitos lógicos se puede llevar a cabo con Logisim empleando las herramientas de análisis que dicho programa software proporciona. De este modo, es posible construir un circuito de partida válido, cuya lógica se puede simular dentro del programa modificando manualmente los valores de las entradas para obtener automáticamente las salidas del circuito. Sin embargo, un modo más completo de obtener el valor de las salidas para todos los valores posibles de las entradas es la obtención de la tabla de verdad de las funciones lógicas implementadas por el circuito.

> La herramienta de análisis de circuitos combinacionales incluida en Logisim proporciona de modo automático tanto la expresión booleana de las funciones asociadas a las salidas del circuito como sus tablas de verdad y las expresiones simplificadas SOP y POS obtenidas mediante K-maps.

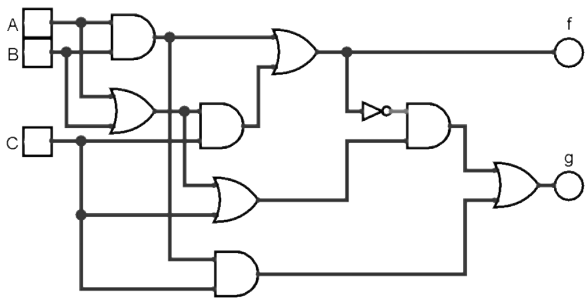
Para interpretar correctamente del análisis de un circuito con Logisim es preciso tener en cuenta el convenio que se emplea para nombrar las señales del circuito en el área de trabajo. En dicha área, las señales obtienen un valor de ponderación decreciente de arriba hacia abajo y de izquierda a derecha. Con el fin de evitar confusiones, se recomienda asignar la etiqueta deseada a cada uno de los pines siguiendo el criterio empleado en esta obra para el nombrado de entradas y salidas de un circuito lógico.

> Recuerda que si no se asignan manualmente etiquetas a las entradas y salidas en los circuitos, Logisin otorga valores de modo automático para el análisis.

Ejemplo 4.7 — Análisis de circuitos lógicos con Logisim. Construye con Logisim el circuito de la figura 4.1 para obtener mediante la utilidad de análisis de circuitos: (1) la tabla de verdad de las funciones $f(A,B,C)$ y $g(A,B,C)$; (2) las expresiones lógicas de las funciones derivadas del circuito; y (3) las expresiones simplificadas de las funciones.

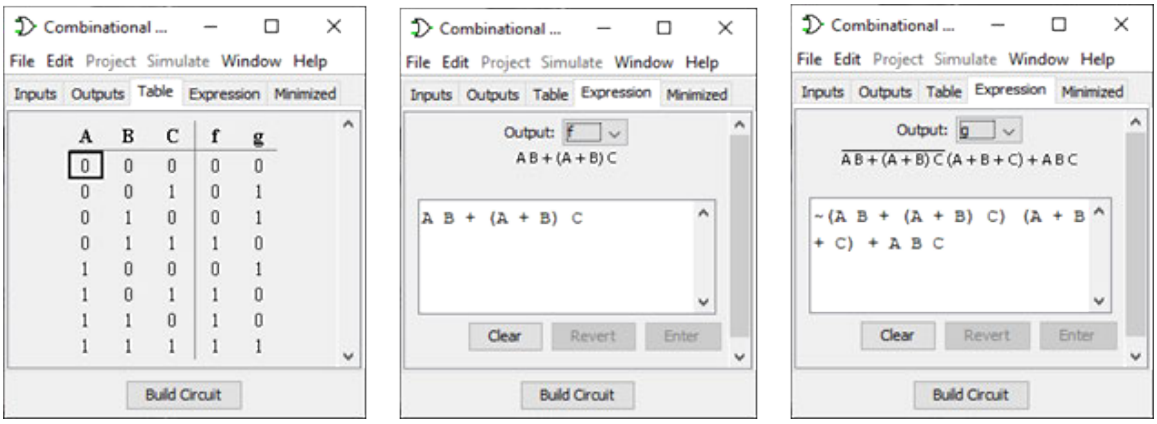
Solución:

En Logisim se construye el circuito de la figura adjunta asignando las etiquetas señaladas en los pines de entrada (A,B,C) y salida (f,g).



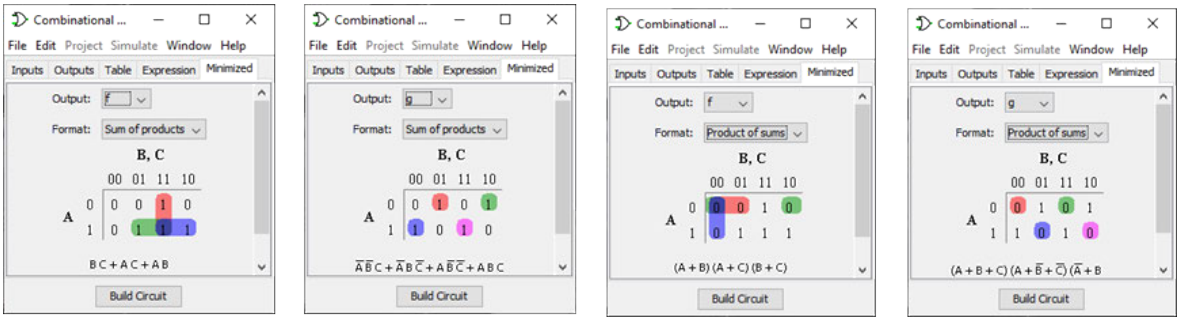
Se inicia la herramienta de análisis del circuito por alguno de los procedimientos alternativos: Project » Analyze Circuit, Window » Combinational Analysis, o clic derecho sobre el nombre de circuito en el menú contextual Analyze Circuit

Según vemos en la figura siguiente, las pestañas *Table* y *Expression* de la herramienta de análisis proporcionan: la tabla de verdad y las expresiones lógicas de las salidas:



$$f = AB + (A+B)C$$
$$g = (AB + (A+B)C)'(A+B+C) + ABC$$

En la pestaña *Minimized* se puede acceder a los K-maps y expresiones simplificadas SOP y POS de las funciones de salida:



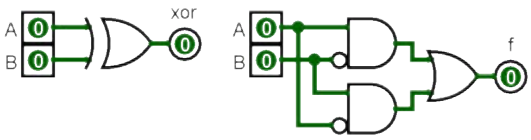
- Formas SOP:
  $f = BC + AC + AB; \quad g = A'B'C + A'BC' + AB'C' + ABC$
- Formas POS:
  $f = (A+B)(A+C)(B+C); \quad g = (A+B+C)(A+B'+C')(A'+B+C')(A'+B'+C)$

Como no es posible efectuar con Logisim el análisis automático de circuitos lógicos si las señales de entrada y salida son tratadas con pines de tipo multibit, estas se deberían separar en sus bits componentes mediante el elemento *splitter*. También se debe evitar el tratamiento de los valores constantes 0/1 como variables, incluyéndolas mediante el elemento *constante* (*Constant*) de la librería *Wiring*. De este modo se evitan las variables innecesarias en el análisis automático con Logisim.

Si se precisa obtener la expresión booleana asociada a un circuito realizado con Logisim, es necesario que en el circuito se omitan las etiquetas de texto de la librería *Base*. Aunque dichas etiquetas facilitan la documentación del circuito, limitan la posibilidad de análisis automático del circuito para acceder a su expresión algebraica. Cuando dichas etiquetas están presentes, es posible obtener tanto la tabla de verdad de la función como sus expresiones simplificadas, pero no la expresión directa obtenida desde el circuito.

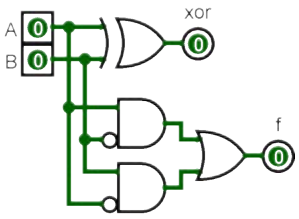
Cuando se emplea Logisim para analizar simultáneamente varios circuitos lógicos presentes en el lienzo de trabajo, es posible emplear los elementos de entrada denominados *túneles* (*tunnels*) de la librería *Wiring*. Estos elementos permiten compartir señales de entrada entre varios circuitos del mismo lienzo de trabajo, sin necesidad de duplicar los pines de entrada para cada circuito produciendo la creación de variables de entrada adicionales en la herramienta de análisis.

Ejemplo 4.8 — Utilización de túneles en Logisim. Comprueba con la herramienta de análisis de Logisim y el empleo de túneles la equivalencia de los circuitos de la figura adjunta.

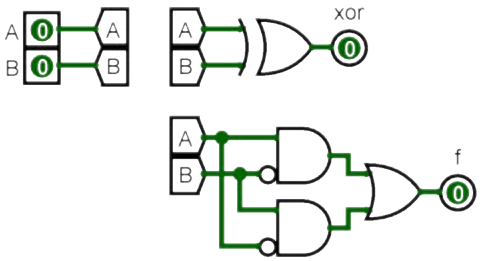


Solución:

Si los dos circuitos proporcionados se encuentran en el mismo lienzo de trabajo, el análisis con Logisim ofrece resultados confusos debido a la duplicación de los pines de entrada A y B. Para evitar la confusión mencionada, se pueden compartir los pines de entrada según se indica en la figura siguiente:



Aunque en este caso el diagrama resultante es muy claro, en circuitos más complejos este método de compartir los pines de entrada produce circuitos de difícil interpretación. El uso de túneles proporciona una herramienta que simplifica el resultado final y su interpretación, como muestra la figura adjunta en la que se añade la tabla de verdad obtenida mediante el análisis automatizado que confirma la equivalencia de los circuitos:



A	B	xor	f
0	0	0	0
0	1	1	1
1	0	1	1
1	1	0	0

La simplificación de funciones lógicas en esta obra se realiza mediante el método de los mapas de Karnaugh o K-maps. El resultado de este método se obtiene automáticamente mediante la herramienta de análisis de Logisim. Sin embargo, es importante advertir que este método no ofrece resultados que permitan implementar los circuitos más óptimos. En general, en la mayoría de sistemas digitales es preciso optimizar conjuntamente numerosas funciones teniendo en cuenta aspectos como: puertas lógicas compartidas para implementar las funciones, restricciones en el número de entradas de las puertas (*fan-in*), empleo de un tipo concreto de puerta lógica, retardos máximos, etcétera. De este modo, la optimización de circuitos se convierte en un problema NP-completo.[2]

> Los algoritmos de optimización empleados en la industria microelectrónica utilizan reglas heurísticas, siendo el algoritmo *ESPRESSO logic minimizer*[3] desarrollado en 1982 por Robert K. Brayton para IBM, uno de los que ha dado lugar a muchas de las variantes integradas actualmente en los programas profesionales de diseño de circuitos lógicos.

La figura 4.10 muestra un ejemplo de minimización de un circuito con dos funciones f y g. A la izquierda, se muestra el circuito resultante que corresponde a la minimización SOP independiente de las funciones. Por el contrario, a la izquierda se realiza la minimización conjunta de ambas funciones. En este último caso la simplificación comparte un número mayor de puertas en las dos funciones.

[2] Problema para el que es imposible encontrar un algoritmo eficiente que obtenga una solución óptima.

[3] Disponible en Internet, URL: `https://is.gd/tFUuM7` como parte del software *Logic Friday* de la Universidad de California.

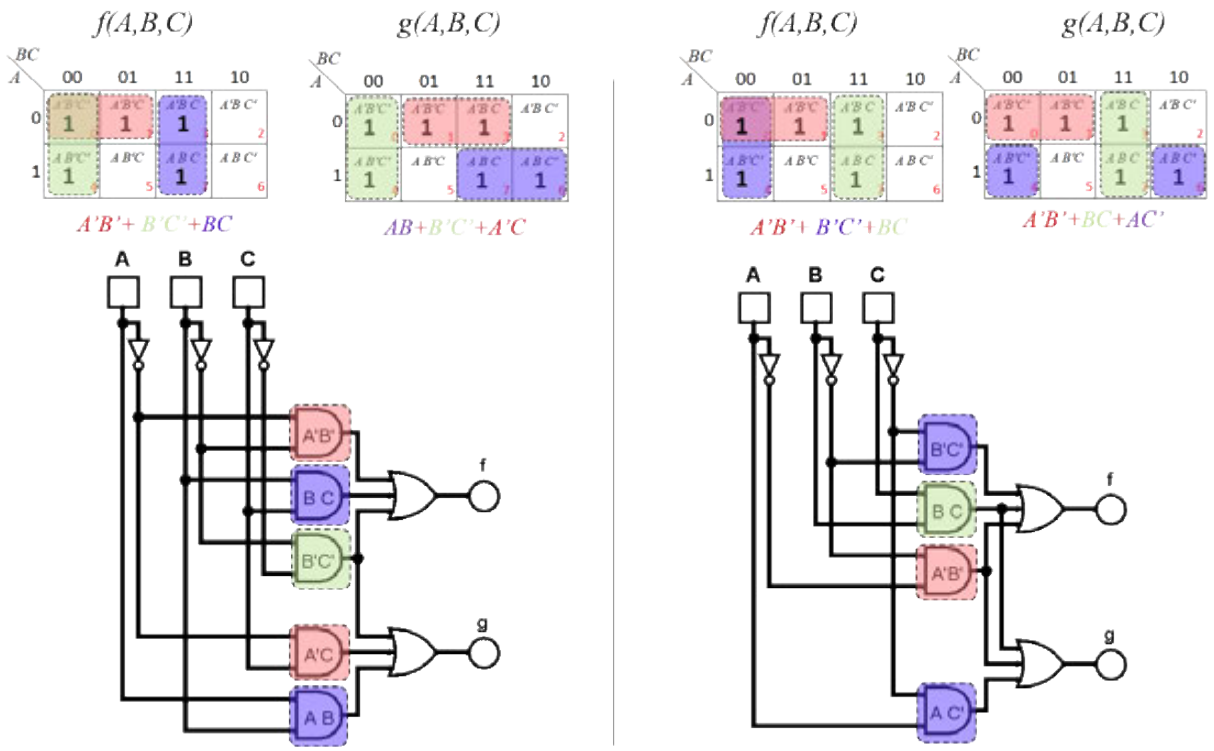


Figura 4.10: Variantes de optimización de circuito lógico con dos funciones.

■ **Ejercicio 4.2** Compara la minimización conjunta SOP y POS de las funciones $f(A,B,C)$ y $g(A,B,C)$ de la figura 4.10, y decide cuál de ellas es la realización óptima. ■

4.4 Diseño de sistemas combinacionales

El diseño de sistemas combinacionales, se plantea como el problema de obtener un circuito digital cuyo comportamiento satisfaga una *especificación* o *descripción funcional* dada. El diseño de circuitos es un proceso que sigue varios pasos en los que se deben tomar decisiones que afectan al resultado final. La figura 4.11 resume las fases del proceso de diseño cuya descripción se explica a continuación.

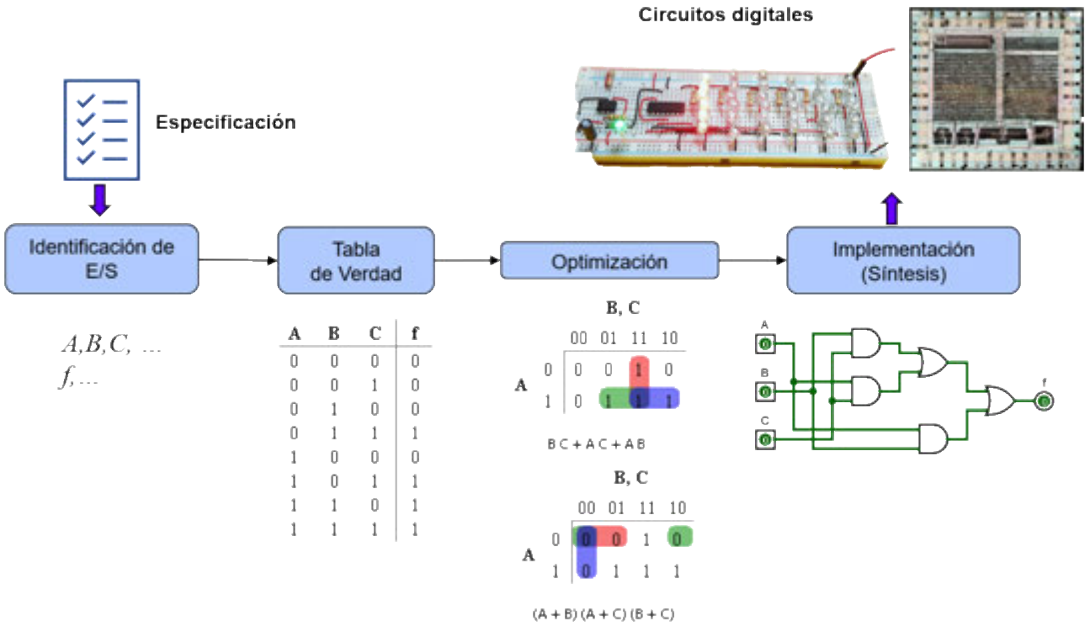


Figura 4.11: Esquema de las fases del proceso de diseño de circuitos combinacionales.

1. **Identificación de variables de entrada y funciones de salida**. A partir de una especificación dada por la descripción textual de la funcionalidad del sistema, se eligen los nombres de las variables de E/S, así como su codificación binaria. Es decir, se decide cómo se interpretan los valores 0 y 1 de las variables de E/S. Si las variables de E/S tienen más de dos posibles valores, es preciso codificar todos los posibles valores mediante variables binarias de tipo multibit. En este caso, cada bit componente se trata como una variable binaria. Por ejemplo, para codificar cuatro posibles valores de entrada, se pueden emplear dos bits A y B para codificar cada uno de los cuatro valores posibles: 00, 01, 10 y 11.

   > Recuerda que las variables de entrada son aquellas que reciben valor de modo independiente al resto. Estas variables están asociadas a las *condiciones* que determinan el comportamiento del sistema. Por contra, el valor de las funciones de salida depende del que tengan las variables de entrada. Se trata de funciones ligadas a las *acciones* o resultados obtenidos en el sistema. La codificación de entradas y salidas otorga significado a sus valores lógicos.

2. **Construcción de la tabla de verdad** para las funciones de salida. La obtención de las tablas de verdad se hace a partir de las condiciones ofrecidas en la especificación inicial del problema. Si dicha especificación asocia a unos valores de entrada sus correspondientes de salida, se trata de una indicación que define directamente la tabla de verdad. Sin embargo, si la especificación contiene declaraciones de tipo *«**SI** (condición) **ENTONCES** (acción)»*, este tipo de descripción se puede convertir en expresiones lógicas con negaciones (NOT), conjunciones (AND o producto lógico) y disyunciones (OR o suma lógica) aplicadas a las variables de entrada. Las expresiones booleanas mencionadas determinan la tabla de verdad. Por ejemplo, si las variables de entrada son A y B, la función de salida es f, y la especificación indica: *«**SI** entradas activadas **ENTONCES** se activa la salida»*, esto se puede traducir en una expresión lógica $f = AB$, asumiendo que la activación se obtiene con valor alto (1).

3. **Optimización de las funciones de salida**. El objetivo es obtener una simplificación de las funciones para que los circuitos que las implementan sean óptimos. En este texto consideramos óptima la forma SOP o POS obtenida mediante el K-map, que precisa menos puertas lógicas para cada una las funciones de salida de modo independiente.[4]

4. **Implementación del circuito** para las funciones de salida minimizadas. La implementación empleará puertas lógicas u otros módulos combinacionales (p. ej., multiplexores, decodificadores, etc.). Las herramientas de diseño automatizado permiten obtener en las fases previas un diagrama del conexionado de los elementos que permite realizar la síntesis del circuito deseado. En el ámbito de este texto, esta fase finaliza con la realización de un diagrama lógico válido del circuito que permita su simulación con Logisim. Para una implementación física del circuito esta fase debe incluir la obtención del diagrama de conexionado eléctrico de componentes o enrutamiento[5] que dependerá de la tecnología elegida, basada en circuitos integrados discretos (familias de circuitos 74xxx) o circuitos lógicos programables (p. ej., FPGA).

Ejemplo 4.9 — Diseño de un circuito de cálculo de mayoría. Diseña un circuito combinacional de obtención de mayoría con 3 bits de entrada. Se trata de un circuito que indica el valor del bit que aparece más veces en la entrada. Es un circuito que suele consistir en una entrada con un número impar de bits para que no sea posible el «empate». Cuando el número de bits de entrada es par, el posible desempate se suele decidir otorgando un peso doble a uno de los bits.

Solución:

Para realizar el diseño se siguen las cuatro fases del proceso de diseño de circuitos combinacionales:

1. **Identificación de variables de E/S**. En este caso hay tres entradas (A, B, C) con los tres bits inspeccionados y una salida (f) que indica el bit que más veces aparece en la entrada.

[4] En la práctica real este proceso es más complejo, pues considera restricciones de espaciales, temporales y energéticas.

[5] El enrutamiento de circuitos electrónicos es un problema NP-completo resuelto de modo heurístico por los programas de diseño hardware.

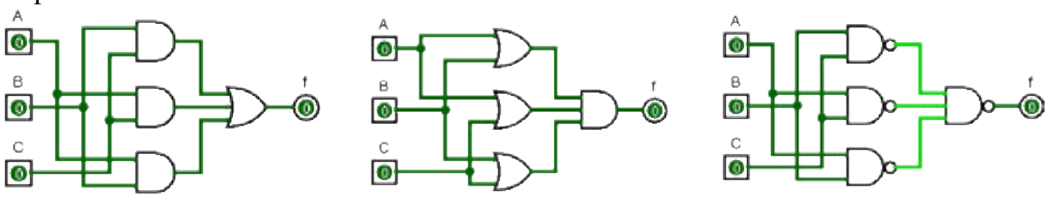
2. **Tabla de verdad**. La tabla de verdad es sencilla de elaborar pues la especificación indica cuál debe ser el valor de la salida para cada valor de la entrada.

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

3. **Optimización**. Con la tabla de verdad se construye el K-map y se obtienen las formas simplificadas SOP y POS de f.

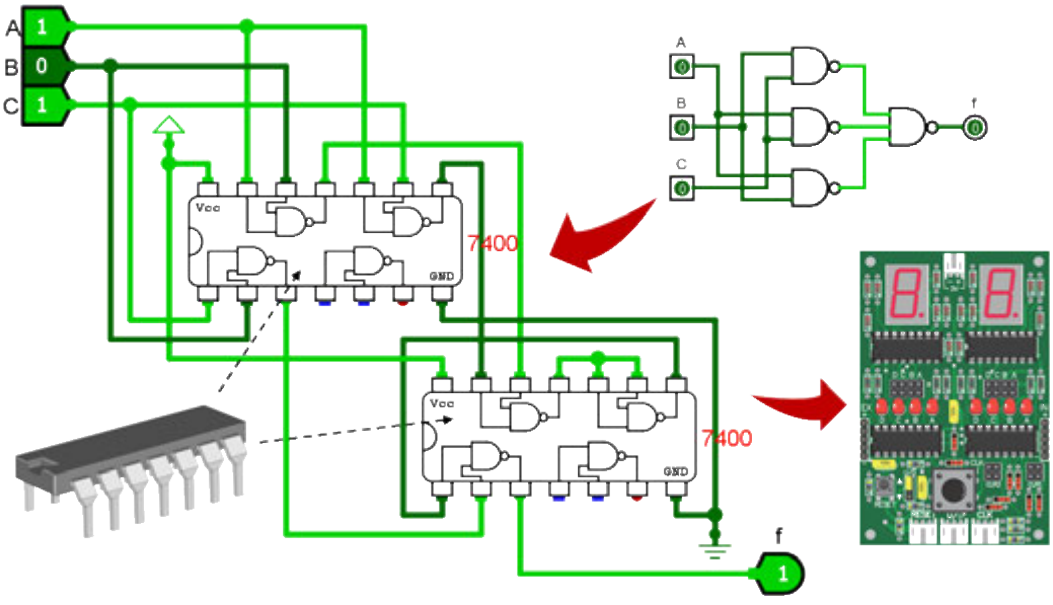
$$f = BC + AC + AB \quad \text{(Forma SOP)}$$
$$f = (A+B) + (A+C) + (B+C) \quad \text{(Forma POS)}$$

4. Implementación.



La figura adjunta muestra, de izquierda a derecha, tres implementaciones alternativas con Logisim: forma simplificada SOP con red AND/OR, forma simplificada POS con red OR/AND y forma simplificada SOP con puertas NAND.

La implementación física del circuito requiere la elección de una tecnología concreta, para determinar el enrutamiento del conexionado entre los componentes. La figura siguiente muestra el esquema de un «posible» enrutamiento empleando circuitos integrados 7400[a] (4 puertas NAND de dos entradas) para montaje en una placa de circuito impreso (PCB).



[a] Lista de circuitos integrados de la serie 7400: https://en.wikipedia.org/wiki/List_of_7400-series_integrated_circuits

En su herramienta de análisis automatizado, Logisim incluye la posibilidad de obtener el diagrama lógico del circuito optimizado. Dicho diagrama se puede generar teniendo en cuenta ciertas restricciones sobre las características de las puertas empleadas: que sean de dos entradas y que solo se empleen puertas de tipo NAND. Aunque el trabajo realizado por Logisim es destacado, presenta una limitación: el uso redundante de inversores. Es decir, se genera un inversor por cada variable negada que aparece en la expresión lógica de la función, incluso aunque se trate de la misma variable.

> Revisa los circuitos lógicos generados automáticamente por Logisim, puede que admitan ligeras modificaciones manuales que optimice su diseño. Por ejemplo, evitando el uso de inversores duplicados.

Ejemplo 4.10 — Diseño de un conversor de BCD a BCD exceso 3 (XS-3). Se debe diseñar un circuito convertidor de código BCD a BCD exceso 3 (o BCD XS-3). El código XS-3 es un código binario de 4 bits que se obtiene sumando 3 al código BCD natural correspondiente.

Solución:

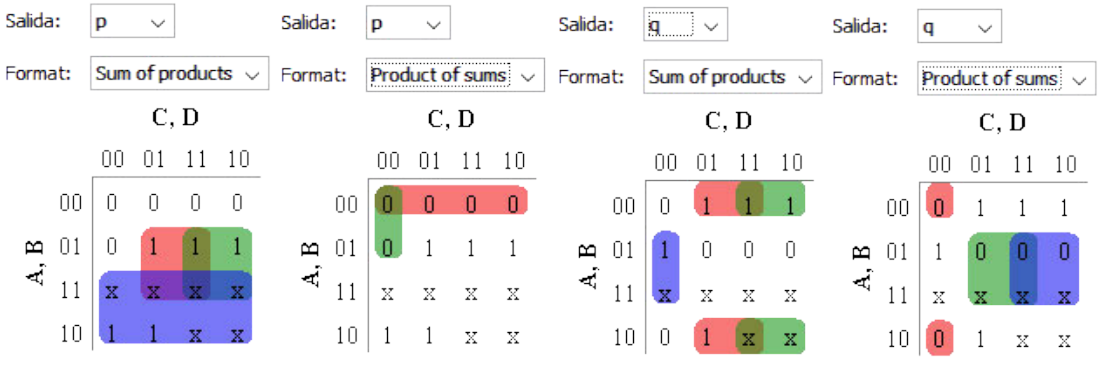
Para llevar a cabo el diseño se siguen los cuatro pasos descritos anteriormente:

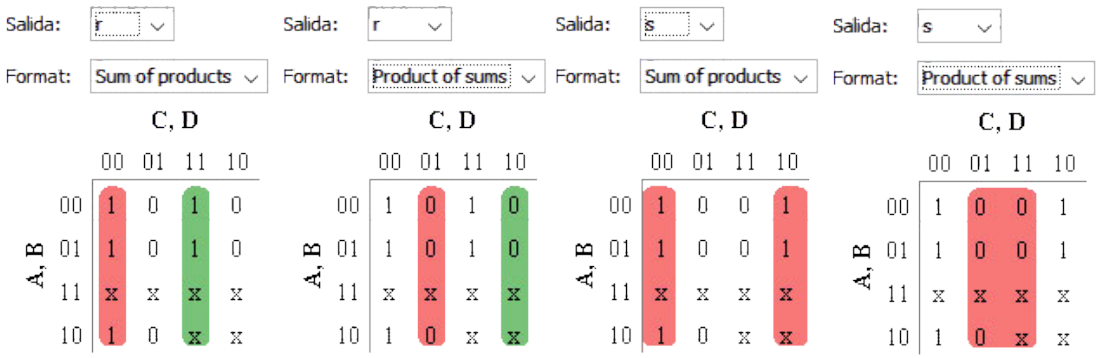
1. Identificación de variables de E/S. En este caso, tanto el código de entrada como el de salida son de 4 bits. Por tanto, tenemos 4 entradas (A,B,C,D) y 4 salidas (p,q,r,s). Observa que las variables se han ordenado según pesos decrecientes. De este modo, las variables más significativas viene dada en la entrada por A y en la salida por p.

2. Tabla de verdad. La tabla de verdad es sencilla de elaborar pues la especificación indica cuál debe ser el valor de la salida para cada valor de la entrada. El código de salida se obtiene como el resultado de sumar 3 al código de entrada. Observa que como en BCD solo se representa hasta el 9, los últimos 6 códigos dan lugar a condiciones libres, ya que para dichas entradas el valor de la función es indiferente.

A	B	C	D	p	q	r	s
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0
1	0	1	0	x	x	x	x
1	0	1	1	x	x	x	x
1	1	0	0	x	x	x	x
1	1	0	1	x	x	x	x
1	1	1	0	x	x	x	x
1	1	1	1	x	x	x	x

3. Optimización. Con la tabla de verdad se construyen los K-maps para las 4 funciones de salida y se obtienen sus formas simplificadas SOP y POS.

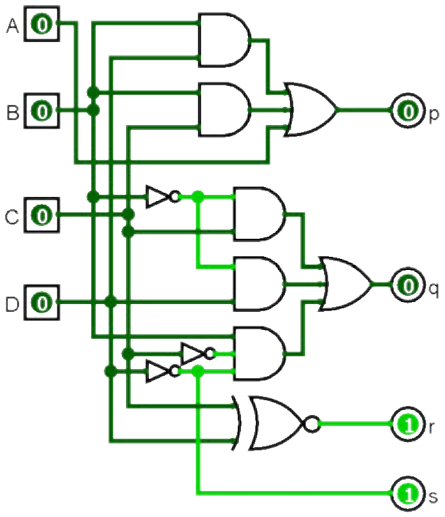




$$
\begin{aligned}
p &= BD + BC + A \quad \text{(Forma SOP)} \\
&= (A + B) + (A + C + D) \quad \text{(Forma POS)} \\
q &= B'D + B'C + BC'D' \quad \text{(Forma SOP)} \\
&= (B + C + D) + (B' + D') + (B' + C') \quad \text{(Forma POS)} \\
r &= C'D' + CD \quad \text{(Forma SOP)} \\
&= (C + D') + (C' + D) \quad \text{(Forma POS)} \\
&= (C \oplus D)' \leftarrow \text{(Implementación con puerta XNOR)} \\
s &= D' \quad \text{(Forma SOP-POS)}
\end{aligned}
$$

Observa que la función s está compuesta de un único elemento. Por tanto, sus formas SOP y POS coinciden.

4. Implementación.

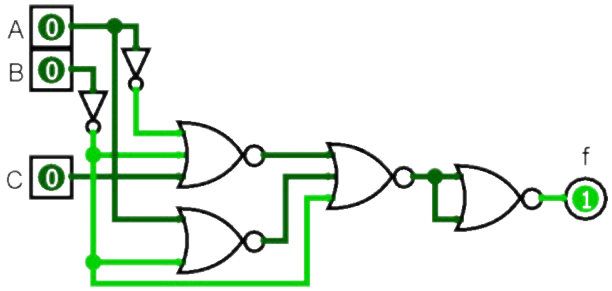


Conceptos clave

- ☑ Implementación de formas SOP y POS mediante redes AND/OR (NAND) y OR/AND (NOR), respectivamente.
- ☑ Diferencia entre análisis y diseño de sistemas digitales.
- ☑ Fases de diseño de sistemas digitales.

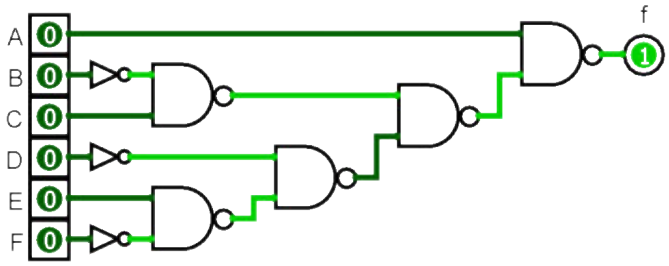
Problemas propuestos

Problema 4.1 ¿Cuál es la función lógica implementada por el circuito de la figura adjunta? ¿Es posible implementar dicha función de un modo más sencillo? Comprueba el resultado analizando el circuito con Logisim.

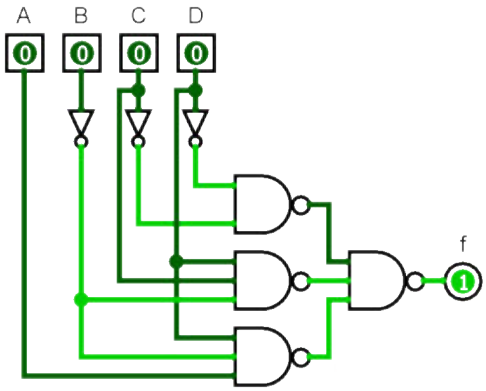


Problema 4.2 Dado el circuito de la figura adjunta, indica cuál de las expresiones siguientes representa una función equivalente al circuito dado:

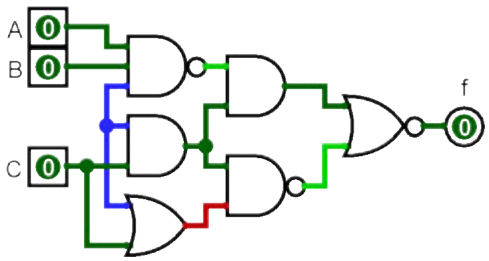
a) $f = A+B+C+D+E+F$
b) $f = A+(B'+C)(D'+EF')$
c) $f = A'+(B+C')(D+EF')$
d) $f = (A'((B+C')+(D+EF')))'$



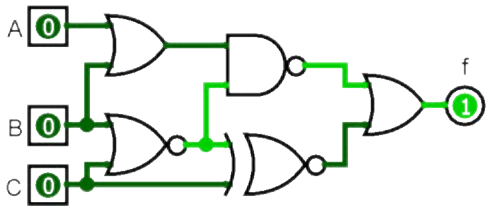
Problema 4.3 ¿Cuál es la función lógica implementada por el circuito de la figura adjunta? Encuentra sus formas canónicas y sus expresiones simplificadas SOP y POS. Comprueba los resultados obtenidos con Logisim.



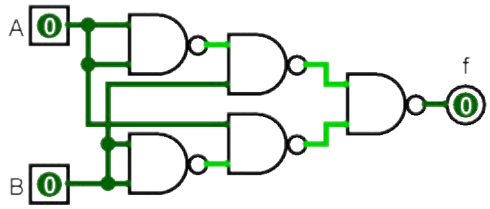
Problema 4.4 Determina la función lógica equivalente al circuito de la figura adjunta.



Problema 4.5 Determina la función lógica equivalente al circuito de la figura adjunta. Indica cuáles son sus formas canónicas y sus expresiones simplificadas SOP y POS equivalentes. Indica cuál sería el circuito lógico equivalente más sencillo.

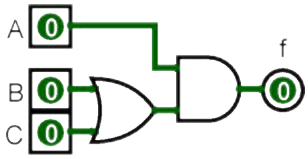


Problema 4.6 Determina la función lógica equivalente al circuito de la figura adjunta. Indica cuáles son sus formas canónicas y sus expresiones simplificadas equivalentes.

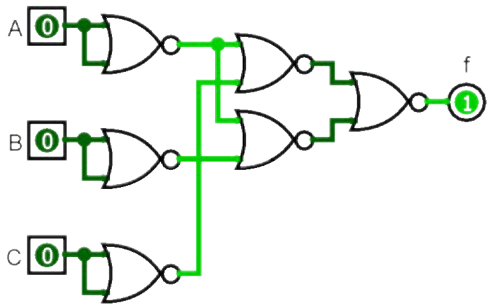


Problema 4.7 Para el circuito de la figura adjunta completa los apartados que se indica:

1.- Calcula las formas canónicas de la función f.
2.- Minimiza la función f obteniendo las expresiones simplificadas SOP y POS.
3.- Implementa f solo con puertas NAND y solo NOR partiendo de las expresiones simplificadas SOP y POS, respectivamente.
4.- Implementa f solo con puertas NAND y solo NOR directamente en el circuito transformando cada puerta en la requerida. Compara los resultados obtenidos con los del apartado anterior.



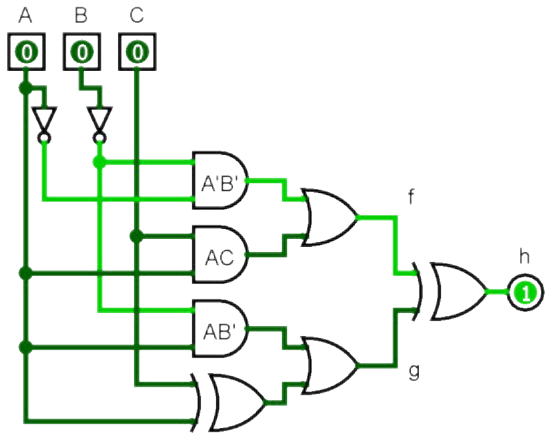
Problema 4.8 Determina la función lógica equivalente del circuito de la figura adjunta, sus formas canónicas y las expresiones simplificadas SOP y POS. Proporciona un circuito final equivalente más simple que el original.



Problema 4.9 Se proporcionan las expresiones de las funciones f, g y la composición de ambas en h, cuyo circuito se muestra en la figura adjunta.

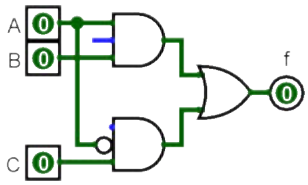
- $f(A,B,C) = A'B' + AC$
- $g(A,B,C) = AB' + A \oplus C$
- $h(A,B,C) = f \oplus g$

Calcula las expresiones canónicas de h a partir de su tabla de verdad obtenida con ayuda de las tablas de verdad correspondiente a f y g. A continuación, simplifica la función h para obtener sus expresiones SOP y POS. Realiza el circuito más sencillo equivalente al circuito original.



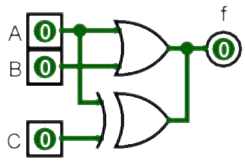
Problema 4.10 Dado el circuito Logisim de la figura adjunta, indicar:

1.- ¿Por qué no es un circuito válido? Comprueba la expresión de la función de salida y su tabla de verdad mediante la herramienta de análisis combinacional de Logisim. Revisa los resultados de la simulación obtenidos con Logisim mediante asignación directa de valores a las entradas.
2.- Propón algún modo de evitar los errores en el circuito.



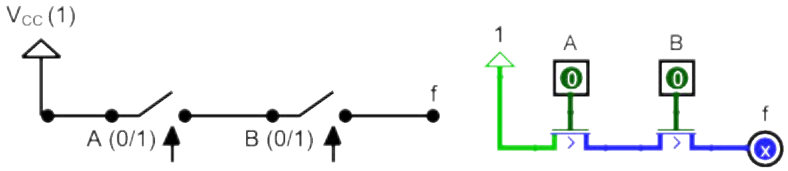
Problema 4.11 Dado el circuito Logisim de la figura adjunta, indica:

1.- ¿Por qué no es un circuito válido? Comprueba la expresión de la función f de salida y su tabla de verdad, mediante la herramienta de análisis combinacional de Logisim. Revisa los resultados de la simulación obtenidos con Logisim mediante asignación directa de valores a las entradas.
2.- Propón algún modo de evitar los errores en el circuito.



Problema 4.12 El diagrama de la figura adjunta corresponde a un circuito lógico constituido por la conexión de dos interruptores en serie gobernados por sendas señales (A, B), de modo que si la señal de gobierno está a nivel bajo (0), el interruptor permanece abierto. De modo dual, el interruptor se cierra cuando la señal de gobierno correspondiente pasa a nivel alto (1). Junto al circuito, se adjunta un circuito Logisim equivalente en el que los interruptores se han sustituido por transistores nMOS cuya funcionalidad es equivalente a la descrita para los interruptores gobernados por señal lógica. Se pide:

1.- Analiza el valor de la señal f de salida del circuito para las distintas combinaciones de valores que pueden presentar las señales de entrada A y B que gobiernan el estado de los interruptores. Presenta el resultado del análisis en un formato de tabla de verdad. El análisis se puede realizar empleando Logisim.
2.- Si se considera f como una función de A y B, ¿es posible implementar esta función mediante puertas lógicas? En caso negativo señala las modificaciones necesarias, para convertir el circuito en la implementación de una función lógica reconocible.



Problema 4.13 Implementa la función $f(A,B,C,D) = AB'CD' + A'BCD' + AB'C'D + A'BC'D$, solo con puertas NAND. Repite la implementación solamente con puertas NOR. Comprueba los resultados con ayuda de Logisim.

Problema 4.14 Dada la función $f(A,B,C,D) = \prod M(1,3,4,5,9,11,12,13)$, calcula:

1.- Primera forma canónica.
2.- Expresiones simplificadas SOP y POS.
3.- Implementación de f solo con puertas NAND y alternativa solo con puertas NOR.

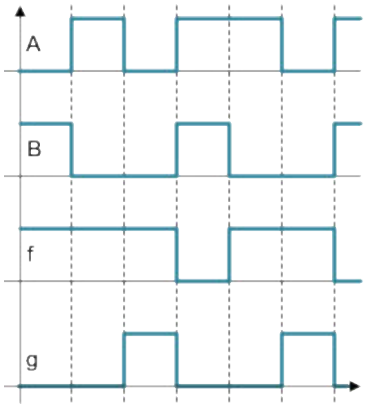
Problema 4.15 Dada la función de cuatro variables:

$$f(A,B,C,D) = [(AB'CD) + (A'BCD) + (AB'C'D) + (A'BC'D)]'$$

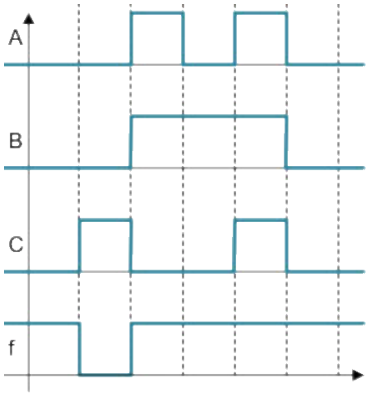
Se desea obtener:

1.- Formas canónicas de f.
2.- Expresiones simplificadas SOP y POS.
3.- Implementación de f solo con puertas NAND y alternativa solo con puertas NOR.

Problema 4.16 Dado el cronograma de la figura adjunta donde $f(A,B)$ y $g(A,B)$ son funciones lógicas de A y B, se pide calcular las expresiones algebraicas correspondientes a f y g.



Problema 4.17 Dado el cronograma de la función $f(A,B,C)$ de la figura adjunta. Obtén la expresión de f indicando si es la única función posible para el cronograma suministrado. Para la función f propuesta, calcula sus formas canónicas y las expresiones simplificadas correspondientes. Propón un circuito con Logisim que genere la función f.

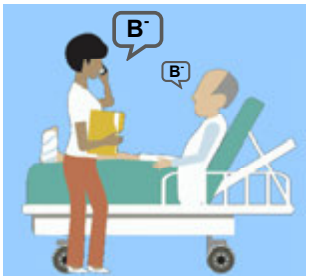


Problema 4.18 Diseña un circuito combinacional con tres entradas y una salida. Esta última debe ser 0 cuando el valor decimal equivalente al valor binario de las entradas es menor que 3, y será 1 en los demás casos. El diseño debe contemplar el cálculo de las expresiones canónicas de la función de salida y las expresiones minimizadas SOP y POS. Propón el circuito lógico más simple para la implementación de f.

Problema 4.19 Diseña un circuito combinacional con tres entradas y tres salidas. En dicho circuito se cumplirá que cuando el valor de entrada en binario natural es 0, 1, 2 o 3, la salida binaria se obtiene sumando 1 a la entrada. Si el valor de la entrada binaria es 4, 5, 6 o 7, la salida binaria se obtiene restando 1 a la entrada. Obtén las expresiones canónicas y simplificadas SOP y POS de las funciones de salida del circuito deseado.

Problema 4.20 Una entidad bancaria desea instalar un sistema de alarma basado en detectores de movimiento. En total se instalarán tres detectores (A, B, C) en las ubicaciones sugeridas por los expertos en seguridad. Para prevenir falsas alarmas provocadas por la activación de un único sensor, la alarma solo se activará cuando al menos dos de los sensores detecten simultáneamente una intrusión. Diseña un circuito lógico combinacional que controle el mecanismo de activación de la alarma.

Problema 4.21 En una población se analizan los grupos sanguíneos de sus individuos para establecer la compatibilidad donante-receptor. Los resultados muestran que los grupos presentes en dicha población son: O^-, AB^+, AB^- y B^-.



Diseña un circuito combinacional capaz de proporcionar una salida f que indique la compatibilidad entre un donante y un receptor. Las entradas del circuito combinacional son el grupo sanguíneo del donante que se codificará con 2 bits denominados D_1 y D_0 y el grupo sanguíneo del receptor codificado también con 2 bits, R_1 y R_0. Para establecer la compatibilidad se sabe que:

- Toda persona puede ser donante-receptor de su propio grupo.
- Una persona del grupo 0^- puede ser donante para receptores de cualquier grupo, pero solo puede ser receptor de donantes O^-.
- Una persona con grupo AB^+ sólo puede ser donante para el grupo AB^+, pero puede ser receptor de cualquier donante.
- Una persona de grupo AB^- puede ser donante para los grupos AB^- o AB^+, y puede ser receptor de donantes O^-, B^- y AB^-.
- Una persona de grupo B^- puede ser donante para los grupos B^-, AB^- y AB^+, y puede ser receptor de donantes O^- o B^-.

Se pide:

1.- Define la codificación de los grupos sanguíneos (la misma para donante y receptor).
2.- Define la tabla de verdad de la función f que establece la compatibilidad donante-receptor.
3.- Encuentra las formas SOP y POS de f, tanto canónicas como simplificadas.
4.- Emplea Logisim para obtener los circuitos de las formas simplificadas SOP y POS.

Problema 4.22 Una máquina de vending dispone de selección de 4 tipos de producto con los precios siguientes: botella de agua (0.50 €), lata de refresco (1 €), paquete de galletas (1.50 €) y caja de bombones (2 €).

La máquina solo acepta una moneda para adquirir el producto. Dicha moneda puede ser de 0.50 €, de 1 € o de 2 €. La máquina está en reposo si no se introduce dinero y no se selecciona producto. Por tanto, asume que al menos se ha seleccionado un producto. Esto es, no es preciso codificar la ausencia de selección. El producto se suministra siempre que se introduce el importe exacto o superior, teniendo en cuenta que:

- La máquina devuelve solo cambio de una moneda.
- Siempre que se introduce el importe exacto del producto seleccionado o se puede devolver cambio exacto en una moneda, se suministra el producto seleccionado.
- Si no puede devolver el cambio correcto en una sola moneda, se devuelve la moneda introducida sin suministrar el producto.
- Si no se introduce moneda, no se suministra producto ni se retorna cambio.

Se pide:

1.- Identifica las variables de entrada, funciones de salida y codificación elegida para ellas.
2.- Elabora la tabla de verdad de las funciones de salida.
3.- Encuentra las formas SOP y POS, tanto canónicas como simplificadas, de las funciones de salida definidas.
4.- Elige una función de salida y para ella realiza los circuitos lógicos empleando: red AND/OR, red OR/AND, sólo puertas NAND y sólo puertas NOR.

5. Módulos combinacionales básicos

Los módulos combinacionales son circuitos que incorporan señales auxiliares que controlan su funcionalidad para su empleo como módulos constructivos en diseños jerarquizados de sistema digitales complejos. Sin embargo, su complejidad es baja/media, ya que poseen un número de puertas lógicas que no supera el centenar. La figura 5.1 muestra una sencilla clasificación de los módulos combinacionales más habituales según su tipo de funcionalidad. En este capítulo se describen las características de los módulos combinacionales del primer bloque (por la izda.), empleados principalmente en la transmisión de datos.

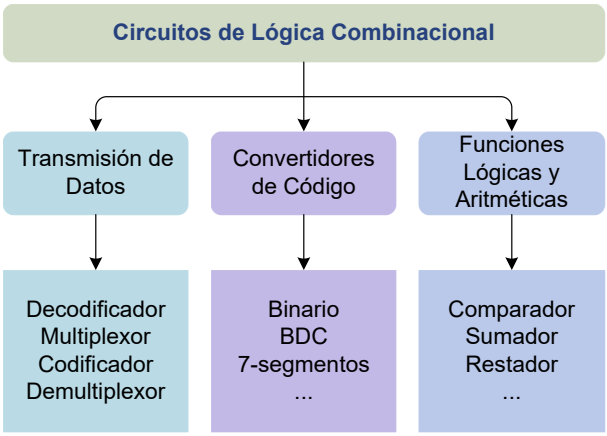


Figura 5.1: Módulos combinacionales agrupados por el tipo de función que implementan.

5.1 El decodificador binario

El *decodificador* $n \times 2^n$ (*decoder*, también conocido como DEC o DECO) es un módulo que tiene n entradas y 2^n salidas. Su función consiste en la activación de la salida correspondiente al código binario que aparece en su entrada. Por tanto, realiza la *decodificación* del código proporcionado en su entrada. El decodificador se puede considerar un módulo que selecciona la línea activada por el código suministrado en su entrada. Por este motivo, a las líneas de entrada se las denomina habitualmente como *entradas de selección*, ya que «seleccionan» la salida activa.

Dado un DEC $n \times 2^n$, se puede suponer que implementa los 2^n productos canónicos asociados a las n variables de entrada. De este modo, para cada combinación de valores de entrada, se activa únicamente el

producto canónico cuya salida es 1 para la combinación de entrada, considerando activación por nivel alto (1).

Corolario 5.1.1 — DEC como generador de minitérminos. Un DEC con salidas activas a nivel alto (1) es un generador de productos canónicos (minitérminos). El producto canónico generado para cada entrada es aquel que vale 1 para la combinación de valores de entrada coincidente con el valor binario del producto canónico generado.

La figura 5.2 muestra el símbolo de un DEC y el módulo correspondiente en Logisim indicando los valores de las salidas como función de las entradas. En ambos representaciones se emplea una señal de control denominada *habilitación* o *enable* (E). Cuando dicha señal es 0 se dice que la salida del DEC está inhibida o deshabilitada. En este caso todas las salidas valen 0, independientemente del valor de las entradas.

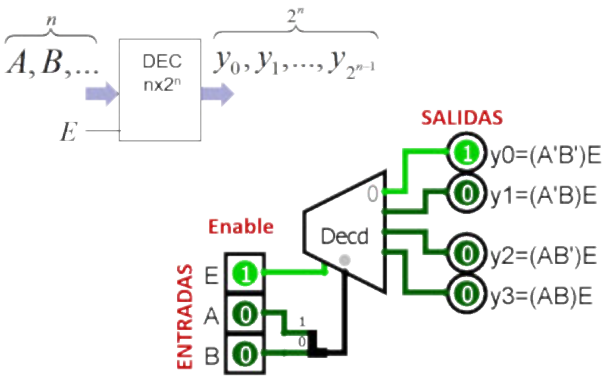


Figura 5.2: Símbolo del decodificador e implementación de DEC 2×4 con Logisim.

> Recuerda que un DEC en Logisim se puede emplear sin señal de *enable*. En este caso, el circuito estaría habilitado siempre. Sin embargo, cuando se emplea señal de *enable*, se debe especificar el comportamiento deseado si el DEC está inhibido o deshabilitado ($E = 0$). En Logisim, el comportamiento por defecto del DEC inhibido es producir salidas en estado flotante. Recuerda que habitualmente el comportamiento deseado en los ejemplos de esta obra es que las salidas sean 0 cuando el DEC está inhibido.

Ejemplo 5.1 — DEC como generador de minitérminos. Dado un DEC 3×8 con entradas A, B, C y salidas activas a nivel alto, indica qué producto canónico se activa a la salida cuando las entradas son $A = 0$, $B = 1$ y $C = 1$.

Solución:

Cuando la entrada es 011 (con el orden ABC) la salida activada a nivel alto (1) es la correspondiente a la decodificación del código de entrada. En este caso particular sería la salida 3. Por tanto, el producto canónico que se activa a la salida es: $m_3 \rightarrow A'BC$. ■

Cuando se emplean decodificadores es fundamental distinguir la ponderación que corresponde tanto a las señales de entrada como a las de salida. De este modo se puede interpretar correctamente el código de entrada y, en consecuencia, la salida activada. En los símbolos del DEC se suele indicar con un '0' la entrada y la salida con menor ponderación (menos significativa).

5.1.1 Implementación de decodificador mediante puertas lógicas

La implementación de un DEC de n entradas y 2^n salidas activas a nivel alto, se consigue mediante los productos canónicos asociados a las variables de entrada. Por tanto, se utilizan 2^n puertas AND y n inversores. Para habilitar o inhibir las salidas de las puertas AND se añade a cada una de ellas una entrada adicional de habilitación (*enable*), de modo que si dicha entrada vale 0, se produce la inhibición de las salidas del DEC.

La figura 5.3 muestra el circuito lógico de un DEC de 2 entradas y 4 salidas con señal de *enable* (E). Todas ellas activadas a nivel alto (1). En dicho circuito se observa que cada salida se corresponde con un producto canónico de las variables de entrada. Cada producto decodifica el código de entrada en el circuito. Además, a cada término producto se añade una entrada de habilitación (E) que lo multiplica (indicado en las puertas AND de la fig. 5.3).

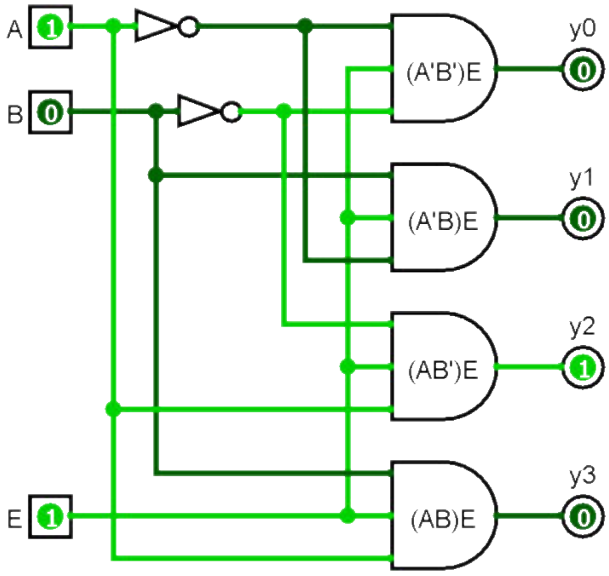


Figura 5.3: Circuito lógico del DEC 2×4 con señales de salida y *enable* activas a nivel alto.

La tabla 5.1 es la tabla de verdad asociada a un DEC 2×4. En ella se indica que si la señal de habilitación está desactivada ($E = 0$) todas las salidas valen 0, independientemente del valor de las entradas. En la tabla también se señala en color rosado, el producto canónico asociado a cada salida. Por tanto, se puede considerar que un DEC con sus salidas activas a nivel alto es un generador de minitérminos, ya que: $y_i = m_i$.

Tabla 5.1: Tabla de verdad del DEC 2×4 con salidas activas a nivel alto.

E	A	B	$(A'B')=m_0$ y_0	$(A'B)=m_1$ y_1	$(AB')=m_2$ y_2	$(AB)=m_3$ y_3
0	x	x	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1

Si se desea implementar un decodificador con salidas activadas a nivel bajo, se debe invertir la salida de las puertas AND. Esto se consigue con el empleo de puertas NAND (ver figura 5.4). Si se tiene en cuenta que la puerta NAND es equivalente a la puerta OR con sus entradas negadas (leyes de De Morgan), se obtiene la decodificación de cada una de las sumas canónicas que corresponden al código de entrada al decodificador. En este caso el DEC se puede considerar un generador de sumas canónicas (maxitérminos).

Si se desea que la habilitación se active a nivel bajo, la señal de *enable* se debe negar antes de entrar en las puertas NAND. De este modo, el circuito se inhabilita con 1 en la entrada *enable* (E), ya que provoca que todas las puertas NAND pongan sus salidas a 1 (ver figura 5.4).

La figura 5.4 muestra el circuito lógico de un DEC 2×4 con salidas y *enable* activos a nivel bajo (0), cuya tabla de verdad se muestra en la tabla 5.2. En dicha tabla también se indica la suma canónica asociada a cada una de las salidas que es decodificada con cada código de entrada. Por tanto, se puede considerar que un DEC con salidas activas a nivel bajo es un generador de maxitérminos, ya que: $y_j = M_j$.

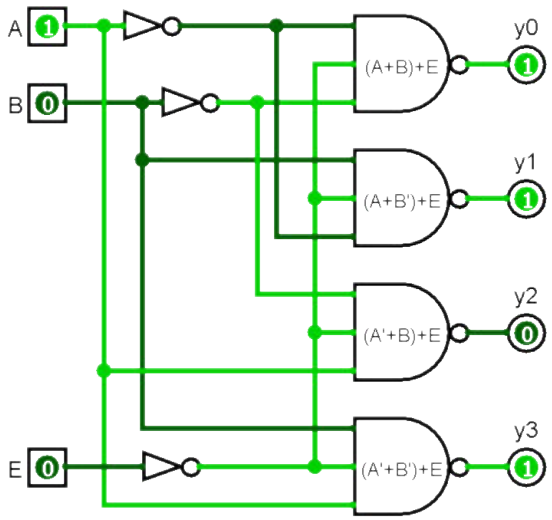


Figura 5.4: Circuito lógico del DEC 2 × 4 con señales de salida y *enable* activas a nivel bajo.

Tabla 5.2: Tabla de verdad del DEC 2 × 4 con salidas activas a nivel bajo.

E	A	B	$(A+B)=M_0$ y_0	$(A+B')=M_1$ y_1	$(A'+B)=M_2$ y_2	$(A'+B')=M_3$ y_3
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0
1	x	x	1	1	1	1

Corolario 5.1.2 — DEC como generador de maxitérminos. El decodificador con salidas activas a nivel bajo (0) se puede considerar un generador de sumas canónicas (maxitérminos). La suma canónica generada es la que vale 0 para la combinación de valores de entrada.

> Recuerda que los módulos DEC en Logisim no se pueden configurar con salidas activas a nivel bajo. Para obtener dicho comportamiento se conectará un inversor en cada salida (ver figura 5.5). Las señales de *enable* en estado flotante (sin conectar) producen habilitación incondicional, siempre activa, aunque en los atributos del módulo es posible configurar la eliminación de dicha señal para evitar ambigüedad.

La figura 5.5 muestra el símbolo para el DEC con señal *enable* y salidas activas a nivel bajo. En dicho símbolo la existencia de un inversor se indica mediante un símbolo de burbuja de inversión ligado a la conexión (○). En la figura se indica con un '0' tanto la entrada como la salida menos significativa.

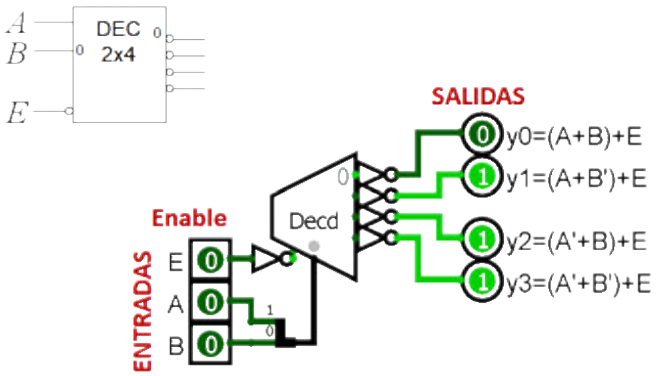


Figura 5.5: Símbolo para DEC 2 × 4 con *enable* (E) y salidas activas a nivel bajo junto a módulo correspondiente en Logisim.

Ejemplo 5.2 — DEC como generador de maxitérminos. Dado un DEC 3×8 con entradas A,B,C y salidas activas a nivel bajo, indica qué suma canónica se activa a la salida cuando la entrada es $A = 1$, $B = 0$ y $C = 1$.

Solución:

Cuando la entrada es 101 (con el orden ABC) la salida que se activa es la correspondiente a la decodificación del código de entrada. En este caso se trata de la salida 5. Como dicha salida se selecciona mediante el valor 0, esta salida activa la suma canónica correspondiente con $A = 1$, $B = 0$ y $C = 1$, que es: $M_5 \rightarrow A' + B + C'$. ■

Una aplicación habitual del DEC es la habilitación selectiva de subsistemas digitales que comparten recursos. En la figura 5.6 se muestra el diagrama lógico de un sistema computador que emplea el circuito integrado 74HC154 —correspondiente a un DEC 4×16— para generar la habilitación selectiva de los periféricos que comparten el bus de datos de un computador. Cuando la CPU precisa acceder o recibir datos de un dispositivo de E/S específico, envía al DEC una solicitud. Junto a dicha solicitud, se introduce —en un registro de direcciones— el valor de la dirección del puerto donde se encuentra conectado el dispositivo que se desea habilitar. El DEC selecciona el dispositivo requerido mediante activación de la señal que lo habilita y desactivación de las señales que inhiben al resto de dispositivos. En esta aplicación la habilitación de un dispositivo periférico deja su E/S conectada al bus de datos, mientras que su inhibición la pone en estado flotante o de alta impedancia.

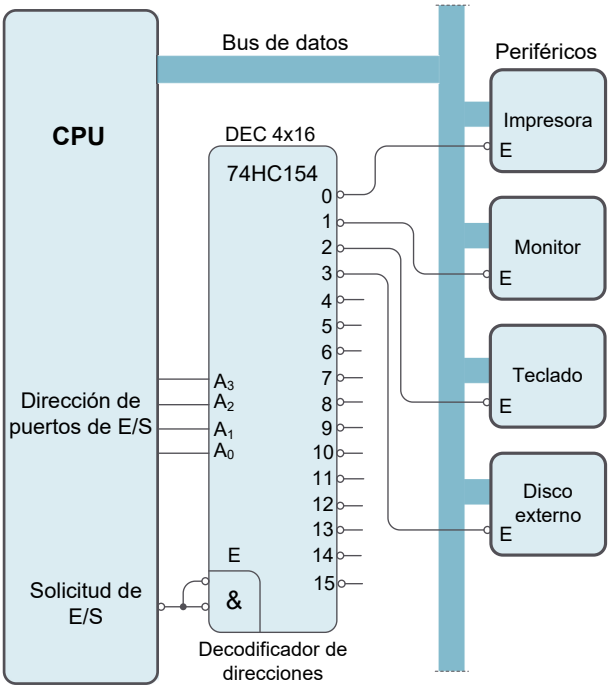


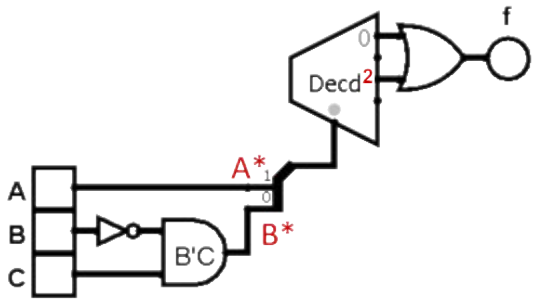
Figura 5.6: Aplicación de un circuito integrado correspondiente a un DEC 4×16 comercial (74HC154) para la habilitación selectiva de periféricos en un sistema computador.

5.1.2 Análisis de circuitos con decodificadores

El análisis de circuitos con DEC debe contemplar su condición de generador de términos canónicos para sus variables de entrada. Si además existe señal de habilitación (*enable*), esta afectará a cada uno de los términos canónicos generados (ver figuras 5.2 y 5.5).

Las señales conectadas a la entrada de un DEC permiten el cálculo de términos canónicos generados por cada una de sus salidas, teniendo en cuenta que serán productos o sumas, si el DEC activa sus salidas por nivel alto o bajo, respectivamente. Las operaciones lógicas sobre las salidas del DEC se aplican a los términos canónicos correspondientes. A continuación se incluyen varios ejemplos de análisis de circuitos con DEC.

Ejemplo 5.3 — Análisis de circuito con DEC de salidas activas a nivel alto. Dado el circuito de la figura adjunta, realiza su análisis para obtener la expresión algebraica de la función $f(A,B,C)$.



Solución:

El DEC se puede tratar como un generador de productos canónicos, ya que sus salidas se activan a nivel alto. Dichos productos no están afectados por la señal de habilitación (*enable*), puesto que dicha señal se ha omitido. Al emplear un DEC 2×4, la función se plantea dependiendo de las dos variables de entrada al DEC, a las que denominamos A^* y B^* cuyo valor depende de las entradas al circuito (A,B,C). Observando del circuito se tiene que:

$$f(A^*,B^*) = m_0 + m_2 \quad \text{con} \quad A^* = A,\ B^* = B'C \quad \text{(puerta AND)}$$

Por tanto, llevando a cabo las sustituciones oportunas, se tiene:

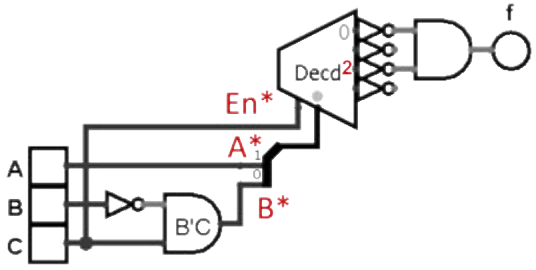
$$m_0 = (A^*)'(B^*)' = A'(B'C)' = A'(B+C')$$
$$m_2 = A^*(B^*)' = A(B'C)' = A(B+C')$$

Operando y aplicando el teorema de absorción (T.7), obtenemos:

$$f(A,B,C) = A'(B+C') + A(B+C') = B+C'$$

Una vez realizado el análisis se comprueba que en realidad f no depende de la variable A. Y el circuito resultante es equivalente a una puerta OR. ■

Ejemplo 5.4 — Análisis de circuito con DEC de salidas activas a nivel bajo. Dado el circuito de la figura adjunta realiza su análisis para obtener la expresión de la función $f(A,B,C)$.



Solución:

Siguiendo un razonamiento análogo al del ejemplo anterior, como el DEC tiene sus salidas activas a nivel bajo, se trata como un circuito generador de sumas canónicas. En este ejemplo además se ha incluido señal de habilitación (En) que afecta a todos los términos canónicos. De este modo, realizando las sustituciones oportunas se tiene:

$$f(A^*,B^*) = (En\cdot m_0)'(En\cdot m_2)' = En' + (M_0M_2), \quad \text{como} \quad A^* = A,\ B^* = B'C,\ En = C$$

Se tiene que:

$$M_0 = (A^*) + (B^*) = A + B'C = A + B'C$$
$$M_2 = (A^*)' + (B^*) = A' + B'C = A' + B'C$$
$$M_0 M_2 = (A + B'C)(A' + B'C) = B'C$$

Operando y aplicando el teorema de absorción (T.7′):

$$f(A,B,C) = C' + B'C = C' + B'$$

■

Ten en cuenta que los ejemplos anteriores se han incluido para ilustrar el método de análisis, puesto que se trata de funciones simples que no requieren el uso de DEC para su implementación.

> Recuerda que en Logisim, los atributos del elemento *splitter* permiten modificar el aspecto de este. También, mediante clic derecho sobre dicho elemento, es posible cambiar el orden ascendente de numeración, fijado por defecto para sus ramas, a descendente. En los diagramas que figuran en este texto se prefiere el orden descendente que coincide con el de las variables que aparecen en el espacio de trabajo. De este modo, el orden obtenido con la herramienta de análisis para las variables de entrada en las tablas de verdad, coincide con el orden en que dichas variables aparecen en el circuito.

■ **Ejercicio 5.1** Emplea Logisim para construir los circuitos de los ejemplos 5.3 y 5.4. Emplea la herramienta de análisis combinacional incluida en Logisim para obtener la función de salida de dichos circuitos y compara los resultados con los obtenidos en el texto. ■

5.1.3 Implementación de funciones lógicas con decodificadores

Ya se ha mostrado que un decodificador es un generador de minitérminos o maxitérminos de sus entradas dependiendo de que sus salidas sean activas a nivel alto o bajo, respectivamente. Como los términos canónicos son los elementos constituyentes de cualquier función lógica, el DEC se puede utilizar para generar cualquier función lógica dependiente de sus variables de selección.

Corolario 5.1.3 — Función lógica a partir de un DEC generador de minitérminos. A partir de un DEC con salidas activas a nivel alto, se puede generar una función lógica dependiente de las variables de selección del DEC empleando una puerta OR que conecte las salidas correspondientes a los minitérminos de la función deseada (forma SOP). Si el número de maxitérminos es menor que el de minitérminos, una alternativa consiste en emplear una puerta NOR para conectar las salidas asociadas a los maxitérminos de la función. Esto es equivalente a implementar la función negada con una puerta OR y añadir un inversor a la salida.

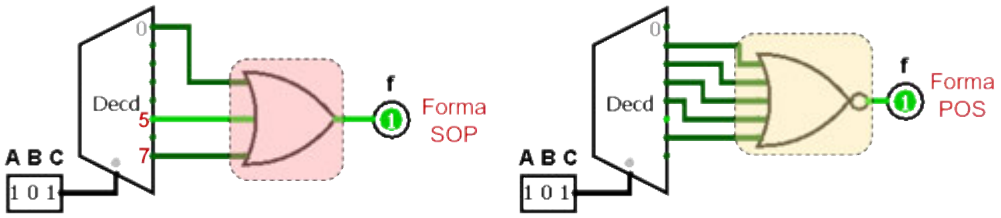
Ejemplo 5.5 — Implementación de una función con DEC generador de minitérminos. Dada la función:

$$f(A,B,C) = \sum m(0,5,7) = \prod M(1,2,3,4,6)$$

realiza su implementación con un DEC 3×8 con salidas activas a nivel alto. No es preciso incluir señal de habilitación.

Solución:

Puesto que el DEC con salidas activas a nivel alto se comporta como un generador de minitérminos, la forma canónica SOP de la función f se puede obtener conectando a una puerta OR las salidas 0, 5 y 7 del DEC. Si se desea implementar la forma POS, se tienen en cuenta los maxitérminos y se cambia la puerta OR por una puerta NOR (ver figura adjunta).



Una posible restricción en la implementación es el uso de puertas lógicas que no superen un determinado número de entradas (*fan-in*). En este caso, la puerta OR se puede sustituir por una concatenación de puertas OR con el número de entradas requerido. Si una puerta OR queda con entradas sobrantes, estas no se dejarán flotantes, sino conectadas a nivel bajo (0) constante o a otra de las entradas de la puerta. En el caso de la implementación con maxitérminos se emplearía también la concatenación de puertas OR, pero la última de ellas se sustituiría por una puerta NOR. ■

Corolario 5.1.4 — Función lógica a partir de un DEC generador de maxitérminos. A partir de un DEC con salidas activas a nivel bajo se puede generar una función lógica dependiente de las variables de selección del DEC empleando una puerta AND que conecte las salidas correspondientes a los maxitérminos de la función deseada (forma POS). Si el número de minitérminos es menor que el de maxitérminos, una alternativa consiste en emplear una puerta NAND para conectar las salidas asociadas a los minitérminos de la función.

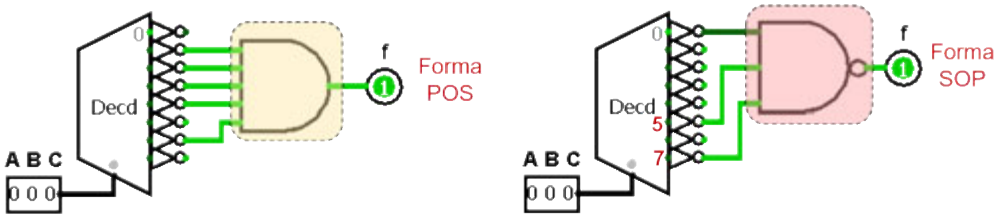
Ejemplo 5.6 — Implementación de una función con DEC generador de maxitérminos. Dada la función:

$$f(A,B,C) = \sum m(0,5,7) = \prod M(1,2,3,4,6)$$

realiza la implementación de dicha función con un DEC 3×8 de salidas activas a nivel bajo. No es preciso incluir señal de habilitación.

Solución:

Puesto que el DEC con salidas activas a nivel bajo se comporta como un generador de maxitérminos, la forma canónica POS de la función f se puede obtener conectando a una puerta AND las salidas 1, 2, 3, 4 y 6 del DEC. Si se desea implementar la forma SOP, se tiene en cuenta los minitérminos y se cambia la puerta AND por una puerta NAND (ver figura adjunta).



Un posible requisito de la implementación es el empleo de puertas lógicas que no superen un determinado número de entradas (*fan-in*). En este caso, la puerta AND se puede sustituir por una concatenación de puertas AND con el número de entradas requerido. Si una puerta AND queda con entradas sobrantes, estas no se dejarán flotantes, sino conectadas a nivel alto (1) constante o a otra de las entradas de la puerta. En el caso de la implementación con minitérminos también se emplearía concatenación de puertas AND, pero la última de ellas se sustituiría por una puerta NAND. ■

> Recuerda que la herramienta de análisis de Logisim se puede emplear en circuitos con DEC, si en estos, las entradas de selección multibit se separan en sus bits componentes mediante el elemento *splitter*.

La figura 5.7 muestra un resumen de las puertas lógicas necesarias para implementar las formas canónicas SOP y POS de una función lógica cuando se emplea un DEC con salidas activas a nivel alto o

bajo, si se emplean las variables de la función buscada como entradas de selección del DEC.

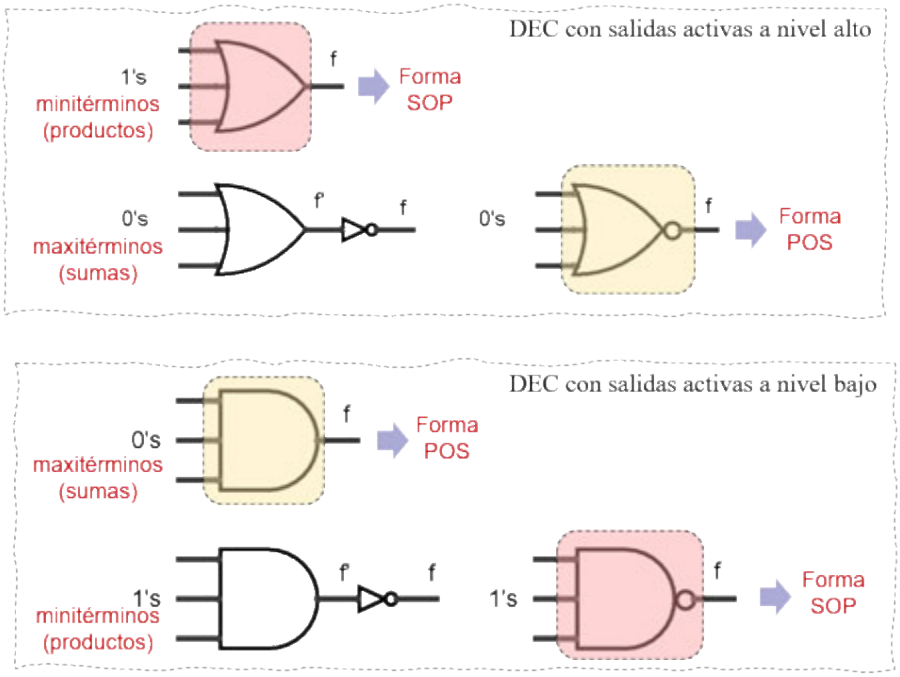


Figura 5.7: Puertas lógicas precisas a la salida de un DEC para implementar las formas canónicas de una función dependiente de las variables de selección del DEC.

5.1.4 Redes modulares con decodificadores

Para obtener la funcionalidad de un DEC con un elevado número de entradas, es posible emplear DEC más sencillos conectados entre sí constituyendo una *red modular de decodificadores*. Las redes de DEC —y en general las de cualquier tipo de módulo— emplean las señales de habilitación (*enable*) de cada módulo para obtener la funcionalidad deseada. A continuación se muestran varios ejemplos de redes modulares con DEC.

La figura 5.8 muestra un DEC 3×8 implementado como una red modular de dos DEC 2×4. En este circuito la señal de entrada más significativa (A) se emplea para seleccionar el DEC apropiado mediante su habilitación. De este modo, si $A = 0$ se habilita el DEC menos significativo (ubicado en la parte superior del circuito —en color rosado—) y cuando $A = 1$, el DEC habilitado es el más significativo (ubicado en la parte inferior del circuito —en color verde—).

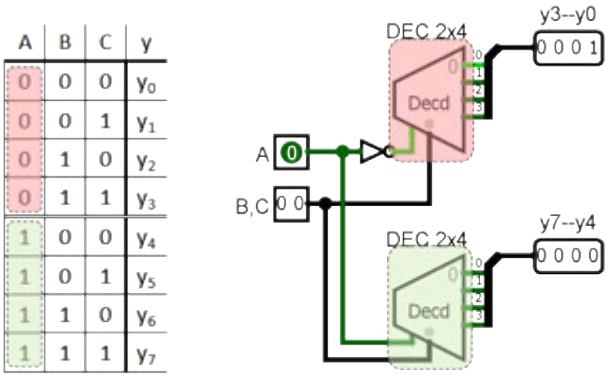


Figura 5.8: Implementación de DEC 3×8 mediante una red de 2 DEC 2×4.

> Recuerda que para la simulación correcta con Logisim de redes de decodificadores, es preciso configurar todos los DEC para que su salida sea 0 cuando están inhibidos (deshabilitados), ya que una configuración con salida en estado flotante produce una simulación con fallos. Dichos errores ocurren cuando una salida en estado flotante se emplea como entrada de otro módulo y no es posible obtener un valor lógico válido.

La habilitación selectiva entre dos módulos se puede llevar a cabo con un simple inversor como se muestra en el circuito de la figura 5.8. Sin embargo, cuando es preciso realizar una habilitación selectiva entre un número de módulos superior a 2, se emplea un DEC auxiliar encargado de generar la señal de habilitación hacia el módulo apropiado. Esta situación se ilustra en la figura 5.9 (izda.) para implementar un DEC 3×8, donde se muestra una red modular que utiliza cuatro DEC 1×2, con un DEC 2×4 adicional que habilita el DEC apropiado para la entrada. En este circuito las dos variables más significativas (A, B) son las entradas al DEC 2×4 encargado de la habilitación de los DEC 1×2.

En el circuito de la figura 5.9 (dcha.) se sigue una estrategia análoga a la descrita en los párrafos previos. En la red modular se obtiene la funcionalidad de un DEC 4×16 con cuatro DEC 2×4. En dicha figura se señala en color cada uno de los DEC habilitado para una combinación concreta de las variables de entrada. En todos estos circuitos, el DEC encargado de generar la habilitación del resto de DEC, está habilitado permanentemente. Aunque la señal de habilitación se podría haber suprimido en dicho DEC, para destacar su habilitación permanente se ha preferido incluir explícitamente una señal de habilitación constante con valor 1.

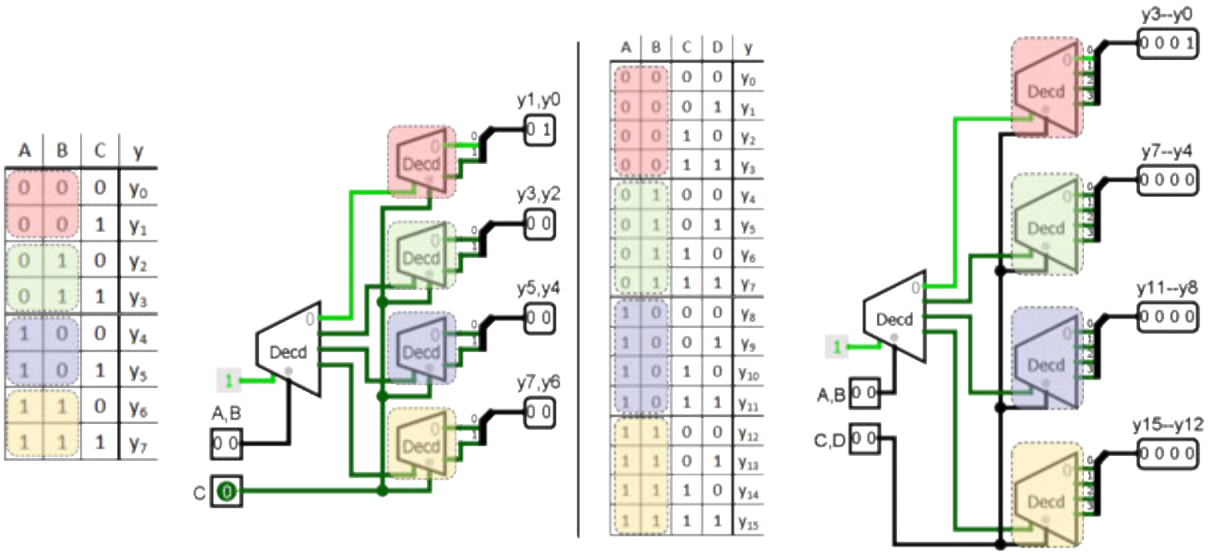


Figura 5.9: Implementación de redes DEC de dos niveles con gestión de habilitación en un nivel.

En la figura 5.10, se muestra un circuito alternativo mediante red con habilitación multinivel, donde el DEC 2×4 se sustituye por tres DEC 1×2. En dicho circuito, cada nivel de DEC comparte como variable de selección una de las variables de entrada. La salida de cada DEC habilita los DEC del siguiente nivel.

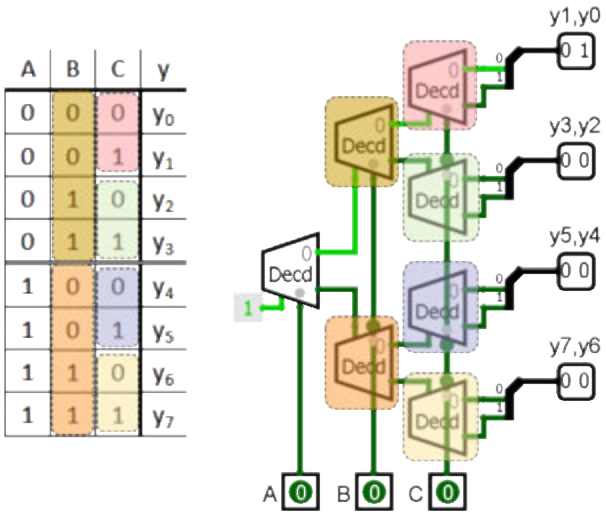


Figura 5.10: Implementación de redes DEC con habilitación multinivel.

> En las redes modulares, todos los DEC deberían poseer el mismo tipo de activación para sus salidas. Si en una red modular se mezclan DEC con salidas activas a nivel alto, con otros cuyas salidas sean activas a nivel bajo, hay que tener mucha precaución para evitar incongruencias en el estado de habilitación de unos y otros.

5.2 El multiplexor

El multiplexor —conocido también como MUX— se emplea en la transmisión de datos para seleccionar el origen de datos a transmitir por un canal de comunicación compartido. La selección del origen de datos se determina por un código de entrada de n bits que es decodificado para activar una de las 2^n líneas de entrada existentes, cuyos datos se transmiten a la salida. La figura 5.11 ilustra la funcionalidad del MUX $2^n \times 1$ con n entradas de selección. Como ejemplo, en dicha figura se muestra el circuito lógico con Logisim correspondiente a un MUX 4×1 con señal de habilitación (E). Como se observa en dicha figura, el módulo tiene tres tipos de entrada:

1. Datos ($D_0, D_1, ..., D_{2^n-1}$). Es la información que se transmite desde las 2^n líneas de entrada a la salida.

2. Selección ($S_0, ..., S_{n-1}$). Constituida por un código de n bits que, mediante su decodificación, selecciona la línea de datos que se transmite a la salida.

3. Habilitación o enable (E). Habilita la salida de datos seleccionada, o bien mantiene la salida a 0, independientemente de la línea de datos seleccionada.

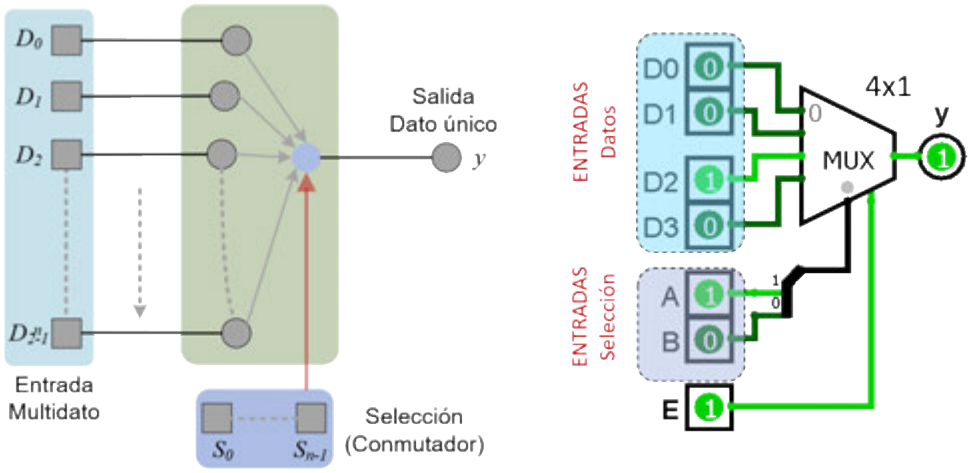


Figura 5.11: Esquema de funcionalidad del multiplexor (izda.) y circuito lógico Logisim de MUX 4×1 con señal de habilitación (dcha.).

> Si en un circuito realizado con Logisim se desea emplear señal de habilitación (*enable*) en el MUX, su salida se puede configurar para valer 0 o pasar a estado flotante cuando se inhabilita el módulo. Si se suprime la señal de habilitación se asume que el módulo está habilitado siempre (equivalente a una conexión constante a 1). El comportamiento deseado se configura en los atributos del MUX. No obstante, si la salida del MUX se va a utilizar como entrada a otro circuito, se recomienda configurar este módulo para que su salida sea 0 cuando está inhabilitado. De este modo se evitan errores en la simulación lógica de los circuitos.

5.2.1 Implementación del multiplexor

Como se muestra en la figura 5.12, la implementación del MUX emplea una red AND/OR. Las puertas AND decodifican los productos canónicos de las entradas de selección al MUX. Dada una combinación concreta de las entradas de selección al MUX, solo se activa la puerta AND que decodifica el producto canónico correspondiente. En consecuencia, a la salida de la puerta AND activada aparece el valor de la entrada de datos presente en dicha puerta. La salida del resto de puertas AND es nula. A la salida del MUX se añade una puerta OR que agrupa la salida de todas las puertas AND.

La señal de habilitación (E) se añade a cada puerta AND como una entrada más para habilitar su salida. A partir del circuito lógico del MUX (ver figura 5.12) es inmediato deducir la expresión booleana de la función lógica de su salida:

$$y = ((A'B')D_0 + (A'B)D_1 + (AB')D_2 + (AB)D_3)E = (m_0D_0 + m_1D_1 + m_2D_2 + m_3D_3)E \qquad (5.1)$$

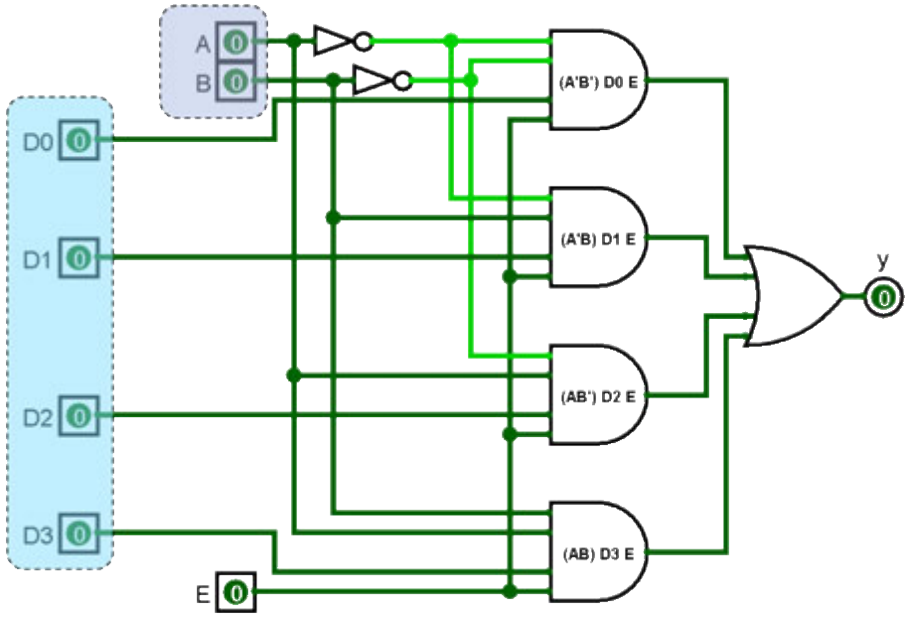


Figura 5.12: Multiplexor 4×1 mediante red AND/OR y señal *enable* (E) activa a nivel alto.

La figura 5.13 muestra el diagrama de conexiones del circuito integrado (CI) comercial 74LS151 correspondiente a un MUX 8×1. En dicha figura, las entradas de selección están etiquetadas como S_2, S_1 y S_0; las entradas de datos son $I_7, \ldots, I_0$, y la entrada de habilitación $\overline{E} \equiv E'$ es activa a nivel bajo. Dicho CI proporciona tanto la salida Z como su valor negado $\overline{Z} \equiv Z'$. En la figura, se señala la numeración de las conexiones (pines) del chip, V_{CC} indica la conexión de alimentación y GND la conexión de tierra (*ground*).

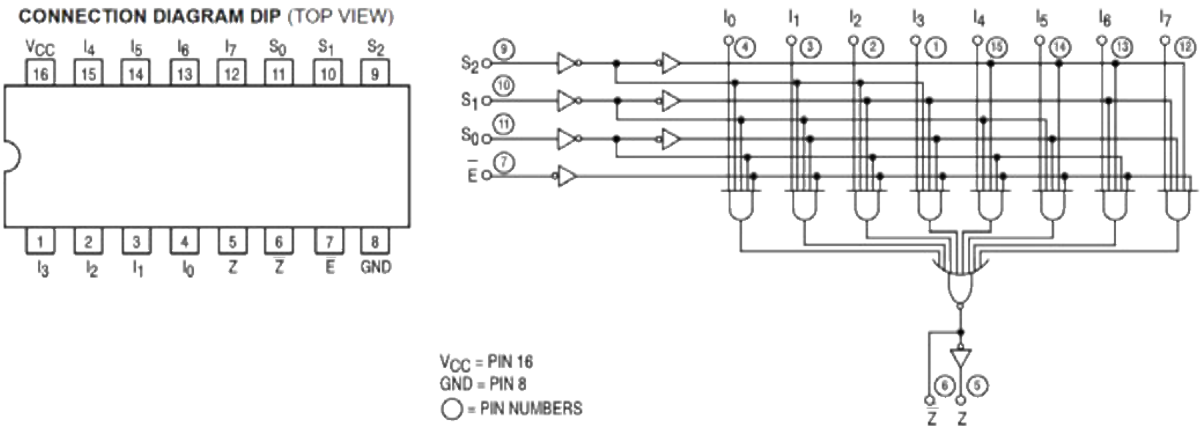


Figura 5.13: Diagrama de conexiones y circuito lógico correspondientes al MUX 8x1 comercializado como el circuito integrado 74LS151 (*Fuente: Motorola*).

■ **Ejercicio 5.2** Realiza una implementación del chip 74LS151 para su simulación lógica con Logisim. Ten en cuenta que los *buffers* con entrada activa a nivel bajo se pueden sustituir por un inversor —a nivel lógico son equivalentes—. Aunque las señales V_{CC} y GND son necesarias para el funcionamiento correcto del circuito físico, su empleo no es preciso para la simulación lógica con Logisim. ■

Al observar el circuito lógico del MUX y la expresión algebraica de su salida (ver ecuación 5.1), se descubre su estrecha relación con el circuito de un DEC. De hecho, el MUX se obtiene añadiendo la entrada de datos correspondiente a cada puerta AND del DEC y una puerta OR a su salida (observa la relación entre DEC y MUX en la figura 5.14).

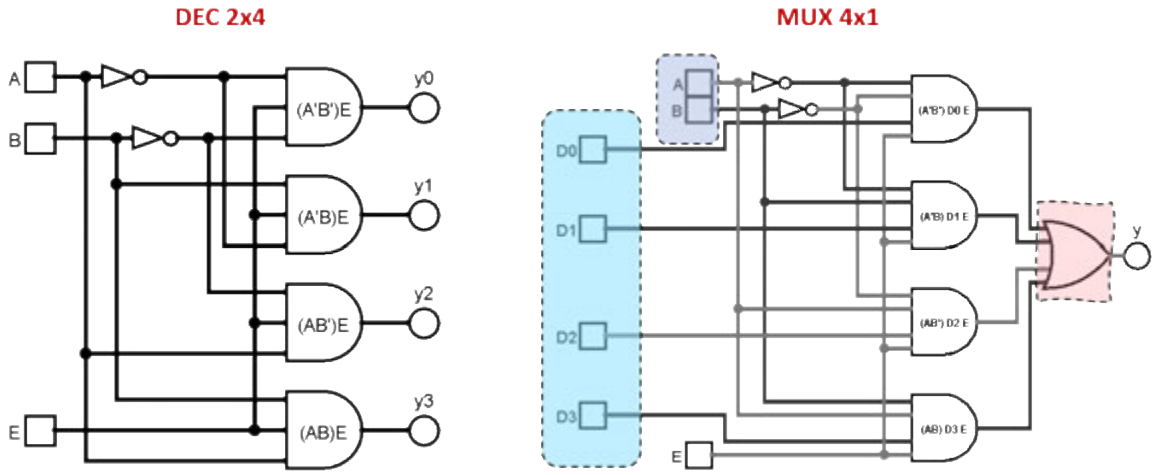


Figura 5.14: Comparación entre los circuitos lógicos del DEC 2×4 y el MUX 4×1.

5.2.2 Teorema de expansión de Shannon aplicado en multiplexores

En la sección 3.6.2 (pág. 68) se explica el *teorema de expansión o descomposición de Shannon*. Dicho teorema señala que toda función lógica admite una expresión de tipo SOP en la que aparecen los productos canónicos de un subconjunto de variables de la función, multiplicados por la evaluación de la función para los valores que hacen 1 cada uno de los productos canónicos mencionados.

Por ejemplo, para una función $f(A,B)$ esta se puede expresar como:

$$f(A,B) = A'B' \cdot f(0,0) + A'B \cdot f(0,1) + AB' \cdot f(1,0) + AB \cdot f(1,1) \tag{5.2}$$

Esta expresión coincide con la forma canónica SOP. Por otra parte, la salida de un MUX 4×1 con entradas de datos D_0, D_1, D_2 y D_3, entradas de selección A y B, y *enable* E es:

$$y(A,B,E) = ((A'B') \cdot D_0 + (A'B) \cdot D_1 + (AB') \cdot D_2 + (AB) \cdot D_3)E \tag{5.3}$$

Si se comparan las ecuaciones 5.2 y 5.3, se deduce que dado un MUX con salida habilitada ($E = 1$), este es capaz de implementar cualquier función $f(A,B)$ siempre que en las entradas de datos del MUX se conecten los valores de la función obtenidos al evaluar sus posibles productos canónicos. Expresado de un modo práctico: $D_0 = f(0,0)$, $D_1 = f(0,1)$, $D_2 = f(1,0)$ y $D_3 = f(1,1)$. Es decir, los valores de la tabla de verdad para f, se conectan en las entradas del MUX. El mismo razonamiento se puede aplicar para implementar la función $f(A,B)$ con un MUX 2×1, teniendo en cuenta que:

$$f(A,B) = A' \cdot f(0,B) + A \cdot f(1,B) = B' \cdot f(A,0) + B \cdot f(A,1) \tag{5.4}$$

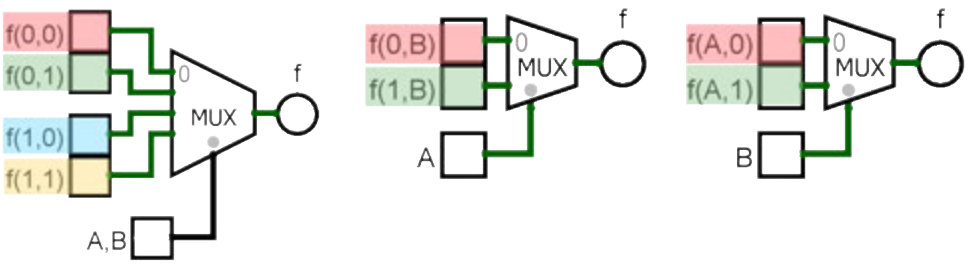
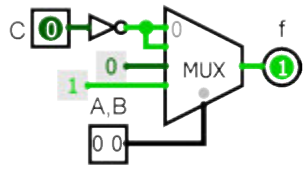


Figura 5.15: Implementación de la función f con MUX aplicando el teorema de expansión de Shannon.

5.2.3 Análisis de circuitos con MUX

El análisis de circuitos que incluyen MUX se realiza teniendo en cuenta la expresión algebraica de la salida del MUX (ver ec. 5.3).

Ejemplo 5.7 — Análisis de un circuito con MUX. Dado el circuito lógico de la figura adjunta, determina la expresión algebraica de la función f y sus formas canónicas.



Solución:

Como el circuito está compuesto de un MUX 4×1, se tiene en cuenta la función de salida de dicho MUX, para obtener:

$$f(A,B) = A'B' \cdot D_0 + A'B \cdot D_1 + AB' \cdot D_2 + AB \cdot D_3 = A'B' \cdot C' + A'B \cdot C' + \cancel{AB' \cdot 0} + AB \cdot 1$$
$$= \underbrace{A'B'C'}_{000\,(0)} + \underbrace{A'BC'}_{010\,(2)} + \underbrace{AB*}_{11\overset{0}{1}\,(6,7)} = \sum m(0,2,6,7) = \prod M(1,3,4,5)$$

■

5.2.4 Implementación de funciones lógicas mediante multiplexores

La salida de un MUX tiene relación directa con la forma canónica SOP de cualquiera de las funciones que es posible construir con las variables de selección del MUX. Dada la expresión de salida del MUX 4×1:

$$y = ((A'B')D_0 + (A'B)D_1 + (AB')D_2 + (AB)D_3)E = (m_0D_0 + m_1D_1 + m_2D_2 + m_3D_3)E$$

si se considera $E = 1$, ajustando a 0 o 1 los valores de D_0, D_1, D_2 y D_3, se puede obtener cualquier expresión canónica dependiente de A y B.

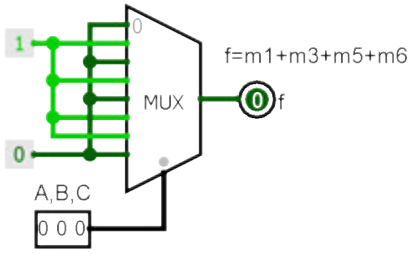
Corolario 5.2.1 — Implementación de funciones con un multiplexor. Para implementar de modo inmediato una función dependiente de n variables con un MUX, cada valor D_i a la entrada del MUX debe coincidir con el valor de la función deseada, en cada fila i de su tabla de verdad.

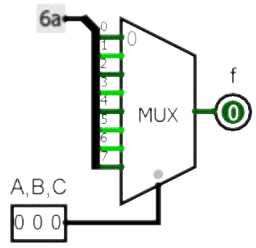
Ejemplo 5.8 — Implementación, con un MUX, de una función a partir de sus formas canónicas. Implementa f mediante un MUX 8×1, a partir de la forma canónica SOP de f:

$$f(A,B,C) = \underbrace{A'B'C}_{001\,(1)} + \underbrace{A'BC}_{011\,(3)} + \underbrace{AB'C}_{101\,(5)} + \underbrace{ABC'}_{110\,(6)} = m_1 + m_3 + m_5 + m_6 = \sum m(1,3,5,6)$$

Solución:

Para implementar la función f con un MUX 8×1, las entradas con el mismo índice de los minitérminos de f se conectan a un valor 1 constante, mientras que el resto de entradas se conectan al valor 0. La figura adjunta muestra el circuito lógico implementado mediante Logisim:





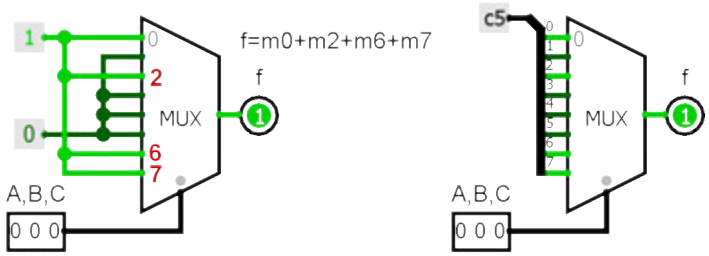
Para conseguir un circuito más compacto en Logisim, en la implementación alternativa equivalente de la derecha se ha utilizado una constante multibit de tamaño 8 con el valor $0110\ 1010 = 6A_{(16}$ conectada a la entrada del multiplexor mediante un elemento separador (*splitter*) de bits. ■

Ejemplo 5.9 — Implementación, con un MUX, de una función a partir de su tabla de verdad. Implementa la función $f(A,B,C)$ con un MUX 8×1 a partir de la tabla de verdad de f:

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	0
1	1	0	1
1	1	1	1

Solución:

La tabla de verdad proporciona el valor que se debe conectar a cada entrada del MUX 8×1. Expresando los valores de la tabla como una constante multibit en hexadecimal, se tiene: $1100\ 0101 = C5_{(16}$. Por tanto, el circuito lógico con Logisim que implementa la función es:



Como se muestra en los ejemplos anteriores, la implementación de una función lógica con un MUX es inmediata a partir de su tabla de verdad o sus formas canónicas, cuando dicha función depende de las mismas variables de selección del MUX. Para evitar errores, es conveniente considerar un orden idéntico para las variables en la conexión al MUX, la tabla de verdad y las formas canónicas de la función.

En la implementación de funciones lógicas con módulos combinacionales, ten en cuenta que un MUX permite implementar una única función, mientras que con un DEC se pueden obtener todas las funciones dependientes de las variables de selección del DEC añadiendo las puertas lógicas oportunas. La ventaja del MUX reside en la sencillez de la implementación resultante.

> La implementación de funciones mediante MUX es habitual en el diseño de circuitos digitales, ya que proporciona un método sencillo de modificación de la funcionalidad del hardware. Si la entrada del multiplexor se conecta a celdas elementales de memoria, es posible reescribir el contenido de dichas celdas para modificar la función implementada por un MUX. Este tipo de metodología de reconfiguración del hardware se observa en los dispositivos lógicos programables avanzados como las FPGA (*Field-Programmable Gate Array*).

El método de implementación de funciones lógicas que se ha descrito requiere que el MUX posea el mismo número de variables de selección que la función a implementar. La cuestión que surge es:

¿es posible emplear un MUX más simple?

La flexibilidad del MUX para implementar una función se pone de manifiesto cuando se permite la conexión de variables en las entradas de datos del MUX. El valor de estas variables no está asociado a los términos canónicos obtenidos en la expresión algebraica de la salida del MUX. En este caso la aplicación del teorema de expansión de Shannon nos indica cuáles son los valores que se deben conectar en la entrada del MUX. Recuerda, que la expansión de la función tiene en cuenta las variables de selección en el MUX.

Al inspeccionar visualmente la tabla de verdad de la función lógica que se desea implementar con el MUX, es posible agrupar las filas asociadas a cada uno de los productos canónicos de las variables de selección del MUX. Para facilitar dicha agrupación, las variables más significativas de la tabla de verdad —columnas más a la izquierda— se eligen como variables de selección del MUX. Para cada agrupación se deriva el valor de la función que se conecta en la entrada correspondiente del MUX. Dichos valores son los mismos que se obtienen mediante expansión de Shannon para la función respecto a las variables de selección elegidas.

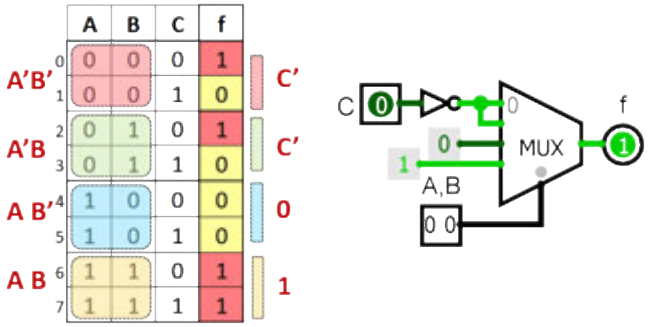
Ejemplo 5.10 — Implementación, con MUX, de función mediante inspección directa de su tabla de verdad. Implementa la función $f(A,B,C) = \sum m(0,2,6,7)$ con un MUX 4×1.

Solución:

Un MUX 4×1 emplea dos variables como entradas de selección. La tercera variable de la función se conectará a las entradas de datos del MUX. Como la implementación se realiza mediante inspección visual directa de la tabla de verdad, se eligen como variables de selección A y B, ya que son las dos variables más significativas. De este modo, en la tabla de verdad se pueden agrupar filas en las que A y B tienen los valores asociados a sus productos canónicos: $A'B'$, $A'B$, AB' y AB. A partir de la tabla de verdad se obtiene el valor de la función f para los valores de las variables A y B, en las agrupaciones asociadas a sus productos canónicos (ver figura adjunta). Así se obtiene que:

$$A = 0,\ B = 0 \Rightarrow f(0,0,C) = C'$$
$$A = 0,\ B = 1 \Rightarrow f(0,1,C) = C'$$
$$A = 1,\ B = 0 \Rightarrow f(1,0,C) = 0$$
$$A = 1,\ B = 1 \Rightarrow f(1,1,C) = 1$$

De este modo se obtiene la implementación de f con un MUX 4×1:



Los valores obtenidos para la función en cada agrupación coinciden son los coeficientes que aparecen en la expresión de f obtenida por aplicación del *teorema de expansión de Shannon* (ver sección 5.2.2, pág. 129). Dicho teorema permite expresar f como:

$$f(A,B,C) = A'B' \cdot f(0,0,C) + A'B \cdot f(0,1,C) + AB' \cdot f(1,0,C) + AB \cdot f(1,1,C)$$

Esta expresión es efectivamente la función de salida del MUX cuando las entradas de datos del mismo se conectan a los valores:

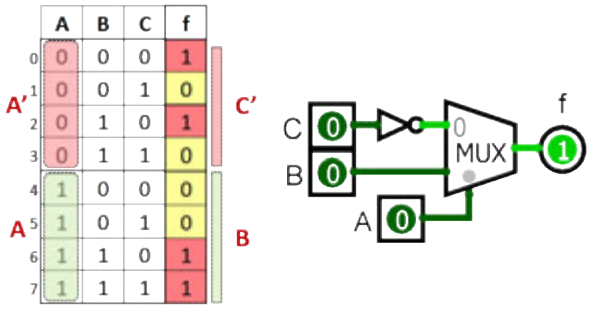
$$D_0 = f(0,0,C), \quad D_1 = f(0,1,C), \quad D_2 = f(1,0,C), \quad D_3 = f(1,1,C)$$

El método seguido se puede aplicar al caso de un MUX 2×1 para implementar la función dada. En este caso, solo la variable más significativa (A) es la que se conecta a la entrada de selección del MUX. Las agrupaciones de 4 filas en la tabla de verdad se asocian a los valores de la función para los dos valores posibles de dicha variable de selección. Para estas agrupaciones se puede calcular el valor de f dependiente de las variables libres (B y C). Este ejemplo con 2 variables libres es sencillo, ya que sigue permitiendo el cálculo de f mediante inspección visual directa de la tabla de verdad. En casos con tres

o más variables libres, se puede emplear la simplificación con K-maps. Mediante la inspección de la tabla de verdad (ver figura adjunta) se obtiene que:

$$A = 0 \Rightarrow f(0,B,C) = C'$$
$$A = 1 \Rightarrow f(1,B,C) = B$$

Como $f(A,B,C) = A' \cdot f(0,B,C) + A \cdot f(1,B,C)$, se obtiene la implementación final que se muestra en la figura siguiente:

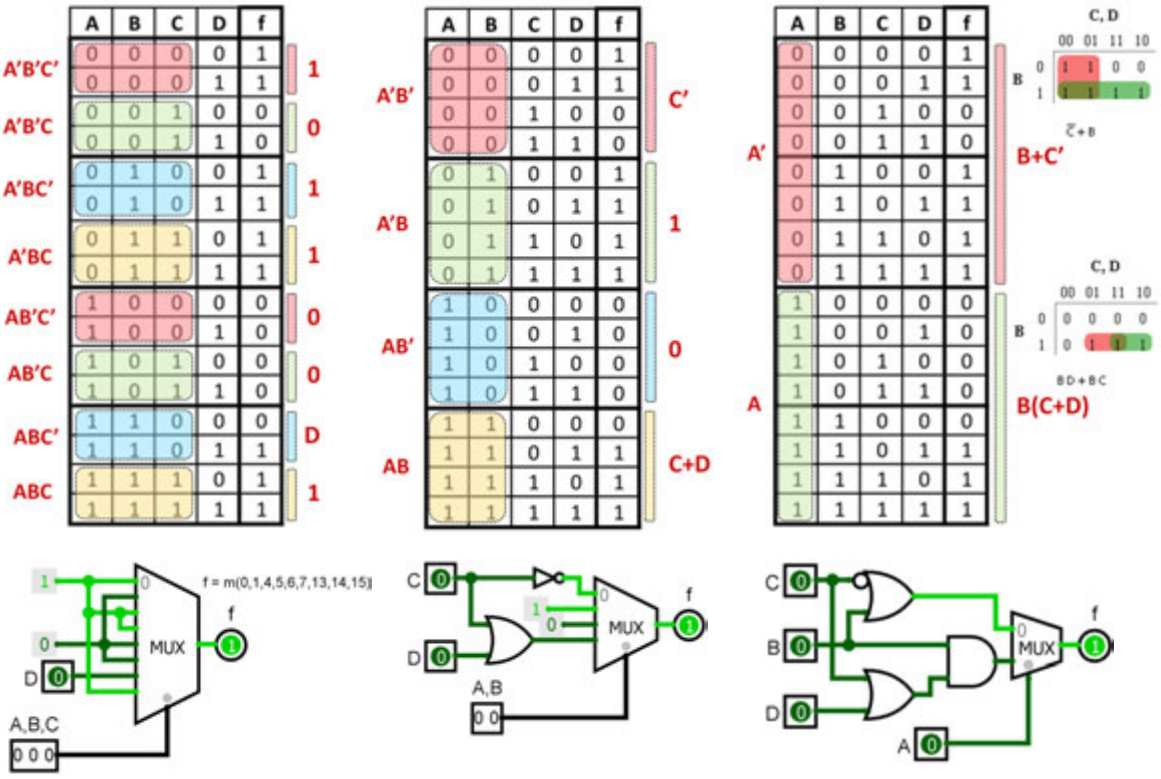


> Recuerda que la inspección visual directa de la tabla de verdad , permite un método bastante rápido y sencillo para implementar con un MUX, funciones de hasta 3 variables. Dicho método es cómodo, incluso con funciones de 4 variables, si las variables de selección se toman desde la posición más significativa de la tabla de verdad.

Ejemplo 5.11 — Implementación de funciones de 4 variables con MUX. Dada la función $f(A,B,C,D) = \sum m(0,1,4,5,6,7,13,14,15)$, realiza su implementación con MUX 8×1, MUX 4×1 y MUX 2×1.

Solución:

En este ejemplo aplicamos la inspección visual directa de la tabla de verdad de f para tomar como variables de selección las más significativas y el resto como libres. La figura adjunta muestra el resultado para los sucesivos casos de MUX 8×1, MUX 4×1 y MUX 2×1.



Las expresiones de f para cada uno de los casos son:

MUX 8x1:

$$A=0,\ B=0,\ C=0 \Rightarrow f(0,0,0,D)=1$$
$$A=0,\ B=0,\ C=1 \Rightarrow f(0,0,1,D)=0$$
$$A=0,\ B=1,\ C=0 \Rightarrow f(0,1,0,D)=1$$
$$A=0,\ B=1,\ C=1 \Rightarrow f(0,1,1,D)=1$$
$$A=1,\ B=0,\ C=0 \Rightarrow f(1,0,0,D)=0$$
$$A=1,\ B=0,\ C=1 \Rightarrow f(1,0,1,D)=0$$
$$A=1,\ B=1,\ C=0 \Rightarrow f(1,1,0,D)=D$$
$$A=1,\ B=1,\ C=1 \Rightarrow f(1,1,1,D)=1$$

MUX 4x1:

$$A=0,\ B=0 \Rightarrow f(0,0,C,D)=C'$$
$$A=0,\ B=1 \Rightarrow f(0,1,C,D)=1$$
$$A=1,\ B=0 \Rightarrow f(1,0,C,D)=0$$
$$A=1,\ B=1 \Rightarrow f(1,1,C,D)=C+D$$

MUX 2x1:

$$A=0 \Rightarrow f(0,B,C,D)=B+C'$$
$$A=0 \Rightarrow f(1,B,C,D)=B(C+D)$$

Los ejemplos anteriores ilustran la implementación de una función lógica con multiplexores a partir de la tabla de verdad de la función. En dichos ejemplos también se pone de manifiesto la relación que existe entre la expansión de Shannon de la función y su implementación mediante un MUX. En realidad, la inspección visual directa de la tabla de verdad permite obtener de un modo rápido los coeficientes que multiplican a los términos canónicos de la función en su expresión mediante expansión de Shannon.

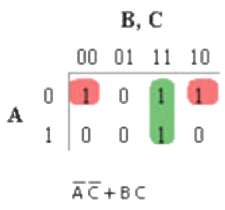
Aunque el método de inspección visual directa de la tabla de verdad es rápido y cómodo, en ocasiones puede resultar más práctico aplicar el método de expansión de Shannon a la expresión algebraica simplificada de la función, si se dispone de esta.

Ejemplo 5.12 — Aplicación de teorema de expansión de Shannon. Implementa con un MUX 2×1, la función $f(A,B,C)=\sum m(0,2,3,7)$.

Solución:

Para emplear un MUX 4×1 es muy rápido recurrir a la inspección visual de la TV tomando como variables de selección A y B. En este caso se tendría que:
$f=A'B'f(0,0,C)+A'Bf(0,1,C)+AB'f(1,0,C)+ABf(1,1,C)$

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



Por inspección visual: $f(0,0,C)=C'$; $f(0,1,C)=1$; $f(1,0,C)=0$; $f(0,0,C)=C$

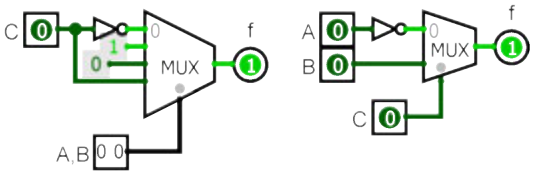
Estos valores son los que se conectan a la entrada del MUX 4×1 (ver figura más abajo). Como la simplificación de la función mediante K-maps proporciona la forma SOP siguiente:

$$f(A,B,C) = A'C' + BC$$

al considerar la expansión de Shannon se puede comparar las expresiones algebraicas para obtener la implementación de f con un MUX 2×1 (ver figura adjunta):

$$f(A,B,C) = C'f(A,B,0) + Cf(A,B,1) \quad \text{con}$$
$$f(A,B,0) = A'; \quad f(A,B,1) = \qquad\qquad B$$

Por tanto, los circuitos resultantes quedan como:



■ **Ejercicio 5.3** Implementa la función f del ejemplo anterior mediante MUX 2×1 e inspección de su tabla de verdad considerando A la variable de selección del MUX. ■

Ejemplo 5.13 — Aplicación de teorema de expansión de Shannon. Implementa mediante MUX 4×1 y MUX 2×1 la función $f(A,B,C,D) = \sum m(0,1,6,8,9,12,13,14)$.

Solución:

La expresión SOP simplificada de f obtenida mediante K-maps es: $f = B'C' + BCD' + AC'$

Aunque B y C no son las variables más significativas, se eligen como variables de selección para la implementación de f con un MUX 4×1, ya que cubren todos los términos de la expresión simplificada de f. De este modo, por aplicación de expansión de Shannon de f respecto a B y C, se tiene que:

$$f = B'C'f(A,0,0,D) + B'Cf(A,0,1,D) + BC'f(A,1,0,D) + BCf(A,1,1,D)$$

Sustituyendo en la expresión SOP simplificada de f se obtienen los coeficientes:

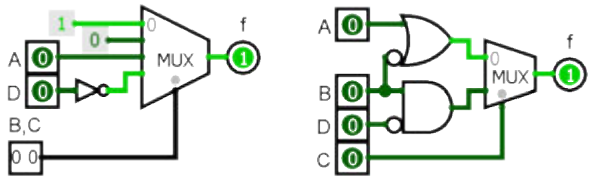
$$f(A,0,0,D) = 1, \quad f(A,0,1,D) = 0, \quad f(A,1,0,D) = A, \quad f(A,1,1,D) = D'$$

Si se desea emplear un MUX 2×1, entonces se tomará C como variable de selección, ya que la expresión algebraica de f se puede escribir:

$$f = B'C' + BCD' + AC' = (B'+A)\cdot C' + (BD')\cdot C; \quad \text{como: } f = C'\cdot f(A,B,0,D) + C\cdot f(A,B,1,D)$$

Se tiene que: $f(A,B,0,D) = B' + A, \quad f(A,B,1,D) = BD'$

Por tanto, los circuitos resultantes quedan:



En las secciones previas se expone la flexibilidad del MUX para implementar funciones lógicas. De hecho, como se muestra en la figura 5.16, solo con MUX e inversores es posible implementar las funciones lógicas elementales (AND y OR), las universales (NAND y NOR) y las secundarias (XOR y XNOR). Esto demuestra que la utilización de MUX sencillos permite componer cualquier función lógica.[1]

[1] Estrategia empleada en las FPGA para componer funciones complejas a partir de módulos lógicos simples.

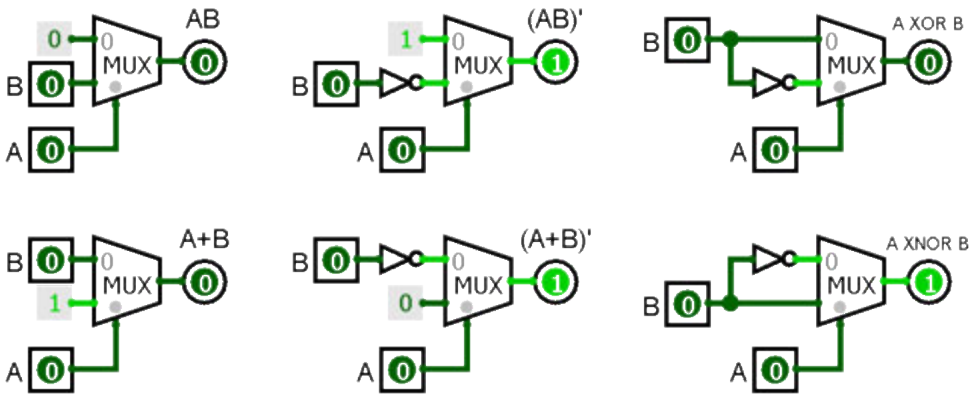


Figura 5.16: Implementación de puertas lógicas de dos entradas con multiplexor 2x1.

■ **Ejercicio 5.4** Implementa A' solo con un MUX 2×1. ■

5.2.5 Redes modulares con multiplexores

Al igual que los decodificadores, los MUX se pueden interconectar en una red modular para obtener la funcionalidad de un multiplexor de orden superior. La conexión de una red modular de MUX puede seguir varias estrategias similares a las mostradas en redes de DEC:

1. Red de dos niveles con gestión de habilitación. Consta de dos niveles. El primero para gestionar la señal de habilitación selectiva de los módulos MUX. En el segundo, la salida se obtiene de una puerta OR cuyas entradas son las salidas de los MUX del nivel previo. Cuando la red posee más de dos MUX en el nivel de salida se emplea un DEC para generar la señal de habilitación hacia cada uno de los MUX de salida (ver figura 5.17 dcha.).

2. Redes multinivel. En este caso, la señal de habilitación no es necesaria, ya que todos los MUX se mantienen siempre habilitados. Por tanto, son redes de implementación más sencilla, debido a que la señal de salida se controla directamente por las señales de selección de los MUX. Son redes multinivel con señales de selección para los MUX compartidas por nivel. Como se observa en la figura 5.18, la estructura sigue fielmente la generación de los valores de la tabla de verdad de las señales de selección. Las variables más significativas se utilizan en orden creciente hacia el nivel de salida de la red.

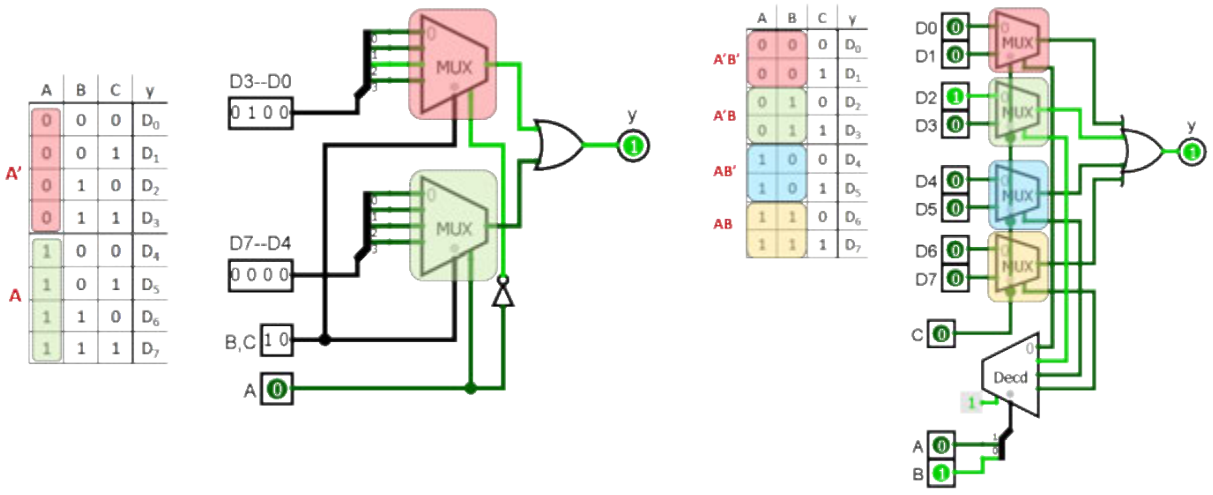


Figura 5.17: Redes de MUX con gestión de señal de habilitación para implementar un MUX 8×1.

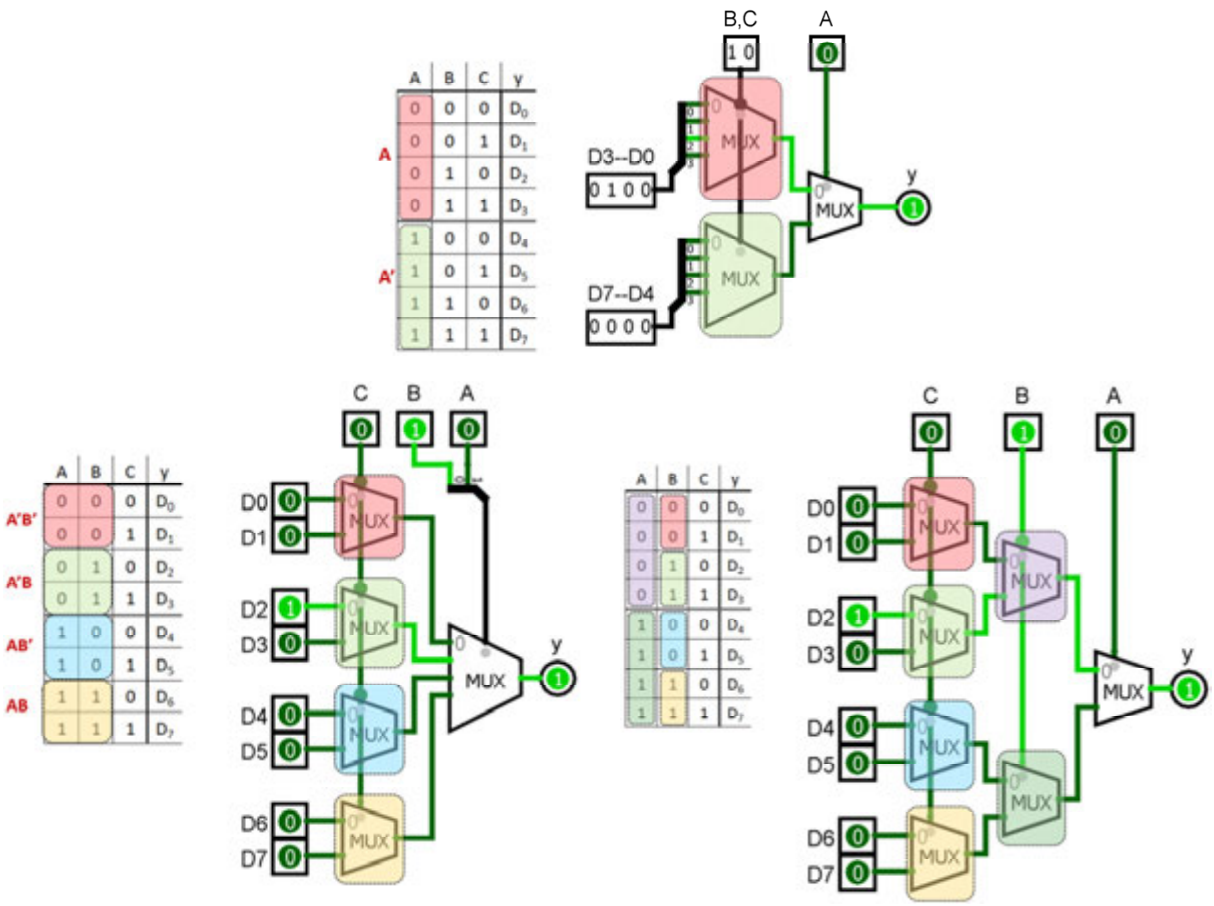


Figura 5.18: Redes de MUX multinivel sin gestión de habilitación y con señal de selección compartida para implementar un MUX 8×1.

> Recuerda que todos los circuitos en los que se emplean MUX (incluidos los que usan redes modulares) son susceptibles de análisis combinacional con la herramienta incluida en Logisim. De este modo es inmediato la obtención de la tabla de verdad de cualquier función implementada con MUX y sus expresiones simplificadas SOP y POS.

5.3 El demultiplexor

El demultiplexor o DEMUX se emplea en transmisión de datos para distribuir los datos desde un bus compartido de comunicación hacia diferentes destinos (ver figura 5.19). Consta de n entradas de selección mediante las que se selecciona una de las 2^n salidas como destino de los datos de entrada. En este sentido se comporta de forma complementaria al MUX y suele trabajar junto a este en las redes de transmisión de datos en el lado opuesto del canal de transmisión.

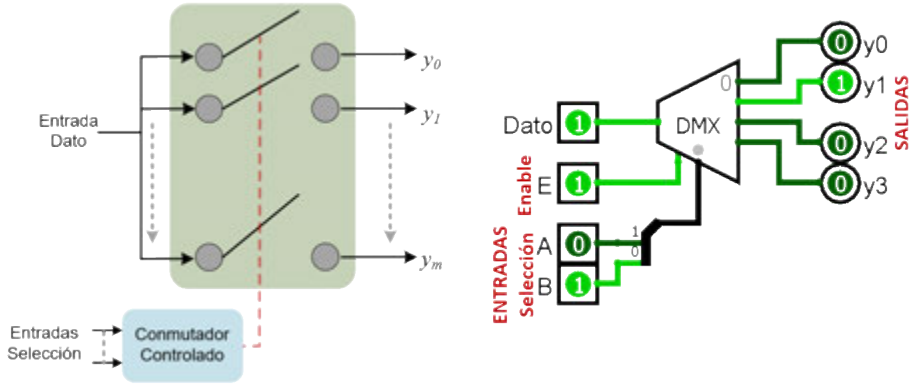


Figura 5.19: Funcionalidad del DEMUX y diagrama lógico de DEMUX 1×4 en Logisim.

La tabla 5.3 resume el comportamiento de un DEMUX 1×4 en el que las salidas son:

$$y_0 = (A'B') \cdot E \cdot D$$
$$y_1 = (A'B) \cdot E \cdot D$$
$$y_2 = (AB') \cdot E \cdot D$$
$$y_3 = (AB) \cdot E \cdot D$$

Tabla 5.3: Tabla de verdad del DEMUX 1×4.

E	D	A	B	y_0	y_1	y_2	y_3
0	x	x	x	0	0	0	0
1	x	0	0	x	0	0	0
1	x	0	1	0	x	0	0
1	x	1	0	0	0	x	0
1	x	1	1	0	0	0	x

La implementación del DEMUX es sencilla, puesto que su funcionalidad se obtiene con un DEC cuya señal de *enable* se emplea como entrada de datos. En la práctica es habitual que la señal de habilitación de los decodificadores sea compuesta para permitir su utilización como DEMUX (ver figura 5.20).

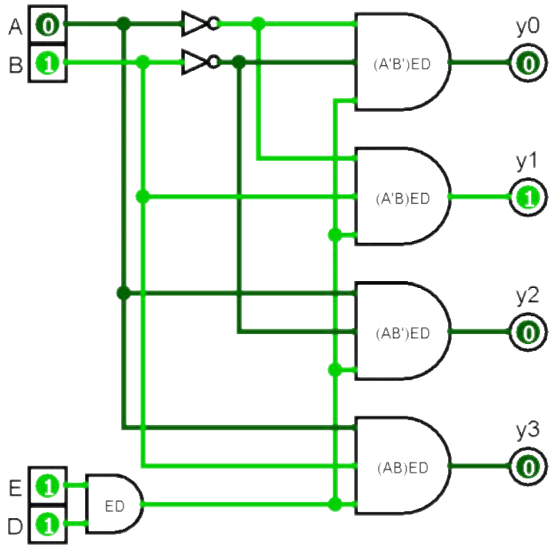


Figura 5.20: Diagrama lógico de DEMUX 1×4 con señal de enable (E) y salidas activas a nivel alto.

Al igual que con los decodificadores se puede dar una implementación del DEMUX con señal de *enable* y salidas activas a nivel bajo, en cuyo caso el circuito resultante emplea puertas NAND (ver figura 5.21).

La tabla 5.4 resume el comportamiento de un DEMUX 1×4 con salidas activas a nivel bajo. Las expresiones algebraicas para las salidas son:

$$y_0 = (A + B) + D + E$$
$$y_1 = (A + B') + D + E$$
$$y_2 = (A' + B) + D + E$$
$$y_3 = (A + B) + D + E$$

Tabla 5.4: Tabla de verdad del DEMUX 1×4 con salidas activas a nivel bajo.

E	D	A	B	y_0	y_1	y_2	y_3
0	x	0	0	x	1	1	1
0	x	0	1	1	x	1	1
0	x	1	0	1	1	x	1
0	x	1	1	1	1	1	x
1	x	x	x	1	1	1	1

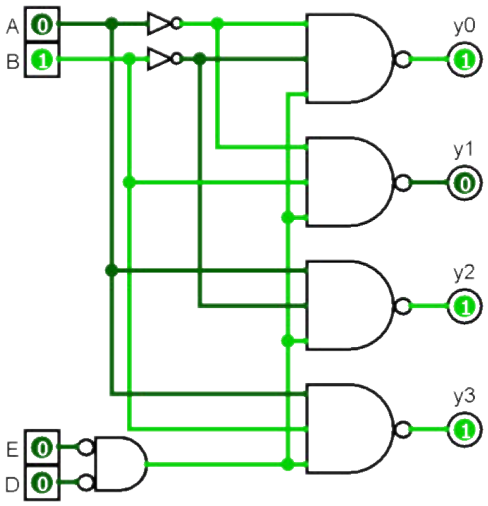


Figura 5.21: Diagrama lógico de DEMUX 1×4 con señal de enable (E) y salidas activas a nivel bajo.

5.4 El codificador binario

El codificador binario (*encoder*) o COD $2^n \times n$ tiene una funcionalidad inversa a la del DEC. En consecuencia, la activación de una de sus 2^n entradas genera a la salida el código binario de n bits correspondiente. Por tanto, el codificador se comporta como un conversor de valor decimal a código binario.

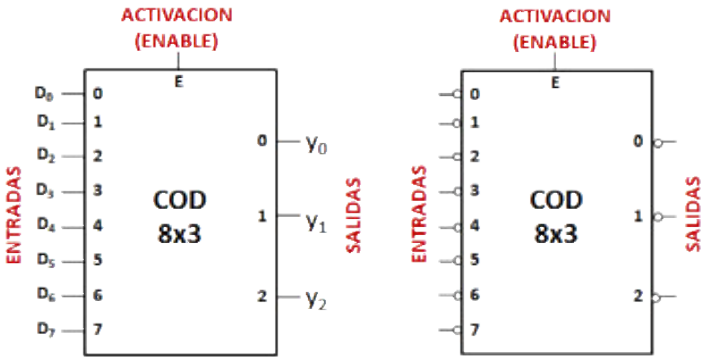


Figura 5.22: Símbolo correspondiente al COD 8×3 con E/S activas a nivel alto/bajo (izda./dcha.).

La activación de las entradas y las salidas del COD se puede realizar tanto a nivel alto (1) como a nivel bajo (0). Como en otros módulos combinacionales la diferencia de activación a nivel alto o bajo se indica en los símbolos empleados mediante '○' (ver figura 5.22).

5.4.1 Implementación del codificador

La tabla de verdad del codificador tiene demasiadas filas (2^n) para representarla de modo exhaustivo. Sin embargo, se puede simplificar teniendo en cuenta que interesan únicamente las combinaciones con una entrada activada. De este modo, se obtiene la tabla 5.5 que constituye la tabla de verdad resumida para el COD 8×3.

Tabla 5.5: Tabla de verdad del COD 8×3 con salidas activas a nivel alto.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	y_2	y_1	y_0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

Puesto que se asume que solo una entrada está activa, las ecuaciones lógicas que se pueden derivar para las salidas del codificador 8×3 son:

$$y_2 = D_7 + D_6 + D_5 + D_4; \quad y_1 = D_7 + D_6 + D_3 + D_2; \quad y_0 = D_7 + D_5 + D_3 + D_1$$

Si se implementan estas funciones lógicas mediante un circuito de puertas lógicas se tiene el resultado que muestra la figura 5.23.

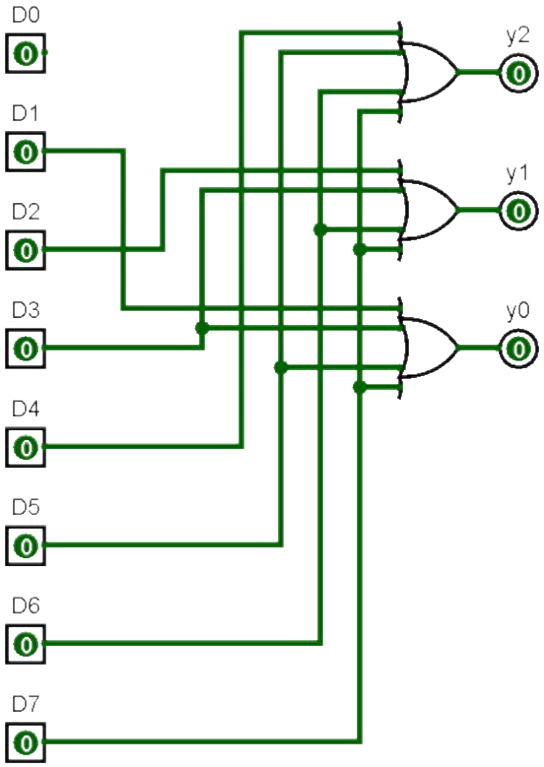


Figura 5.23: Implementación, mediante puertas lógicas, del COD 8×3 con E/S activas a nivel alto.

En el circuito lógico de la figura 5.23 se observa que la entrada D_0 queda sin conectar porque ninguna de las salidas depende de su estado. Por esta razón, en algunas implementaciones del COD, dicha entrada se suprime. En consecuencia, cuando todas las entradas están desactivadas (0), las salidas también lo están, lo que equivale a la codificación del valor 0.

La implementación de un codificador con entradas y salidas activas a nivel bajo se deriva del anterior, negando tanto las entradas como las salidas. En este caso las negaciones en la entrada de las puertas OR convierten a estas en puertas NAND y como la salida también debe estar negada, el circuito final resulta con puertas AND.

5.4.2 Codificador con prioridad

La implementación del codificador binario plantea varias situaciones singulares, a saber:

1. *¿Cuál debe ser la salida del COD si se activa simultáneamente más de una entrada?* Empleando las expresiones lógicas para el codificador derivadas previamente se llega a resultados erróneos, ya que para obtenerlas se tuvo en cuenta que había una única entrada activada.

2. *¿Cómo distinguir la salida con código 0 correspondiente a la activación de entrada D_0, de una situación con todas las entradas desactivadas?*

3. *¿Cómo diferenciar la salida cuando todas las entradas están desactivadas, de la situación con* enable *desactivado ($E = 0$)?*

La solución que se adopta para determinar la salida correcta cuando hay varias entradas activadas simultáneamente, consiste en la aplicación del criterio de prioridad en las entradas. De este modo, la salida corresponde a la entrada activa de mayor prioridad. La implementación de prioridad en un COD se consigue mediante un circuito combinacional que activa únicamente la entrada de mayor prioridad cuando hay varias entradas activas.

La implementación del circuito de prioridad es sencilla, ya que cada salida se activa si la entrada correspondiente está activa (vale 1) y las de mayor orden no lo están (su valor es 0). Por tanto, las ecuaciones lógicas de las salidas de un circuito de prioridad con 8 entradas y 8 salidas serían las siguientes:

$$
\begin{aligned}
D_0^* &= D_7' \cdot D_6' \cdot D_5' \cdot D_4' \cdot D_3' \cdot D_2' \cdot D_1' \cdot D_0 \\
D_1^* &= D_7' \cdot D_6' \cdot D_5' \cdot D_4' \cdot D_3' \cdot D_2' \cdot D_1 \\
D_2^* &= D_7' \cdot D_6' \cdot D_5' \cdot D_4' \cdot D_3' \cdot D_2 \\
&\ldots \\
D_7^* &= D_7
\end{aligned}
$$

La figura 5.24 muestra la implementación mediante diagrama lógico del circuito con Logisim en el caso concreto de un circuito de prioridad con 8 entradas y 8 salidas para un COD 8×3.

Para distinguir la generación de código 0 con entrada D_0 activa o todas las entradas desactivadas, algunos codificadores omiten la entrada D_0. Ya se ha visto en la figura 5.23 que en el circuito lógico del codificador que dicha entrada no tiene lógica asociada.

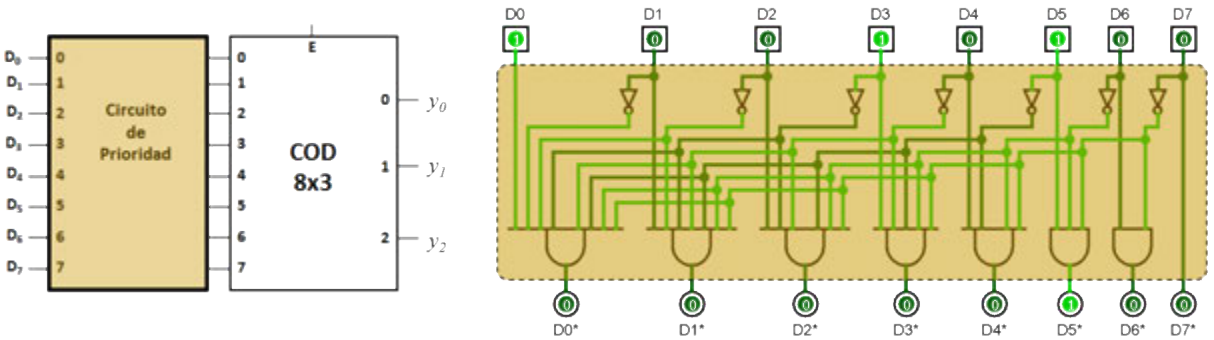
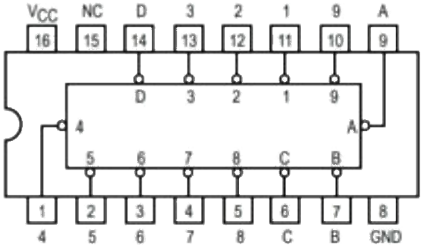


Figura 5.24: Diagrama lógico de circuito de prioridad a la entrada de un COD 8×3.

El circuito 74LS147 es un codificador con prioridad comercial, conocido también como conversor decimal a BCD, ya que tiene 9 entradas y 4 salidas (ver figura 5.25 izda.). En este circuito la entrada 0 se omite, de modo que cuando todas las entradas están desactivadas, también lo están las salidas. Se trata de un circuito con entradas y salidas activas a nivel bajo como se indica en su tabla de verdad (ver figura 5.25 dcha.).



FUNCTION TABLE – '147, 'LS147

INPUTS									OUTPUTS			
1	2	3	4	5	6	7	8	9	D	C	B	A
H	H	H	H	H	H	H	H	H	H	H	H	H
X	X	X	X	X	X	X	X	L	L	H	H	L
X	X	X	X	X	X	X	L	H	L	H	H	H
X	X	X	X	X	X	L	H	H	H	L	L	L
X	X	X	X	X	L	H	H	H	H	L	L	H
X	X	X	X	L	H	H	H	H	H	L	H	L
X	X	X	L	H	H	H	H	H	H	L	H	H
X	X	L	H	H	H	H	H	H	H	H	L	L
X	L	H	H	H	H	H	H	H	H	H	L	H
L	H	H	H	H	H	H	H	H	H	H	H	L

H = high logic level, L = low logic level, X = irrelevant

Figura 5.25: Diagrama de conexiones externas del circuito integrado 74LS147 (izda.) y tabla de verdad correspondiente (dcha.) (*Fuente: Texas Instruments*).

Para diferenciar la salida del COD cuando está inhabilitado o cuando todas las entradas están desactivadas, se añaden algunas señales auxiliares. Además, dichas señales facilitan la conexión en cadena (*daisy chain*) de varios COD para crear redes modulares. Las señales auxiliares utilizadas son:

- *Enable input* (E_i). Cuando está desactivada, el COD pone sus salidas en estado flotante (alta impedancia).
- *Group select* (G_s). Se activa solo si el COD está habilitado ($E_i = 1$) y alguna de sus entradas está activada.
- *Enable output* (E_o). Se activa solo si el COD está habilitado ($E_i = 1$) y todas sus entradas están desactivadas.

> Logisim proporciona en la librería *Plexers* un módulo que implementa un codificador de prioridad en el que es posible configurar el número de salidas (n) y el comportamiento cuando la entrada de habilitación está desactivada (salida flotante o 0).

La figura 5.26 muestra el circuito lógico completo en Logisim para un COD 8×3 con prioridad y E/S activas a nivel alto, en el que se incluye un módulo de codificación (COD 8×3, ver figura 5.22, un módulo de prioridad (Prior 8×3, ver figura 5.24) y la lógica para obtener las señales auxiliares *Enable input* (E_i), *Enable output* (E_o) y *Group select* (G_s). El circuito incluye a la salida un *buffer controlado* (incluido en la librería *Gates*) que deja las salidas en estado flotante cuando el circuito está inhibido. El chip comercial 74LS148 es un circuito comercial con la funcionalidad descrita, pero con E/S activas a nivel bajo.

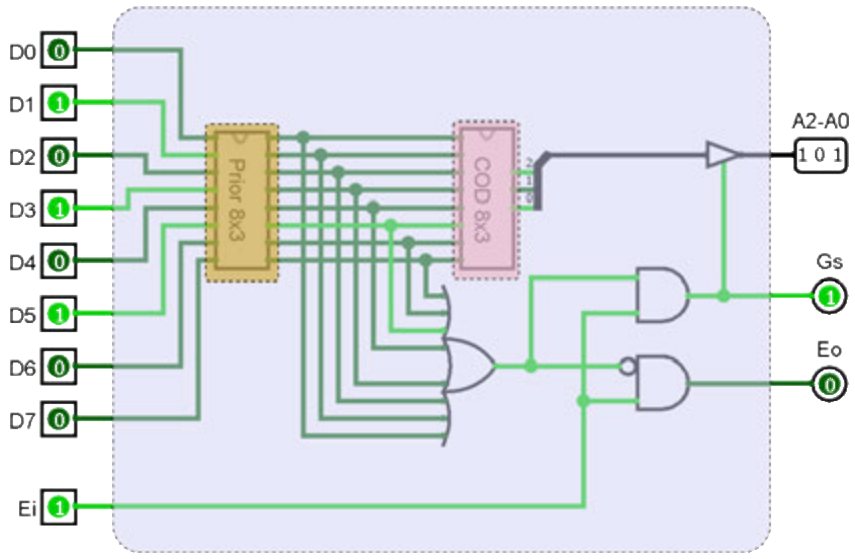


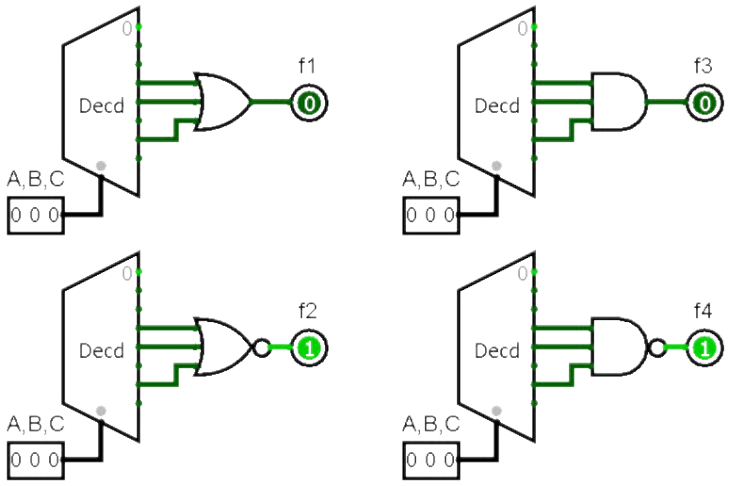
Figura 5.26: COD 8×3 completo implementado con Logisim que incluye circuitos de codificación, prioridad y señales de auxiliares.

Conceptos clave

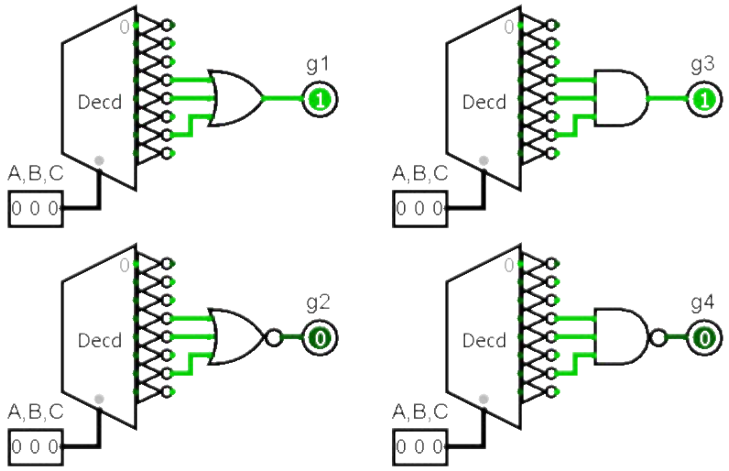
- ☑ DEC como generador de términos canónicos.
- ☑ Implementación de un DEC con puertas lógicas.
- ☑ Implementación de funciones lógicas con DEC.
- ☑ Implementación de un MUX con puertas lógicas.
- ☑ Implementación de funciones lógicas con MUX.
- ☑ Redes modulares de decodificadores y multiplexores.
- ☑ Relación entre un DEMUX y un DEC.
- ☑ Implementación de un circuito de prioridad.

Problemas propuestos

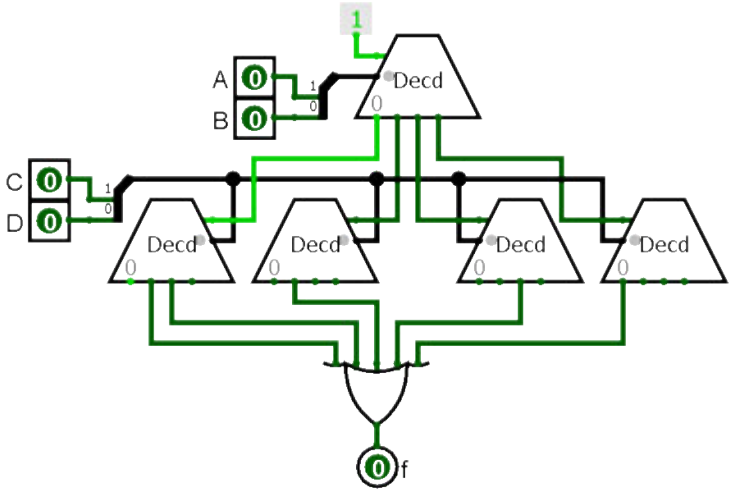
Problema 5.1 ¿Cuáles son las formas canónicas de las funciones lógicas f_1, f_2, f_3 y f_4, implementadas por cada uno de los circuitos de la figura adjunta? Calcula también las versiones simplificadas con K-maps para cada una de ellas.



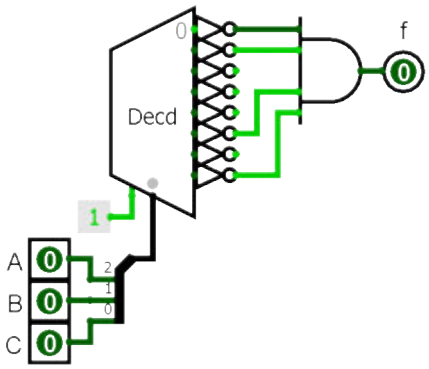
Problema 5.2 ¿Cuáles son las formas canónicas de las funciones lógicas g_1, g_2, g_3 y g_4, implementadas por cada uno de los circuitos de la figura adjunta? Calcula también las versiones simplificadas con K-maps para cada una de ellas.



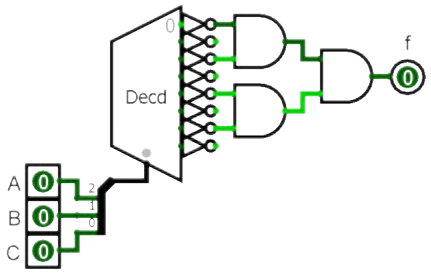
Problema 5.3 ¿Cuáles son las formas canónicas y las versiones simplificadas de la función lógica f implementada por el circuito de la figura adjunta? Comprueba el resultado con Logisim.



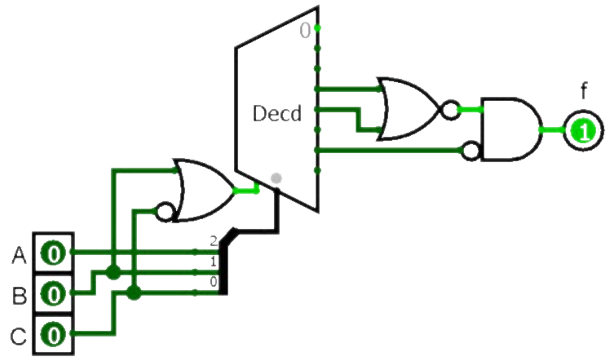
Problema 5.4 ¿Cuáles son las formas canónicas y las versiones simplificadas de la función lógica f implementada por el circuito de la figura adjunta?



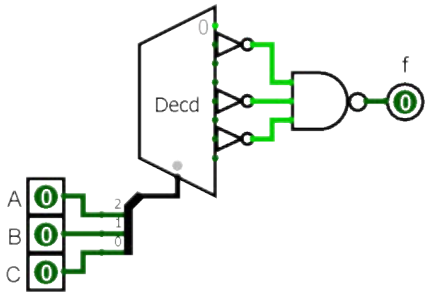
Problema 5.5 ¿Cuáles son las formas canónicas y las versiones simplificadas de la función lógica f implementada por el circuito de la figura adjunta?



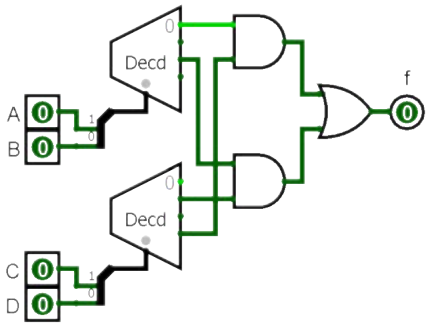
Problema 5.6 ¿Cuáles son las formas canónicas y las versiones simplificadas de la función lógica f implementada por el circuito de la figura adjunta?



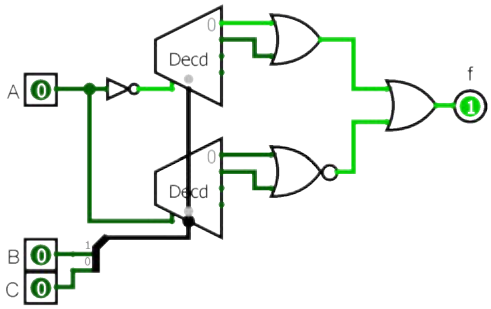
Problema 5.7 Calcula las formas canónicas y simplificadas de la función lógica f implementada por el circuito de la figura adjunta.



Problema 5.8 Analiza el circuito de la figura adjunta y calcula la expresión lógica de la función f para determinar tanto sus formas canónicas como sus expresiones simplificadas.



Problema 5.9 ¿Cuáles son las formas canónicas de la función lógica f implementada por el circuito de la figura adjunta?



Problema 5.10 Dada la función:

$$f(A,B,C) = \sum m(1,3,5,6,7)$$

Se pide:

1.- Determina sus formas canónicas y expresiones simplificadas.
2.- Implementa el circuito con Logisim para las expresiones simplificadas de f mediante cualquier tipo de puerta lógica de 2 entradas y los inversores necesarios.
3.- Implementa la función f con Logisim empleando un DEC de salidas activas a nivel alto y puertas de dos entradas.

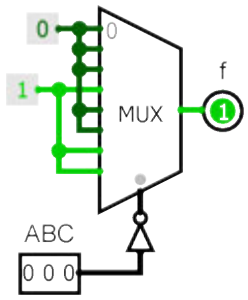
Problema 5.11 Dada la función:

$$f'(A,B,C,D) = A'B'C'D' + A'BC'D' + AB'C'D' + AB'CD'$$

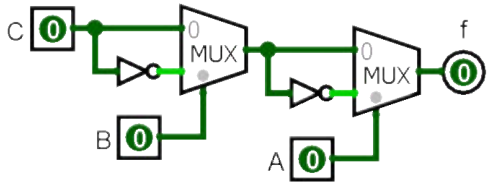
Se pide:

1.- Determina las formas canónicas de la función $f(A,B,C,D)$.
2.- Calcula las expresiones simplificadas —SOP y POS— de f.
3.- Implementa f empleando una red de dos niveles compuesta por 2 DEC 3×8 y puertas lógicas de dos entradas.
4.- Implementa f con una red de dos niveles con gestión de habilitación, compuesta por DEC 2×4.
5.- Implementa f con una red de dos niveles con gestión de habilitación compuesta por DEC 3×8 y DEC 1×2.
6.- Implementa de f con una red multinivel compuesta solo por DEC 1×2.

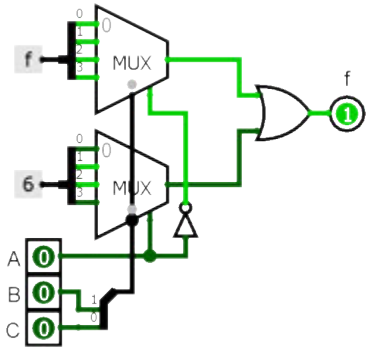
Problema 5.12 Dado el circuito de la figura adjunta determina las expresiones canónicas de la función implementada f y sus expresiones simplificadas SOP y POS. Comprueba el resultado con ayuda de Logisim.



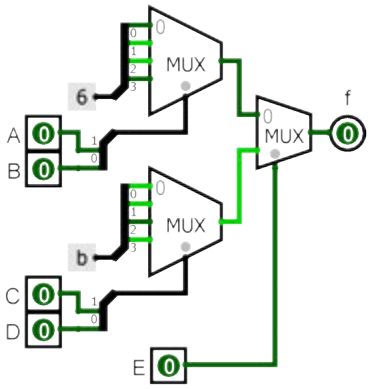
Problema 5.13 ¿Cuáles son las formas canónicas y las versiones simplificadas de la función lógica f implementada por el circuito de la figura adjunta?



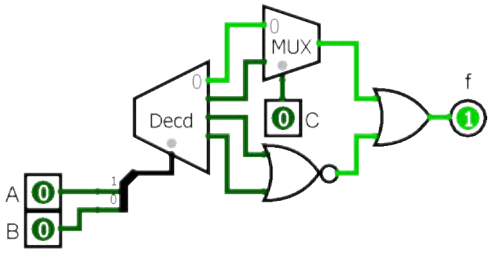
Problema 5.14 Dado el circuito de la figura adjunta, analiza su propósito obteniendo las expresiones SOP y POS de la función f.



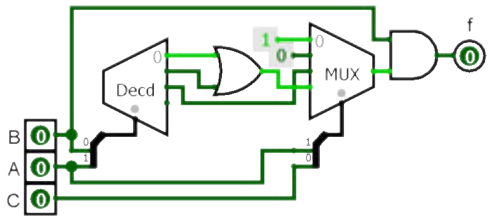
Problema 5.15 Dada la función $f(A,B,C,D,E)$ implementada con el circuito de la figura adjunta, calcula su expresión lógica y los valores para $f(0,0,1,1,1)$ y $f(1,1,0,0,1)$.



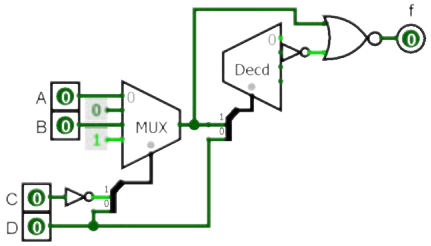
Problema 5.16 ¿Cuáles son las formas canónicas y las versiones simplificadas de la función lógica f implementada por el circuito de la figura adjunta?



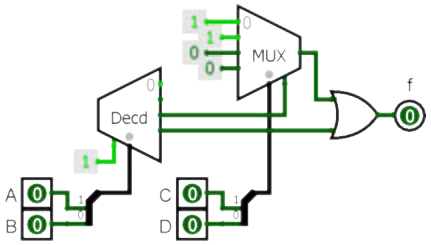
Problema 5.17 ¿Cuáles son las formas canónicas y las versiones simplificadas de la función lógica f implementada por el circuito de la figura adjunta?



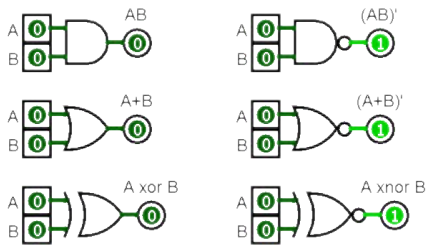
Problema 5.18 ¿Cuáles son las formas canónicas y las versiones simplificadas de la función lógica f implementada por el circuito de la figura adjunta?



Problema 5.19 ¿Cuáles son las formas canónicas y las versiones simplificadas de la función lógica f implementada por el circuito de la figura adjunta?



Problema 5.20 Expresa con MUX 2×1 las funciones básicas implementadas por las puertas lógicas de dos entradas: AB, $A+B$, $(AB)'$, $(A+B)'$, $A\oplus B$, $A\odot B$:



Problema 5.21 Implementa, con MUX 2×1 y MUX 4×1, alternativas del circuito equivalente a una puerta AND de tres entradas.

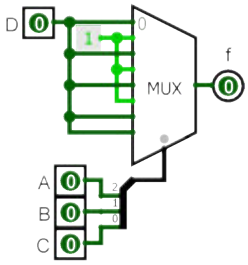
Problema 5.22 Implementa la función $f(A,B,C)=A'B'+(B\oplus C)'$ mediante multiplexor(es) empleando como señales de selección las señales A o B.

Problema 5.23 Implementa la función $f(A,B,C,D)=\sum m(1,5,9,10,11,13)$ mediante MUX de distintos tipos:

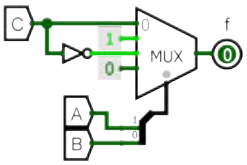
1.- MUX 8×1,
2.- MUX 4×1,
3.- MUX 2×1. Encuentra la implementación más sencilla e intenta llevarla a cabo mediante concatenación de MUX 2×1.
4.- Mediante redes de MUX de distinto tipo.

Problema 5.24 Implementa la función $f(A,B,C,D) = \sum m(0,1,6,8,9,12,13,14)$ del modo más simple posible mediante MUX 4×1 y después con MUX 2×1. En este último caso sugiere una implementación que solo emplee MUX, excluyendo puertas lógicas e inversores.

Problema 5.25 Indica cuáles son las expresiones canónicas de la función $f(A,B,C,D)$ implementada por el circuito lógico de la figura adjunta. Encuentra las expresiones simplificadas de f y sugiere una implementación basada en MUX 4×1 y otra en MUX 2×1.

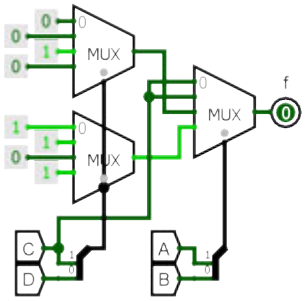


Problema 5.26 Analiza cuál es la función $f(A,B,C)$ implementada por el circuito de la figura adjunta. Elabora una implementación de dicha función con un MUX 2×1.



Problema 5.27 Implementa la función $f(A,B,C) = \sum m(0,1,3,4,7)$ con un MUX 8×1 y de modo alternativo con redes multinivel de MUX de orden inferior.

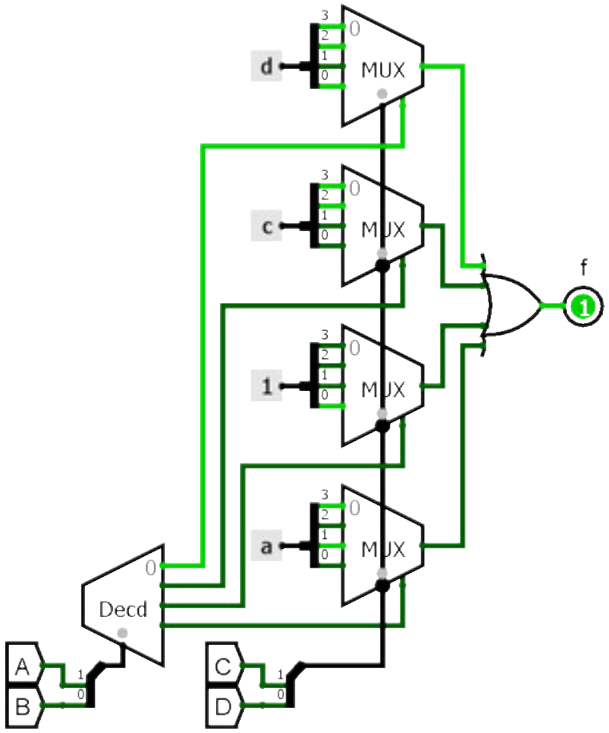
Problema 5.28 Dada la función f implementada por el circuito de la figura adjunta:



Se pide:

1.- Determina las formas canónicas de la función f.
2.- Calcula las expresiones simplificadas de f.
3.- Implementa f con un único MUX 4×1 en el que se utilicen A y C como señales de selección.
4.- Implementa f con un MUX 2×1 en el que B sea la señal de selección.

Problema 5.29 Dada la función f implementada por el circuito de la figura adjunta:

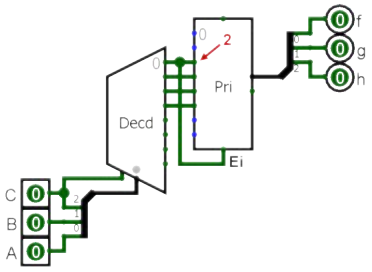


Se pide:

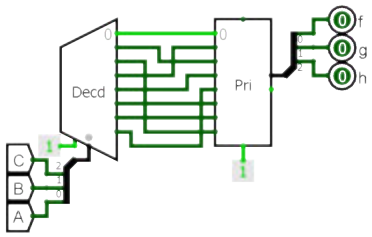
1.- Determina las formas canónicas de f.
2.- Calcula las expresiones simplificadas de f.
3.- Implementa el mismo circuito con una red de MUX 4×1 en la que se elimine el DEC y la puerta OR a la salida.
4.- Elabora una implementación alternativa con un MUX 4×1 y las puertas lógicas auxiliares necesarias.
5.- Sugiere una implementación alternativa empleando solo MUX 2×1 e inversores.

Problema 5.30 Implementa un circuito de prioridad de 4 bits con salidas activas a nivel alto.

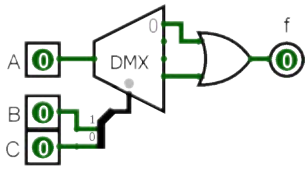
Problema 5.31 Analiza el circuito de la figura adjunta describiendo su funcionamiento. En este circuito E_i representa la señal *enable input* que habilita la salida del COD, ya que si $E_i = 0$ la salida del COD es 0.



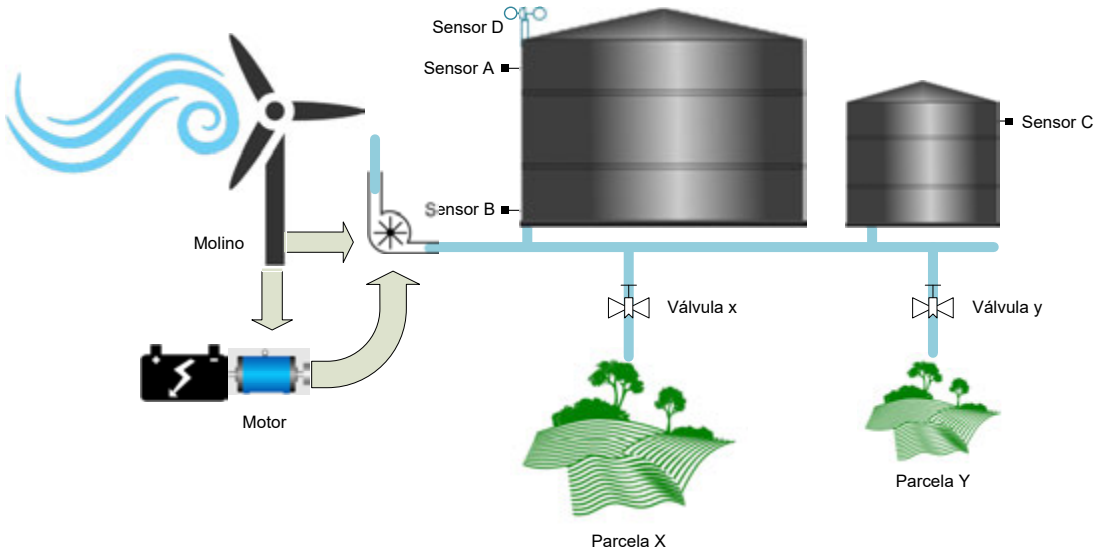
Problema 5.32 Analiza el circuito de la figura adjunta y calcula la tabla de verdad para sus tres funciones de salida f, g, h. Expresa dichas funciones en sus formas canónicas y simplificadas.



Problema 5.33 Determina las formas canónicas y simplificadas de f, dada por el circuito constituido por un DEMUX 1×4 de la figura adjunta. Realiza también la implementación de f empleando puertas lógicas elementales.



Problema 5.34 La figura adjunta representa un sistema de regadío de dos parcelas (X e Y). Para bombear el agua se cuenta con un aerogenerador y un motor (m) que pueden accionar la bomba de agua. El agua es bombeada y almacenada en dos depósitos intercomunicados, uno para cada parcela. El motor m y las válvulas de los aspersores conectados a los depósitos (que llamaremos v_x e v_y) se accionan en función de: los sensores de nivel A, B y C de los depósitos, así como de un sensor D que mide si hay viento suficiente para que el aerogenerador accione la bomba hidráulica.



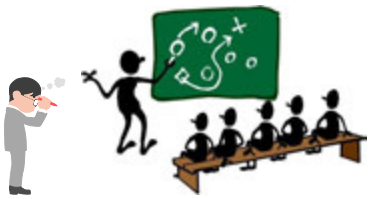
Las condiciones de funcionamiento del sistema se resumen en las siguientes reglas:

- El motor m se pone en marcha siempre que ambos depósitos no estén llenos y no haya suficiente viento.
- La parcela X se riega siempre que su depósito esté lleno y no haya viento suficiente o bien, su depósito esté al menos medio lleno y haya viento suficiente.
- Por su parte, la parcela Y sólo se riega cuando su depósito supera el nivel mínimo marcado por el sensor de nivel B y hay viento suficiente.

Se pide:

1.- Determina las variables de entrada y salida del sistema digital que controla el regadío de las dos parcelas.
2.- Establece claramente cuáles son las situaciones indiferentes (no importa) en este problema.
3.- Elabora la tabla de verdad del sistema digital para las funciones de salida y calcula sus expresiones simplificadas.
4.- Implementa la función de activación del motor m empleando DEC o MUX. Elije las señales de selección que produzcan la implementación más sencilla.
5.- Implementa la función de apertura de la válvula v_x usando DEC o MUX. Elije las señales de selección que produzcan la implementación más sencilla.
6.- Implementa la función de apertura de la válvula v_y utilizando DEC o MUX. Elije las señales de selección que produzcan la implementación más sencilla.

Problema 5.35 El presidente de un club de fútbol debe decidir al final de la temporada si se renueva (f) o no el contrato al entrenador ($f = 1 \Rightarrow$renueva, $f = 0 \Rightarrow$no renueva). Diseña un circuito combinacional para resolver el problema lógico de toma de decisión teniendo en cuenta las condiciones que se indican a continuación.



- SI el equipo queda campeón y obtiene más de 70 puntos, ENTONCES se renueva contrato.
- SI quedando campeón el equipo no supera los 70 puntos, ENTONCES el entrenador correrá la misma suerte que el entrenador del equipo rival al que se le disputa el campeonato.
- SI el equipo no queda campeón, pero supera los 70 puntos, ENTONCES el entrenador correrá la suerte contraria que el entrenador del equipo rival.
- SI el equipo no queda campeón y no supera los 70 puntos, ENTONCES no se renueva al entrenador.

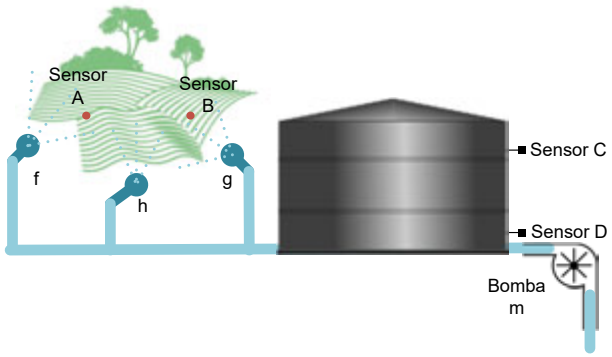
Diseña el sistema completando los apartados siguientes:

1.- Define las variables lógicas del problema.
2.- Elabora la tabla de verdad de la función f que codifica la renovación del entrenador.
3.- Calcula las formas canónicas y simplificadas de f.
4.- Implementa f usando solo puertas NAND. Alternativamente, elabora la implementación solo con puertas NOR.

Para tomar una decisión más acertada, el presidente decide pedir consejo a su junta directiva. Si el equipo queda campeón se aplican los mismos criterios que en los apartados anteriores, incluyendo la renovación del entrenador del equipo rival. Pero si el equipo no queda campeón, la decisión dependerá de la junta directiva. Llamaremos g a la nueva función lógica de decisión. Ante estas nuevas circunstancias completa los apartados siguientes:

5.- Elabora la tabla de verdad de g y calcula sus expresiones simplificadas.
6.- Implementa la nueva función g utilizando alternativamente un MUX 8×1, un MUX 4×1 y un MUX 2×1, colocando en las líneas de selección las variables que consideres oportunas.

Problema 5.36 Diseña un circuito combinacional que permita controlar el sistema de riego automático esquematizado en la figura adjunta. El sistema consta de 2 sensores de humedad (A y B) y 3 aspersores (f, g y h) conectados a un depósito de agua. El depósito dispone de dos sensores de nivel (C y D) y se llena mediante una moto bomba hidráulica (m).



El sistema debe funcionar de la siguiente manera:

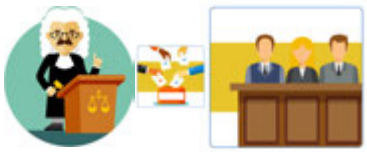
- La moto bomba m se debe activar si:
  - El depósito está vacío, a menos que no sea preciso el riego porque tanto A como B detectan humedad;

    - El depósito está medio lleno y es preciso regar porque algún sensor o ambos no detectan humedad.
- La moto bomba m nunca se activa si el depósito está lleno.
- El aspersor f se activa si el sensor A no detecta humedad.
- El aspersor g se activa si el sensor B no detecta humedad.
- El aspersor h se activa si alguno de los aspersores f o g está activo, pero no ambos.
- Los aspersores no se deben activar si el depósito está vacío, salvo si:
    - Solo el sensor A no detecta humedad, en cuyo caso se activa únicamente f,
    - Solo el sensor B no detecta humedad, en cuyo caso se activa únicamente g.

Se pide:

1.- Realiza la tabla de verdad de las funciones de salida a partir de la especificación dada.
2.- Implementa m con MUX.
3.- Implementa f con DEC con salidas activas a nivel alto.
4.- Implementa g con DEC con salidas activas a nivel bajo.
5.- Implementa h con DEC y con MUX.

Problema 5.37 Un tribunal de justicia está compuesto por cuatro miembros, el Presidente (P), el Secretario (S), el Vocal 1 (V_1) y el Vocal 2 (V_2). A la hora de dictar el veredicto del tribunal (f), cada uno de sus miembros toma una decisión individual sobre el acusado sin conocer la decisión del resto.



Para obtener el veredicto final del tribunal han acordado tener en cuenta la decisión de la mayoría de sus componentes con el siguiente criterio:

- Si el Presidente considera que el acusado es inocente, ocurre lo siguiente:
    - SI al menos uno de los restantes miembros declaran inocente al acusado, ENTONCES este es declarado inocente.
    - SI todos los demás miembros consideran que el acusado es culpable, ENTONCES es declarado culpable.
- Si el Presidente declara culpable al acusado, el veredicto se toma de la siguiente forma:
    - SI los dos Vocales lo declaran inocente, ENTONCES el veredicto final será el del Secretario.
    - SI los dos Vocales coinciden en declararlo culpable, ENTONCES será declarado culpable, independientemente de lo que diga el Secretario.
    - SI los dos Vocales discrepan, ENTONCES el veredicto final será el del Secretario, siempre que coincida con el del Presidente, y en caso contrario (si Presidente y Secretario discrepan) el acusado será declarado inocente.

El abogado del acusado podrá recurrir (r) la sentencia a otro tribunal de mayor rango si el acusado ha contado con: al menos dos votos favorables de entre los cuatro del tribunal, o bien si el Presidente votó a favor de su inocencia. Se desea diseñar un circuito combinacional para modelar la toma de decisión del tribunal. Para ello se completarán los pasos siguientes:

1.- Identifica las entradas y salidas del sistema.
2.- Determina la tabla de verdad correspondiente a las funciones de salida. Explica si existen condiciones libres para alguna función de salida.
3.- Calcula las expresiones canónicas y simplificadas de las funciones de salida.
4.- Implementa las funciones de salida con varios DEC 3×8 o más sencillos.
5.- Implementa las funciones de salida con un MUX 8×1. Repite el proceso con un MUX 4×1 y con un MUX 2×1.
6.- Implementa las funciones de salida solo con puertas NAND. Alternativamente, elabora una implementación solo con puertas NOR.

6. Módulos lógicos y aritméticos

Además de los módulos combinacionales especializados en operaciones de transmisión de datos, existen otros destinados a llevar a cabo operaciones de tipo lógico y aritmético. Este capítulo aborda el estudio de dichos módulos. Algunas de estas funcionalidades se encuentran implementadas por módulos especializados y otras están integradas en unidades complejas multifuncionales denominadas *Unidades Aritmético Lógicas* (UAL o *Arithmetic Logic Units*, ALU).

6.1 El comparador

Un *comparador* es un circuito combinacional que a partir de dos entradas A y B ofrece tres bits de salida indicando el resultado de las comparaciones: $A < B$, $A = B$ y $A > B$.

El diseño combinacional del comparador de un bit es sencillo partiendo de la tabla de verdad de las funciones asociadas a los tres bits de salida. De este modo se obtiene que:

- La «igualdad» ($A = B$) se consigue como resultado de aplicar la puerta XNOR. Esto es: $A \odot B = (A \oplus B)'$
- La condición «mayor que» ($A > B$) se evalúa mediante una puerta AND. Es decir: $A = 1 \wedge B = 0 \Rightarrow A \cdot B' = 1$
- La condición «menor que» ($A < B$) también se logra mediante una puerta AND. Por tanto: $A = 0 \wedge B = 1 \Rightarrow A' \cdot B = 1$

La figura 6.1 muestra el circuito comparador de un bit resultante para simulación lógica con Logisim.

A	B	f1	f2	f3
0	0	0	1	0
0	1	0	0	1
1	0	1	0	0
1	1	0	1	0

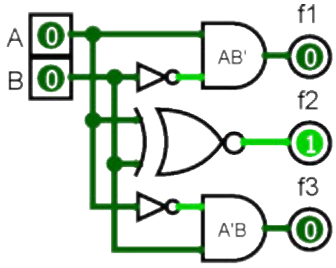


Figura 6.1: Tabla de verdad y circuito lógico del comparador de un bit.

> Recuerda que la condición «distinto que» al comparar una pareja de bits se puede evaluar mediante la puerta XOR. Esto es, $A \neq B \Rightarrow A \oplus B = 1$.

Con idéntica metodología se puede realizar el diseño del comparador de 2 bits a partir de su tabla de verdad según se ilustra en la figura 6.2.

				A>B	A=B	A<B
A_1	A_0	B_1	B_0	f_1	f_2	f_3
0	0	0	0	0	1	0
0	0	0	1	0	0	1
0	0	1	0	0	0	1
0	0	1	1	0	0	1
0	1	0	0	1	0	0
0	1	0	1	0	1	0
0	1	1	0	0	0	1
0	1	1	1	0	0	1
1	0	0	0	1	0	0
1	0	0	1	1	0	0
1	0	1	0	0	1	0
1	0	1	1	0	0	1
1	1	0	0	1	0	0
1	1	0	1	1	0	0
1	1	1	0	1	0	0
1	1	1	1	0	1	0

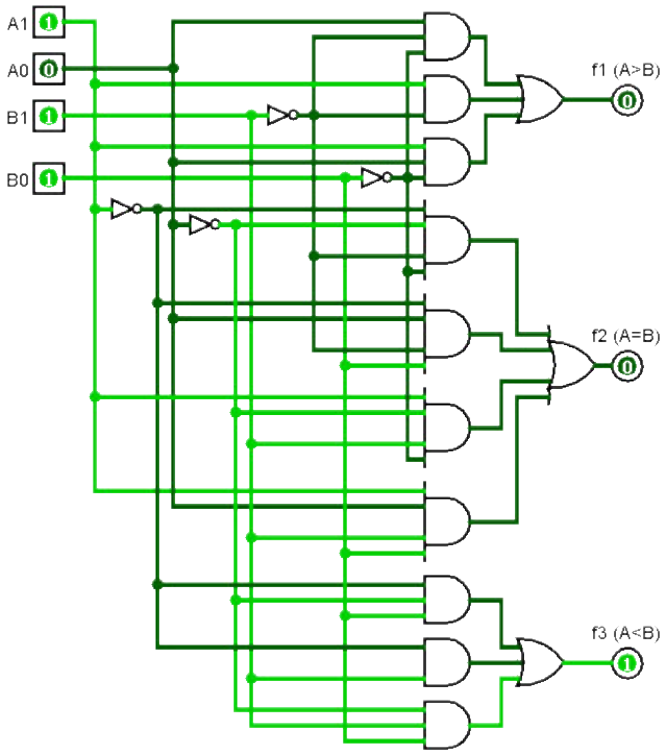


Figura 6.2: Tabla de verdad y diagrama lógico del circuito comparador de 2 bits.

Este circuito se puede obtener a partir del comparador de un bit mediante diseño modular jerárquico. En este caso es preciso emplear puertas lógicas auxiliares para obtener el valor deseado.

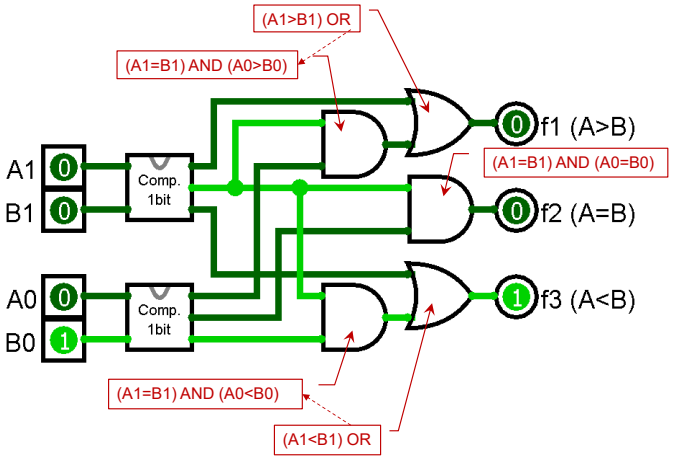


Figura 6.3: Diagrama del circuito lógico del comparador de 2 bits mediante comparadores de 1 bit y puertas lógicas adicionales.

Para prescindir de las puertas adicionales en el circuito de la figura 6.3, es preciso utilizar un comparador completo que incorpore entradas adicionales destinadas a la conexión de las salidas del módulo previo. En este diseño jerárquico la conexión de módulos se realiza propagando el resultado de la comparación obtenida de los bits menos significativos hacia el módulo que compara los bits más significativos, como se muestra en la figura 6.4.

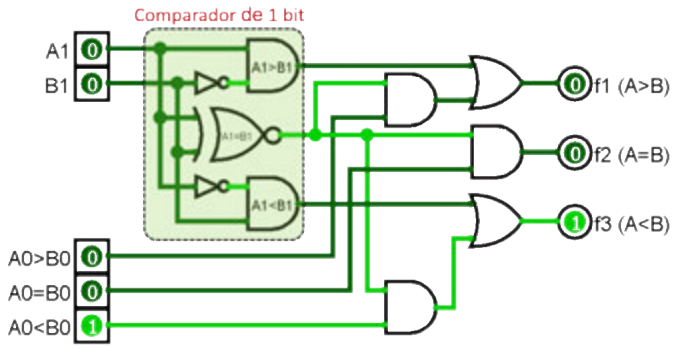


Figura 6.4: Diagrama lógico del circuito comparador completo de 1 bit.

En la figura 6.4 se observa que el comparador completo de un bit tiene en cuenta nuevas entradas que son el resultado de la comparación de los bits menos significativos. Este módulo permite la comparación de un número arbitrario de bits mediante conexión en cascada de módulos comparadores. En consecuencia, el resultado de la comparación solo tiene en cuenta los bits menos significativos cuando se produce igualdad en los más significativos. La figura 6.5 ilustra el diagrama lógico del comparador de 4 bits (*nibble*) mediante empleo de módulos comparadores de 1 bit completos conectados en cascada.

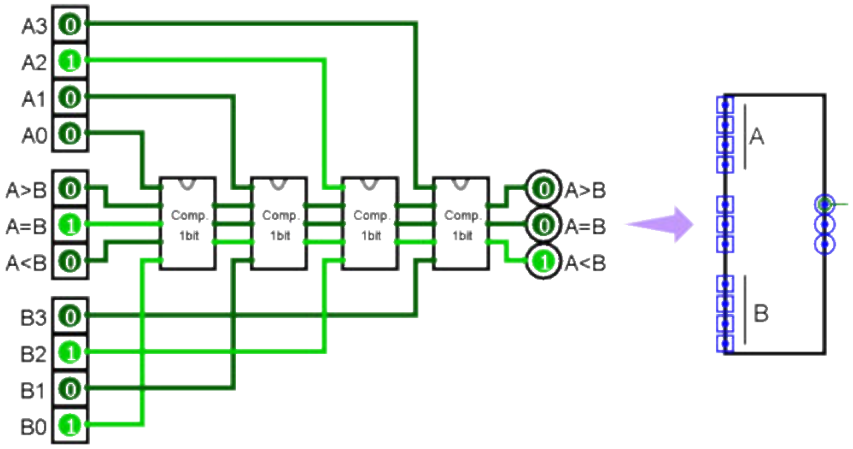


Figura 6.5: Diagrama lógico del circuito comparador de 4 bits a partir de 4 comparadores completos de 1 bit (izda.) y módulo Logisim generado (dcha.).

El módulo comparador de 4 bits se puede tratar a su vez como un nuevo módulo con el que implementar comparadores más complejos. El circuito integrado comercial 74HC85 es un comparador de 4 bits que permite obtener comparadores de 8 bits como el que se muestra en el diagrama lógico de la figura 6.6 para simular su comportamiento mediante Logisim.

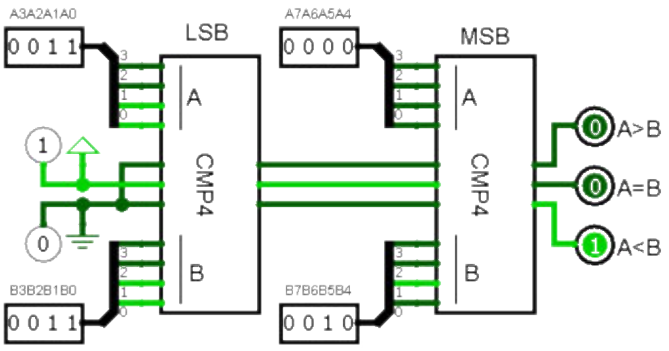


Figura 6.6: Diagrama lógico en Logisim para un comparador de 8 bits similar al obtenido mediante CI 74HC85.

Los módulos de comparación expuestos anteriormente comparan números binarios sin signo. Si se desea comparar dos números enteros (A, B en C1 o C2) se emplearán los módulos descritos anteriormente, añadiendo algunas puertas lógicas. En dicha comparación se pueden dar los casos siguientes:

1. Si ambos números tienen el mismo signo, la comparación que ofrece el comparador es correcta e independiente del signo (véase en figura 6.7-izda. las tablas adjuntas de codificación en C1 y C2).

2. Si los números tienen distinto signo y A es positivo, entonces $A > B$.

3. Si los números tienen distinto signo y A es negativo, entonces $A < B$.

Las tres condiciones mencionadas se pueden evaluar mediante un comparador de 1 bit en el que se compara el complemento del bit de signo. Por tanto, si A y B representan valores con signo (en C1 o C2) su comparación se realiza con los módulos ya estudiados en los que el bit de signo se introduce invertido (ver circuito en figura 6.7-dcha.).

C1	Dec.
0111	7
0110	6
0101	5
0100	4
0011	3
0010	2
0001	1
0000	0
1111	-0
1110	-1
1101	-2
1100	-3
1011	-4
1010	-5
1001	-6
1000	-7

C2	Dec.
0111	7
0110	6
0101	5
0100	4
0011	3
0010	2
0001	1
0000	0
1111	-1
1110	-2
1101	-3
1100	-4
1011	-5
1010	-6
1001	-7
1000	-8

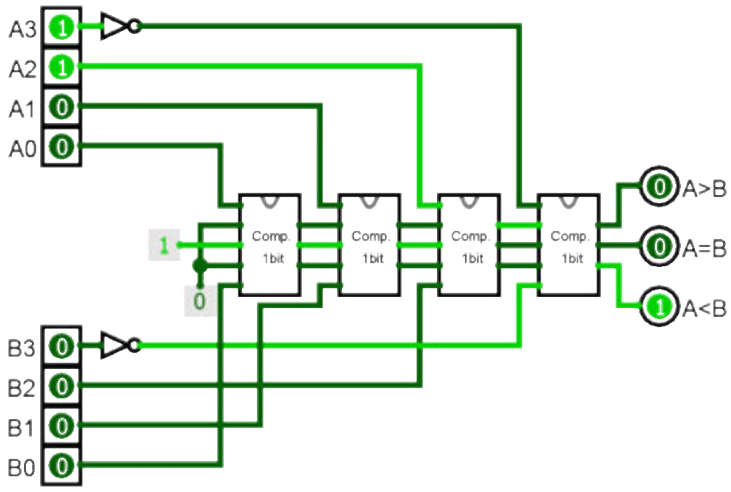


Figura 6.7: Códigos de 4 bits en C1 y C2 junto a su equivalente entero en decimal (izda.), y diagrama lógico del circuito comparador de 4 bits con bit de signo (dcha.).

> Ten en cuenta que Logisim proporciona mediante su librería aritmética (*Arithmetic*), un módulo comparador que se puede configurar para tener en cuenta el signo (C2) y el número de bits. Recuerda que los números negativos en Logisim emplean codificación en C2.

6.2 Generador de paridad

Las comunicaciones digitales se llevan a cabo mediante envío y recepción de bloques de bits. Un método sencillo de detección de errores en la comunicación, consiste en la adición de un *bit de paridad* al bloque de información que se va a transmitir. El bit de paridad se puede calcular desde los unos —lo más habitual— o desde los ceros, y puede ser de dos tipos: par o impar. El cálculo del bit de paridad (para los unos) se realiza de modo que la cantidad de unos en el bloque de datos —incluyendo el bit de paridad— sea par (*paridad par*, P_p) o impar (*paridad impar*, P_i). Para el cálculo de la paridad desde los ceros se sigue un razonamiento dual con el número de ceros.

El cálculo del bit de paridad se basa en la idea de que la suma de un número par de unos es 0. De este modo, el bit de paridad par desde los unos se calcula mediante la concatenación de la función XOR (OR-exclusivo) para los bits de datos (ver figura 6.8). Así se tiene que el circuito generador de paridad posee la expresión booleana:

$$P_p = D_7 \oplus D_6 \oplus D_5 \oplus D_4 \oplus D_3 \oplus D_2 \oplus D_1 \oplus D0$$
$$P_i = P'_p$$

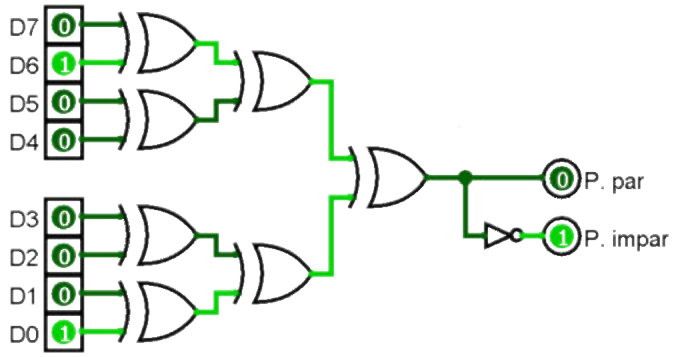


Figura 6.8: Diagrama lógico de circuito *generador de paridad* de un bloque de 8 bits.

El bit de paridad calculado para un bloque de datos se añade a los datos que se transmiten. En el extremo receptor se comprueba el valor de la paridad del bloque recibido para detectar la presencia de error. En este caso, el circuito recibe el nombre de *detector de paridad*. No obstante, se debe tener presente que el bit de paridad solo permite saber si se ha producido el error de un solo bit, sin indicar su posición.

Ejemplo 6.1 — Generador de paridad par desde los unos. Dado el bloque de datos de 1 byte de longitud 0100 0001 que representa la letra A en código ASCII, determina el bit de paridad par asociado a dicho bloque de datos.

Solución:

Como el número de unos en el bloque de datos es 2 (par), el bit de paridad es 0. El bloque resultante sería 0100 0001 0. Para que la comprobación de paridad indique que el bloque es correcto, la comprobación se realiza con el complemento del cálculo del bit de paridad sobre el bloque completo (incluido el bit de paridad). ■

> En Logisim no existe un módulo para determinación de la paridad, ya que las puertas XOR se pueden configurar con el comportamiento de determinación de la paridad. De este modo, su salida es 1 cuando el número de entradas con valor 1 es impar.

Existen circuitos integrados comerciales específicos para el cálculo del bit de paridad, como el 74HC180 que tiene 8 entradas de datos y dos salidas correspondientes al cálculo y la comprobación de la paridad. Dicho circuito añade dos entradas adicionales (T_p, T_i) que indican el tipo de paridad empleado de modo que se facilita la conexión en cascada de varios circuitos para trabajar con bloques de datos de mayor longitud. En la figura 6.9 se ilustra el diagrama lógico equivalente del circuito 74HC180 y su aplicación para generar la paridad de un bloque de 16 bits.

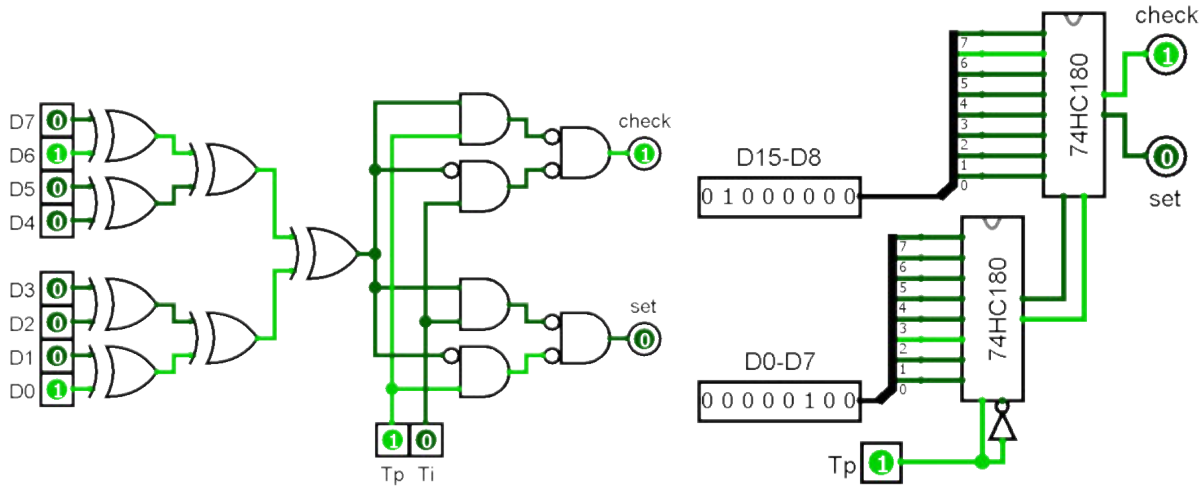


Figura 6.9: Diagrama lógico en Logisim correspondiente al circuito integrado 74HC180 (izda.), y esquema de uso para obtener un comprobador de paridad de 16 bits (dcha.).

6.3 Conversores de código

Un *conversor de código* es un módulo combinacional que transforma un código digital en otro diferente. Para cualquier pareja de códigos binarios dados se puede implementar un conversor partiendo de la tabla de verdad cuya entrada es el código original y la salida el código de destino.

Ejemplo 6.2 — Especificación de conversor de código mediante tabla de verdad. Las tablas de verdad de la figura adjunta indican el cambio de código para dos conversores $A \rightarrow B$. Determina las funciones lógicas que permiten obtener el código B a partir del código A para cada uno de los conversores.

Conversor 1

	Cod. A		Cod.B		
	A_1	A_0	B_1	B_0	
0	0	0	0	1	1
1	0	1	1	0	2
2	1	0	1	1	3
3	1	1	0	0	0

Conversor 2

	Cod. A		Cod.B				
	A_1	A_0	B_3	B_2	B_1	B_0	
0	0	0	0	0	0	1	1
1	0	1	0	0	1	0	2
2	1	0	0	1	0	1	5
3	1	1	1	1	1	0	14

Solución:

Por inspección directa de las tablas de verdad es posible obtener las expresiones lógicas para el código de destino en cada uno de los dos conversores:

Conv. 1

$$B_1 = A_1 \oplus A_0$$
$$B_0 = A_0'$$

Conv. 2

$$B_3 = A_1 \cdot A_0$$
$$B_2 = A_1$$
$$B_1 = A_0$$
$$B_0 = A_0'$$

Este tipo de módulo se puede implementar alternativamente mediante:

(A) Puertas lógicas elementales. La función lógica de cada uno de los bits del código de salida se obtiene a partir de la tabla de verdad mediante minimización (SOP o POS) obtenida con K-maps o alguna estrategia alternativa de minimización de funciones lógicas.

(B) Decodificador (DEC). La función de cada bit del código de salida se puede implementar a partir de sus términos canónicos mediante un DEC cuya entrada es el código a convertir. También es posible emplear multiplexores para la implementación del conversor. Sin embargo, es preciso un MUX por cada uno de los bits de código de salida. Por tanto, en una simulación con Logisim es preciso configura MUX multibit, con un número de bits igual al del código deseado en la salida del conversor de código.

(C) DEC-COD concatenados. Empleando una pareja DEC/COD para decodificar a decimal el código a convertir y emplear un COD conectado en cascada para obtener la nueva codificación.

(D) Memorias ROM. En este caso el código de entrada corresponde con la dirección de memoria en la que se almacena el código de salida.

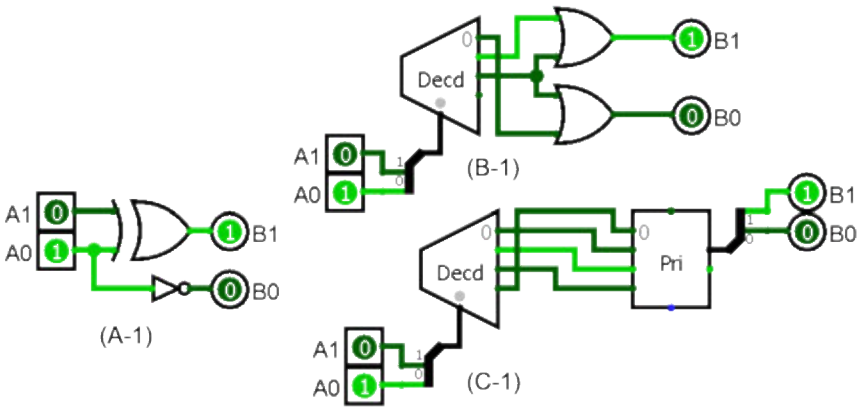
Ejemplo 6.3 — Implementación de un conversor de código. Dados los conversores de código del ejemplo anterior, implementa los circuitos de conversión correspondientes para simulación lógica con Logisim mediante: (A) puertas lógicas, (B) DEC, y (C) concatenación DEC-COD.

Solución:

Las figuras adjuntas muestran los circuitos de implementación resultantes empleando: (A) puertas lógicas, (B) DEC, y (C) concatenación DEC-COD.

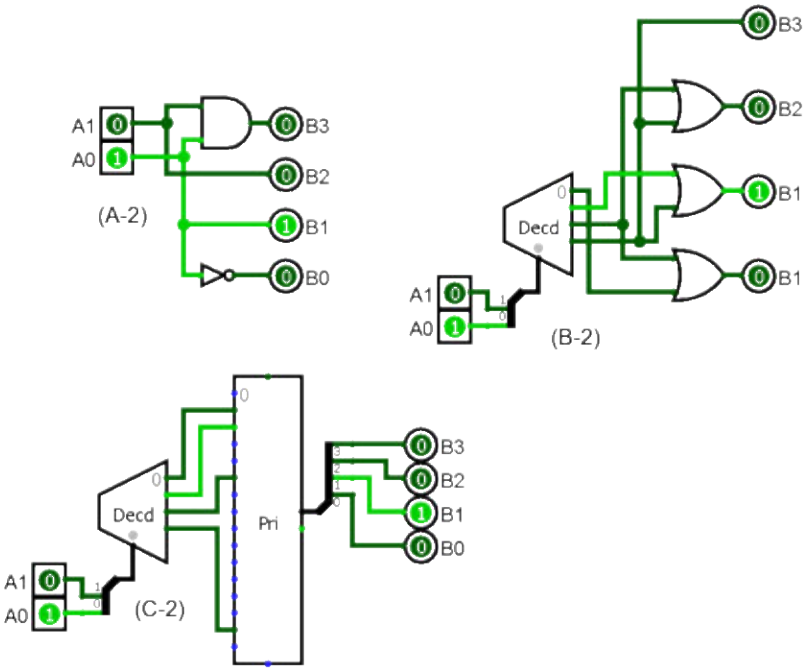
Conversor 1

	Cod. A		Cod.B		
	A_1	A_0	B_1	B_0	
0	0	0	0	1	1
1	0	1	1	0	2
2	1	0	1	1	3
3	1	1	0	0	0



Conversor 2

	Cod. A		Cod.B				
	A_1	A_0	B_3	B_2	B_1	B_0	
0	0	0	0	0	0	1	1
1	0	1	0	0	1	0	2
2	1	0	0	1	0	1	5
3	1	1	1	1	1	0	14



> Cuando los conversores de código son complejos por tener un número elevado de bits, su diseño se puede simplificar mediante empleo de memorias ROM. En una memoria ROM la entrada indica la dirección de memoria del elemento almacenado que se obtiene a la salida. De este modo la entrada constituye el código que se transforma en código almacenado en la posición de memoria direccionada.

6.4 Módulos aritméticos

En las secciones siguientes se describe los módulos combinacionales relacionados con las operaciones aritméticas elementales.

6.4.1 El semisumador

El *semisumador* (*half adder* o HA) es el módulo aritmético más sencillo, ya que a partir de dos bits de entrada A y B, obtiene la suma S y el acarreo correspondiente C_{out}, sin tener en cuenta la existencia de un acarreo de entrada C_{in}.

La suma de los bits de entrada se obtiene mediante la operación lógica XOR y el valor del acarreo mediante la operación lógica AND. Esto es, el acarreo de salida se produce cuando A y B valen simultáneamente 1. En la figura 6.10 se muestra el diagrama de bloques del semisumador, su tabla de verdad y la implementación mediante puertas lógicas. Con dicho circuito no es posible propagar el acarreo pues se necesita una entrada correspondiente al acarreo (C_{in}) obtenido como salida de módulos previos.

A	B	C_{out}	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

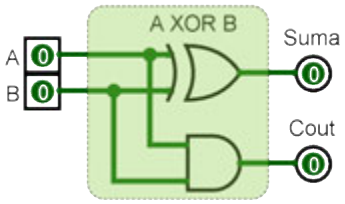


Figura 6.10: Tabla de verdad del semisumador de 2 bits y diagrama del circuito correspondiente.

Como se observa en la tabla de verdad del semisumador (ver figura 6.10-izda.) si se considera el valor binario [C_{out} S] este codifica tanto el valor de la suma total, como el número de unos existentes en las entradas A y B. En este caso el valor del acarreo C_{out} es el bit más significativo y la suma S, el menos significativo.

> Recuerda la diferencia entre la suma aritmética —tratada en esta sección— y la suma lógica (operación OR), ya que su valor no coincide cuando ambos bits de entrada valen 1.

6.4.2 El sumador completo

Como se indicó en la sección anterior, el módulo semisumador no es capaz de propagar el acarreo, ya que no considera acarreo en su entrada. Por este motivo surge la necesidad de considerar el diseño del *sumador completo* (*full adder* o FA). Como se puede observar en la figura 6.11-izda, dicho módulo tiene en cuenta el acarreo de entrada (C_{in}).

C_{in}	A	B	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

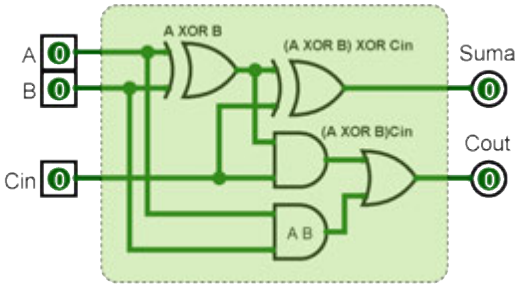


Figura 6.11: Tabla de verdad del sumador completo de 2 bits y diagrama del circuito correspondiente.

En la tabla de verdad del sumador completo se comprueba que el valor binario codificado por C_{out} (MSB) y S (LSB) ofrece tanto el valor de la suma total, como el número de entradas activas del FA (C_{in}, A y B). Este es un resultado interesante, pues es la base para determinar el número de entradas activas a un circuito, o el número de unos en un bloque de datos. De modo alternativo el sumador completo se puede obtener mediante conexión en cascada de dos semisumadores (HA), según se muestra en la figura 6.12.

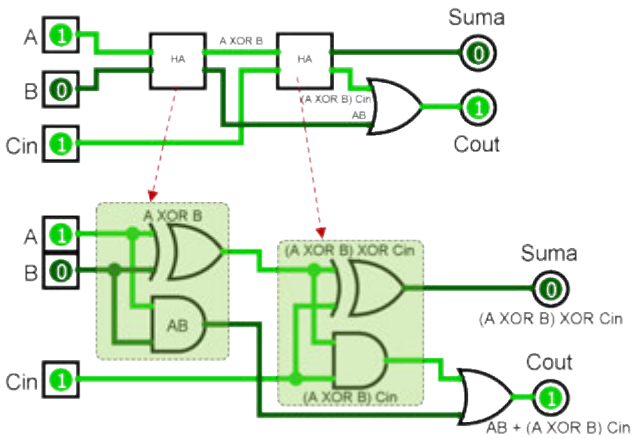


Figura 6.12: Diagrama del circuito lógico del sumador completo (FA) de dos bits obtenido a partir de dos semisumadores (sup.), y detalle (inf.).

6.4.3 Análisis de desbordamiento

En una operación aritmética se produce desbordamiento (*overflow*) cuando el resultado de la operación no es representable con el número de bits disponible.

El rango de representación en C2 con n bits es $[-2^{n-1}, 2^{n-1}-1]$. Por ejemplo, con $n=4$ dicho rango es $[-8,7]$ y con $n=5$ es $[-16,15]$. Cuando se suman dos números del mismo signo en C2, se produce desbordamiento si el resultado excede los límites del rango para el número de bits empleado. Cuando se suman dos números de n bits en C2, el resultado puede tener un tamaño de $n+1$ bits, siendo el bit extra señalizado mediante el bit de acarreo obtenido en la suma. Sin embargo, el valor de dicho bit no es indicativo de una situación de desbordamiento.

Ejemplo 6.4 — Desbordamiento para la suma en C2. Calcula la suma $7+1$ en C2 con 4 bits.

Solución:

Como se emplean 4 bits, el bit MSB se reserva para el signo. Sin embargo, esto no afecta a la representación en los sumandos de este ejemplo, ya que se trata de dos números positivos: $7_{(10} = 0111_{C2}$ y $1_{(10} = 0001_{C2}$. Realizando la suma en binario se tiene:

$$\begin{array}{cccccl} & 0 & 1 & 1 & 1 & \rightarrow 7_{(10} \\ + & 0 & 0 & 0 & 1 & \rightarrow 1_{(10} \\ \hline \mathbf{0} & 1 & 0 & 0 & 0 & \rightarrow -8_{(10}\ \textit{¡ERROR!} \end{array}$$

En este caso, aunque no hay acarreo final (el bit correspondiente es 0), el resultado no es correcto pues se produce desbordamiento, ya que el resultado correcto $8_{(10}$ queda fuera del rango representable en C2 con 4 bits, que es $[-8,7]$. Para resolver esta situación habría que añadir un bit a la representación de los números empleados en esta operación. ■

En los sistemas digitales es muy práctico detectar automáticamente la situación de desbordamiento al realizar una operación aritmética. Veamos cómo se consigue en una operación de suma como la del ejemplo anterior. Para ello, se debe analizar el bit de signo de los sumandos (debe ser igual), el bit de signo del resultado (distinto al de los sumandos), y los acarreos obtenidos en las distintas fases de la suma. El desbordamiento se produce si concurren las condiciones siguientes (ver figura 6.13):

- Al sumar dos números positivos el resultado obtenido es negativo. Esto es, $A_{n-1} = B_{n-1} = 0$ y $C_{\text{in}} = 1$, ya que en este caso se tiene que $S_{n-1} = 1$ con $C_{\text{out}} = 0$.
- Al sumar dos números negativos el resultado obtenido es positivo. Esto es, $A_{n-1} = B_{n-1} = 1$ y $C_{\text{in}} = 0$, puesto que $S_{n-1} = 0$ con $C_{\text{out}} = 1$.

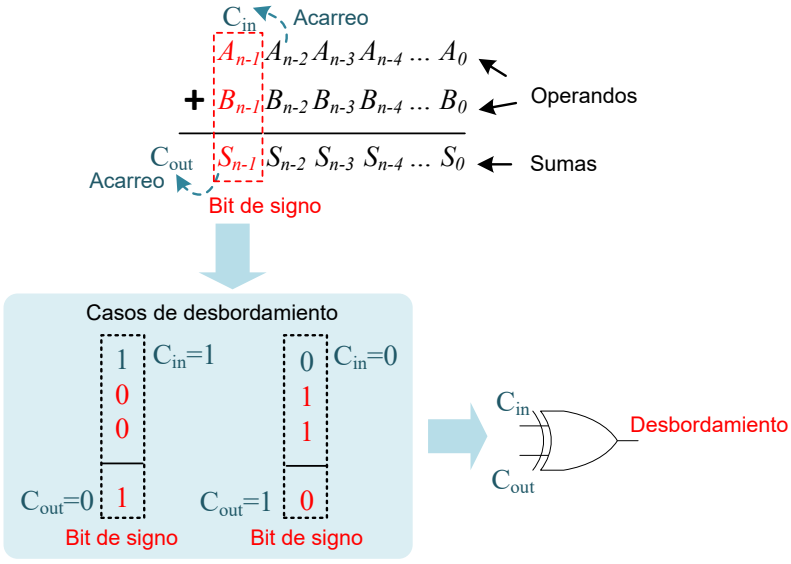


Figura 6.13: Condiciones de aparición de desbordamiento en la suma.

Para comprobar las condiciones mencionadas, es posible realizar una comparación en la que solo intervengan el acarreo previo (C_{in}) y el resultante (C_{out}) cuando se suma el bit MSB de los operandos, ya que se produce desbordamiento siempre que los acarreos mencionados son distintos. Por tanto, la condición de desbordamiento se detecta mediante la función XOR de dichos acarreos. Es decir, el desbordamiento se produce siempre que en la suma del bit más significativo se cumple:

$$C_{in} \oplus C_{out} = 1$$

6.4.4 Sumador paralelo con acarreo serie

El *sumador paralelo con acarreo serie* es un circuito combinacional en el que la suma se realiza bit a bit mediante sumadores completos (FA), teniendo en cuenta que el acarreo generado en la suma de dos bits se propaga a la suma siguiente de mayor orden. En la figura 6.14 se muestra el diagrama del circuito lógico del sumador paralelo de 4 bits (*nibble*) en el que se emplean cuatro sumadores completos de 1 bit. En este diagrama se tiene en cuenta también la presencia de un acarreo de entrada y de salida para todo el conjunto, así como la comprobación de desbordamiento en la suma mediante la operación XOR explicada en la sección anterior.

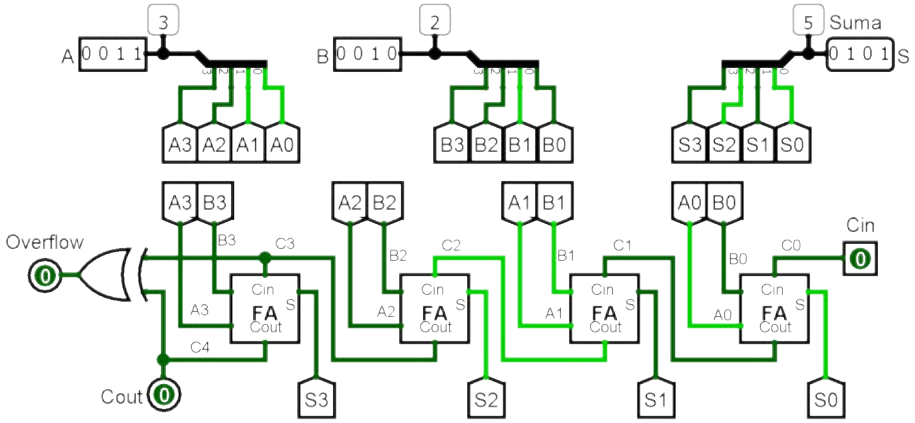


Figura 6.14: Diagrama de circuito lógico de sumador paralelo de 4 bits con acarreo serie.

Este sumador realiza la suma de cada pareja de bits en paralelo. Sin embargo, el acarreo final se obtiene una vez que se han propagado los acarreos por todas las etapas del sumador. Dicho de otro modo, el acarreo de entrada en una etapa no está disponible hasta que se genera el acarreo de salida de la etapa anterior. Por tanto, este sumador también se conoce como *sumador paralelo con acarreo retardado* (*ripple carry adder*).

La figura 6.15 muestra el retardo para obtener el acarreo de salida para cada etapa. Por ejemplo, si el retardo en cada etapa es de 20 ns, no se puede garantizar la lectura correcta del acarreo de salida del sumador C_{out} hasta que no han transcurrido 80 ns. De modo similar, no es posible garantizar que el valor del bit S_3 de la suma sea correcto hasta que han transcurrido 60 ns.

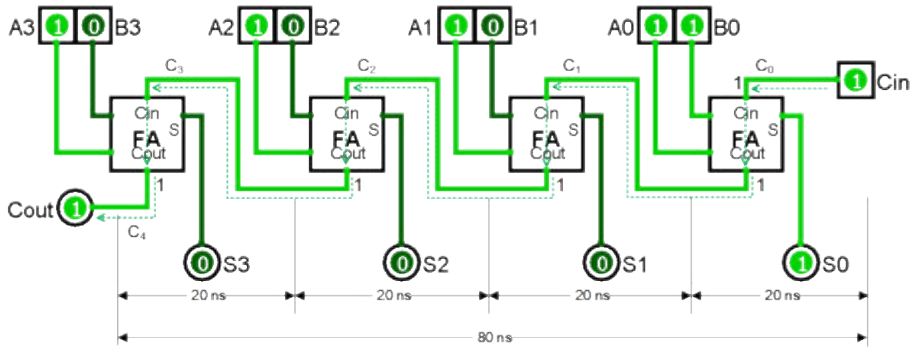


Figura 6.15: Diagrama del circuito lógico del sumador paralelo con acarreo serie señalando el retardo acumulado en la propagación del acarreo.

Por tanto, el sumador paralelo con acarreo retardado presenta un retraso en la obtención del resultado final por la acumulación del retardo en cada etapa de la suma. Cuando se desea un sistema más rápido se emplean sumadores más complejos que realizan el cálculo del acarreo de modo anticipado (*carry look-ahead adder*). En estos últimos se realiza un cambio de variable para expresar tanto los acarreos como las sumas en función de dos nuevas variables definidas para cada etapa.

- Acarreo propagado: $P_i = A_i \oplus B_i$
- Acarreo generado: $G_i = A_i \cdot B_i$

La figura 6.16 muestra en el circuito lógico del sumador completo los valores de los acarreos mencionados.

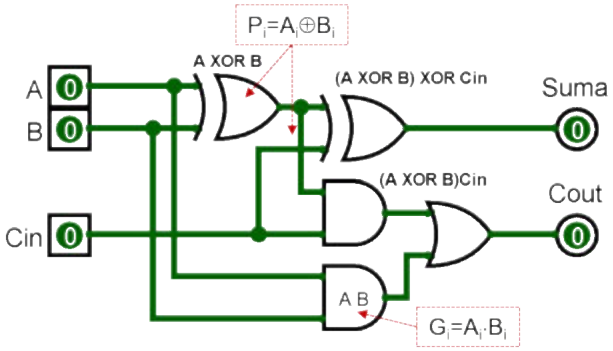


Figura 6.16: Sumador completo de 1 bit mostrando el acarreo propagado (P_i) y generado (G_i).

La figura 6.16 muestra que la suma y el acarreo de salida de cada etapa de la suma son:

$$S_i = P_i \oplus C_i$$
$$C_{i+1} = G_i + P_i \cdot C_i$$

Teniendo en cuenta las ecuaciones del acarreo C_{i+1}, para un sumador paralelo con acarreo anticipado de 4 bits, los acarreos de salida de cada etapa se obtienen como (con $C_0 \equiv C_{\text{in}}$ y $C_4 \equiv C_{\text{out}}$):

$$C_1 = G_0 + P_0 C_0$$
$$C_2 = G_1 + P_1 C_1 = G_1 + P_1 G_0 + P_1 P_0 C_0$$
$$C_3 = G_2 + P_2 C_2 = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_0$$
$$C_4 = G_3 + P_3 C_3 = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 C_0$$

Este sumador se puede emplear como un módulo para realizar diseño jerárquico y obtener sumadores más complejos. Para facilitar dicho diseño el módulo sumador de 4 bits debería proporcionar como salidas adicionales los acarreos generado y propagado del bloque completo:

$$GG = G_3 + P_3 C_3 = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 \quad \text{(Acarreo generado)}$$
$$PG = P_3 P_2 P_1 P_0 \quad \text{(Acarreo propagado)}$$

> Para la simulación de circuitos aritméticos, Logisim proporciona un módulo sumador que permite indicar —en sus atributos— el número de bits de las entradas de datos a sumar. Además, incluye tanto un acarreo de entrada como de salida.

6.4.5 Sumador-Restador

Para realizar la operación de resta se diseñan circuitos aritméticos combinacionales análogos a los sumadores. Si tenemos en cuenta que la representación binaria de números negativos en C2 permite efectuar la suma de números con signo, la resta se puede considerar la suma del sustraendo expresado en C2. Dicho de otro modo, tanto la suma como la resta se implementan como una suma si se emplea complemento a dos para representar los números negativos.

El circuito de la figura 6.17 emplea el subcircuito sumador de 4 bits de la figura 6.14 para sumar dos números A y B expresados en C2. Además, se añade una entrada de control (Suma$'$/Resta) que controla el

tipo de operación aritmética realizada. Las puertas XOR permiten cambiar el signo de B en la operación de resta si Suma$'$/Resta $= 1$. Al sumar el acarreo de entrada se obtiene el cambio de signo de B expresado en C2. Resumiendo:

- SI Suma$'$/Resta $= 0$, se lleva a cabo $A + B$, ya que las puertas XOR dejan inalterada la señal B y en la suma no se añade ningún acarreo de entrada.
- SI Suma$'$/Resta $= 1$, se realiza la operación $A + B' + 1 = A - B$. Esto es, la suma de A y B con B expresado en C2.

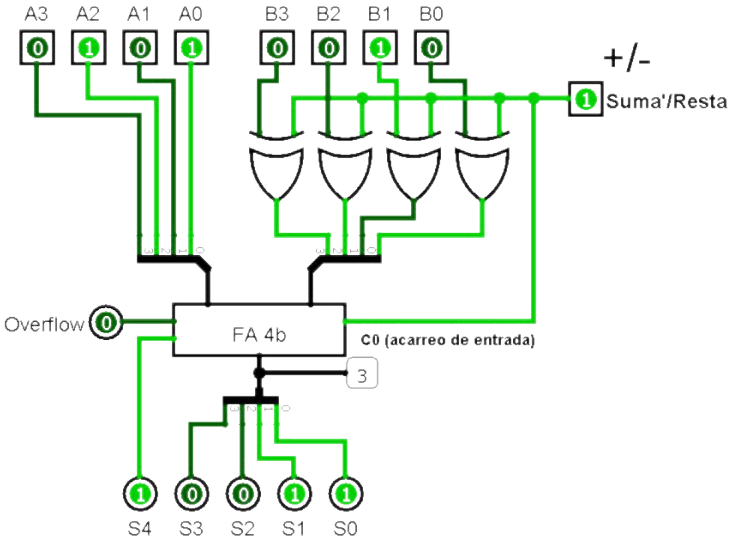


Figura 6.17: Diagrama de circuito lógico sumador-restador de 4 bits.

6.4.6 Sumador BCD

La suma en BCD presenta diferencias respecto a la suma en binario natural, ya que el código BCD no emplea los últimos 6 códigos construidos con 4 bits (del 10 al 15). Por tanto, se trata de un código incompleto. Para hacer la suma en BCD se debe añadir un circuito que detecte si el resultado excede el valor $1001_{(2} = 9_{(10}$. Esta circunstancia se indica mediante el bit de acarreo. Entonces para obtener un código BCD válido, se suma el valor $0110_{(2} = 6_{(10}$ al resultado. El circuito de la figura 6.18 muestra el diagrama de un circuito lógico utilizando el subcircuito del sumador completo de 4 bits de la figura 6.14 para simular el funcionamiento del sumador de BCD. Dicho circuito añade el elemento display de dígitos hexadecimales (*hex digit display*) para mostrar los operandos y el resultado final.

El circuito que detecta que el resultado excede el valor $1001_{(2} = 9_{(10}$ es un circuito combinacional cuyas entradas son los bits del resultado de la suma en binario natural y el acarreo de salida de dicha suma.

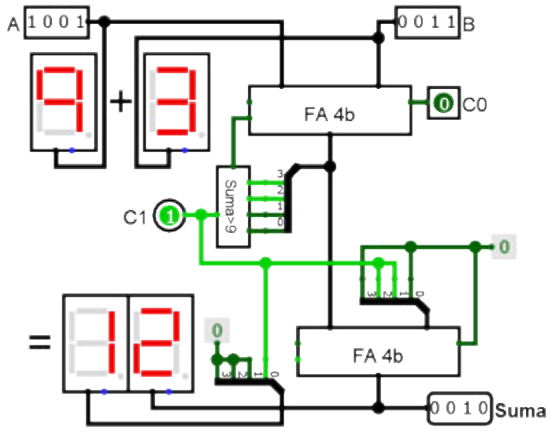


Figura 6.18: Diagrama de circuito lógico sumador de BCD.

6.4.7 Módulos de multiplicación y división

La multiplicación y la división son dos operaciones aritméticas que se pueden calcular fácilmente mediante operaciones de suma y resta. La multiplicación se puede calcular como la suma del multiplicando repetida tantas veces como indique el multiplicador. La división consiste en la resta sucesiva del sustraendo al minuendo para determinar el número de veces que el primero está contenido en el minuendo. En cualquier caso es posible diseñar circuitos dedicados que realicen las operaciones de multiplicación y división. Por ejemplo, en la figura 6.19 se ilustra el diagrama de un circuito combinacional de multiplicación que emplea sumadores completos (FA) de un bit y puertas lógicas adicionales para obtener la multiplicación de operandos con 3 bits.

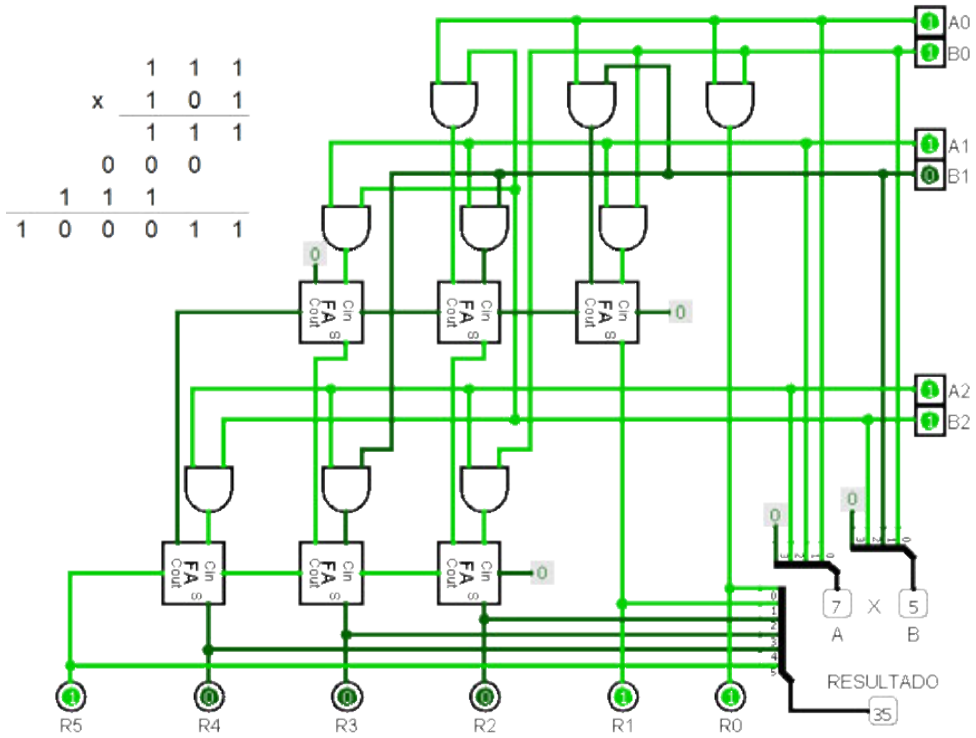


Figura 6.19: Diagrama de circuito lógico multiplicador de 3 bits.

> Recuerda que cuando el factor de multiplicación/división es potencia de 2, el resultado se puede obtener mediante desplazamiento hacia la izquierda/derecha tantas posiciones como indique el exponente empleado para obtener el multiplicador/ divisor. Así, el desplazamiento de una posición equivale a un factor de 2, dos posiciones a un factor de 4 y así sucesivamente. Logisim incorpora el elemento *shifter* de la librería aritmética que proporciona esta funcionalidad de desplazamiento.

6.4.8 Cálculo del número de entradas activas

En ciertas situaciones es preciso conocer el número de entradas activas (a nivel alto) en un sistema digital. Para realizar este cálculo todas las entradas son igualmente significativas. Por tanto, el orden en el que se toman es indiferente. El cálculo emplea sumadores completos, ya que la salida de estos es un valor binario de dos bits $[C_{\text{out}}\, S]$ cuyo valor decimal es el número de entradas activas en el sumador. Si el número de entradas es superior a 3, entonces se pueden hacer dos sumas independientes con dos sumadores completos que permiten evaluar hasta 6 entradas. El resultado final se obtiene mediante una suma en paralelo de las dos sumas parciales previas.

En la figura 6.20 se muestra el diagrama de un circuito lógico que obtiene el número de entradas activas de un total de 7. Para conseguirlo, emplea varios sumadores completos de 1 bit en los que se realiza la suma bit a bit de modo ordenado para las 7 entradas.

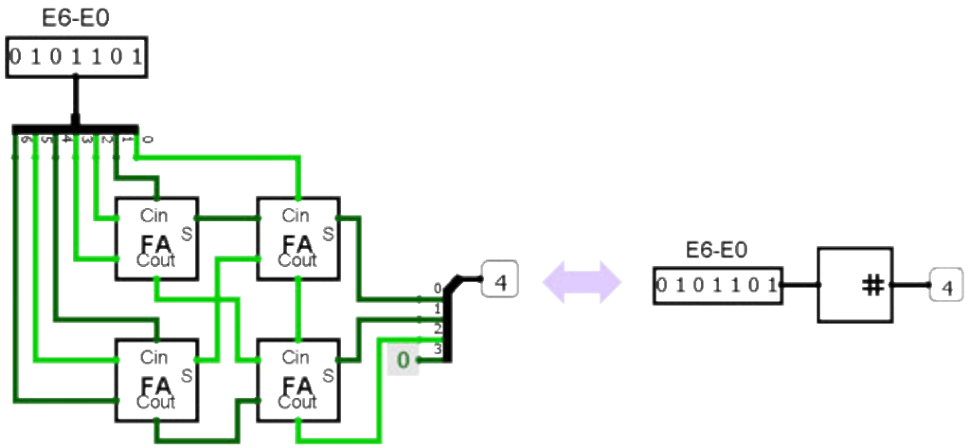


Figura 6.20: Diagrama de circuito lógico para cálculo de número de entradas activas mediante sumadores completos de 1 bit.

> Logisim proporciona el elemento *bit adder* de la librería aritmética que permite obtener el número de bits activos en su entrada multibit (ver figura 6.20-dcha.).

6.5 Unidad Aritmética Lógica

Una unidad aritmética lógica o UAL (en inglés, *arithmetic logic unit o ALU*) es un módulo combinacional capaz de efectuar varias operaciones con sus señales de entrada: de tipo lógico (OR, AND, etc.) y aritmético (suma, resta, etc.). Una ALU tiene tres tipos de señales:

1. *Datos*. Proporcionan los operandos de las operaciones, su resultado y el acarreo final.
2. *Control*. Indican el tipo de operación que se obtiene como resultado.
3. *Auxiliares*. Son señales que facilitan la interconexión con otros módulos funcionales.

A modo de ejemplo, a continuación se muestra el proceso de diseño de una ALU sencilla para simulación lógica, con las características siguientes:

- Dos operandos de entrada de 4 bits: A y B.
- Cuatro señales de control para seleccionar la operación de salida: $[S_3\,S_2\,S_1\,S_0]$. Con ellas se podría codificar hasta 16 operaciones distintas. Sin embargo, en este caso se implementan solo 12: 4 operaciones lógicas y 8 aritméticas.
- Cinco señales de salida para el resultado de la operación $[f_3\,f_2\,f_1\,f_0]$ y el acarreo de salida (C_{out}).

La figura 6.21 muestra el diagrama global de la ALU propuesta y la estrategia de implementación para simulación lógica mediante Logisim. En el diagrama del circuito lógico se ilustra como se emplea la señal S_3 para discriminar entre las operaciones lógicas y las aritméticas al utilizarla como señal de selección del MUX 2×1 de salida que dirige hacia la salida el resultado de la operación deseada.

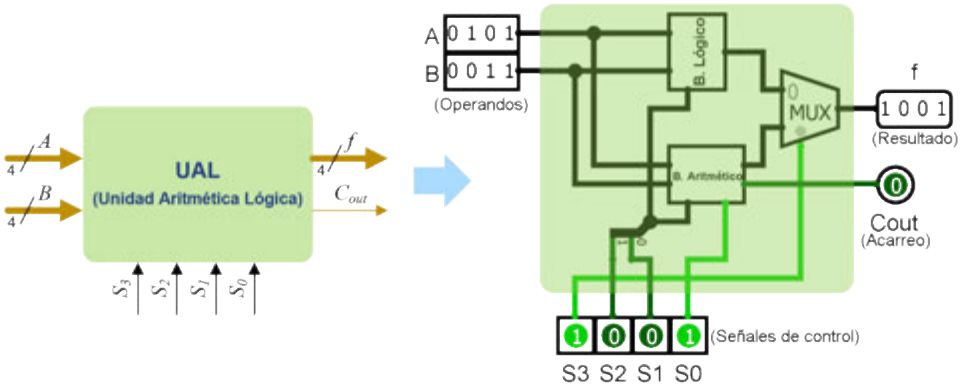


Figura 6.21: Diagrama de la ALU propuesta (izda.), y circuito lógico correspondiente para simulación (dcha.).

A continuación se describe en detalle la implementación de los bloques de operaciones lógicas y aritméticas. El bloque lógico se implementa mediante un MUX 4×1 multibit, con S_2 y S_1 como entradas de selección para obtener a la salida el resultado de la operación lógica seleccionada. La figura 6.22 muestra los detalles del diagrama del circuito que permite simular su funcionamiento lógico. Es importante distinguir el empleo de elementos multibit para simplificar el circuito resultante. En este caso, las puertas lógicas actúan bit a bit sobre los operandos multibit.

S_3=0 (Op. lógica)		
S_2	**S_1**	**Op.**
0	0	AB
0	1	$A+B$
1	0	A'
1	1	$A \oplus B$

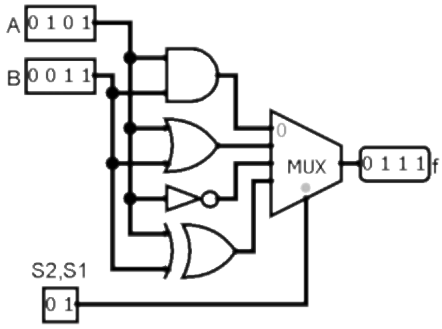


Figura 6.22: Tabla de selección de operación lógica (izda.), y diagrama de circuito correspondiente con puertas lógicas y MUX 4×1 multibit (dcha.).

El diseño del bloque aritmético de la ALU (ver figura 6.23) se basa en el uso de un sumador completo de 4 bits (ver figura 6.14), ya que en todas las operaciones aritméticas se realiza la operación de suma. Para completar la operación deseada es preciso que el operando B sufra una conversión dependiente de la operación seleccionada.

S_3=1 (Op. aritmética)				
S_2	**S_1**	**S_0**	**Op.**	**D**
0	0	0	$A+B$	B
0	0	1	$A+B+1$	B
0	1	0	$A-1$	1111
0	1	1	$A+1111+1$	1111
1	0	0	A	0000
1	0	1	$A+1$	0000
1	1	0	$A-B-1$	B'
1	1	1	$A-B$	B'

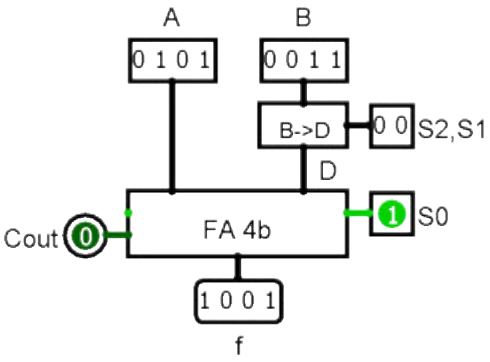


Figura 6.23: (izda.) Tabla de selección de operación aritmética, y (dcha.) diagrama de circuito lógico correspondiente.

La figura 6.24 muestra los detalles del diseño del bloque combinacional de conversión del operando B. Para realizar dicho diseño se tiene en cuenta los valores de las entradas S_2 y S_1. Dichos valores seleccionan tanto el tipo de operación, como el valor de salida D del bloque en función del valor de la entrada B. En este caso se han empleado subíndices tanto en B_i como D_i para señalar que se trata de señales multibit. El diseño combinacional proporciona el valor para D_i:

$$D_i = S_2' B_i + S_1 B_i'$$

Para la implementación de este bloque en Logisim se emplea el elemento extensor de bits (*bit extender*), de la librería *Wiring*, que permite realizar operaciones de tipo multibit entre las variables S_i y B_i. La extensión de bits para S_2 y S_1, se obtiene replicando su valor al conjunto de los 4 bits requeridos para realizar las operaciones bit a bit con la variable B.

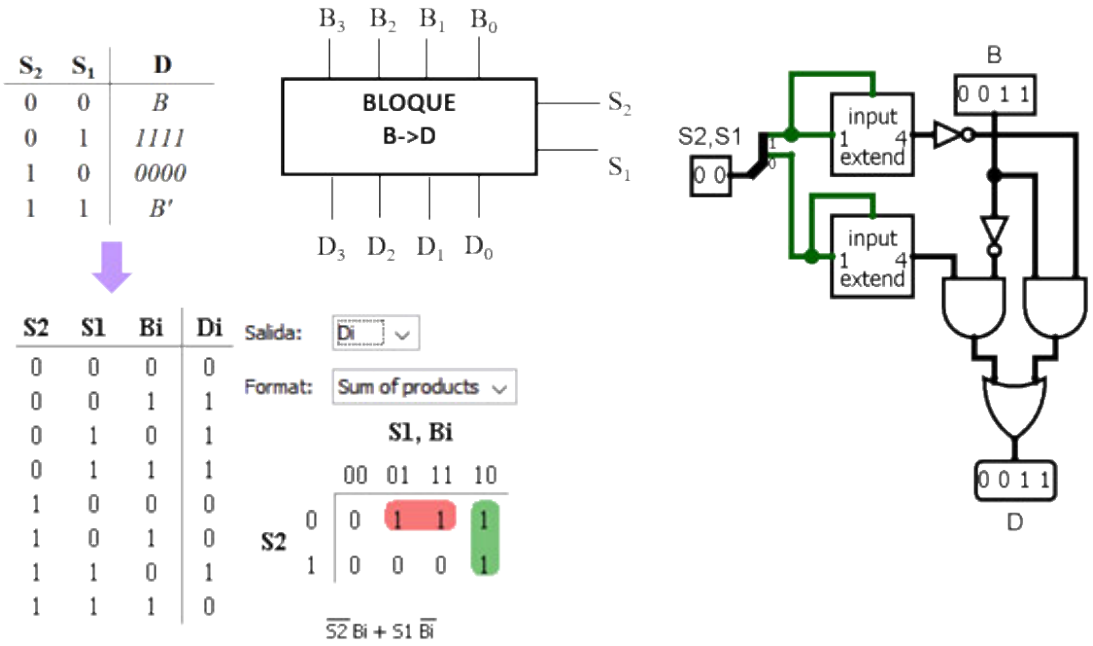


Figura 6.24: Diseño del bloque de conversión del operando *B* de entrada al módulo aritmético de la ALU junto con el diagrama del circuito lógico para simulación.

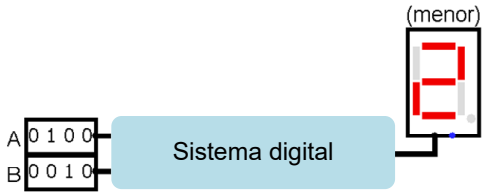
Conceptos clave

- ☑ Implementación de comparador multibit.
- ☑ Bit de paridad: generación y detección.
- ☑ Implementación del semisumador y el sumador completo de 1 bit.
- ☑ Implementación del sumador en paralelo con acarreo serie.
- ☑ Condiciones de desbordamiento y detección.
- ☑ Implementación Sumador/restador y sumador BCD.
- ☑ Implementación de ALU sencilla y uso de MUX para selección de resultado en la salida.

Problemas propuestos

Problema 6.1 Diseña un circuito digital que partiendo de dos entradas (A y B) de 4 bits obtenidas mediante sendos diales, muestre el valor menor en un display de 7 segmentos según muestra la figura adjunta. Contesta razonadamente a las cuestiones siguientes:

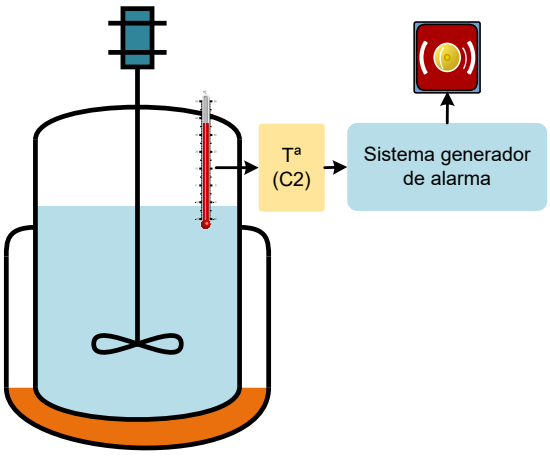
1.- ¿Qué tipo de operación se debe efectuar para determinar el valor menor? ¿Es necesario algún tipo de memoria?
2.- Implementa un circuito con Logisim que permita simular el comportamiento lógico del sistema (*Nota: puedes emplear cualquiera de los módulos funcionales incluidos en las librerías del programa*).
3.- Realiza las modificaciones oportunas al circuito, añadiendo un segundo display que muestre el mayor de los valores. En caso de igualdad ($A = B$) no se debe mostrar nada en ambos displays.



Problema 6.2 En un sistema digital se introducen tres números de 4 bits (A, B y C) sin signo. Diseña el circuito lógico con Logisim correspondiente a un sistema digital que muestre en un display el mayor de los tres números. Modifica el diseño para que el display no muestre nada en caso de que $A = B = C$.

Problema 6.3 Para garantizar la calidad del producto en una planta de producción farmacéutica se debe controlar la temperatura T de un reactor químico en el rango $[-3°\text{C}, +5°\text{C}]$. Para conseguirlo, si el valor de T queda fuera del rango mencionado se debe activar una alarma (mediante nivel alto). El valor de la temperatura se obtiene mediante un sistema sensorial —ya existente— que proporciona el valor de T codificado en complemento a 2. Diseña el sistema digital que activa la alarma mencionada dependiendo del valor de T contestando razonadamente a las cuestiones siguientes:

1.- ¿Cuántos bits se precisan como mínimo para expresar el valor de T?
2.- ¿Qué tipo de operación se debe efectuar para determinar si T está dentro del rango deseado?
3.- Implementa un circuito con Logisim que permita simular el comportamiento del sistema empleando los módulos funcionales incluidos en las librerías disponibles. Recuerda que el sistema sensorial ya proporciona la información de temperatura en C2.



Problema 6.4 Diseña un circuito combinacional de dos entradas con 4 bits A y B cuya salida f sea:

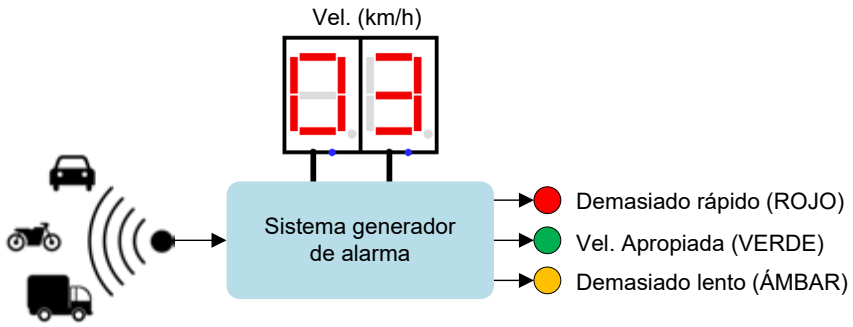
- 0000, si $A > B$ o $A = B$
- B en C1, si $A < B$

Contempla los requisitos siguientes:

1.- Emplea un comparador y los módulos combinacionales necesarios de Logisim para simular el comportamiento lógico del sistema.
2.- Modifica el circuito para que $f = B$ en C2 cuando $A < B$.

Problema 6.5 En la salida de un parking se instala un sistema sensorial que proporciona la velocidad en km/h de los vehículos que abandonan el parking, codificada en BCD. La resolución de dicho sistema sensorial es de 1 km/h. Diseña un circuito de alerta luminosa a los conductores cuando la velocidad del vehículo sea excesiva, al superar 15 km/h (ROJO), o cuando sea demasiado reducida, al no superar los 5 km/h (ÁMBAR). Si la velocidad está en el rango permitido la señal luminosa será de color VERDE. Contesta razonadamente las cuestiones siguientes:

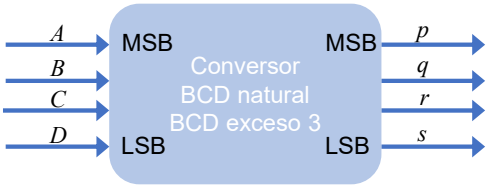
1.- ¿Cuántos bits se precisan como mínimo para codificar la velocidad de los vehículos?
2.- Diseña un circuito con Logisim que muestre la velocidad del vehículo mediante displays de 7 segmentos y active la alerta luminosa correspondiente cuando la velocidad del vehículo está fuera del rango requerido.



Problema 6.6 Implementa un conversor de 4 bits de código BCD a BCD-ex3 (BCD exceso 3). El código de entrada será $[ABCD]$ y el de salida $[pqrs]$, donde cada una de las variables lógicas representa un bit del código correspondiente. Contesta razonadamente a las cuestiones siguientes:

1.- Construye la tabla de verdad del conversor,
2.- Determina las expresiones simplificadas para las funciones de salida p, q, r y s.
3.- ¿Cuáles son las formas canónicas de las funciones de salida?

NOTA: Recuerda que el código BCD-ex3 se obtiene sumando 3 al BCD natural.



Problema 6.7 Diseña un circuito combinacional que convierta un código Gray de cuatro bits en el número binario correspondiente la tabla adjunta. Además del diseño convencional intenta la implementación del circuito con puertas lógicas de tipo OR exclusivo (XOR).

Cod. Gray	Bin. natural
0000	0000
0001	0001
0011	0010
0010	0011
0110	0100
0111	0101
0101	0110
0100	0111
1100	1000
1101	1001
1111	1010
1110	1011
1010	1100
1011	1101
1001	1110
1000	1111

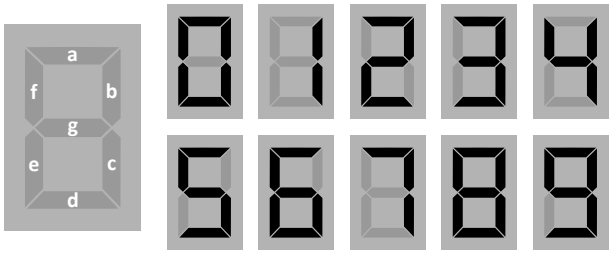
Problema 6.8 Diseña un circuito combinacional para convertir a BCD, un dígito decimal codificado con código binario ponderado $(8,-4,-2,1)$. De este modo, por ejemplo el código 1111 corresponde al dígito: $1\times(+8)+1\times(-4)+1\times(-2)+1\times(+1)=3$

Completa razonadamente los apartados siguientes:

1.- Calcula la tabla de verdad del conversor de código.
2.- Implementa el conversor mediante puertas lógicas.
3.- Repite la implementación mediante combinación de DEC y COD.

Problema 6.9 Diseña un circuito lógico que, partiendo de un código BCD, active un display de 7 segmentos para mostrar el dígito decimal correspondiente. El display de 7 segmentos está formado por LED encargados de la iluminación de cada segmento del display. Cada LED se activa con un nivel alto (1), si su configuración es en cátodo común o con un nivel bajo (0), si su configuración es en ánodo común. Suponer para este ejercicio una configuración en cátodo común y la denominación de segmentos según se indica en la figura adjunta. Completa razonadamente cada uno de los apartados siguientes:

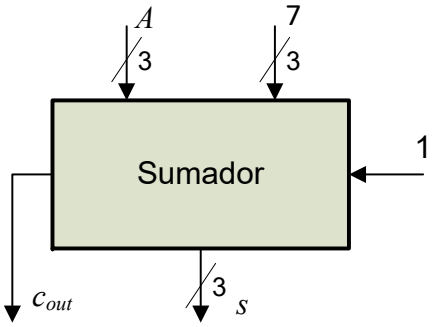
1.- Construye la tabla de verdad para las funciones de activación de los segmentos del display.
2.- Determina las expresiones simplificadas de las funciones de salida.
3.- Implementa el conversor de código mediante una memoria ROM empleando Logisim para simular su comportamiento.



Problema 6.10 Diseña un conversor de código de cuatro bits que convierta un número binario de cuatro bits representado en C2 $[ABCD]$ a su representación en SM (signo magnitud) con 4 bits $[pqrs]$. Se debe tener en cuenta que el código 1000 permanecerá inalterado. Completa razonadamente los apartados siguientes:

1.- Construye la tabla de verdad de las funciones de conversión.
2.- Determina las formas canónicas de las funciones de salida p, q, r y s.
3.- Calcula las expresiones simplificadas de las funciones de salida.
4.- Implementa el conversor mediante concatenación de DEC-COD para simulación con Logisim.

Problema 6.11 El diagrama de la figura adjunta representa un sumador completo (*full adder*) de 3 bits.



Completa razonadamente los apartados siguientes:

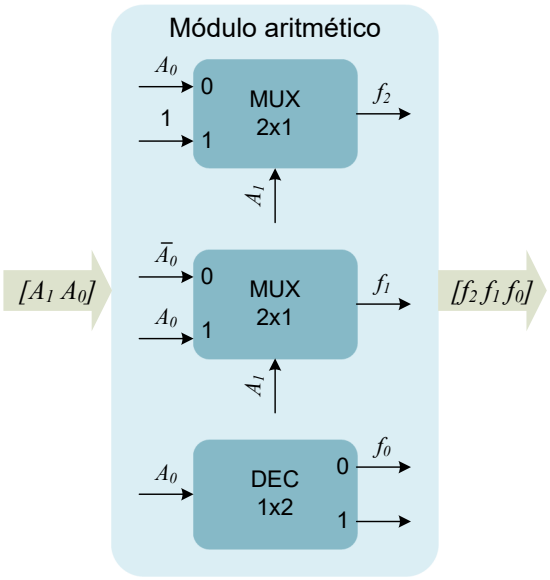
1.- Indica cuáles son las señales de entrada y cuáles las de salida señalando el número de bits de cada una y su valor.
2.- ¿Especifica qué tipo de operación realiza el sistema: lógica, aritmética u otra?
3.- ¿Cuáles son los valores menor y mayor posibles a su salida y con qué valores de entrada se produce cada uno?
4.- Implementa un circuito equivalente con puertas lógicas para simulación con Logisim.

Problema 6.12 Implementa el sumador completo de 1 bit mediante un DEC y puertas lógicas de dos entradas (Sugerencia: construye la tabla de verdad del sumador completo de 1 bit).

Problema 6.13 Implementa mediante Logisim un sumador completo de 4 bits con acarreo anticipado. Emplea diseño jerárquico usando un módulo sumador completo de 1 bit que proporcione como salidas: el acarreo propagado ($P_i = A_i \oplus B_i$), el acarreo generado ($G_i = A_i \cdot B_i$) y la suma ($S_i = P_i \oplus C_i$).

Problema 6.14 Dado el circuito de la figura adjunta contesta razonadamente las cuestiones de los apartados siguientes:

1.- Enumera tanto las variables de entrada como de salida del circuito y determina la tabla de verdad de las salidas respecto de las entradas.
2.- Calcula las formas canónicas de las funciones lógicas de salida.
3.- Expresa la relación aritmética entre la entrada y la salida del sistema.
4.- Implementa un circuito aritmético equivalente al sistema proporcionado inicialmente. Emplea Logisim para realizar la simulación lógica de los circuitos.



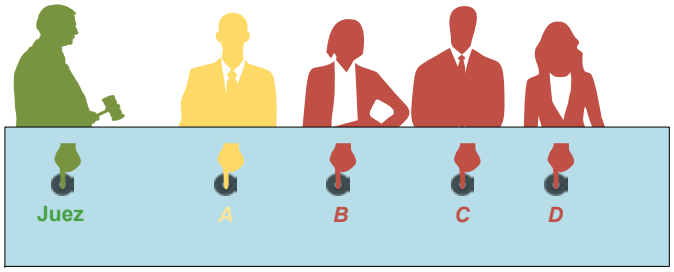
Problema 6.15 Dado un número A de 4 bits codificado en C2 (con signo). Diseña un circuito lógico con Logisim que muestre en un display el valor absoluto del número indicando en el punto decimal (dp) si A es negativo ($dp = 1$).

Problema 6.16 En un sistema digital se introducen dos números A y B de 4 bits en C2. Completa razonadamente los apartados siguientes:

1.- Diseña un circuito que muestre en un display el resultado de la resta $A - B$ cuando esta sea de signo positivo o 0. En caso contrario no se debe mostrar nada.
2.- Modifica el diseño para que se muestre en el display el valor absoluto del resultado indicando el valor del signo en el punto decimal del display (activo para negativo) y (apagado para positivo).

NOTA: Utiliza Logisim con cualquiera de los módulos disponibles.

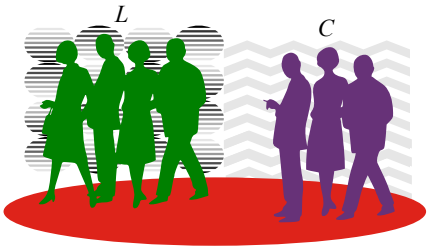
Problema 6.17 Un acusado será juzgado en un juicio con jurado popular formado por cuatro miembros: A, B, C y D. Cada miembro del jurado emitirá su voto (inocente=0 o culpable=1) mediante activación de un interruptor. El veredicto final tendrá en cuenta la suma de votos individuales, pero en caso de empate decidirá el voto de calidad del miembro A del jurado. Se reservará un interruptor especial para el juez, cuya activación cambia el sentido del veredicto final. Diseña un circuito combinacional para simulación lógica con Logisim que implemente la funcionalidad descrita ofreciendo como salida el veredicto.



Problema 6.18 Una comisión parlamentaria está formada por miembros de dos partidos políticos L y C, siendo su composición: 4 miembros de L y 3 de C. Las decisiones de la comisión sobre una propuesta (1=a favor, 0=en contra) se tomarán de acuerdo al siguiente criterio:

- SI hay mayoría de votos entre los miembros de L, ENTONCES esta será la decisión adoptada.
- SI se produce un empate entre los miembros de L, ENTONCES la decisión será la que adopte la mayoría de los miembros de C.

Diseña un circuito lógico que proporcione la decisión de la comisión tomando como entrada los votos de cada uno de los miembros de la misma. Sugiere el esquema de un circuito lógico con Logisim para poder simular el funcionamiento lógico de la comisión.



Problema 6.19 Se tienen las dos calificaciones N_1 y N_2 de un estudiante expresadas en BCD con 4 bits. La calificación máxima posible es 9. Diseña un circuito lógico que obtenga la nota final del estudiante en función del valor de entrada de dos señales de control $[C_1\, C_0]$ según el siguiente criterio:

- 00 $\rightarrow$ nota mínima,
- 01 $\rightarrow$ nota media redondeada por defecto,
- 10 $\rightarrow$ nota media redondeada por exceso, y
- 11 $\rightarrow$ nota máxima.

Elabora un circuito lógico con Logisim que permita simular el funcionamiento lógico del sistema.

III Sistemas Secuenciales

7. Biestables

En un circuito digital sin realimentación, la salida solo depende del valor de sus entradas. La salida cambia en cuanto se modifica el valor de las entradas. En realidad, existe un pequeño retardo en los cambios de las señales de salida derivado del tiempo requerido para la propagación de las señales eléctricas a través de las puertas lógicas que componen el circuito. Los *circuitos combinacionales* estudiados en los capítulos previos se comportan de este y a pesar de sus limitaciones son muy útiles resolviendo problemas habituales en los sistemas digitales.

En los circuitos secuenciales la salida depende tanto de la entrada al circuito, como de su historia previa. Por tanto, son sistemas que precisan de elementos capaces de almacenar información sobre su historia pasada. En este capítulo se inicia el estudio de los fundamentos de los circuitos secuenciales comenzando por sus elementos más simples: los biestables.

7.1 Circuitos digitales realimentados

En ocasiones es preciso que los sistemas digitales actúen de acuerdo a su historia previa. Por ejemplo, si es preciso activar una alarma doméstica de intrusión mediante un detector PIR,[1] la señal de activación no puede ser una función que activa momentáneamente la alarma en función de las entradas, ya que cuando el detector se desactiva la alarma deja de sonar. La figura 7.1 muestra un posible circuito lógico de *activación momentánea* de la alarma.

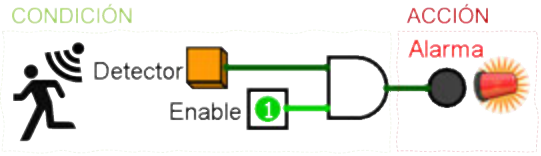


Figura 7.1: Sistema combinacional de activación momentánea de una alarma.

En este circuito se precisa memoria que permita la *activación memorizada o sostenida* de la alarma una vez detectada la intrusión. Dicho circuito se denomina *sistema secuencial* y en él la salida estaría gobernada por el valor de las señales de entrada y el estado previo del circuito (p. ej., la detección de

[1] Sensor pasivo de infrarrojo de movimiento de tipo TODO/NADA.

intrusión). En dicho circuito, cuando se detecta a un intruso, se activa la alarma de modo sostenido, aunque el detector deje de detectar movimiento un instante después.

Definición 7.1 — Acciones momentáneas vs. memorizadas. Las *acciones momentáneas* son aquellas que tienen lugar mientras se cumple la condición que las activa. Por el contrario, las *acciones memorizadas* o *sostenidas* son aquellas que se inician con una condición (*set*) y se mantienen aunque la condición que las activó se deje de cumplir. Estas acciones terminan cuando se cumple una condición de finalización (*reset*). ■

La relación entre condiciones y acciones momentáneas se implementa mediante lógica combinacional. Por el contrario, la relación entre las condiciones de inicio (*set*), fin (*reset*) y las acciones memorizadas se consigue mediante lógica secuencial.

Definición 7.2 — Estado de un sistema digital. El *estado* de un sistema está constituido por el valor de salida de los elementos de memoria del sistema en un instante determinado. En este texto se representa con la letra Q. El estado queda almacenado en los elementos de memoria hasta que se produce un cambio en los mismos. ■

La salida de un sistema secuencial está determinada por el estado almacenado en sus elementos de memoria, bien directamente —coincidiendo en este caso con el estado— o bien mediante una función lógica combinacional dependiente del estado y de la entrada al sistema. Los cambios en la salida de este tipo de sistema se pueden producir de dos modos:

- **Modo asíncrono**. Si el cambio de estado se realiza inmediatamente tras la modificación de las entradas al sistema.
- **Modo síncrono**. Si el cambio de estado se produce cuando cambian las entradas al sistema, pero solo si una señal de control habilita dicho cambio. Dicha señal de control se denomina señal de sincronismo o disparo, ya que es responsable de «sincronizar» o «disparar» el cambio de estado del sistema. La señal de sincronismo puede activar el cambio de estado mientras permanece en un nivel determinado, en este caso se habla de *activación o disparo por nivel*. Mientras que si el cambio de estado tiene lugar cuando la señal de sincronismo sufre un cambio o transición, se conoce como *activación o disparo por flanco*.

Definición 7.3 — Transición de una señal digital. El cambio de nivel lógico de una señal digital se denomina *transición* de dicha señal. ■

Las transiciones de una señal digital generan flancos en su forma de onda de dos tipos (ver figura 7.2):

a) *Flanco de subida* o positivo (*rising edge*): si la señal cambia de nivel bajo a nivel alto (↑).

b) *Flanco de bajada* o negativo (*falling edge*): si la señal cambia de nivel alto a nivel bajo (↓).

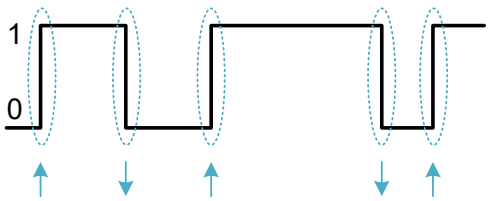


Figura 7.2: Señal lógica en la que se muestran las transiciones y los flancos en su forma de onda.

Para que un circuito pueda almacenar información es preciso que posea *realimentación o feedback* de las señales de salida hacia las entradas del circuito. De este modo la información «recircula» y no se pierde al modificar el valor en las entradas. Por ejemplo, el circuito de la figura 7.3 es capaz de mantener un estado estable en su salida mientras se suministre energía al mismo —proporcionada por la alimentación eléctrica de los inversores—.

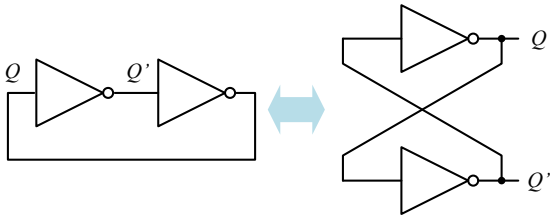


Figura 7.3: Circuito realimentado capaz de mantener su estado.

> Recuerda que cualquier circuito de puertas lógicas necesita alimentación eléctrica para funcionar. El símbolo de la puerta no incluye indicaciones sobre el conexionado eléctrico implicado en el circuito aunque dichas conexiones y elementos son precisos para su funcionamiento. Estos detalles se ilustran en los diagramas de los circuitos electrónicos correspondientes a las puertas lógicas implementadas con transistores (ver figura 3.2, pág. 41).

Los circuitos electrónicos con realimentación se denominan *multivibradores*, ya que oscilan entre los dos posibles estados (0/1) de un circuito digital. Entre los circuitos multivibradores se pueden diferenciar los tipos enumerados a continuación:

1. *Astable*. Sus dos estados no son estables, ya que el sistema oscila permanentemente entre dichos estados. Permanece en cada estado un tiempo fijo[2] antes de evolucionar al estado alternativo.
2. *Monoestable*. Tiene solo un estado estable en el que puede permanecer por tiempo indefinido. El estado no estable se activa mediante una señal de entrada y en él permanece por un tiempo fijo, transcurrido el cual, regresa al estado estable.
3. *Biestable*. Sus dos estados son estables y se activan mediante una o dos señales de entrada. En cada uno de los estados puede permanecer por tiempo indefinido. Este tipo de circuito constituye la celda elemental de memoria de los sistemas digitales. Los biestables pueden ser de tipo *síncrono* o *asíncrono* dependiendo de que su cambio de estado precise o no, respectivamente, de una señal de disparo o sincronismo.

Veamos algunas definiciones importantes relativas a la activación de los circuitos realimentados.

Definición 7.4 — Señal de disparo de un circuito multivibrador. Se denomina *señal de disparo* o *trigger* de un circuito a una señal de sincronismo que habilita (activa o dispara) un cambio de estado en dicho circuito. ■

Definición 7.5 — Disparo por nivel y por flanco. El *disparo por nivel* (*level triggering*) activa el cambio de estado de un circuito cuando la señal de sincronismo se encuentra en un nivel lógico determinado. El *disparo por flanco* (*edge triggering*) activa el cambio de estado del sistema en los instantes de transición de la señal de sincronismo. Dependiendo del tipo de flanco que dispara el cambio de estado del sistema, se habla de disparo por *flanco de subida* o *flanco positivo* y disparo por *flanco de bajada* o *flanco negativo*. ■

Las señales de disparo o sincronismo suelen estar asociadas a una señal periódica o reloj (*CLK*) que permite sincronizar el funcionamiento de sistemas complejos para coordinar sus módulos componentes.

En las secciones siguientes se describen los distintos tipos de biestables, analizando su comportamiento y características principales.

7.1.1 Biestable S-R asíncrono

El circuito presentado en la figura 7.3 exhibe una importante dificultad operativa, ya que no es posible modificar su estado. Para resolver esta cuestión se pueden sustituir los inversores por puertas NOR de dos entradas con una de ellas conectada a nivel bajo (ver figura 7.4). De este modo, la puerta NOR se

[2] Determinado por las características del circuito.

comporta del mismo modo que el inversor, puesto que mientras la entrada libre se mantenga a nivel bajo, el circuito retiene su estado.

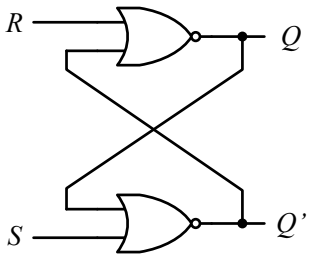
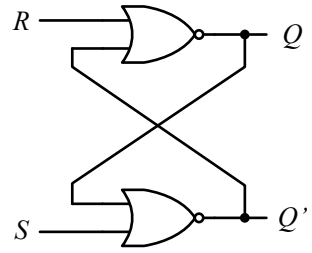


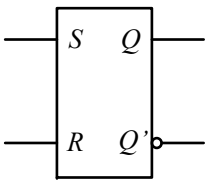
Figura 7.4: Circuito lógico de un *latch* S-R con entradas activas a nivel alto.

Este circuito mantiene el valor de salida Q y Q' mientras $S = R = 0$. En dichas circunstancias, ambas puertas NOR se comportan como los inversores del circuito de la figura 7.3. Si $S = 1$ y $R = 0$ entonces $Q = 1$ ($Q' = 0$); y si $S = 0$ y $R = 1$ entonces $Q = 0$ ($Q' = 1$). En este circuito hay que evitar la situación «prohibida» $S = R = 1$, ya que en este caso los valores de Q y Q' no son consistentes, pues ambos valdrían 0. Si después ambas entradas S y R pasan a 0, es imposible predecir el valor final que alcanzan Q y Q', puesto que dependerá de cuál de las dos puertas NOR sea más rápida en fijar su valor de salida.

El circuito de la figura 7.4 se denomina *biestable S-R asíncrono* o *latch S-R* con entradas activas a nivel alto. Las señales S (*Set*) y R (*Reset*) permiten ajustar el estado Q del *latch* a 1 o 0, respectivamente. Como este circuito emplea puertas NOR, la entrada que vale 1 fija el estado del *latch*, ya que fuerza el valor de salida de la puerta NOR a 0, independientemente del estado realimentado en la otra entrada de la puerta.

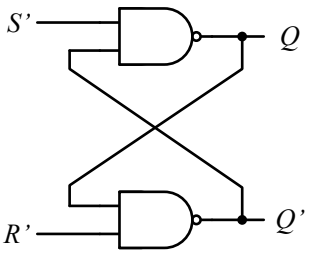
El *latch* S-R con entradas activas a nivel bajo se obtiene sustituyendo las puertas NOR por NAND. En este caso, la entrada que vale 0 es la que fija el valor del estado del *latch*, ya que fuerza el valor de salida de la puerta NAND a 1, independientemente del estado realimentado en la otra entrada de la puerta. La figura 7.5 compara los circuitos lógicos del *latch* S-R con entradas activas a nivel alto y bajo junto a sus símbolos estandarizados y sus tablas de verdad.

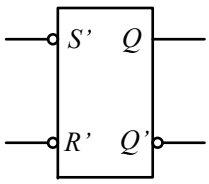




Entradas Salida

S	R	Q_{t+1}	Resultado
0	0	$\mathbf{Q_t}$	*Retiene estado*
0	1	**0**	*Puesta a 0 (reset)*
1	0	**1**	*Puesta a 1 (set)*
1	1	**0**	*Prohibido*





Entradas Salida

S'	R'	Q_{t+1}	Resultado
0	0	**1**	*Prohibido*
0	1	**1**	*Puesta a 1 (set)*
1	0	**0**	*Puesta a 0 (reset)*
1	1	$\mathbf{Q_t}$	*Retiene estado*

Figura 7.5: Circuitos lógicos del *latch* S-R con entradas activas a nivel alto (sup.) y bajo (inf.) junto a sus símbolos estandarizados y las tablas de verdad para cada uno de ellos.

En las tablas de verdad de la figura 7.5 se observa que la salida o estado Q_{t+1} de este tipo de circuito no solo depende de las entradas al mismo S y R, sino también del estado (o salida) previo Q_t. Para distinguir entre los estados de partida y resultante del circuito, se emplean los subíndices t y $t+1$, respectivamente. El instante t viene determinado por cada instante en que el biestable puede cambiar de estado, siendo

Q_t su estado de partida en dicho instante. Mientras que Q_{t+1} el estado alcanzado por el biestable en un instante inmediatamente posterior.

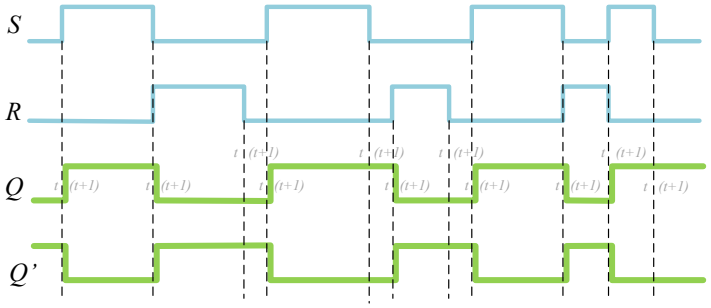
> Recuerda que la nomenclatura Q_t y Q_{t+1} es un modo de distinguir el valor binario para la misma variable lógica en dos instantes sucesivos. Una característica común a todos los biestables es que ofrecen en su salida las dos señales Q y Q' debido la propia construcción física del circuito. Esto evita la necesidad de un inversor adicional.

El biestable S-R asíncrono se emplea para controlar acciones iniciadas con una condición (*Set*) y sostenidas hasta que se cumple una condición de terminación (*Reset*). Es un biestable asíncrono, ya que su estado cambia en el mismo instante en que lo hacen las entradas, sin intervención de una señal de disparo o sincronismo.

En el análisis de sistemas secuenciales cobra un interés especial el empleo del *cronograma* o *diagrama de tiempo* (*timing diagram*) cuya definición se introduce a continuación.

Definición 7.6 — Cronograma de un circuito digital. Un *cronograma* es un diagrama de la evolución temporal de las señales de entrada y salida de un circuito digital. Adquiere especial interés en sistemas cuya salida depende tanto de sus entradas como de su estado previo. ■

Ejemplo 7.1 — Cronograma de un biestable S-R asíncrono. Dado el cronograma adjunto de un biestable S-R asíncrono, realiza el análisis que permite su elaboración partiendo de las señales de entrada S, R y el valor inicial del estado Q.



Solución:

En la elaboración y análisis de cronogramas es útil dibujar líneas guía verticales asociadas a los instantes en que el sistema puede cambiar de estado. Dichas líneas facilitan tanto el análisis del diagrama de tiempos, como su elaboración. Por esta razón en el cronograma proporcionado, las guías mencionadas se sitúan en los instantes en que S y R adquieren el valor 1 que marca el posible cambio de valor de Q. Observa que en las gráficas de Q y Q' los cambios se han desplazado ligeramente a la derecha de las líneas guía. De este modo se indica que el cambio de estado se produce un instante posterior al cambio en las entradas del biestable.

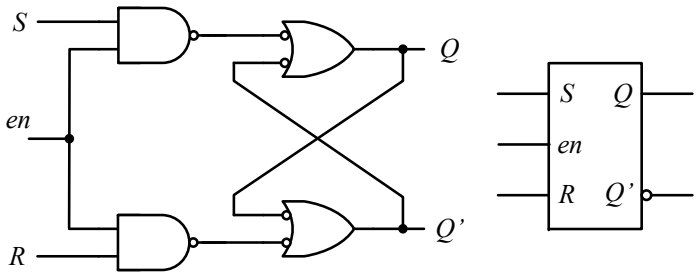
Los valores de S y R no deben valer simultáneamente 1, porque esta situación no permite predecir el estado del sistema. Cuando ambas entradas son 0, el estado se mantiene en su valor precedente. Si $S = 1$ y $R = 0$, entonces $Q = 1$ y cuando $S = 0$ y $R = 1$, entonces $Q = 0$. En este cronograma se ha añadido la señal Q', que es el reflejo especular de la señal Q, respecto al eje de tiempo. Por esta razón lo habitual es representar solo una de dichas señales. ■

7.1.2 Biestable S-R activado por nivel

Para tener mayor control del estado del biestable S-R asíncrono se añade una señal de habilitación (*enable*). En el circuito de alarma empleado como ejemplo, interesa que la alarma se dispare solo si está habilitada evitando de este modo avisos indeseados, aunque el detector de movimiento esté activado. Así se permite la presencia de personas en horarios autorizados sin que la alarma se active.

Para conseguir el comportamiento mencionado empleando el *latch* S-R, se añade una puerta AND en cada entrada del circuito. Dicha puerta habilita, mediante una señal adicional (*en*), el cambio de estado del

biestable. En este caso, se dice que el biestable con la señal de habilitación adicional, se activa por nivel (*gated latch*). La figura 7.6 muestra el circuito lógico del biestable S-R activado por nivel, su símbolo estándar y su tabla de verdad.



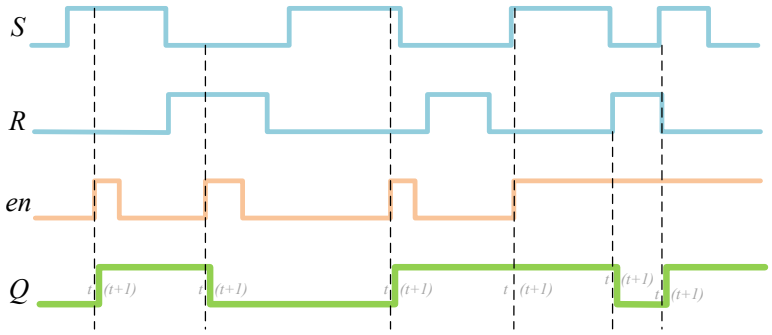
Entradas			Salida	
en	*S*	*R*	Q_{t+1}	**Resultado**
0	x	x	$\mathbf{Q_t}$	*Retiene estado*
1	0	0	$\mathbf{Q_t}$	*Retiene estado*
1	0	1	**0**	*Puesta a 0 (reset)*
1	1	0	**1**	*Puesta a 1 (set)*
1	1	1	**0**	*Prohibido*

Figura 7.6: Circuito lógico del biestable S-R activado por nivel alto con su símbolo estándar y la tabla de verdad correspondiente.

Este tipo de biestable se considera síncrono porque es preciso activar la señal de habilitación (*en*) para que pueda cambiar su estado. Si la señal de habilitación se conecta a través de un inversor se consigue la habilitación por nivel bajo. La activación a nivel bajo se indica en el símbolo del biestable colocando el símbolo de negación (○) en la entrada asociada.

> Recuerda que siempre es posible utilizar una puerta AND para añadir una señal de habilitación a cualquier señal de entrada de un circuito. En el *gated latch* S-R, dicha puerta es interna al módulo y habilita simultáneamente ambas entradas, pero se puede añadir una condición externa (habilitación) mediante puertas adicionales en las entradas del circuito.

Ejemplo 7.2 — Cronograma de un biestable S-R activado por nivel alto. Dado el cronograma del biestable S-R activado por nivel alto de la figura adjunta, realiza el análisis que permita su elaboración partiendo de las entradas *S*-*R*, la señal de activación *en* y el valor inicial del estado *Q*.



Solución:

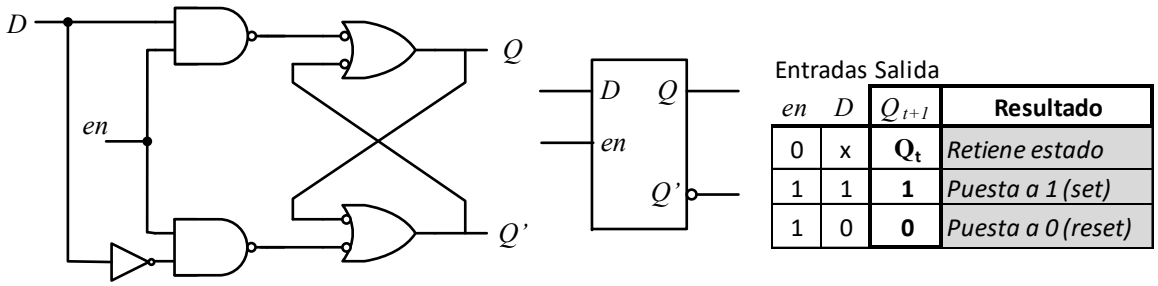
En este caso las líneas guías verticales se dibujan en el flanco de subida de la señal *en* y en los flancos de subida de *S* y *R* cuando $en = 1$, ya que solo en dichos instantes el biestable puede cambiar de estado. En los instantes señalados, el valor de *Q* está determinado por la tabla de verdad correspondiente al biestable (ver figura 7.6). ■

> Observa que un biestable sincronizado por nivel se comporta como el biestable asíncrono sin más que hacer que la señal de disparo permanezca siempre activa con $en = 1$, si el biestable se activa a nivel alto, o $en = 0$, si el biestable se activa a nivel bajo.

7.1.3 Biestable D

En las secciones previas se ha descrito cómo la activación simultánea de las dos entradas en los biestables S-R provoca una situación que impide predecir el estado final del sistema. Por este motivo es preciso diseñar circuitos que eviten dichas situaciones.

Tanto el biestable D (*Delay o Data*) asíncrono, como el biestable D activado por nivel se obtienen mediante la conexión de las entradas S y R del biestable S-R a través de un inversor para tener $D = S$ y $R = S'$ (ver figura 7.7). De este modo se garantiza siempre que $S \neq R$.



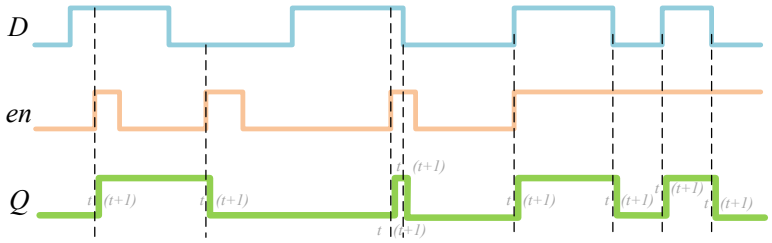
Entradas Salida

en	*D*	Q_{t+1}	**Resultado**
0	x	$\mathbf{Q_t}$	*Retiene estado*
1	1	**1**	*Puesta a 1 (set)*
1	0	**0**	*Puesta a 0 (reset)*

Figura 7.7: Circuito del biestable D activado por nivel alto junto a su símbolo estándar y tabla de verdad.

El estado del biestable D almacena el valor expuesto en su entrada. Si está activado por nivel, para que cambie su estado es preciso que también esté habilitado. El biestable D activado por nivel recibe el nombre de *latch transparente* o *seguidor* (*transparent latch*), ya que una vez activado, la salida sigue a la entrada.

Ejemplo 7.3 — Cronograma de un biestable D activado por nivel. Dado el cronograma de la figura adjunta correspondiente a un biestable D activado por nivel alto, realiza el análisis que permita su elaboración empleando como datos de partida: la señal de entrada D, la señal de activación *en* y el valor inicial del estado Q.



Solución:

En este caso las líneas guías verticales se dibujan en el flanco de subida de la señal *en* y en los flancos de subida y bajada de D cuando $en = 1$, ya que solo en dichos instantes es cuando el biestable puede cambiar de estado. En dichos instantes el valor del estado sigue al valor de D. ■

7.1.4 Biestables disparados por flanco

En los sistemas digitales interesa que el estado cambie en instantes determinados y predecibles. Esto se consigue si el estado de los sistemas cambia solo en las transiciones de la señal de sincronismo. La idea subyacente es la sustitución de la señal de activación de los biestables, por un pulso de activación de duración muy corta (⎍) generado en los flancos de la señal de sincronismo.

Para generar el pulso deseado, se precisa un *circuito detector de flanco*, cuya entrada es la señal de sincronismo y cuya salida es un pulso de duración muy corta[3] en el flanco deseado —de subida ⎍ o de bajada ⏉— de la señal de sincronismo. Una vez aplicada la salida del circuito detector de flanco al biestable original, los cambios de estado en el biestable se habilitan solo en los flancos detectados. Se puede decir que el biestable «captura» el valor de las entradas en los instantes en que se produce el flanco de sincronismo. El circuito resultante se muestra en la figura 7.8.

[3] Del orden de nanosegundos.

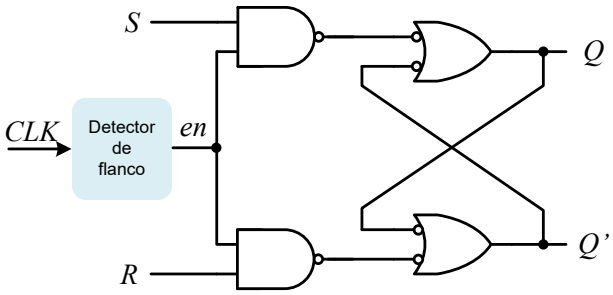


Figura 7.8: Circuito lógico del biestable S-R disparado por flanco.

El detector de flanco se puede implementar mediante una estrategia sencilla en la que se aprovecha el retardo de la señal de respuesta de las puertas lógicas, como se muestra en la figura 7.9. En el circuito de dicha figura, los pulsos positivos de entrada al inversor generan a su salida un pulso negativo retrasado. Ambos pulsos son las entradas a la puerta AND que genera un pulso positivo de duración muy corta en su salida, correspondiente al intervalo de tiempo en que ambas entradas a la puerta están a nivel alto. Si el pulso de salida de la puerta AND se aplica como señal de activación del biestable, este queda habilitado para su cambio de estado en el flanco de subida de la señal de entrada del detector de flanco. Un razonamiento similar se aplica para la sincronización en flanco de bajada. En general, como la señal de sincronismo es una señal de reloj (*clock*), su denominación suele ser *CLK*, *CK* o *C*.

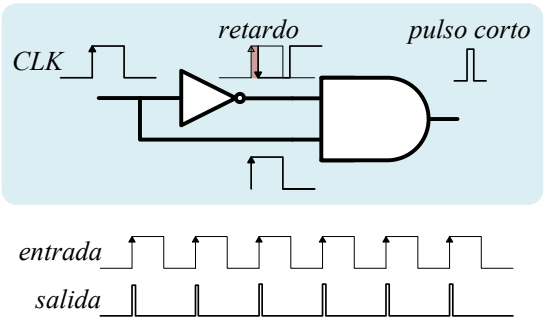


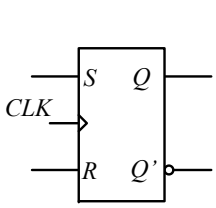
Figura 7.9: Circuito lógico para detección de flanco de subida.

Los biestables disparados por flanco se denominan popularmente *flip-flops* para diferenciarlos de los biestables asíncronos (*latch*) y de los síncronos activados por nivel (*gated latch*). La activación por flanco también se conoce como disparo por flanco (*edge triggering*), que puede ser de subida (*rising edge*) o de bajada (*falling edge*).

> Recuerda que una señal de reloj (*CLK*) es periódica con idéntica duración en nivel alto y nivel bajo. Esto es, un *ciclo de trabajo* del 50 %.

Biestable S-R disparado por flanco

El biestable S-R activado —o disparado— por flanco, solo puede cambiar de estado en el flanco de sincronismo elegido —de subida o de bajada—. El símbolo estándar de los biestables activados por flanco añade un triángulo en la entrada de la señal de sincronismo (ver figura 7.10). En la tabla de verdad del biestable se ha añadido el símbolo ↑ para señalar que el cambio de estado se produce en el flanco de subida de la señal de sincronismo (*CLK*).



Entradas Salida

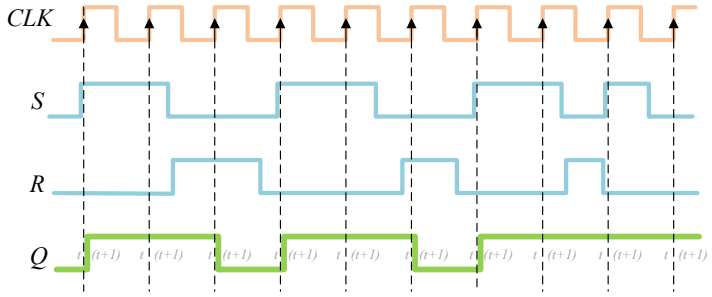
CLK	*S*	*R*	Q_{t+1}	Resultado
↑	0	0	$\mathbf{Q_t}$	*Retiene estado*
↑	0	1	**0**	*Puesta a 0 (reset)*
↑	1	0	**1**	*Puesta a 1 (set)*
↑	1	1	**0**	*Prohibido*

Figura 7.10: Símbolo estándar y tabla de verdad del biestable S-R disparado por flanco de subida.

Para señalar que el biestable se sincroniza en el flanco de bajada, el símbolo del mismo se modifica añadiendo el símbolo de negación (○) en la entrada de sincronismo.

> El biestable S-R disparado por flanco carece de interés práctico real, solo presenta interés teórico, aunque se puede emplear sin problema en ejercicios de simulación lógica con Logisim.

Ejemplo 7.4 — Cronograma de un biestable S-R disparado por flanco de subida. Dado el cronograma que se muestra en la figura adjunta correspondiente a un biestable S-R activado por flanco de subida, realiza el análisis que permita su elaboración empleando como datos de partida: las señales de entrada S y R, la señal de reloj CLK y el valor inicial del estado Q.



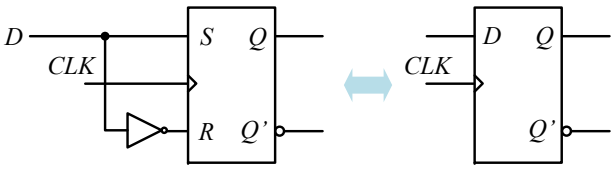
Solución:

En este caso las guías verticales se dibujan en el flanco de subida (⎍) de la señal CLK. Solo en dichos instantes puede cambiar el estado del biestable. Para obtener su estado se comprueba el valor de las entradas y se tiene en cuenta la tabla de verdad del biestable (ver figura 7.10). ■

Biestable D disparado por flanco

El biestable D activado —o disparado— por flanco almacena el estado de la señal conectada a su entrada en los instantes en que se produce un flanco de la señal de sincronismo. Esto es, se activa el biestable.

El biestable D se obtiene a partir del biestable S-R cuando se obliga a que $R = S'$ mediante un inversor. De este modo se asegura siempre que $S \neq R$. La figura 7.11 muestra el símbolo estándar del biestable D disparado por flanco de subida junto a su tabla de verdad.



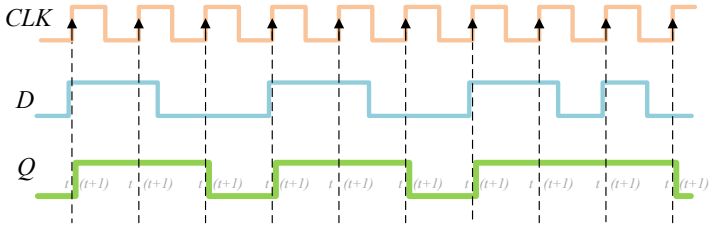
Entradas Salida

CLK	D	Q_{t+1}	Resultado
↑	X	Q_t	*Retiene estado*
↑	1	**1**	*Puesta a 1 (set)*
↑	0	**0**	*Puesta a 0 (reset)*

Figura 7.11: Símbolo estándar del biestable D disparado por flanco de subida junto a su tabla de verdad.

> Recuerda que si no se indica el tipo de sincronización de un biestable, sin proporcionar el símbolo asociado, se sobreentiende que se trata de un *flip-flop*. Es decir, un biestable síncrono, disparado por flanco de subida (⎍ o ↑).

Ejemplo 7.5 — Cronograma de un biestable D disparado por flanco de subida. Dado el cronograma que se muestra en la figura adjunta correspondiente a un biestable D activado por flanco de subida, realiza el análisis que permita su elaboración empleando como datos de partida: la señal de entrada D, la señal de reloj CLK y el valor inicial del estado Q.



Solución:

En este caso las guías verticales se dibujan en el flanco de subida (⌐) de la señal *CLK*. Solo en dichos instantes puede cambiar el estado del biestable que tomará el valor de la entrada *D*. ■

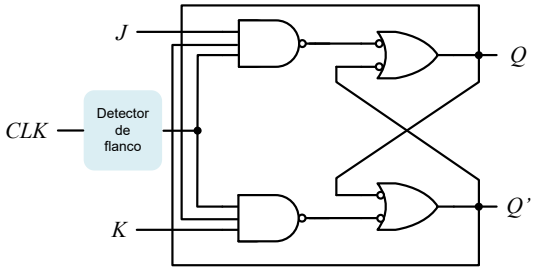
El biestable D se emplea como memoria temporal de retención en los registros. Una posible analogía para este biestable es el de una cámara fotográfica por la que se observa el entorno (valor binario presente en la entrada *D*) que captura una instantánea del mundo observado (*Q*) cuando se activa el obturador (señal de disparo *CLK*).

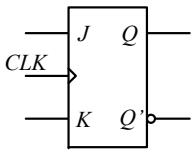
Otra aplicación inmediata del biestable D es la sincronización de señales asíncronas. De este modo cuando una señal asíncrona se conecta a la entrada de un biestable D, a la salida del biestable se obtiene el valor de la señal de entrada sincronizada. En algunos sistemas digitales complejos (p. ej., FPGA) este tipo de biestable aparece conectado a la salida de un sistema combinacional para asegurar la sincronización del valor de salida. Como la salida del biestable D aparece un ciclo de reloj más tarde que la entrada correspondiente, también se denomina *biestable de retardo* (*Delay*).

Biestable J-K

El biestable J-K se comporta de modo similar al biestable S-R. Sin embargo, el circuito se modifica para evitar el estado de indeterminación que se alcanza cuando ambas entradas tienen simultáneamente valor 1. Si $J = K = 1$, el biestable se comporta como una báscula (*toggle*) que conmuta el valor del estado del biestable: si está en 0 pasa a 1 y si está en 1 pasa a 0.

El biestable J-K *siempre se sincroniza por flanco, nunca por nivel*, ya que sería imposible determinar el estado final del biestable cuando estuviese activado basculando su estado (con $J = K = 1$). La figura 7.12 muestra el circuito lógico del biestable J-K, junto a su símbolo estándar y su tabla de verdad.





Entradas Salida

CLK	J	K	Q_{t+1}	Resultado
↑	0	0	Q_t	*Retiene estado*
↑	0	1	**0**	*Puesta a 0 (kill)*
↑	1	0	**1**	*Puesta a 1 (jump)*
↑	1	1	Q_t'	*Báscula del estado*

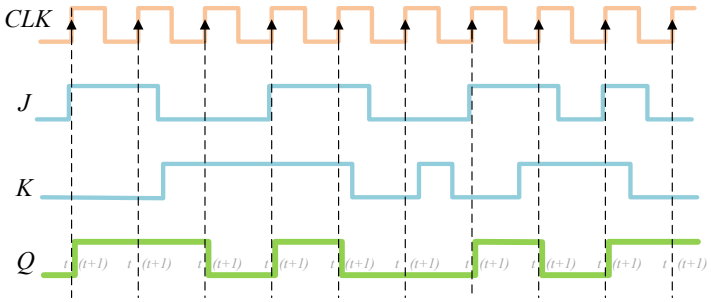
Figura 7.12: Circuito, símbolo estándar y tabla de verdad del biestable J-K disparado por flanco de subida.

> No está claro el verdadero origen de la denominación J-K para este biestable, pero la versión más aceptada es que J-K coinciden con las iniciales de Jack Kilby (1923-2005, EEUU) el inventor del primer circuito integrado,[4] un biestable de este tipo. Jack Kilby —ingeniero eléctrico y físico estadounidense— inventó el circuito integrado (basado en germanio) en 1958 cuando trabajaba para Texas Instruments. En el año 2000 recibió el merecido reconocimiento del Premio Nobel de Física por un invento que ha revolucionado nuestro mundo. En su discurso mencionó a Robert Noyce,[5] otro de los pioneros relevantes en el desarrollo de los sistemas digitales y creador del primer circuito integrado basado en silicio.

Ejemplo 7.6 — Cronograma de un biestable J-K disparado por flanco de subida. Dado el cronograma que se muestra en la figura adjunta correspondiente a un biestable J-K activado por flanco de subida, realiza el análisis que permita su elaboración empleando como datos de partida: las señales de entrada *J* y *K*, la señal de reloj *CLK* y el valor inicial del estado *Q*.

[4] U.S. Patent 3 138 743: *Miniaturized Electronic Circuits*, presentada en febrero de 1959, aprobada en junio de 1964.

[5] U.S. Patent 2 981 877: *Semiconductor Device and Lead Structure*, presentada en julio de 1959, aprobada en 1961, asignada a Fairchild Semiconductor.

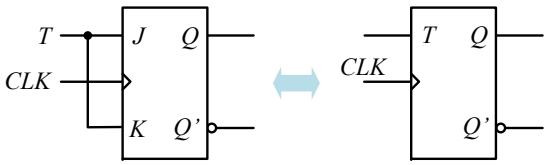


Solución:

En este caso las guías verticales se dibujan en el flanco de subida de *CLK*. Solo en dichos instantes puede cambiar el estado del biestable. Para obtener su estado se comprueba el valor de las entradas y se tiene en cuenta la tabla de verdad del biestable (ver figura 7.12). ■

Biestable T

El biestable T (*Toggle* o báscula) es el resultado de unir las dos entradas del biestable J-K. En este caso cuando la entrada es $T = 0$ ($J = K = 0$), el biestable mantiene su estado y cuando $T = 1$ ($J = K = 1$), su estado bascula. Al igual que el biestable J-K, *siempre se dispara por flanco*. En la figura 7.13 se muestra el símbolo del biestable T y su tabla de verdad.

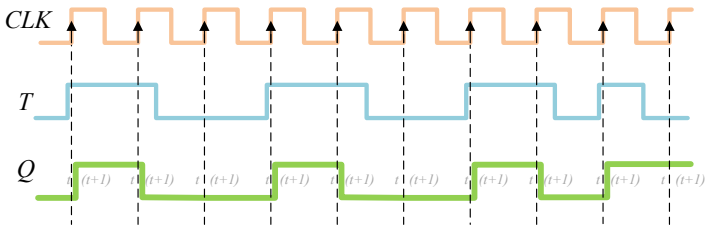


Ent. Salida

CLK	T	Q_{t+1}	Resultado
↑	0	Q_t	*Retiene estado*
↑	1	Q_t'	*Báscula del estado*

Figura 7.13: Obtención del biestable T desde un J-K (izda.); símbolo estándar y tabla de verdad del biestable T (dcha.).

Ejemplo 7.7 — Cronograma de un biestable T disparado por flanco de subida. Dado el cronograma que se muestra en la figura adjunta correspondiente a un biestable T activado por flanco de subida, realiza el análisis que permita su elaboración empleando como datos de partida: la señal de entrada T, la señal *CLK* y el valor inicial del estado Q.



Solución:

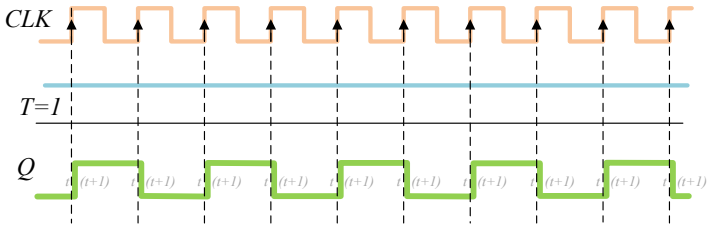
En este caso las guías verticales se dibujan en el flanco de subida (⌐) de la señal de reloj *CLK*. Solo en dichos instantes puede cambiar el estado del biestable. Para obtener su estado se comprueba el valor de la señal T y se tiene en cuenta la tabla de verdad del biestable (ver figura 7.13). ■

Ejemplo 7.8 — El biestable T como divisor de frecuencia de la señal de reloj. Dado un biestable T elabora el cronograma de su salida cuando su entrada se mantiene siempre a nivel alto $T = 1$ y su estado inicial es 0.

Solución:

Para realizar el cronograma se dibujan líneas guías verticales en el flanco de subida (⌐) de la señal *CLK*. Solo en dichos instantes puede cambiar el estado del biestable. Como la entrada del biestable es siempre un nivel alto $T = 1$, en cada flanco positivo de sincronización el biestable conmuta su estado.

De este modo, se obtiene el cronograma de la figura adjunta. En dicho cronograma se aprecia que la salida del biestable es similar a la señal de reloj, pero con un periodo que es el doble. Por tanto, la frecuencia de la señal obtenida a la salida del biestable es la mitad de la que posee la señal *CLK*. Es decir, el biestable T se comporta como un divisor de frecuencia.



La aplicación fundamental del biestable T, derivada de su capacidad como divisor de frecuencia, es la implementación de contadores.

7.1.5 Ajuste asíncrono del estado de un biestable

Al trabajar con biestables se plantea la necesidad de ajustar su estado a conveniencia bajo unas condiciones determinadas (p. ej., en la puesta en marcha del sistema), independientemente del valor de cualquier entrada. Para conseguirlo se incluyen dos entradas adicionales en el circuito digital del biestable para su puesta a 1/0, de modo asíncrono. Es decir, de modo independiente al valor de su entrada de sincronismo.

Las señales de entrada en el biestable para su ajuste asíncrono a 1/0, reciben respectivamente el nombre de *preset* (*PRE*) y *clear* (*CLR*). El resto de señales de entrada a los biestables son síncronas, ya que su efecto se deja sentir solo cuando se produce la transición de la señal de sincronismo que activa al biestable.

Las señales de ajuste asíncrono pueden ser activas por nivel bajo y en ese caso las entradas correspondientes aparecen en el circuito lógico con el símbolo de negación (○) y sus etiquetas negadas (*PRE*′ y *CLR*′). Como se observa en la figura 7.14, la inclusión de una entrada adicional en las puertas OR de salida del biestable J-K permite fijar la salida de dicha puerta a 1 cuando la entrada correspondiente es 0. Así se fija de modo inmediato el valor de la salida del biestable, sin intervención de la señal de habilitación.

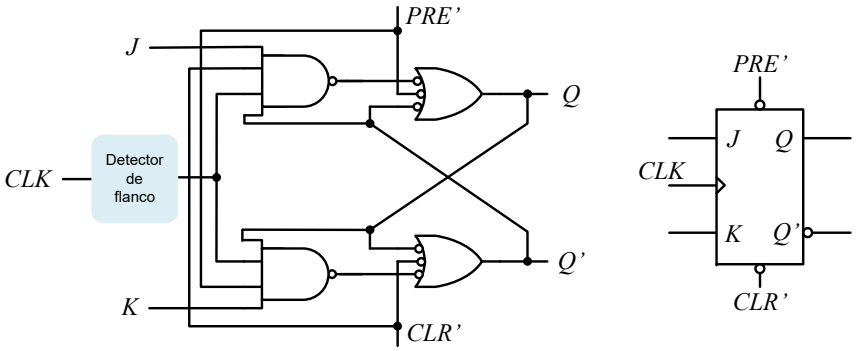
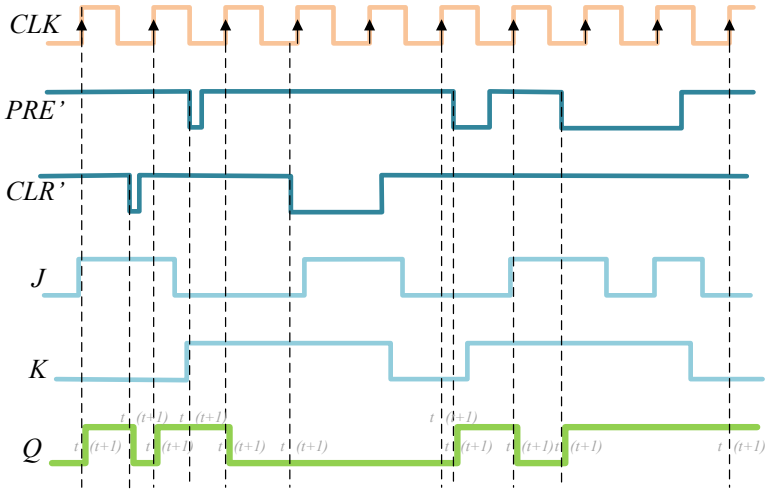


Figura 7.14: Circuito lógico y símbolo estándar del biestable J-K disparado por flanco de subida y señales de *preset* y *clear* activas a nivel bajo.

Si se activan simultáneamente las señales *PRE* y *CLR* se puede presentar una situación anómala similar a la que se produce en un biestable S-R cuando sus entradas se ponen a 1 simultáneamente, por tanto dicha situación se debe evitar priorizando una de las dos.

Además de las entradas mencionadas, todos los biestables disparados por flanco pueden emplear una entrada adicional de habilitación (*enable*, *EN*) del biestable. Cuando dicha entrada está inactiva se retiene el estado del biestable. Su implementación se puede conseguir con una entrada adicional en las puertas AND de entrada al biestable. De este modo, cuando dicha entrada se desactiva, el biestable retiene su estado sin responder a los cambios en sus entradas, exceptuando a las señales de ajuste asíncrono *PRE* y *CLR*.

Ejemplo 7.9 — Cronograma de un biestable J-K con señales PRE' y CLR'. Dado el cronograma que se muestra en la figura adjunta correspondiente a un biestable J-K disparado por flanco de subida, realiza el análisis que permita su elaboración empleando como datos de partida: las entradas PRE', CLR', J y K, la señal CLK y el valor inicial del estado Q.



Solución:

En este caso las guías verticales se dibujan preferentemente en los flancos de bajada de las señales PRE' y CLR', ya que ajustan incondicionalmente el valor del biestable a 1 y 0, respectivamente. Se añaden guías verticales en los flancos de subida de CLK siempre que $PRE' = CLR' = 1$, puesto que en dichos instantes el biestable adquiere el valor del estado indicado por su tabla de verdad para los valores de J, K y el estado de partida. ■

> Recuerda que se denomina Q_t al *estado de partida* de un biestable en los instantes en que dicho estado puede cambiar (por su disparo o ajuste asíncrono). En dichos instantes el *estado alcanzado* por el biestable se denomina Q_{t+1}. Observa que en los cronogramas de los ejemplos previos se han marcado los instantes t y $t+1$ donde se analiza la evolución del estado del biestable en estudio.

7.2 Simulación de circuitos con biestables en Logisim

En la tabla 7.1 se muestra una clasificación de los tipos de biestables que se han presentado en las secciones previas. En dicha tabla se ha incluido la denominación en inglés que permite distinguir los tipos de biestable por su método de sincronización. Para todos ellos, en castellano, se emplea la denominación de *biestables*, añadiendo el tipo de sincronización solo cuando es necesario.

Tabla 7.1: Tipos de biestables clasificados por su método de sincronización.

Tipo	S-R	D	J-K	T
Asíncrono	✓	✓	✗	✗
	*latch**			
Activado por nivel	✓	✓	✗	✗
(alto o bajo)	*gated latch*			
Disparado por flanco	✓	✓	✓	✓
(de subida ↑ o bajada ↓)	*flip-flop*			

Todos los tipos de biestables que aparecen en la tabla 7.1 se pueden utilizar en las simulaciones lógicas con Logisim. Los cuatro tipos de biestables están disponibles mediante la librería *Memory*. Aunque los símbolos empleados en Logisim son muy intuitivos, no responden al estándar internacional. Sin embargo,

las discrepancias respecto a los símbolos estándar no dificultan la comprensión de su funcionalidad en los circuitos implementados.

En Logisim el modo de sincronismo de los biestables no se puede determinar examinando a simple vista su símbolo, sino a través de sus atributos mostrados en el panel lateral de la ventana de la aplicación cuando el biestable está seleccionado en el espacio de trabajo. Aunque no se proporciona el comportamiento asíncrono como *latch* para los biestables S-R y D (marcado con asterisco en la tabla 7.1), dicho comportamiento se obtiene adoptando el modo de disparo por nivel con una señal constante que mantenga permanentemente activado el biestable.

Todos los biestables cuentan en su parte izquierda con las entradas que determinan el tipo de biestable (S-R, D, J-K y T) y la entrada de sincronismo (señalada con la punta de flecha). El ajuste del tipo de sincronismo del biestable determina el tipo de elemento que se conecta en su entrada de sincronismo. En la parte derecha de los biestables aparecen las salidas Q (con etiqueta) y Q' (sin etiqueta).

Durante una simulación con Logisim, cada biestable muestra su estado mediante un indicador en su centro, similar al que se presenta en los pines de E/S. Al igual que en los pines de entrada, el valor del estado de los biestables se puede modificar manualmente haciendo clic sobre él.

> Recuerda que en Logisim para indicar que un biestable se sincroniza por flanco de bajada, puedes emplear un inversor en la entrada de sincronismo. Del mismo modo que para emplear señales activas a nivel bajo. También puedes recurrir al empleo de etiquetas intuitivas para el biestable. Por ejemplo, '+-' para disparo por flanco de bajada ⌝, '+' para activación por nivel alto y '-' por nivel bajo.

7.2.1 Señales de ajuste asíncrono y habilitación

Todos los biestables en Logisim cuentan con tres entradas asíncronas ubicadas en la parte inferior del símbolo (ver figura 7.15) marcadas como '1' (*preset*, *PRE*), 'en' (*enable*, *EN*) y '0' (*clear*, *CLR*). Las tres son activas por nivel alto, pero precedidas por un inversor se consigue el efecto de activación por nivel bajo. Si no son necesarias, no es preciso conectarlas a valores constantes, pueden quedar en estado flotante permitiendo una simulación correcta.

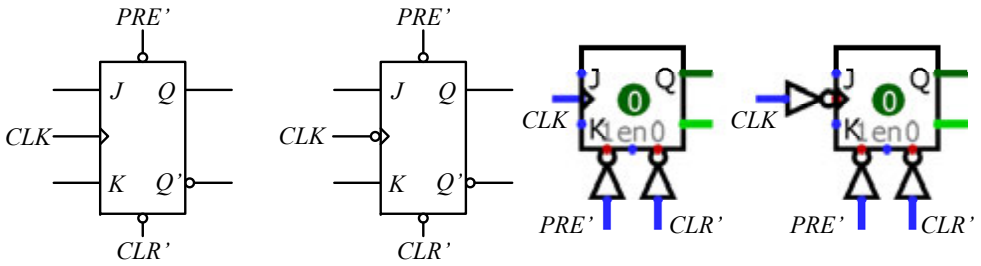


Figura 7.15: Símbolos estándar del biestable J-K y equivalentes en Logisim.

Estas señales se emplean para inicializar (*INIT*) un sistema digital a un estado deseado. Dicho reinicio o *reset* puede requerir el ajuste selectivo de los biestables a valores 0/1 obtenidos mediante la activación selectiva de las señales *PRE* o *CLR*.

Aunque es conveniente evitar situaciones en que *PRE* y *CLR* están activas en el mismo instante, Logisim otorga prioridad a *CLR*. De este modo, cuando en una simulación ambas señales están activas simultáneamente, los biestables afectados pasan a estado 0.

La señal de inhibición de los biestables (*EN* o *en*) se emplea para retener el estado del sistema. En los biestables de Logisin (ver figura 7.15) se incluye una señal específica (*en*) para obtener dicho comportamiento. En dicha situación el sistema no responde a los cambios de la señal de sincronismo. En las simulaciones lógicas si esta entrada se deja flotante el funcionamiento es equivalente a una habilitación permanente con $EN = 1$. Esto es, el sistema se comporta de modo normal. Esta funcionalidad se puede conseguir de modo alternativo mediante puertas AND para habilitar las entradas del biestable (ver figura 7.6). Las señales de *preset* y *clear* de los biestables son señales que actúan de modo inmediato sobre el estado del biestable aunque esté inhibido con $EN = 0$.

■ **Ejercicio 7.1** Reflexiona sobre la modificación requerida para dar prioridad a la señal de *preset* sobre *clear* en las simulaciones de biestables con Logisim. ■

7.2.2 Señales de reloj

La simulación del comportamiento de los biestables síncronos con Logisim utiliza el componente reloj (*clock*) de la librería *Wiring*. Este elemento genera una señal periódica basada en una señal interna de medida de tiempo denominada *tick* que controla la actualización de señales durante la simulación.

El *tick* es la unidad mínima de tiempo interno de simulación, de modo que todos los tiempos que se emplean en una simulación son múltiplos de dicho reloj interno, cuya frecuencia es ajustable en el menú Simulate. De este modo, para el elemento reloj se pueden ajustar independientemente la duración tanto de su nivel alto como bajo, a un número entero de *ticks*. Por defecto la duración de ambos niveles está ajustada a un *tick* y se recomienda mantener dicho valor para conocer su relación con la frecuencia fijada para el *tick*. Por tanto, un elemento reloj con duración de nivel alto y bajo de un *tick*, tiene un periodo de 2 *ticks*. Como el periodo de la señal de reloj es el doble del *tick*, su frecuencia es la mitad.

Ejemplo 7.10 — Cálculo del periodo de una señal de reloj en Logisim. Dada una simulación en Logisim con un elemento *clock* etiquetado como *CLK*, se observa en sus atributos que posee una duración de 1 *tick* tanto para el nivel alto como para el nivel bajo. Si la frecuencia de *tick* (*tick frequency*) de una simulación se fija a 2 Hz. ¿Cuál será la frecuencia de la señal *CLK* y el intervalo entre dos flancos positivos consecutivos de dicha señal?

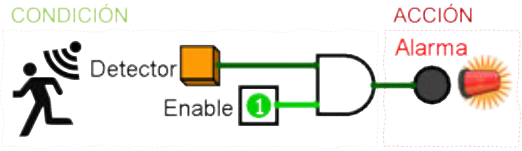
Solución:

Puesto que la señal *CLK* tiene una duración de 2 *ticks*, su frecuencia es la mitad que la correspondiente a 1 *tick*. Esto es, 1 Hz. Por tanto, el periodo de la señal *CLK* es 1 segundo, que es la separación entre dos flancos consecutivos de dicha señal. ■

La pulsación de Ctrl + K en Logisim permite conmutar el avance autónomo de las señales de reloj. De este modo, el sistema evoluciona automáticamente gobernado por el *tick* interno de simulación, a la frecuencia seleccionada en el menú Simulate, cambiando automáticamente el valor de los elementos de tipo reloj. Si se desea controlar manualmente el reloj de simulación (*tick*), este se puede avanzar paso a paso —un *tick* cada vez— mediante la pulsación de Ctrl + T. El estado de cada elemento reloj se puede modificar independientemente mediante clics de ratón.

En el ejemplo siguiente se aborda el problema de la activación memorizada de una alarma doméstica de intrusión mediante el empleo de biestables. En dicho ejemplo se proponen circuitos lógicos para simulación de los mismos con ayuda de Logisim.

Ejemplo 7.11 — Acciones memorizadas mediante el uso de biestables. Transforma el circuito combinacional de alarma doméstica de intrusión que se muestra en la figura adjunta para satisfacer los requisitos de funcionamiento que se indican. El nuevo circuito debe ser capaz de memorizar la detección de una intrusión accionando la alarma si el circuito está habilitado mediante señal de *Enable*. Además, el circuito propuesto debe incluir una señal de *Reset* que permita devolver el sistema a su estado inicial.



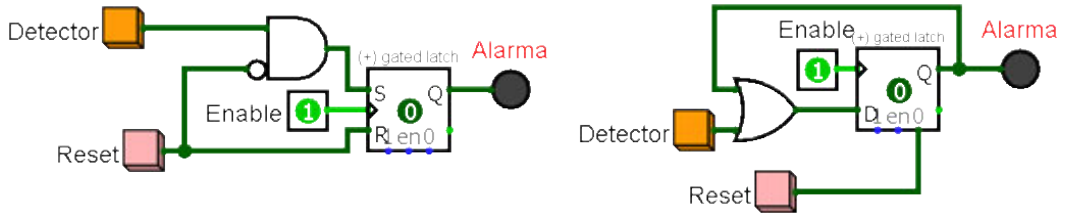
Solución:

La limitación del circuito combinacional proporcionado en el enunciado reside en que el accionamiento de la alarma es momentáneo. Esto es, se activa solo si el detector de intrusión está activado. Este comportamiento no es deseado porque una vez detectada la intrusión, el detector ubicado en un lugar visible podría ser destruido por el intruso o este podría ocultarse para evitar la detección.

Para accionar la alarma de modo permanente, después de detectar una intrusión, es preciso emplear un biestable que memorice la detección de intrusión. De este modo la alarma solo puede ser interrumpida mediante una señal de *Reset* o reinicio.

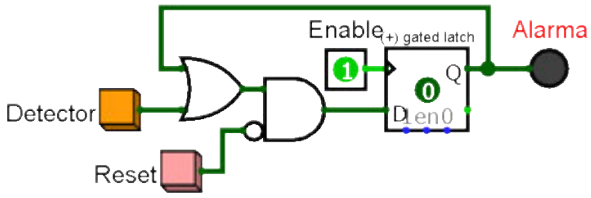
Abordamos el problema mediante soluciones alternativas en las que se emplean distintos tipos de biestables. Los casos 1 y 2 utilizan biestables de tipo *latch* (activados por nivel). En el caso 3 se propone el uso de biestables disparados por flanco. Para las soluciones se muestran los circuitos para simulación lógica con Logisim.

Caso 1.- Circuito con biestable asíncrono (*latch*) o biestables activados por nivel (*gated latch*):

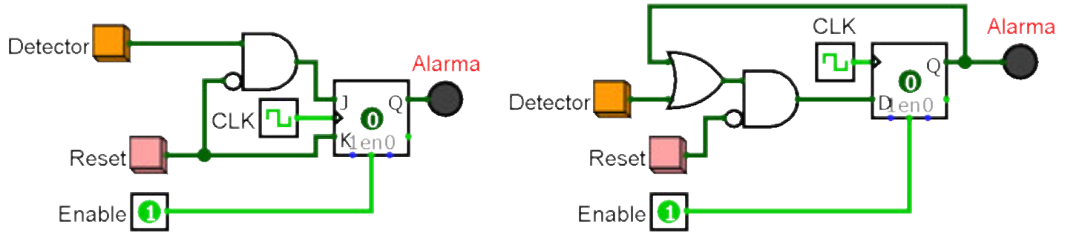


Como Logisim carece de biestables en modo *latch*, el comportamiento se obtiene con biestables activados por nivel alto de modo constante. Con el *gated latch* S-R la detección acciona la entrada *Set* del biestable y la entrada *Reset* provoca el apagado de la alarma. La puerta AND con entrada negada desde el *Reset* garantiza que nunca se obtiene la condición «prohibida» $S = R = 1$. La señal de activación del latch es la señal que habilita la alarma (*enable*).

El circuito alternativo emplea un *latch* de tipo D (dcha.). Con este tipo de biestable la memorización se consigue mediante realimentación del estado en la entrada a través de una puerta OR. Es decir, la alarma se activa si está activado el detector o previamente la alarma ya se activó. Este tipo de estrategia de memorización recibe el nombre de *enclavamiento*. En este caso, el enclavamiento del biestable está condicionado a la detección de intrusión. Para «salir» del estado de enclavamiento, la señal de *Reset* activa de modo asíncrono la entrada *Clear* del *latch*. El *Reset* del enclavamiento también admite una implementación síncrona con puerta AND en serie con la puerta OR (ver figura adjunta).



Caso 2.- Circuito con biestable activado por flanco (*flip-flop*): Los circuitos son similares a los mostrados en el caso anterior, pero en este caso la entrada de sincronismo está conectada a una señal de reloj (*CLK*). El cambio de estado del biestable se produce en los flancos de subida de dicha señal de reloj. En este tipo de implementación se debe asegurar que el evento a detectar es mucho más lento que la señal de reloj empleada para disparar el biestable. De este modo se garantiza que cualquier cambio en la señal de entrada es «capturada» en el biestable.



> Recuerda que un biestable disparado por flanco «captura» el valor de las entradas en los instantes en que se produce el flanco de sincronismo. Los valores de las entradas en otros instantes son «invisibles», ya que son más «rápidos» que el mecanismo de «captura» del biestable.

7.3 Caracterización de biestables

El comportamiento de los biestables descritos en las secciones previas se puede caracterizar formalmente mediante herramientas de descripción funcional que se explican a continuación.

Definición 7.7 — Tabla de transición del estado de un biestable. La tabla de transición de un biestable es similar a su tabla de verdad, pero añade como primera columna el *estado de partida* del biestable en el instante de sincronismo (Q_t). Dicho estado de partida se coloca en la tabla como si se tratase de una entrada al sistema. El resto de columnas de entrada en la tabla corresponden a las entradas del biestable. Como columna de salida se tiene el *estado resultante* al que evoluciona el biestable en un instante posterior a su sincronización (Q_{t+1}). ■

La tabla de transición contiene la información presente en un cronograma de evolución del estado del biestable cuando se ordenan los valores binarios de su posible estado de partida y sus entradas. A diferencia de las tablas de verdad de los sistemas combinacionales, en la tabla de transición de un biestable, el cambio de estado está condicionado a la ocurrencia del flanco de sincronización del biestable. Por tanto, la salida cambia de modo síncrono, mientras que en los sistemas combinacionales la salida cambia de modo asíncrono, ya que se produce en cuanto las entradas cambian de valor.

Ejemplo 7.12 — Tabla de transición del biestable J-K. Elabora la tabla de transición de estado del biestable J-K a partir de su tabla de verdad (ver figura 7.12).

Solución:

La tabla de transición del biestable se construye del mismo modo que una tabla de verdad de una función booleana, tomando como entradas: el estado de partida en el instante de sincronismo (Q_t) y las entradas del biestable (J y K). Como salida se tiene el valor del estado alcanzado por el biestable (Q_{t+1}). De este modo, tendiendo en cuenta la tabla de verdad proporcionada, se tiene:

Entradas			Salida
Q_t	J	K	Q_{t+1}
0	0	0	**0**
0	0	1	**0**
0	1	0	**1**
0	1	1	**1**
1	0	0	**1**
1	0	1	**0**
1	1	0	**1**
1	1	1	**0**

■

La información recogida en la tabla de transición de estado de un biestable es la misma que se obtendría realizando una gráfica de la evolución temporal de su estado. Por ejemplo, en la gráfica de la figura 7.16 se observan las transiciones del estado ($Q_t \xrightarrow{t_s} Q_{t+1}$) de un biestable T sincronizado por flanco de subida. En dicha gráfica se muestra como el estado del biestable (Q) cambia en los instantes de sincronismo t_s (flancos de subida del reloj).

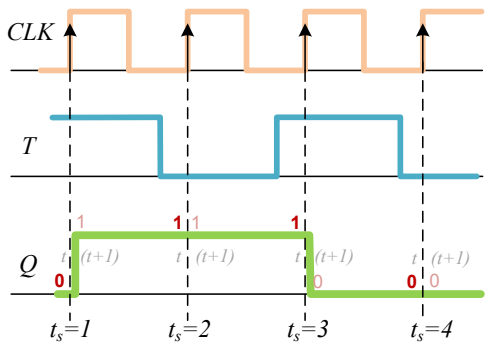


Figura 7.16: Evolución temporal del estado de un biestable de tipo T disparado por flanco de subida, para una entrada dada.

Definición 7.8 — Ecuación característica de un biestable. La ecuación característica de un biestable se obtiene a partir de su tabla de transición. Es la ecuación lógica de la función del estado resultante Q_{t+1} al que evoluciona el biestable en cada instante de sincronismo t_s en función de su estado de partida Q_t y el valor de sus entradas. ■

Tanto la tabla de transición del biestable como su ecuación característica permiten responder a la pregunta: *¿cuál será el estado Q_{t+1} alcanzado por el biestable en un instante de sincronismo t_s, partiendo del estado Q_t y unas entradas determinadas?*

Por tanto, la ecuación característica del biestable adquiere la forma:

$$Q_{t+1} = f(Q_t, \text{ENTRADAS})$$

> Ten en cuenta que el estado resultante Q_{t+1} en un instante previo t_s, se convierte en el instante de partida Q_t en el siguiente instante. Con esta nomenclatura debería entenderse que Q_{t+1} es el estado del sistema en el siguiente instante de sincronismo. Observa los cronogramas de evolución de estado en los ejemplos previos.

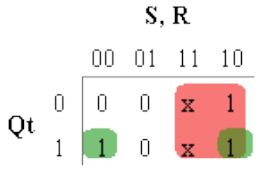
Ejemplo 7.13 — Cálculo de la ecuación característica de un biestable. Calcula la ecuación característica de un biestable S-R.

Solución:

La ecuación característica es la expresión del estado resultante Q_{t+1} que alcanza el biestable como una función de su estado de partida Q_t y sus entradas. Para obtener esta función se parte de la tabla de transición de estado del biestable teniendo en cuenta que sus entradas son S y R:

Q_t	S	R	Q_{t+1}
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	x
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	x

En la tabla de transición los casos en que $S = R = 1$ se han considerado condiciones indiferentes (*don't care*), ya que son situaciones que se evitan. Aplicando simplificación mediante K-maps se obtiene:



$$Q_{t+1} = S + Q_t \cdot R'$$

■

Aplicando el método descrito en el ejemplo anterior, es posible obtener las ecuaciones características del resto de tipos de biestables. La tabla 7.2 muestra las ecuaciones características de los biestables disparados por flanco: S-R, J-K, D y T.

Tabla 7.2: Ecuaciones características de los biestables disparados por flanco (*flip-flops*).

Tipo	Ecuación
S-R	$Q_{t+1} = S + Q_t \cdot R'$
J-K	$Q_{t+1} = Q'_t \cdot J + Q_t \cdot K'$
D	$Q_{t+1} = D$
T	$Q_{t+1} = Q_t \oplus T$

> Recuerda que el nombre de las variables en Logisim puede contener letras, números y el carácter de guion bajo. Por tanto, no es posible nombrar una variable como `Qt+1`. En consecuencia, en Logisim, para obtener las ecuaciones características de los biestables se emplea el convenio de etiquetar dichas señales como `Qt` para Q_t y `Qtt` para Q_{t+1}.

■ **Ejercicio 7.2** Calcula las ecuaciones características de los biestables J-K, D y T, siguiendo la metodología mostrada del ejemplo 7.13. ■

Definición 7.9 — Tabla de excitación de un biestable. La tabla de excitación de un biestable expresa cuáles deben ser sus entradas (o excitaciones) para conseguir cada uno sus posibles cambios de estado. ■

La tabla de excitación es un modo alternativo de expresar la información de la tabla de transición del biestable. Es especialmente útil en el diseño de los sistemas secuenciales, ya que permite calcular la expresión lógica de las entradas al biestable para obtener el cambio de estado requerido. La figura 7.17 muestra las tablas de excitación correspondientes a los biestables disparados por flanco: S-R, J-K, D y T.

Q_t	Q_{t+1}	**S**	**R**
0	0	**0**	**x**
0	1	**1**	**0**
1	0	**0**	**1**
1	1	**x**	**0**

Q_t	Q_{t+1}	**J**	**K**
0	0	**0**	**x**
0	1	**1**	**x**
1	0	**x**	**1**
1	1	**x**	**0**

Q_t	Q_{t+1}	**D**
0	0	**0**
0	1	**1**
1	0	**0**
1	1	**1**

Q_t	Q_{t+1}	**T**
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**0**

Figura 7.17: Tablas de excitación de los biestables disparados por flanco.

7.3.1 Obtención de cronogramas

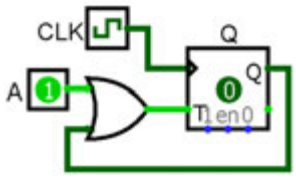
El cronograma de las señales lógicas es una herramienta de gran utilidad para el análisis de los circuitos lógicos que emplean biestables. En los circuitos reales, el análisis de la evolución temporal de las señales del sistema permite determinar si su comportamiento es el requerido. Se obtiene mediante el *osciloscopio* y el *analizador lógico*. En esta obra los cronogramas se emplean como una herramienta de análisis para conocer la evolución temporal esperada de las señales del sistema, dadas unas señales de entrada.

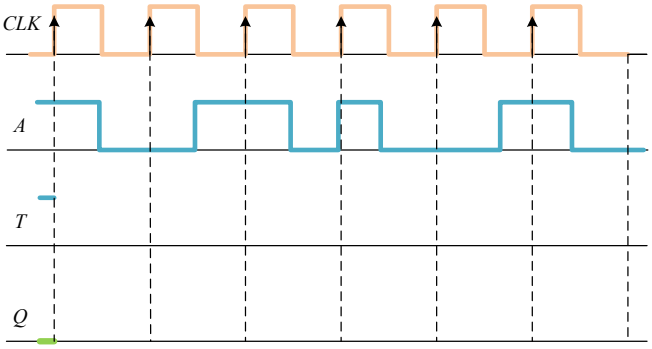
Para elaborar el cronograma de la señal de salida de un biestable, se deben seguir los pasos siguientes:

1. Identificar y calcular las entradas al biestable en función de las entradas del circuito y el estado del biestable antes del flanco de disparo.
2. Calcular el valor de la salida del biestable tras el flanco de disparo en función del valor de sus entradas y su tipo.
3. Mantener el valor del estado del biestable durante el ciclo de reloj hasta el siguiente flanco de disparo o hasta que se active una señal de *preset* o *clear*.
4. Si se activa una señal de *preset* o *clear* actualizar el valor del estado a 1 o 0 según corresponda y mantenerlo mientras esté activa la señal.
5. Actualizar el valor de las entradas al biestable hasta el siguiente flanco de disparo.
6. Si no se ha llegado al tiempo final de simulación volver al paso 2.

Es importante que al seguir los pasos enumerados se tenga en cuenta que el estado de los biestables solo cambia en los instantes de disparo a menos que existan entradas asíncronas activas de *preset* o *clear*. Por el contrario, las entradas al biestable cambiarán de modo independiente a las señales de disparo del biestable.

Ejemplo 7.14 — Obtención de cronograma de salida de un biestable. Dado el circuito de la figura adjunta, elabora el cronograma de la salida (Q) del biestable T disparado por flanco de subida, sabiendo que inicialmente $Q = 0$ y $T = 1$.



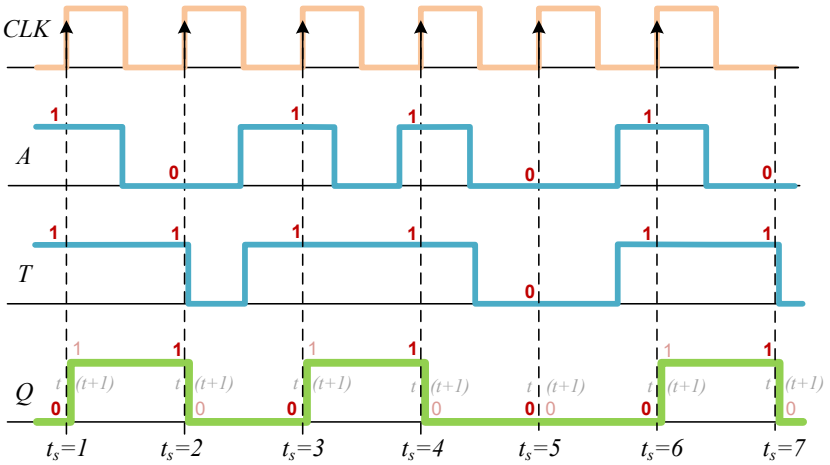


Solución:

A partir del circuito proporcionado es sencillo obtener la expresión booleana de la entrada al biestable como salida de la puerta OR con entradas A y Q. En este caso el valor de Q es el valor de partida Q_t. De este modo, se tiene:

$$T = A + Q$$

Teniendo en cuenta los valores de partida de A y Q_t, es posible saber el valor que tiene la entrada del biestable (T). Con este valor es posible obtener —en cada instante de sincronismo (t_s)— el valor resultante de Q_{t+1} para el biestable. Este nuevo valor junto al de A permite actualizar el valor de T en el siguiente instante de sincronismo. Así, es posible completar el cronograma final de la figura siguiente:



Observa que en los instantes de sincronismo (t_s) el estado de partida es Q_t y el valor resultante es Q_{t+1}. El tiempo transcurrido entre instantes de sincronismo consecutivos está determinado por el periodo de la señal de sincronismo (reloj), que es el inverso de su frecuencia.

El cronograma síncrono de este sistema se puede plantear también como una *tabla de evolución temporal* del sistema en la que para cada instante de sincronismo se inspeccionan los valores del estado de partida Q_t, la entrada A para obtener el valor de T, y el valor resultante del estado Q_{t+1} para la entrada del biestable.

t	Q_t	A	T	Q_{t+1}
$t_s = 1$	0	1	1	1
$t_s = 2$	1	0	1	0
$t_s = 3$	0	1	1	1
$t_s = 4$	1	1	1	0
$t_s = 5$	0	0	0	0
$t_s = 6$	0	1	1	1
$t_s = 7$	1	0	1	0

Si hubiese que considerar entradas asíncronas de puesta a 1 o *preset* (PRE) y borrado o *clear* (CLR), estas se incorporan en la tabla de evolución temporal incluyendo los instantes de su activación $t = t_{\text{PRE}}$ y $t = t_{\text{CLR}}$ con los valores asociados al estado resultante Q_{t+1}. ■

7.3.2 Intercambio de biestables

Como ocurre en el caso del biestable J-K actuando como biestable D o como biestable T, la funcionalidad de un tipo de biestable se puede obtener mediante un tipo de biestable diferente. De este modo, es posible implementar la funcionalidad de cualquier tipo de biestable empleando cualquier otro tipo y añadiendo convenientemente algunas puertas lógicas adicionales. El método a seguir para conseguir la transformación deseada consta de los pasos siguientes:

1. Se parte de la tabla de transición del biestable objetivo (esto es, el que se desea conseguir).
2. Se completa la tabla anterior con una columna por cada variable de entrada del biestable dado. Estas variables son las funciones que deseamos encontrar.
3. Los valores de las columnas añadidas en el paso anterior se completan teniendo en cuenta la tabla de excitación del biestable dado, de modo que el valor de las entradas provoque el cambio de estado requerido en los instantes de sincronismo.
4. Se calcula la expresión lógica de las entradas del biestable dado en función de su estado y las entradas del biestable objetivo.

La figura 7.18 muestra el caso de algunas conversiones entre biestables inmediatas y fáciles de recordar.

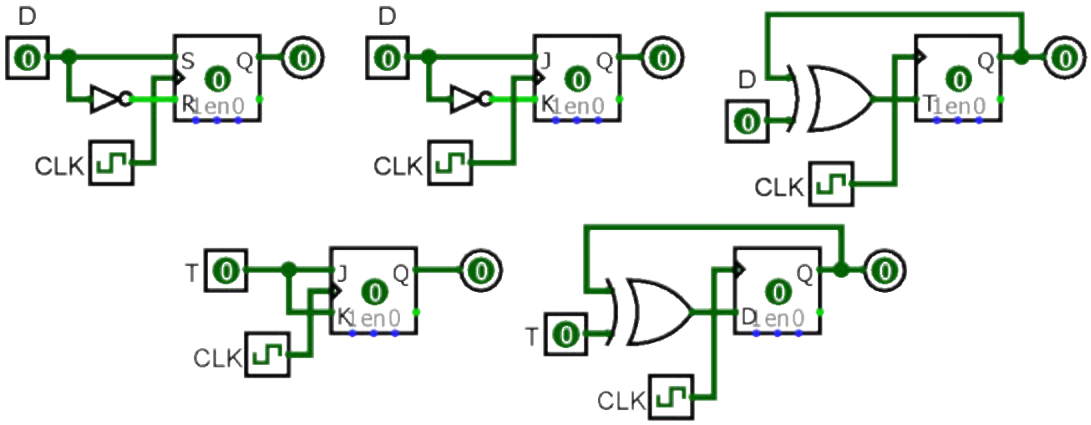
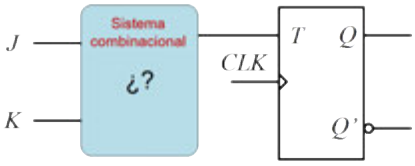


Figura 7.18: Conversiones comunes entre biestables.

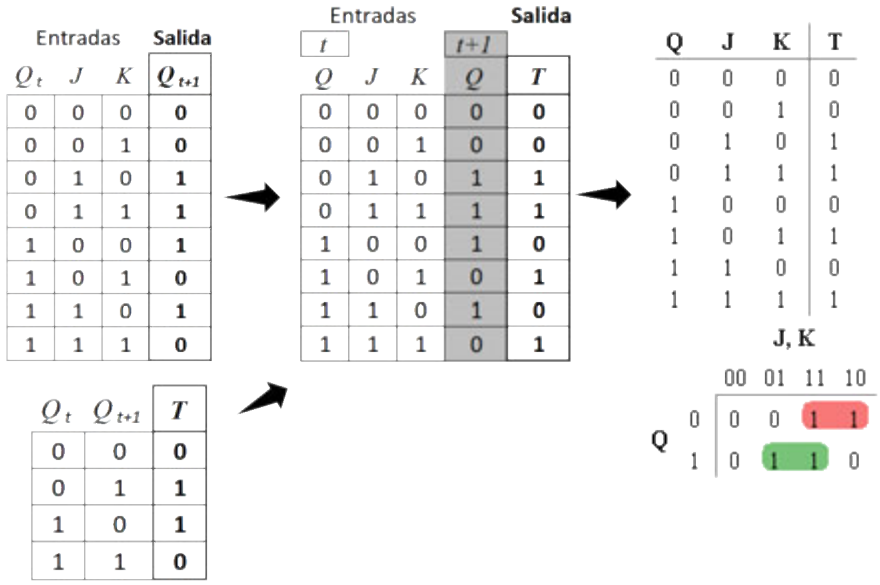
Ejemplo 7.15 — Obtención de un tipo de biestable a partir de otro. Dado un biestable T, diseña un subsistema combinacional —constituido por puertas lógicas— para obtener la funcionalidad de un biestable J-K.



Solución:

En primer lugar se debe obtener la tabla de transición del biestable objetivo J-K que se debe completar con la entrada T del biestable proporcionado. Dicha entrada se considera la función a calcular. Es decir, deseamos obtener el valor de T como una función del estado Q y de las entradas J, K del nuevo biestable: $T = f(Q,J,K)$.

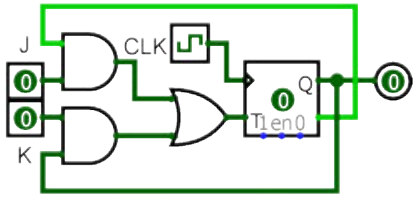
La columna de la salida, correspondiente a la variable T se completa teniendo en cuenta la tabla de excitación del biestable tipo T. Con esta tabla de excitación podremos poner los valores correspondientes a T para que el estado evolucione de Q_t a Q_{t+1} según los valores especificados en la tabla. Si el estado se mantiene $Q_t = Q_{t+1}$, el valor de T es 0, pero si el estado cambia, T debe valer 1. Finalmente, con la columna de salida completa, es posible calcular el valor de T en función de Q, J y K. En la figura siguiente se resume el proceso y el resultado final aplicando K-maps:



Por tanto:

$$T = Q' \cdot J + Q \cdot K$$

Añadiendo las puertas apropiadas se puede obtener el circuito de la figura siguiente. En dicho circuito se observa como el valor de Q'_t se obtiene directamente del biestable sin necesidad de un inversor adicional.



7.3.3 Características operativas de los biestables reales

La descripción y análisis realizados en las secciones previas asumen un comportamiento ideal para los biestables. La simulación lógica de los sistemas con Logisim responde de modo congruente a dicha asunción y permite adquirir un conocimiento funcional básico de los sistemas estudiados. Sin embargo, en los circuitos electrónicos reales que implementan sistemas con biestables, se debe tener presente las características físicas de operación de los mismos.

A continuación se ofrece un listado de características o parámetros que condicionan el funcionamiento de los sistemas secuenciales en su implementación real:

- *Tiempo de establecimiento* (*setup time*). Tiempo mínimo que se debe mantener sin cambio la entrada del biestable antes de que se produzca el flanco de disparo.
- *Tiempo de mantenimiento* (*hold time*). Tiempo mínimo que se debe mantener sin cambio la entrada al biestable después de que se produzca el flanco de disparo.
- *Retardo de propagación* (*propagation delay*). Es el tiempo que transcurre desde el flanco de la señal de disparo hasta el cambio de la señal de salida del biestable. Este cambio no es instantáneo y este parámetro ofrece un valor promedio.

- *Frecuencia máxima del reloj*. Determina la velocidad máxima a la que se puede disparar el biestable de modo fiable. Está condicionada por el valor de la suma de los tiempos de establecimiento, mantenimiento y retardo de propagación.
- *Anchura mínima de pulso*. Especifica el valor mínimo del ancho de los pulsos de las señales de reloj *CLK*, *preset PRE* y *clear CLR* del biestable para asegurar su funcionamiento esperado.
- *Disipación de potencia*. Potencia eléctrica promedio, medida en vatios, consumida por el biestable durante su funcionamiento. Este parámetro permite diseñar la fuente de alimentación de un sistema para mantener su funcionamiento correcto.

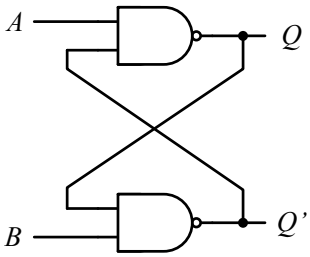
Conceptos clave

- ☑ Comportamiento de biestables asíncronos S-R y D (*latches*).
- ☑ Comportamiento de biestables asíncronos activados por nivel S-R y D (*gated latches*).
- ☑ Comportamiento de biestables síncronos disparados por flanco S-R, J-K, D y T (*flip-flops*).
- ☑ Tabla de transición y tablas de excitación de biestables disparados por flanco.
- ☑ Diferencia entre el estado de partida (Q_t) y el estado resultante (Q_{t+1}) de un biestable en los instantes de sincronismo (t_s).
- ☑ Obtención y análisis de cronograma de un biestable (tabla de evolución temporal).
- ☑ Obtención de biestables T y D a partir del biestable J-K.
- ☑ Obtención de comportamiento de un tipo de biestable a partir de uno dado de distinto tipo.

Problemas propuestos

Problema 7.1 Analiza en el circuito de la figura adjunta, los valores de Q y Q' en cada una de las siguientes situaciones:

1.- $A = 0$ y $B = 1$.
2.- $A = 1$ y $B = 0$.
3.- Ambas entradas son 1, $A = B = 1$.
4.- Ambas entradas son 0, $A = B = 0$.
5.- Ambas entradas pasan de valer 0 a 1.



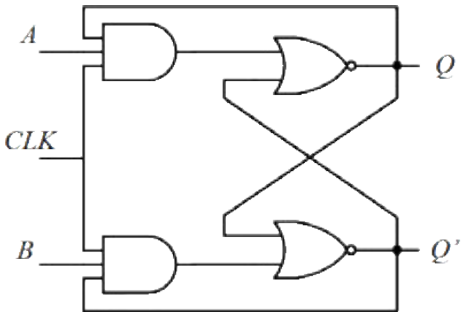
NOTA: Recuerda que en los esquemas de los circuitos lógicos, se omite la conexión a la fuente de alimentación (V_{DD}) y tierra (GND) aunque son necesarios para el funcionamiento de los circuitos físicos.

Problema 7.2 Calcula las ecuaciones características de los biestables J-K, D y T, describiendo detalladamente el proceso de obtención de dichas ecuaciones.

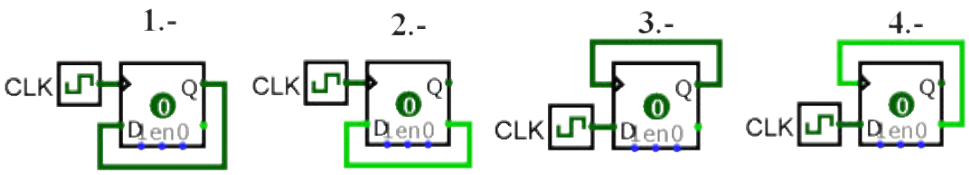
Problema 7.3 ¿Si se unen las entradas de un biestable J-K, el circuito resultante sigue siendo un biestable? ¿En caso afirmativo, de qué tipo de biestable se trata?

Problema 7.4 Para el circuito de la figura adjunta, responde razonadamente:

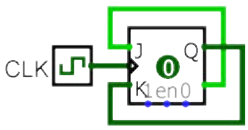
1.- ¿Se trata de un circuito combinacional o secuencial?
2.- ¿La salida posee algún tipo de sincronización respecto de la señal *CLK*? ¿De qué tipo?
3.- Calcula la ecuación característica del circuito y describe qué tipo de comportamiento implementa dicho circuito.



Problema 7.5 Analiza los circuitos siguientes con biestables sincronizados por flanco de subida e indica cuál de ellos corresponde a un divisor de frecuencia de la señal *CLK*. ¿Cuál es la frecuencia obtenida respecto a la señal *CLK*? ¿Qué relación existe entre cada circuito y el comportamiento de un biestable T? Emplea Logisim para hacer la simulación de cada circuito.



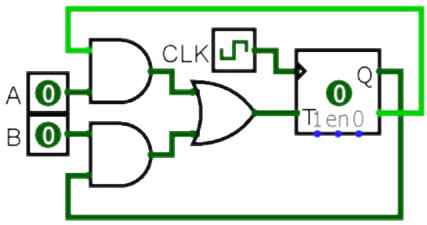
Problema 7.6 Dibuja el cronograma de *CLK* y Q en el circuito de la figura adjunta, partiendo del estado inicial $Q = 0$. ¿Qué relación existe entre la señal de reloj y la salida del biestable? Utiliza Logisim para simular el circuito y asume que el biestable tiene sincronismo por flanco de subida.



Problema 7.7 Indica cómo obtener un biestable T a partir de un biestable D, describiendo detalladamente el proceso. Comprueba el resultado con Logisim.

Problema 7.8 Dado el circuito de la figura adjunta cuyo biestable está sincronizado por flanco de subida, completa razonadamente los apartados siguientes y comprueba los resultados obtenidos con Logisim:

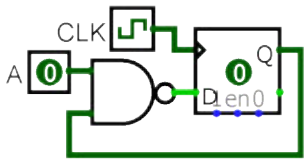
1.- Calcula la ecuación característica del circuito que relaciona el estado del biestable T con su estado previo y el valor de las entradas A y B.
2.- Relaciona del comportamiento del circuito con el de algún biestable conocido.
3.- Si se parte del estado $Q = 0$ para el biestable T y $A = B = 1$, ¿cuál es la evolución del estado del biestable?



Problema 7.9 Diseña un circuito con el comportamiento de un biestable J-K empleando un biestable de tipo D. Comprueba el resultado con Logisim.

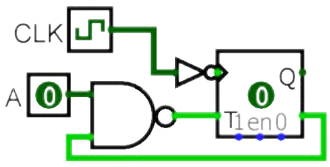
Problema 7.10 Para el circuito de la figura adjunta con el biestable D sincronizado por flanco de subida, completa los apartados siguientes:

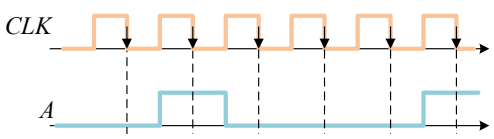
1.- Calcula la ecuación característica para el estado del circuito tras el sincronismo del biestable.
2.- Determina el estado del biestable cuando el valor de la señal A se mantiene a 1 y el estado inicial del biestable es $Q = 1$.
3.- ¿Se comporta el conjunto como un biestable T?



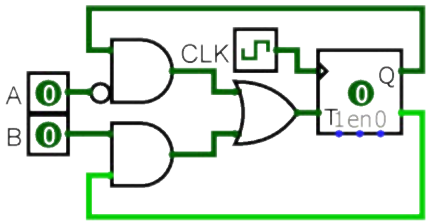
Problema 7.11 Dado el circuito de la figura adjunta, completa los apartados siguientes:

1.- Calcula la función de entrada al biestable T (disparado por flanco de bajada).
2.- Calcula la ecuación característica de la salida del biestable.
3.- Completa el cronograma con las señales T y Q, asumiendo que inicialmente $Q = 0$. Comprueba el resultado con ayuda de Logisim.





Problema 7.12 Calcula la ecuación característica del biestable de la figura adjunta expresando el valor del estado Q_{t+1} en función de las entradas A, B y el estado de partida Q_t. Indica que modificación sería precisa para que el comportamiento fuese equivalente al de un biestable J-K. Comprueba este último apartado con ayuda de Logisim.



Problema 7.13 Configura un biestable J-K disparado por flanco positivo, que otorgue prioridad a la señal de *preset* (PRE) frente a *clear* (CLR), cuando ambas señales se activan por nivel bajo. Comprueba la funcionalidad del biestable obtenido mediante Logisim.

8. Registros y contadores

Los biestables constituyen las celdas básicas de memoria empleadas en los sistemas secuenciales. Con ellos es posible obtener los módulos constructivos para el diseño jerárquico de los sistemas secuenciales más complejos. Este capítulo se dedica a describir los *registros* y *contadores*, considerados como los módulos básicos empleados en el diseño de los sistemas secuenciales.

8.1 Registros de desplazamiento

En los sistemas digitales se hace necesario almacenar información temporal como por ejemplo para:

- almacenar el resultado intermedio de operaciones aritméticas,
- suministrar información estable a un sistema de visualización,
- almacenar direcciones de memoria y datos relacionados con operaciones de lectura y escritura en la memoria principal de un sistema digital, etcétera.

La principal aplicación de los biestables D es el almacenamiento de un bit de información. Por tanto, un arreglo de varios biestables D almacena información hasta completar el tamaño de un elemento básico denominado *registro*.

Definición 8.1 — Registro. Es un circuito digital síncrono capaz de almacenar y desplazar un conjunto de n bits. ■

Los biestables que componen un registro comparten todas las señales de control del mismo: *CLK* (*clock*), *CLR* (*clear*) y *EN* (*enable*). En los registros la señal *PRE* (*preset*) se emplea con menos frecuencia que *CLR*, porque es más habitual su borrado ajustando su valor a 0. Los registros son muy versátiles y admiten varias clasificaciones:

- Dependiendo del modo de E/S de los bits en el registro, ya que tanto la entrada como la salida pueden ser *en serie* (bit a bit) o *en paralelo* (todos los bits de una vez).
- Dependiendo de la dirección de desplazamiento de bits en el registro pueden ser *unidireccionales* o *bidireccionales*.

La figura 8.1 muestra los distintos tipos de registros dependiendo del tipo de E/S. Además, es posible considerar el desplazamiento serie en dos sentidos: desde el bit más significativo al menos significativo (MSB→LSB) y dirección inversa (MSB←LSB).

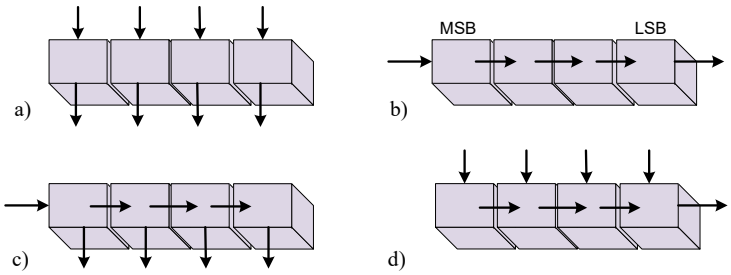


Figura 8.1: Tipos de registros dependiendo del tipo de E/S: a) E/S en paralelo, b) E/S en serie, c) entrada serie y salida en paralelo, y d) entrada en paralelo y salida serie.

> Recuerda que tanto en los registros como en el almacenamiento en memoria, existe una posición determinada a escala de hardware para los bits MBS y LSB. En este texto adoptamos por convenio el orden natural de bits con el MSB colocado a la izquierda.

En las secciones siguientes se describen los tipos más frecuentes de registros y se aportan los correspondientes circuitos con Logisim para llevar a cabo su simulación lógica.

8.1.1 Registro con E/S en paralelo

Es un registro cuya implementación es sencilla, ya que cada biestable del registro tiene una entrada y una salida independiente, pero todos comparten la señal de sincronismo. De este modo la entrada y salida de información en todos los biestables del registro se produce al unísono. Todos los bits de la entrada al registro aparecen en la salida con el primer flanco de sincronismo.

La figura 8.2 presenta el diagrama del circuito lógico correspondiente a un registro de 4 bits (*nibble*) con E/S en paralelo. Dicho circuito emplea biestables D síncronos disparados por flanco de subida, con señales comunes de sincronismo *CLK*, habilitación *EN* activa a nivel alto y borrado asíncrono activo a nivel bajo.

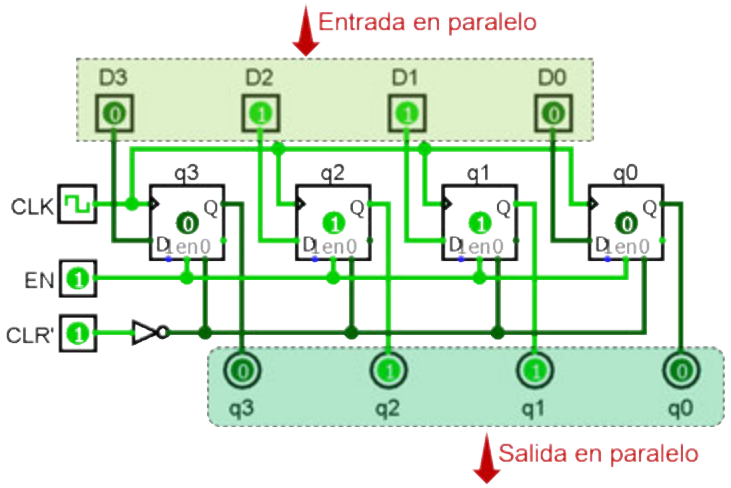


Figura 8.2: Registro de 4 bits con E/S en paralelo y señal de borrado CLR' activa a nivel bajo.

Para limpiar el registro se emplea la entrada *clear* de cada biestable. La retención del estado del registro se consigue mediante la señal de *enable* de cada biestable.

> Logisim proporciona el bloque *Register* en la librería *Memory* que recoge la funcionalidad de este tipo de registro. Para este elemento se puede definir atributos como el número de bits del registro y el tipo de sincronismo, similar al proporcionado para un biestable D. Además, posee una entrada asíncrona de borrado o puesta a cero.

8.1.2 Registro con E/S en serie

En este tipo de registro la entrada de cada biestable se conecta a la salida del anterior. Los datos se introducen por la entrada del biestable inicial y se obtienen uno a uno por la salida del último biestable del registro. La entrada y salida de cada bit se produce con cada flanco de la señal de sincronismo. También se denomina *registro de desplazamiento* (*shift register*), ya que con cada flanco de sincronismo se carga un bit en el primer biestable y el bit cargado previamente se desplaza hacía el siguiente biestable.

Para cargar y descargar por completo este registro se precisan tantos flancos de la señal de disparo como bits componen el registro. La figura 8.3 muestra un registro de desplazamiento hacia la derecha con 4 bits. El circuito se completa con señales de habilitación *EN* y borrado CLR' del registro, esta última activa a nivel bajo.

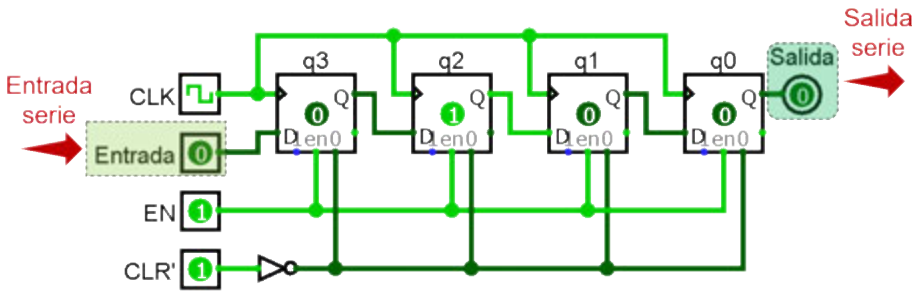


Figura 8.3: Registro de desplazamiento hacia la derecha de 4 bits con señal de borrado CLR' activa a nivel bajo.

8.1.3 Registro conversor serie a paralelo

Este registro es idéntico al anterior, pero añade una salida individual desde cada biestable para tener en cada instante acceso al contenido completo del registro. Este registro también se conoce como registro *conversor serie a paralelo* (*SIPO, Serial Input Parallel Output*). La figura 8.4 presenta un registro SIPO de 4 bits con señal de borrado activa a nivel bajo.

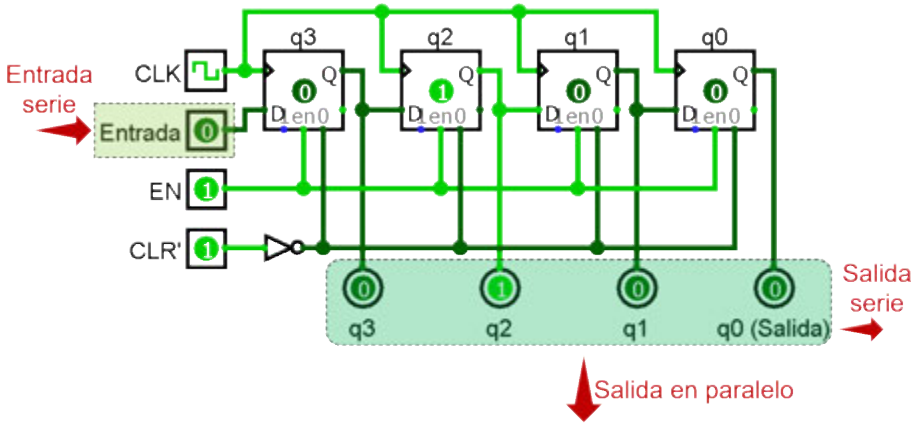


Figura 8.4: Registro conversor serie a paralelo (SIPO) de 4 bits.

8.1.4 Registro conversor paralelo a serie

Al registro anterior se puede añadir una señal $LOAD'/SHIFT$ para controlar la operación de carga y desplazamiento. De este modo se reúne la funcionalidad de los tres registros explicados previamente, ya que permite entrada serie con $LOAD'/SHIFT = 1$ o paralelo con $LOAD'/SHIFT = 0$, con salida en serie a través de q_0 cuando está en modo desplazamiento.

Cuando funciona como conversor paralelo a serie (PISO, *Parallel Input Serial Output*), el registro precisa de un único ciclo de reloj para cargar los datos en el registro. Sin embargo, para obtener los datos a la salida se requieren tantos ciclos de reloj como etapas tenga el registro.

La figura 8.5 presenta el diagrama del circuito lógico de este tipo de registro cuya característica más interesante es la inclusión de MUX 2×1. Estos MUX permiten controlar la entrada de datos en cada biestable obteniendo la funcionalidad de carga del registro o bien de desplazamiento de su contenido. Dicho control se consigue mediante la señal de control *LOAD'/SHIFT* empleada como señal de selección en los MUX.

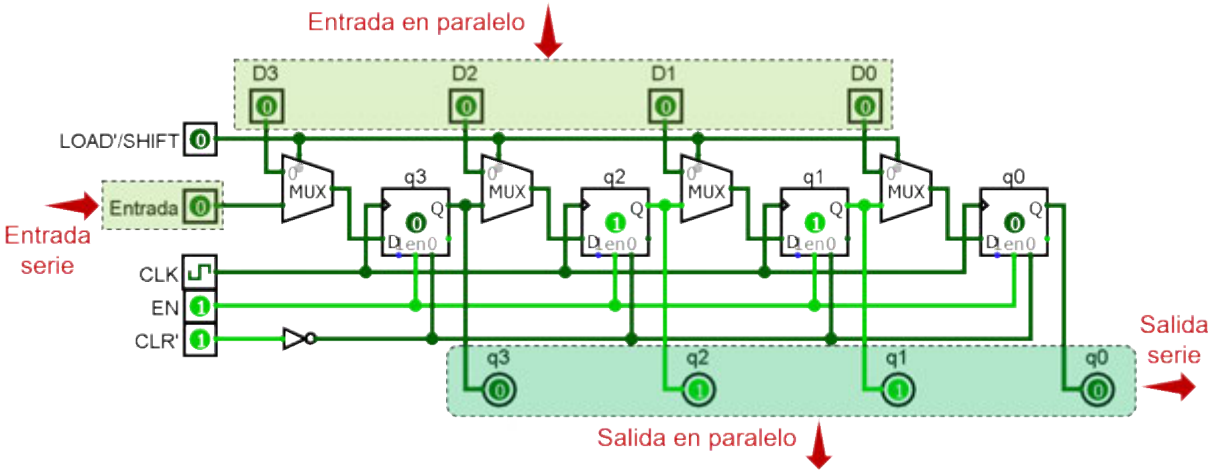


Figura 8.5: Registro conversor paralelo a serie (PISO) de 4 bits con entrada alternativa serie y empleo de multiplexores.

Logisim proporciona el bloque *Shift Register* en la librería *Memory* que recoge la funcionalidad de este tipo de registro añadiendo posibilidad de carga en paralelo. Para este elemento se puede definir atributos como el número de etapas del registro y el tipo de flanco de sincronismo.

Es posible diseñar un circuito alternativo equivalente sustituyendo los MUX por puertas lógicas. Este circuito se expone en la figura 8.6. En dicho circuito cuando se activa el modo de desplazamiento, el registro se completa con ceros, pues no existe entrada serie.

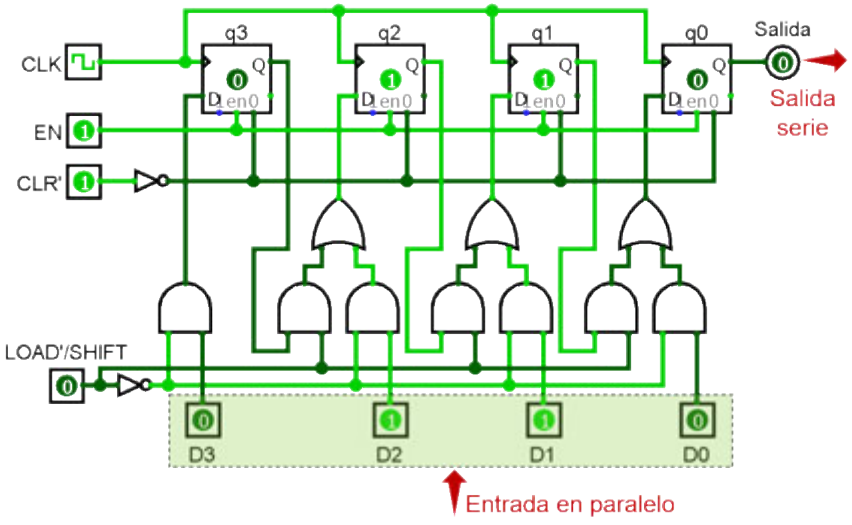


Figura 8.6: Registro conversor paralelo a serie (PISO) de 4 bits con puertas lógicas.

> Recuerda que en cualquier circuito de registro, la salida en paralelo no requiere elementos adicionales, solo una conexión individual desde la salida de cada biestable.

8.1.5 Otros tipos de registros

Además de los registros mencionados en las secciones anteriores, existen algunos más, como:

- **Registro de desplazamiento bidireccional** (ver figura 8.7). Permite controlar el origen de los datos y el sentido de desplazamiento de los bits en el registro.

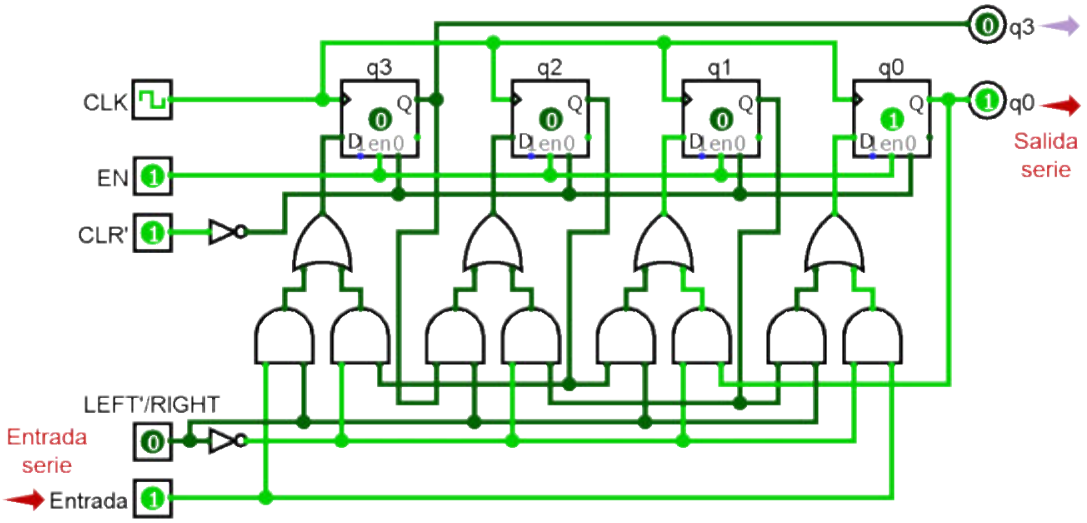


Figura 8.7: Registro de desplazamiento bidireccional de 4 bits.

- **Registro universal**. Es un registro de desplazamiento bidireccional que permite la carga de datos en paralelo. Además, este registro posee una señal de habilitación que permite la inhibición del registro. En el diagrama del circuito lógico de la figura 8.8 se muestra como los dos bits de selección de los MUX 4×1 controlan la operación del registro: $m = 00$ para desplazamiento hacia la derecha, $m = 01$ para carga, $m = 10$ para inhibición, y $m = 11$ para desplazamiento hacia la izquierda.

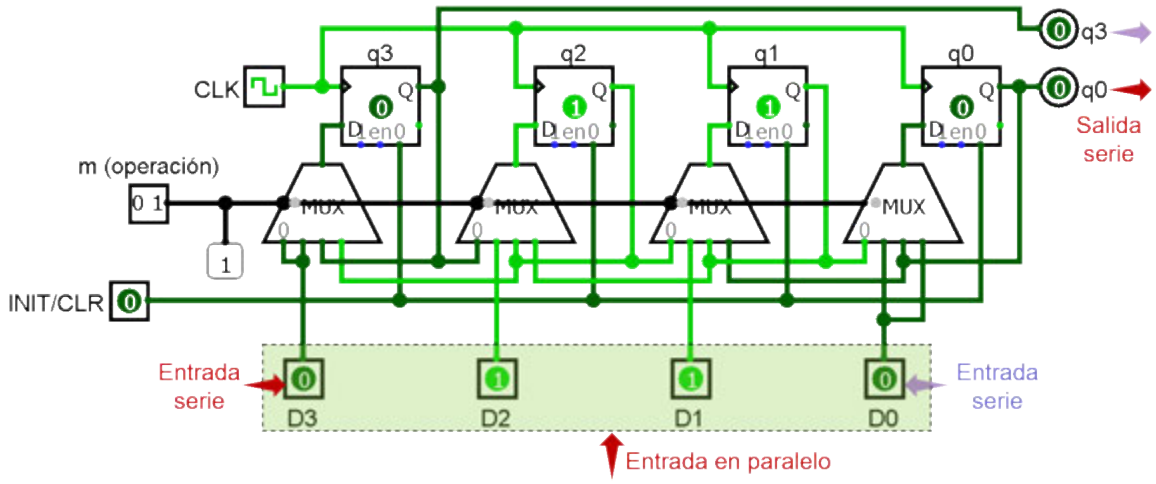


Figura 8.8: Registro universal de 4 bits.

- **Registro multibit por etapa**. En los circuitos presentados en esta sección, los registros emplean un biestable por etapa. Esto es, el dato desplazado de una etapa a la siguiente consiste en un único bit. Sin embargo, en muchas aplicaciones es preciso desplazar más de un bit por etapa. En la figura 8.9 se presenta la implementación de un circuito correspondiente a un registro de desplazamiento con 4 etapas y 2 bits por etapa, con posible carga en paralelo o serie. Dicho circuito muestra el esquema básico para obtener registros de un número determinado de bits por etapa.

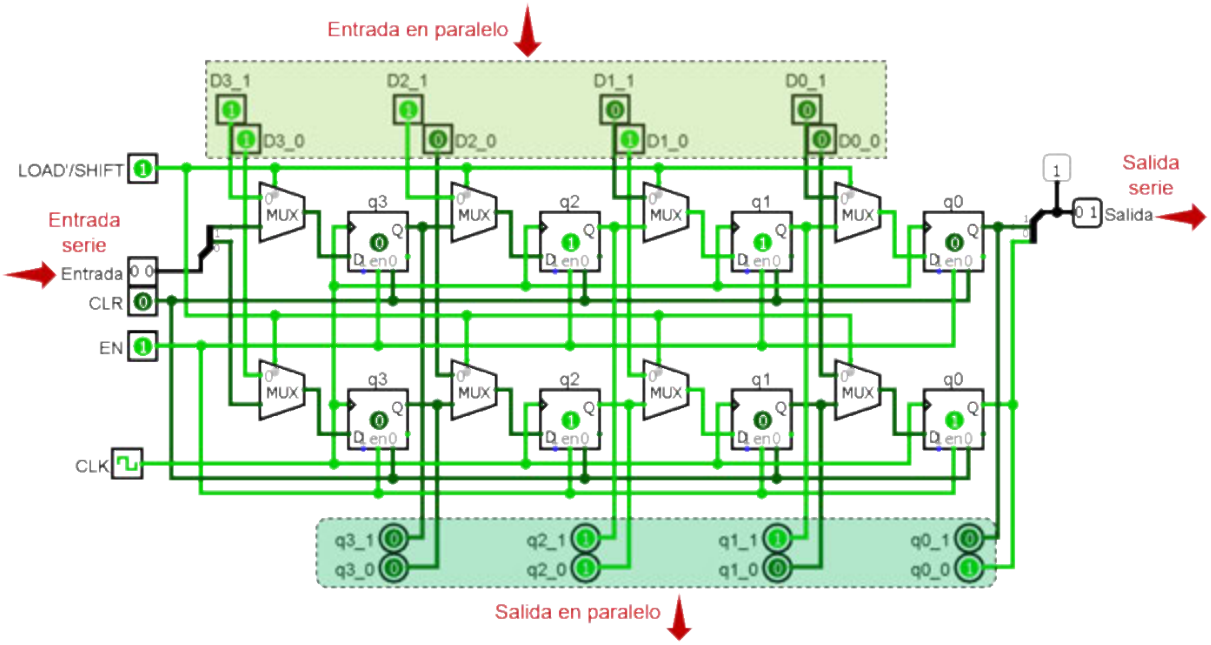


Figura 8.9: Registro multibit de 4 etapas con 2 bits por etapa.

> Recuerda que en Logisim es posible definir los registros no solo con un número de etapas determinado, sino también especificar el número de bits de datos para cada una de ellas.

8.2 Contadores

En los sistemas digitales hay aplicaciones que requieren el empleo de elementos contadores para obtener distinta funcionalidad, por ejemplo:

- *Divisor de frecuencia*. El reloj de los sistemas digitales suele oscilar a una frecuencia muy elevada, del orden de megahercios o gigahercios (10^6-10^9 Hz). Dicha frecuencia es apropiada para gobernar subsistemas rápidos. Sin embargo, si se necesitan frecuencias más reducidas, estas se obtienen mediante divisores de la frecuencia del reloj principal del sistema.
- *Temporizador*. Cuando es preciso temporizar acciones en un sistema (p. ej., retardos, esperas, etc.) se recurre a elementos capaces de contar el tiempo mediante la cuenta del número de flancos de disparo de una señal de reloj.
- *Contador de eventos*. En este caso se debe contar las transiciones de una señal asociada a la ocurrencia de eventos (p. ej., detección de objetos, personas, etc.).
- *Generador de secuencia cíclica*. Cuando es preciso generar repetidamente una sucesión de códigos binarios ordenados o desordenados, con elementos repetidos o sin ellos (p. ej., generación de señales de selección de MUX o DEC, generación de direcciones de memoria, activación de sistemas en secuencia, etc.).

Todas las aplicaciones mencionadas hacen uso de los contadores que son sistemas secuenciales constituidos habitualmente por biestables de tipo T que generan una secuencia cíclica sin elementos repetidos en ella.

Definición 8.2 — Secuencia. Es una lista ordenada posicionalmente de un conjunto finito de números naturales que generan una sucesión de valores en los que se repite de modo cíclico la secuencia. En este texto se representan encerrando entre corchetes los valores que componen la secuencia. Por ordenada se entiende que dicho orden es el que aparece en la sucesión de valores generados por la secuencia. ■

Ejemplo 8.1 — Secuencia. ¿Cómo se representan las dos secuencias que generan las sucesiones de valores que se indican a continuación?

a) $0,1,3,0,1,3,0,1,3,\ldots$

b) $1,0,3,1,0,3,1,0,3,\ldots$

Solución:

a) $[0,1,3] \rightarrow \overline{0,1,3},\overline{0,1,3},\overline{0,1,3},\ldots$

b) $[1,0,3] \rightarrow \overline{1,0,3},\overline{1,0,3},\overline{1,0,3},\ldots$

Observa, que aunque las secuencias anteriores contienen los mismos valores, son secuencias diferentes, al igual que las sucesiones generadas por ellas. ■

Definición 8.3 — Contador. Circuito lógico secuencial constituido por biestables que genera en su salida una sucesión correspondiente a una secuencia de valores únicos, sin repetir. Además, la secuencia debe admitir una codificación de modo que coincida con el estado codificado en el sistema. Se dice que en este caso el estado del sistema y su salida coinciden. ■

Ejemplo 8.2 — Secuencia de salida en un contador. Dadas las secuencias siguientes: $[0,1,2,3]$, $[0,1,2,3,2]$ y $[0,1,8,9]$; indica qué sucesión de valores genera cada una de ellas y si corresponden a un contador.

Solución:

a) $[0,1,2,3] \rightarrow \overline{0,1,2,3},\overline{0,1,2,3},\ldots$
SI representa la secuencia de salida de un contador porque está compuesta por elementos únicos. Esto es, dentro del corchete no existe ningún valor que se repita.

b) $[0,1,\mathbf{2},3,\mathbf{2}] \rightarrow \overline{0,1,\mathbf{2},3,\mathbf{2}},\overline{0,1,\mathbf{2},3,\mathbf{2}},\ldots$
NO representa la secuencia de salida de un contador, ya que dentro del corchete que define la secuencia se repite el valor 2. Esto es, tiene un valor repetido.

c) $[0,1,8,9] \rightarrow \overline{0,1,8,9},\overline{0,1,8,9},\ldots$
Dependiendo de la implementación, puede representar la salida de un contador o no. Si se emplean 4 bits para codificar el estado, se emplearían 4 biestables de modo que la salida y el estado coincidan. En este caso se trataría de un contador. Sin embargo, como el sistema tiene cuatro estados, es posible una implementación óptima utilizando dos biestables para el contador. Para obtener la salida deseada se aplicaría una decodificación a la salida del contador para obtener los valores deseados.

■

En los contadores, se emplea habitualmente biestables de tipo T u otro con la función de un T, como el J-K con $J = K$. El cambio de valor de la secuencia de salida del contador está gobernado por los flancos de la señal de disparo de los biestables. El reinicio asíncrono del contador se consigue mediante el empleo de las señales *PRE* (*preset*) y *CLR* (*clear*) de los biestables. Para su reinicio síncrono, se puede emplear la funcionalidad de carga del contador con un valor determinado, o bien el diseño de un circuito de excitación completo para sus biestables.

Definición 8.4 — Módulo de un contador. El módulo (MOD) de un contador es el número de valores (estados) distintos que genera. Dicho valor coincide con la longitud de la secuencia asociada al contador. ■

Ejemplo 8.3 — Módulo de un contador. Un circuito genera la sucesión de valores:

$$0,1,2,3,0,1,2,3,...$$

Indica a qué secuencia corresponden dichos valores, si se trata de un contador y en caso afirmativo, cuál es el módulo de dicho contador.

Solución:

$\overline{0,1,2,3},\overline{0,1,2,3},... \rightarrow [0,1,2,3]$: Es una secuencia que corresponde a un contador porque no tiene valores repetidos. El módulo del contador es 4 porque esta es longitud de la secuencia. También se dice que se trata de un contador MOD 4. ■

Si un contador está compuesto de n biestables entonces puede tener un número máximo de estados igual a 2^n, que son los posibles valores distintos de la salida del contador. Los contadores también se denominan circuitos divisores de frecuencia en donde el factor de división es igual al módulo del contador.

En general, para todos los sistemas secuenciales y en particular para los contadores, se debe distinguir cada estado global del sistema Q_i del estado local particular de cada biestable q_j, siendo j el índice del biestable. Siempre que sea posible, la codificación del estado Q_i empleará binario natural para codificar el valor decimal de i con los bits individuales q_j correspondientes al estado de cada biestable. De este modo es sencillo obtener el valor del contador, ya que se obtiene por concatenación directa de la salida binaria de los biestables.

Recuerda que en los contadores, el estado es la salida del contador. Además, a partir del estado Q_i es inmediato derivar el valor q_j de cada biestable, ya que dicho valor está dado por el bit j-ésimo del valor de i expresado en binario natural.

Ejemplo 8.4 — Codificación de estado en contadores. Dado un contador de 2 bits, indica el mínimo número n de biestables necesarios, número máximo de estados que se pueden obtener, la codificación de cada estado Q_i y los valores de estado individual de cada biestable q_j para cada estado Q_i del contador.

Solución:

Para almacenar cada bit se precisa un biestable. Por tanto, el contador es de dos biestables que denominamos q_1 y q_0. Con dos biestables (2 bits) el número máximo de estados que se pueden codificar son $2^2 = 4$. Para codificar cada uno de los 4 estados del contador empleando binario natural se tiene:

$$Q_0 = \underbrace{00}_{q_1\,q_0} \quad Q_1 = \underbrace{01}_{q_1\,q_0} \quad Q_2 = \underbrace{10}_{q_1\,q_0} \quad Q_3 = \underbrace{11}_{q_1\,q_0}$$

Observa que en este ejemplo se habla de los estados del contador sin indicar el orden en que se generan estos en una sucesión de valores de salida. ■

8.2.1 Clasificación de contadores

Los contadores se puede clasificar atendiendo a distintos criterios:

- Sincronismo. En un contador *síncrono*, los biestables comparten la señal de sincronismo *CLK*, pero en los *asíncronos*, la señal de sincronismo no es común a todos los biestables.
- Codificación. La secuencia binaria generada por el contador puede responder a distintos métodos de codificación: binario natural, BCD, código Gray, etcétera.
- Ordenación de la secuencia. Dependiendo de la ordenación de los valores de la secuencia generada por el contador, estos se denominan ordenados o desordenados. Cuando el contador es ordenado, dependiendo del sentido de la ordenación pueden ser: ascendentes o descendentes.

- Módulo. Un contador es de *secuencia completa* o completo, si su módulo (número de estados) coincide con el valor máximo de estados que se puede codificar con los biestables empleados. En caso contrario se denomina contador incompleto o de *secuencia incompleta*. Si la secuencia está ordenada, sin «huecos» entre valores consecutivos, pero es incompleta porque faltan valores al final de la ordenación, entonces se denomina contador de *secuencia truncada*.

Ejemplo 8.5 — Clasificación de contadores. Dadas las secuencias: [1,2,3], [2,1,3,0], [0,1,2] y [0,1,2,4]. Para cada una de ellas determina: número de estados, número mínimo de biestables necesarios y codificación empleada. Si se trata de un contador, indica su clasificación de acuerdo a los criterios establecidos en esta sección.

Solución:

- [1,2,3]: Esta secuencia tiene 3 estados, por tanto precisa de al menos 2 biestables (q_1, q_0) para codificarlos. Se puede emplear una codificación en binario natural para el estado: $Q_1 = 1$ con $[q_1\, q_0] = [0\,1]$, $Q_2 = 2$ con $[q_1\, q_0] = [1\,0]$, y $Q_3 = 3$ con $[q_1\, q_0] = [1\,1]$. Esta secuencia es un contador porque no tiene valores repetidos y el estado coincide con la salida. Se trata de un contador ordenado, ascendente e incompleto (no posee el valor 0). No es una secuencia truncada porque el contador tiene un «hueco» al comienzo de la secuencia, correspondiente al valor 0.
- [2,1,3,0]: Esta secuencia tiene 4 estados, por tanto precisa de al menos 2 biestables (q_1, q_0) para codificarlos. Se puede emplear una codificación en binario natural para el estado: $Q_2 = 2$ con $[q_1\, q_0] = [1\,0]$, $Q_1 = 1$ con $[q_1\, q_0] = [0\,1]$, $Q_3 = 3$ con $[q_1\, q_0] = [1\,1]$, y $Q_0 = 0$ con $[q_1\, q_0] = [0\,0]$. Esta secuencia es un contador porque no tiene valores repetidos y el estado coincide con la salida. Se trata de un contador desordenado y completo.
- [0,1,2]: Esta secuencia tiene 3 estados, por tanto precisa de al menos 2 biestables (q_1, q_0) para codificarlos. Se puede emplear una codificación en binario natural para el estado. Esta secuencia es un contador porque no tiene valores repetidos y el estado coincide con la salida. Se trata de un contador ordenado, ascendente, incompleto (no posee el valor 3) y de secuencia truncada, porque el valor que falta en la secuencia está al final de esta.
- [0,1,2,4]: Esta secuencia tiene 4 estados, por tanto precisa de al menos 2 biestables (q_1, q_0) para codificarlos. Con 2 biestables no se puede emplear una codificación en binario natural para el estado, pues el valor 4 no se puede representar con 2 bits. Para conseguir que el estado y la salida coincidan como en un contador serían precisos 3 biestables: $Q_0 = 0$ con $[q_2\, q_1\, q_0] = [0\,0\,0]$, $Q_1 = 1$ con $[q_2\, q_1\, q_0] = [0\,0\,1]$, $Q_2 = 2$ con $[q_2\, q_1\, q_0] = [0\,1\,0]$, y $Q_4 = 4$ con $[q_2\, q_1\, q_0] = [1\,0\,0]$. Si esta secuencia se implementa con 3 biestables es considerado un contador porque no tiene valores repetidos y el estado coincide con la salida. Se trataría de un contador ordenado, ascendente, incompleto (no posee el valor 3), pero no es de secuencia truncada porque la secuencia tiene un hueco.

  En este caso se pueden emplear solo dos biestables con un circuito combinacional que recodifique el código de salida de un contador completo ascendente MOD 4, al valor de salida deseado. Las entradas al circuito combinacional mencionado serían q_1, q_0 (el estado) y la salida f_1, f_2, f_3 (el valor de la secuencia). En este caso el sistema combinacional resultantes es muy sencillo como se deduce de la tabla de verdad adjunta:

q_1	q_0	f_2	f_1	f_0	val. sec.
0	0	0	0	0	(0)
0	1	0	0	1	(1)
1	0	0	1	0	(2)
1	1	1	0	0	(4)

$f_2 = q_1 \cdot q_0 \qquad f_1 = q_1 \cdot q_0' \qquad f_0 = q_1' \cdot q_0$

■

8.2.2 Contadores asíncronos

Para describir la implementación de un contador partimos del análisis de la secuencia asociada. Para ello se considera el caso concreto de un contador ascendente MOD8 de secuencia completa para el que se emplean 3 biestables. La tabla de la figura 8.10 presenta los estados globales del contador (Q_0 a Q_7) y los estados individuales (q_2, q_1, q_0) que debería tener cada uno de los biestables para codificar el valor del estado. En la misma figura junto a la tabla se muestra el cronograma de la evolución temporal del estado individual de cada biestable.

La sucesión de estados del biestable menos significativo q_0 $(0,1,0,1,...)$ se puede obtener de modo sencillo mediante un biestable T con su entrada conectada permanentemente a 1. De este modo el biestable cambiará de estado en cada flanco de disparo de la señal de sincronismo *CLK*.

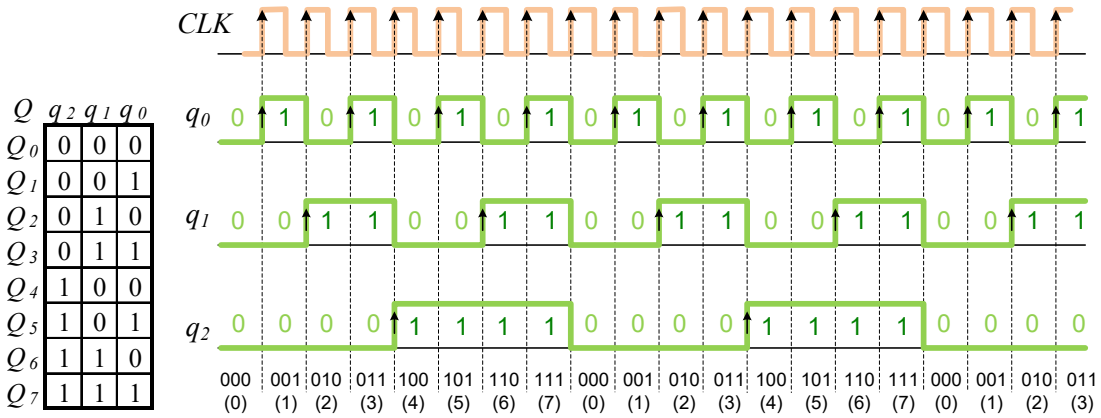


Figura 8.10: Secuencia de estados de un contador ascendente de 8 estados y 3 biestables, junto con el cronograma del estado de cada biestable.

> Observa que en el cronograma de los contadores presentados en este texto es habitual emplear un orden creciente de significancia de los bits hacia abajo del cronograma. Así, en la figura 8.10 se observa que en la parte superior se coloca el estado del biestable menos significativo (q_0) y en la parte inferior el más significativo (q_2).

Para el biestable q_1 se observa un comportamiento similar a q_0, pero su cambio de estado se produce cuando q_0 pasa de valer 1 a 0. Esto es, en el flanco negativo de q_0. Por tanto, si se desea emplear solo biestables sincronizados por flanco positivo (de subida), se puede afirmar que q_1 cambia de estado con los flancos positivos de q_0', ya que los flancos negativos de una señal, son los positivos de la señal invertida. De este modo, el comportamiento de q_1 se obtiene con un biestable T con entrada $T = 1$ y señal de sincronismo conectada a la señal q_0' del biestable previo. Con un razonamiento análogo se llega a que el comportamiento del biestable q_2 se obtiene mediante biestable T con entrada $T = 1$ y señal de sincronismo conectada a q_1'. Por tanto, la implementación del contador asíncrono ascendente MOD 8 resultante se muestra en la figura 8.11.

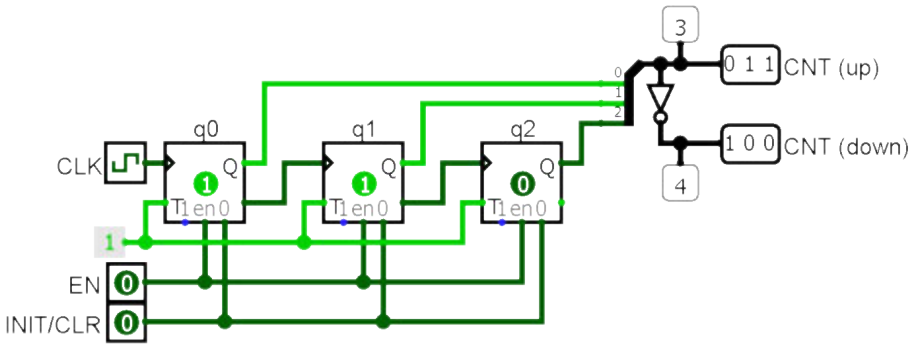


Figura 8.11: Diagrama lógico de contador asíncrono MOD 8 ascendente con biestables T.

En el circuito de la figura 8.11 se ha incluido una señal para el reinicio asíncrono del contador (*INIT*/*CLR*) conectada a la entrada *CLR* de los biestables. También se incluye una señal de habilitación (*EN*) conectada a la entrada *enable* de cada biestable. La figura añade una sonda de prueba (*probe*) para mostrar el valor decimal de la salida del contador. Dicha salida se complementa con un inversor para obtener de modo inmediato el contador descendente.

Es muy importante reconocer que el contador implementado es de tipo asíncrono —aunque los biestables sean de tipo síncrono—, ya que los biestables no comparten la señal de sincronismo. Como se observa en la figura 8.11, el valor binario de cada biestable se indica con un subíndice, de modo que en la codificación del estado, q_2 y q_0 son el biestable más y menos significativo, respectivamente.

Para obtener un contador descendente se pueden seguir varias estrategias:

1. Intercambiar las conexiones de Q y Q' para cada biestable. Si en el ascendente la salida Q' de un biestable se conecta a la entrada de sincronismo del siguiente biestable, en el descendente se conecta la salida Q con la entrada de sincronismo del siguiente biestable. El análisis se puede repetir a partir de la tabla de evolución de los estados del contador para concluir que los cambios de estado de un biestable se producen en los flancos positivos del estado del biestable previo.
2. Sincronizar los biestables por flanco de bajada. Manteniendo todas las conexiones y cambiando el flanco de disparo de todos los biestables, se obtiene un contador descendente.
3. Complementar los bits de salida del contador ascendente. La comprobación es sencilla a partir de la tabla de estados del contador. El complemento es sencillo de obtener, ya que cada biestable ofrece directamente el valor de su estado Q y el complemento Q'.

> En el diagrama de la figura 8.11 para obtener el contador descendente se ha añadido un inversor, pero dicho valor se obtiene directamente de cada biestable, ya que cada uno proporciona tanto su estado Q como el complemento Q'. Recuerda que en un contador descendente, el valor de *CLR* debe poner todos los biestables a 1. De este modo se obtiene el valor de partida del contador que es el valor más alto del contador.

Contadores de secuencia truncada

Un contador de secuencia truncada es un contador ascendente ordenado incompleto en su cuenta superior, o bien descendente ordenado e incompleto en su cuenta inferior. El contador de secuencia truncada más popular en los sistemas digitales es el contador MOD 10, conocido también como contador BCD o *contador de décadas*. Es un contador con 10 estados (de Q_0 a Q_9) y 6 estados no alcanzados. Para su implementación se precisan 4 biestables (q_3, q_2, q_1, q_0).

Para implementar los contadores asíncronos de secuencia truncada se puede recurrir al *truncamiento asíncrono* del contador al alcanzar el siguiente valor al máximo deseado. De este modo, en el contador MOD 10 se incorpora una puerta AND que decodifica el valor 10, alcanzado cuando $[q_3\, q_2\, q_1\, q_0] = [1\,0\,1\,0]$, que activa el *reset* asíncrono de los biestables mediante la señal *CLR* que los lleva inmediatamente al valor 0. La figura 8.12 muestra el diagrama lógico del circuito correspondiente en Logisim.

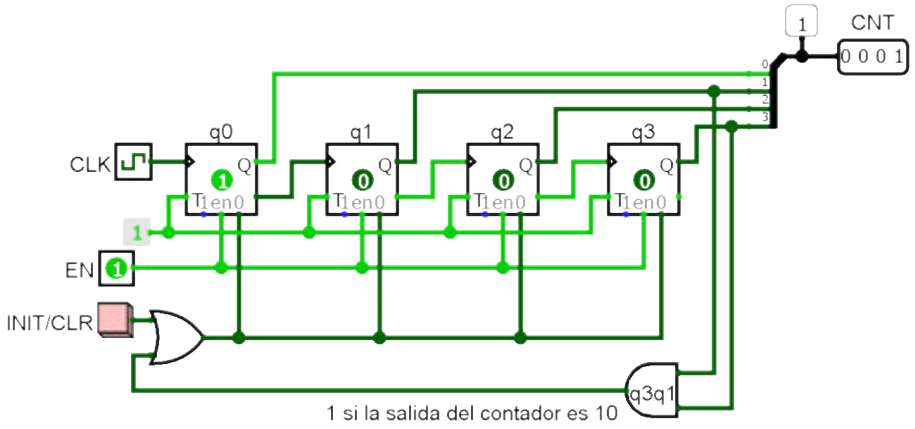


Figura 8.12: Diagrama lógico de contador de décadas asíncrono con truncamiento asíncrono.

El ajuste de la condición de truncamiento del contador se debe realizar con cuidado. En el circuito de la figura 8.12, el valor decodificado con la puerta AND en realidad no es $10_{(10}$, puesto que el producto q_3q_1 con código binario 1*1* se puede expandir a los minitérminos: $1010_{(2} \equiv 10_{(10}$, $1011_{(2} \equiv 11_{(10}$, $1110_{(2} \equiv 14_{(10}$ y $1111_{(2} \equiv 15_{(10}$. En consecuencia la puerta AND genera un 1 para cualquiera de dichos valores, pero como el contador es ascendente, el primero que se alcanza es el 10 y de este modo, con una puerta de dos entradas más simple se consigue el efecto deseado. Para el truncamiento en sentido descendente se podría seguir un esquema análogo decodificando los biestables que deben ser 0.

Limitaciones de los contadores asíncronos

Los biestables de un contador asíncrono no emplean la misma señal de sincronismo. Por tanto, no es posible garantizar que su cambio de estado se produce de modo simultáneo. De hecho la señal de sincronismo de cada biestable —a excepción del inicial— es la consecuencia del cambio de estado del biestable precedente. Como resultado, en este tipo de contador se produce un retardo (*delay*) que se propaga en cascada de un biestable al siguiente.

En consecuencia, el retardo en una etapa se acumula al de la etapa previa hasta completar el total de etapas. El retardo acumulado afecta al cambio de estado de cada biestable conectado en la cascada de biestables, para ser tanto mayor cuanto más profundo es el biestable en la cascada de conexión. Este efecto se debe a los retardos de propagación de señales en los biestables.

El efecto del retardo acumulado en la salida del contador produce un «rizado» (*ripple*) de la salida del contador. Por esta razón, este tipo de contador se conoce como *contador con rizado* (*ripple counter*) o contador de propagación (*propagation counter*). La figura 8.13 muestra el efecto del retardo en la respuesta de cada uno de los biestables del contador —exagerado para mayor claridad— y cómo este puede dar lugar a valores incorrectos de cuenta.

En la figura 8.13 se presenta la evolución del estado de los tres biestables del contador para mostrar cómo entre el valor 3 y 4 existen instantes en los que se puede obtener un valor erróneo de cuenta (2 y 0). Por ejemplo, si el valor de este contador generase los valores de las señales de selección de un MUX 8×1, durante algunos instantes se podría obtener una salida errónea.

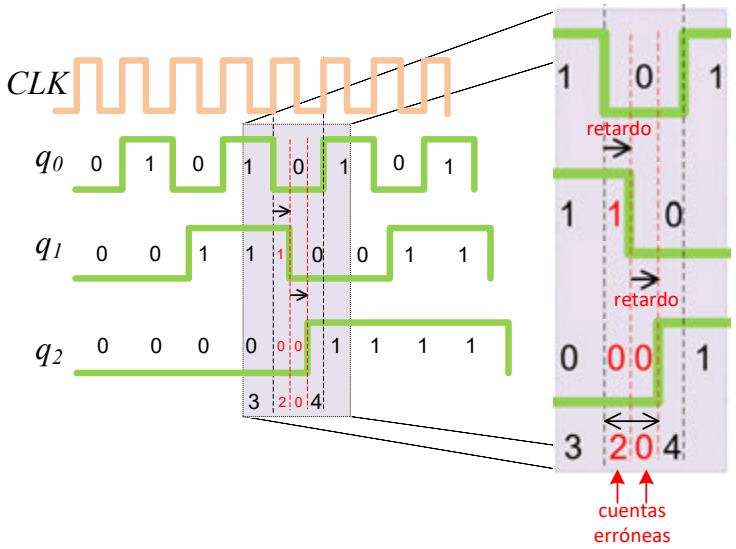


Figura 8.13: Cronograma del estado de un biestable asíncrono MOD 8 ascendente mostrando el retardo acumulado y los posibles errores de cuenta que provoca (dcha.).

Para evitar la situación de cuentas erróneas derivadas de los contadores con rizado, se recurre a una técnica sencilla denominada *strobbing*. Dicha estrategia consiste en habilitar en el circuito de destino el valor procedente del contador mediante el nivel bajo de la señal de reloj (si el contador emplea biestables disparados por flanco de subida). De este modo hasta que la señal de reloj no está en nivel bajo la salida del contador no tiene efecto en el circuito de destino. Esta técnica también es válida para utilizar satisfactoriamente los valores de contadores obtenidos mediante truncamiento asíncrono. Por ejemplo, se

podría decodificar el valor máximo deseado para el contador habilitando el reinicio del mismo en el nivel bajo de la señal de sincronismo si los biestables son disparados por flanco de subida.

■ **Ejercicio 8.1** Modifica el circuito de la figura 8.12 para que el truncamiento asíncrono del contador se realice cuando el valor del contador es 9 y se habilite el borrado de los biestables con el nivel bajo de la señal de reloj (*CLK*). ■

El rizado de los contadores asíncronos limita la frecuencia máxima permitida del reloj, ya que el retardo acumulado debe ser menor que el periodo de la señal de reloj. Por este motivo los contadores síncronos pueden ser más rápidos que los asíncronos. Una aplicación de los contadores asíncronos es el conteo de la ocurrencia asíncrona de eventos (p. ej., paso de personas, vehículos, material, etc.) utilizando la transición en la señal del evento como señal de sincronismo del contador.

8.2.3 Contadores síncronos

El diseño de sistemas asíncronos y en particular de contadores, presenta dificultades que limitan su aplicación en favor de los sistemas síncronos. En un contador síncrono todos los biestables comparten la señal de sincronismo. De este modo, el cambio de estado en todos los biestables se produce de forma simultánea. Como todos los biestables comparten la señal de reloj *CLK*, la funcionalidad deseada se obtiene calculando el valor apropiado de las entradas —señales de excitación— para cada biestable.

Para analizar la implementación de un contador síncrono, partimos del estudio del contador ascendente MOD 8 para el que se emplean 3 biestables. La tabla 8.1 muestra la evolución temporal de estados del contador y los estados individuales de cada biestable. También se añaden las señales de excitación a cada biestable (T_2, T_1, T_0) para conseguir la evolución de estado requerida en los instantes de sincronismo. Con los valores de la tabla se obtiene el valor de la excitación de cada biestable como una función del estado (Q_t).

Tabla 8.1: Tabla de transición de estado del contador síncrono ascendente MOD8.

t				$t+1$													
Q	q_2	q_1	q_0	Q	q_2	q_1	q_0	T_2	T_1	T_0	Q	q_2	q_1	q_0	T_2	T_1	T_0
Q_0	0	0	0	Q_1	0	0	1	0	0	1	Q_0	0	0	0	0	0	1
Q_1	0	0	1	Q_2	0	1	0	0	1	1	Q_1	0	0	1	0	1	1
Q_2	0	1	0	Q_3	0	1	1	0	0	1	Q_2	0	1	0	0	0	1
Q_3	0	1	1	Q_4	1	0	0	1	1	1	Q_3	0	1	1	1	1	1
Q_4	1	0	0	Q_5	1	0	1	0	0	1	Q_4	1	0	0	0	0	1
Q_5	1	0	1	Q_6	1	1	0	0	1	1	Q_5	1	0	1	0	1	1
Q_6	1	1	0	Q_7	1	1	1	0	0	1	Q_6	1	1	0	0	0	1
Q_7	1	1	1	Q_0	0	0	0	1	1	1	Q_7	1	1	1	1	1	1

Para realizar el cálculo de la excitación en los biestables, la tabla de transición se trata como una tabla de verdad para las funciones de excitación mencionadas. Las entradas son los estados en t para cada biestable (q_j) y las salidas son las entradas (excitaciones) a dichos biestables (T_j). Una vez realizado el cálculo se obtiene el circuito de la figura 8.14 con las funciones de excitación: $T_0 = 1$, $T_1 = q_0$ y $T_2 = q_1 q_0$.

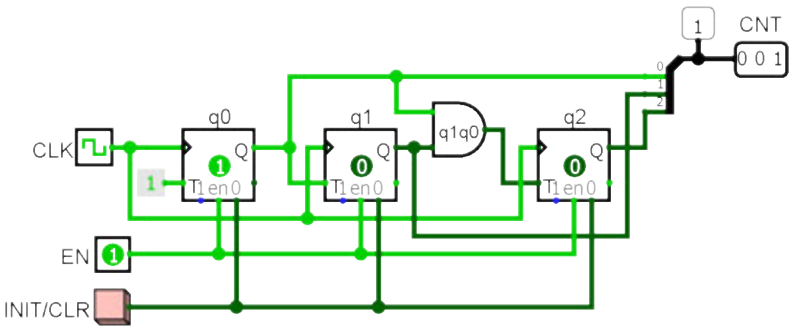


Figura 8.14: Circuito de contador síncrono ascendente MOD 8.

La implementación de un contador con un número mayor de biestables seguiría un esquema en el que la entrada de cada biestable añadido (a la dcha. del circuito) es el resultado del AND de las salidas de los biestables previos como se muestra en la figura 8.15.

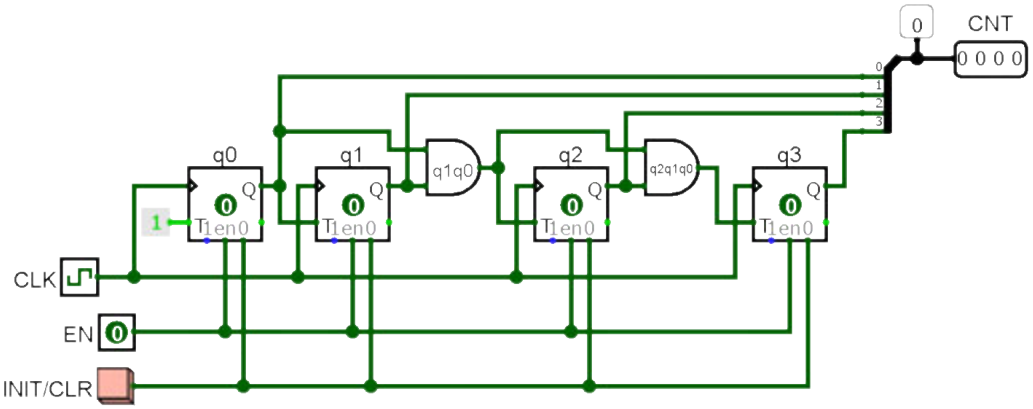


Figura 8.15: Circuito de contador síncrono ascendente MOD 16.

Para conseguir contadores descendentes se puede seguir un estudio análogo al explicado anteriormente. De este modo, en el caso del contador descendente MOD 8 se tendría el circuito de la figura 8.16 con las funciones de excitación: $T_0 = 1$, $T_1 = q_0'$ y $T_2 = q_1' q_0'$.

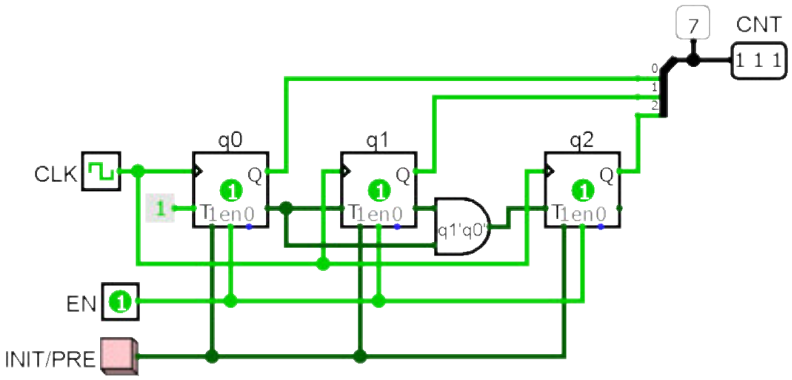


Figura 8.16: Circuito de contador síncrono descendente MOD 8.

> Observa que en los contadores descendentes el valor inicial de cuenta es el valor binario más alto del contador. Por tanto, su señal de reinicio (*INIT/PRE*) activa la puesta a 1 de los biestables (*preset*).

Contador de décadas síncrono

Para la implementación del contador de décadas (MOD 10) síncrono se puede emplear una estrategia de truncamiento similar a la empleada en el caso del contador asíncrono (ver figura 8.17).

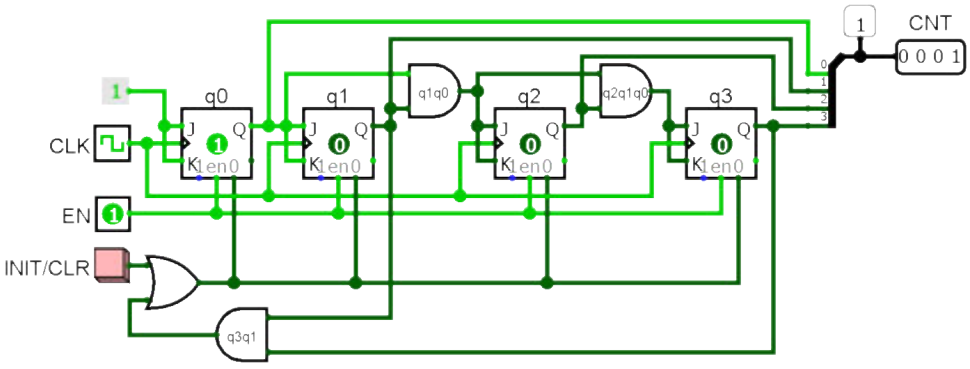


Figura 8.17: Circuito lógico de contador décadas ascendente síncrono con reinicio asíncrono.

Desde un punto de vista lógico, el reinicio asíncrono de un contador síncrono ofrece un resultado correcto. Sin embargo, un análisis temporal riguroso de la salida del contador determina que durante un breve instante la salida del contador es incorrecta (10) superando el valor máximo (9) permitido para el contador. Aplicando la técnica denominada *strobing*, el truncamiento asíncrono realiza el reinicio del contador en el valor máximo de cuenta, habilitado con el nivel bajo de la señal de reloj (*CLK*).

El truncamiento síncrono evita los problemas mencionados evitando el reinicio del contador mediante la señal de borrado asíncrono de los biestables. En su lugar, se modifica la excitación de los biestables para conseguir la secuencia de estados requerida. La figura 8.18 muestra el circuito resultante con las funciones de excitación de los biestables obtenidas a partir de la tabla de transición de estados del contador, considerando todos los estados superiores a Q_9 como condiciones libres (*don't care*). Las funciones resultantes son: $T_0 = 1$, $T_1 = q_3' q_0$, $T_2 = q_1 q_0'$ y $T_3 = q_2 q_1 q_0 + q_3 q_0$

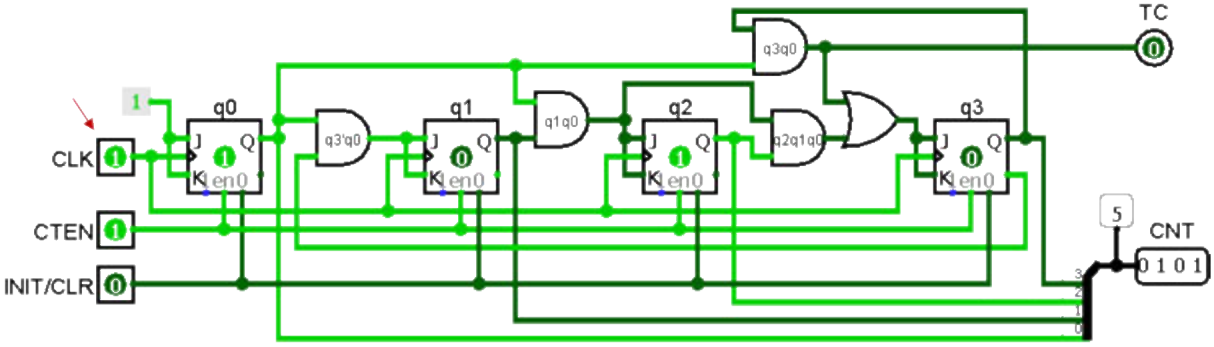


Figura 8.18: Diagrama lógico de contador de décadas con truncamiento síncrono y señales de habilitación de cuenta (*counter enable*) y de término de cuenta (*terminal count*).

El circuito de la figura 8.18 incluye señal de habilitación de cuenta (*CTEN*, *counter enable*) y señal de término de cuenta (*TC*, *terminal count*). Dichas señales facilitan la conexión en cascada (*daisy chain*) de módulos contadores conectando la salida *TC* a la entrada *CTEN* del módulo de mayor orden. En consecuencia, es posible conseguir divisores de 10, 100, 1000, etcétera. La figura 8.19 muestra el esquema conseguido para simulación lógica con Logisim del comportamiento de dos módulos contadores con la implementación presentada en la figura 8.18 para obtener un contador de 0 a 99.

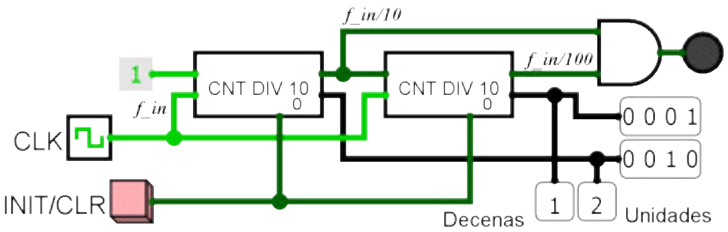


Figura 8.19: Contadores de décadas conectados en cascada para obtener un contador de unidades y decenas.

> Recuerda que para crear en Logisim un módulo como el contador de décadas de la figura 8.18, debes incluir la señal de sincronismo con un pin de entrada convencional en lugar de emplear el elemento reloj. Dicho pin aparece marcado con una flecha en la figura mencionada. En dicho pin se conectará la señal de reloj deseada cuando el módulo sea empleado en un diseño jerárquico.

Una alternativa práctica al reinicio síncrono de un contador truncado, consiste en el empleo de la decodificación del valor final de cuenta para activar la señal de carga (*load*) del contador con el valor deseado de reinicio, habitualmente 0. Este método exige que el contador incorpore la funcionalidad de carga en paralelo de sus biestables. La figura 8.20 muestra el diagrama lógico de un contador de 4 bits con funcionalidad de carga en paralelo y cómo se aplica a un contador de décadas mediante reinicio síncrono. Este circuito se considera de interés para ilustrar un método de carga en un contador mediante el uso de biestables D y multiplexores.

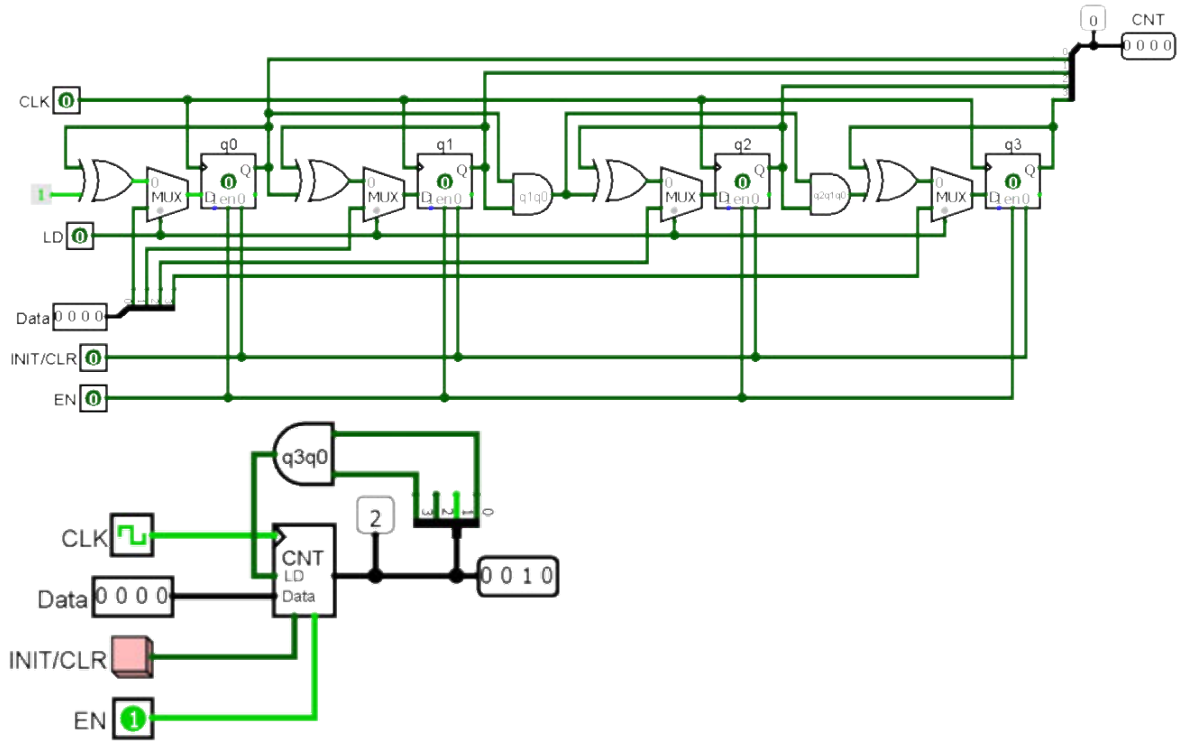


Figura 8.20: Contador con carga en paralelo (sup.) y aplicación a contador de décadas con reinicio síncrono.

Contador síncrono bidireccional

La implementación del contador descendente es análoga a la empleada para el contador ascendente considerando las señales complementarias que intervienen en la excitación de los biestables. Esto es, se cambia q_j por q'_j en las expresiones algebraicas de las funciones de excitación de los biestables.

En un contador bidireccional es preciso añadir al contador, una señal de control (*UP'/DOWN*) que permita controlar el sentido de la cuenta. Esta señal se puede emplear como señal de selección de un MUX 2×1 para conectar a cada biestable la entrada q_j o q'_j según corresponda (ver figura 8.21). En la figura 8.21 también se muestra el proceso de decodificación de la señal de dirección del contador para reiniciar la cuenta en 0 o 7 activando el *clear* o *preset* de todos los biestables.

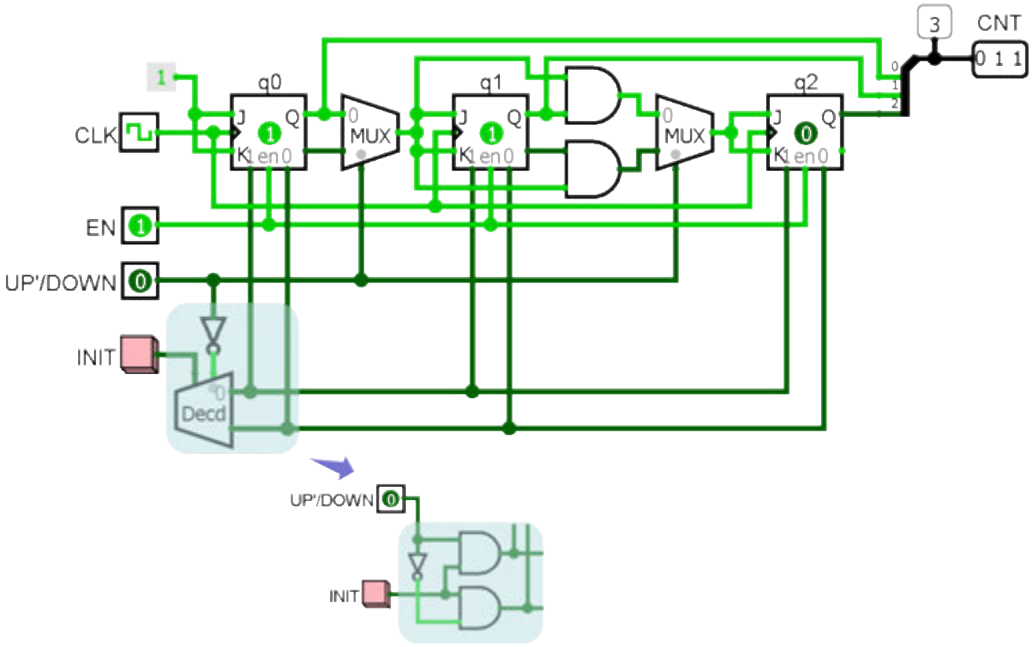


Figura 8.21: Circuito lógico de contador bidireccional mediante MUX 2×1.

La figura 8.22 ilustra una estrategia alternativa para implementar la bidireccionalidad del contador partiendo de su implementación ascendente y tomando el complemento de su salida para obtener el contador descendente —válido también para los contadores asíncronos—. En este caso mediante una señal adicional ($UP'/DOWN$) de entrada en las puertas XOR se puede controlar si la salida del circuito es la del contador empleado o su complemento. El reinicio del contador es correcto al realizar la puesta a 0 de todos los biestables, pues en el caso de cuenta descendente el complemento ya ofrece el valor correcto al inicio.

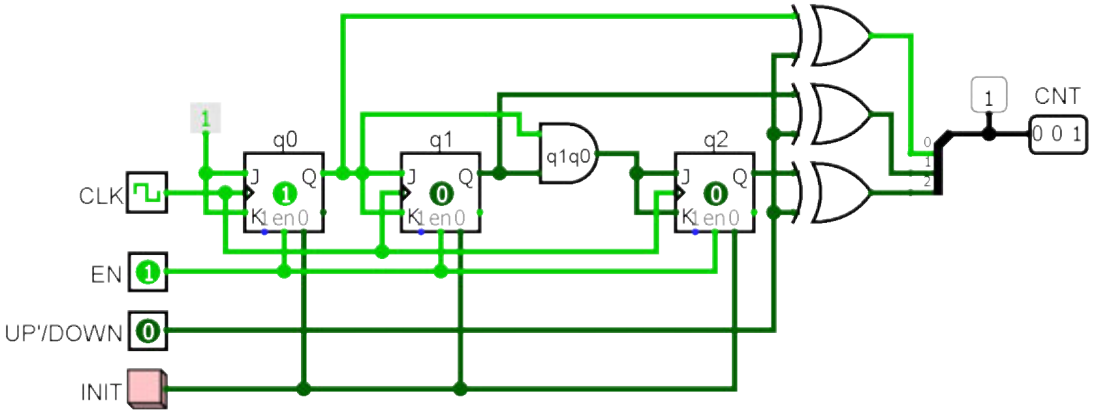


Figura 8.22: Circuito lógico de contador bidireccional mediante puertas XOR.

El elemento contador de Logisim

Logisim proporciona un elemento contador universal (*counter*) en la librería *Memory* con una funcionalidad fijada por el valor de sus atributos descritos a continuación:

- Número de bits del contador, con posibilidad de definir el valor máximo.
- Selección de comportamiento al alcanzar valor máximo prefijado:
  - Vuelta a cero en cuenta continua (*wrap around*).
  - Enclavamiento en valor máximo (*stay at value*).
  - Continua hasta valor máximo (*continue counting*)
  - Carga valor (*load next value*). Cuando se alcanza el valor máximo de cuenta se carga el próximo valor del contador. Necesario en el caso de cuenta descendente.
- Tipo de flanco de disparo (de subida o bajada).
- Cuenta ascendente o descendente.
- Carga de un valor en el contador.

Como entradas al contador se fijan la señal de disparo (*CLK*), el valor de carga (*Data*), el bit de carga (*LD*) y la habilitación de cuenta (*CNT*). Dichas señales permiten fijar la dirección de cuenta y la inhibición del contador. Por último, se incluye como entrada el valor de borrado asíncrono (*CLR*). Como salidas se obtiene: el valor de cuenta (Q) y el bit de fin de cuenta (TC).

8.2.4 Contadores basados en registros de desplazamiento

Los registros de desplazamiento con la salida de su etapa final realimentando a la etapa inicial, se caracterizan por repetir continuamente una secuencia. El valor del registro cambia con el desplazamiento de un bit de un biestable al siguiente. Las secuencias generadas cumplen la condición de un contador, ya que no hay valores repetidos en la secuencia. Se trata de contadores de secuencia incompleta, puesto que cuando el número de biestables empleado es n tienen un número de estados (n) inferior al máximo permitido (2^n). A continuación se explica dos tipos de contadores basados en registros de desplazamiento.

Contador en anillo

Se implementa con un registro de desplazamiento cuya salida q_n del último biestable realimenta la entrada al primero ($D_0 = q_n$). El contador genera la secuencia derivada de rotar el valor de inicialización del registro. Por tanto, es un contador con un número de estados igual al de biestables que componen el registro empleado. En el contador de la figura 8.23 los estados posibles son: [0001, 0010, 0100, 1000].

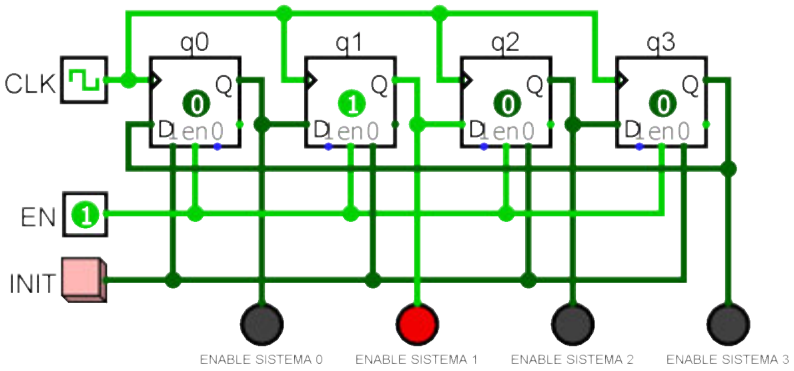


Figura 8.23: Contador en anillo de 4 bits con *one-hot encoding*.

Esta estrategia ofrece un método simple de implementación de un contador MOD n (divisor por n), para el que es preciso disponer de n biestables. Aunque este método es válido para rotar cualquier secuencia binaria, cuando la secuencia tiene un único 1, se denomina codificación *one-hot encoding*. De modo dual, cuando la secuencia tiene un único 0, recibe el nombre *one-cold enconding*. En estos casos el contador en anillo genera un valor decodificado del estado. Con esta estrategia se simplifica el diseño a costa de aumentar el número de biestables, pero elimina la necesidad de un decodificador a la salida como se muestra en la figura 8.24.

La salida decodificada del contador en anillo es interesante en aplicaciones de activación secuencial de distintos subsistemas, como por ejemplo el mostrado en la figura 5.6 de la pág. 121. En estas aplicaciones la salida decodificada del contador se conecta a las entradas de habilitación (*enable*) del conjunto de subsistemas que se desea habilitar secuencialmente. Cuando la habilitación es activa a nivel bajo se emplea *one-cold encoding*.

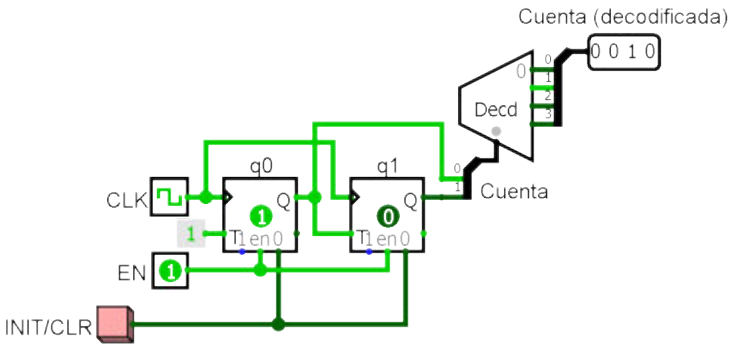


Figura 8.24: Circuito lógico de contador de 2 bits seguido de DEC 2×4, equivalente a contador en anillo de 4 bits con *one-hot encoding*.

Las desventajas principales de los contadores en anillo son las siguientes:

1. Necesitan el mismo número de biestables que de estados. Esto restringe la capacidad de representación máxima del contador.
2. Requieren la inicialización de los biestables en el arranque del sistema. En este contador se debe partir de la configuración de ceros y unos que se desea rotar. Por ejemplo, con uno solo de los biestables a 1 y el resto a 0 para la codificación *one-hot encoding*.

Contador Johnson

Este contador también emplea un registro de desplazamiento, pero se obtiene por realimentación de la salida negada del último biestable q'_n a la entrada del primero ($D_0 = q'_n$). De este modo si el registro comienza con todos los biestables a cero, se completa progresivamente con unos. A continuación se va completando con ceros hasta completar el ciclo. La figura 8.25 presenta el circuito lógico de un contador Johnson de cuatro bits para simulación lógica con Logisim.

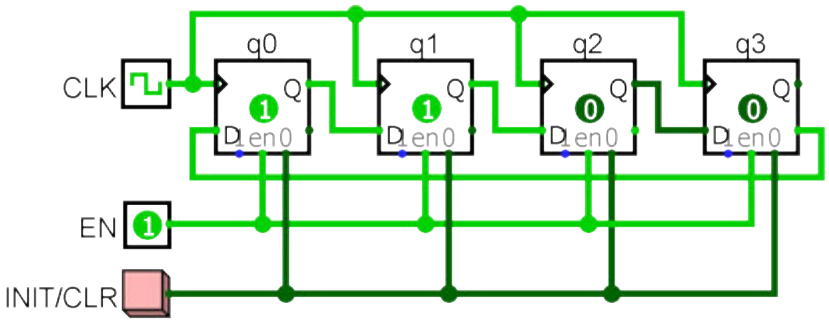


Figura 8.25: Circuito lógico de contador Johnson de 4 bits.

El número total de estados del contador Johnson es $2 \cdot n$, siendo n el número de biestables del contador. Los estados del contador de la figura 8.25 serían: [0000, 0001, 0011, 0111, 1111, 1110, 1100, 1000].

El contador Johnson se emplea de modo muy similar al contador en anillo, pero con la mitad de biestables que este último. Si se implementa con un registro de n biestables se puede conseguir un contador MOD $2n$ (divisor $2 \cdot n$) al que se añaden puertas decodificadoras —como en el circuito de la figura 8.26— para obtener un contador con salidas decodificadas para aplicaciones similares a las de un contador en anillo.

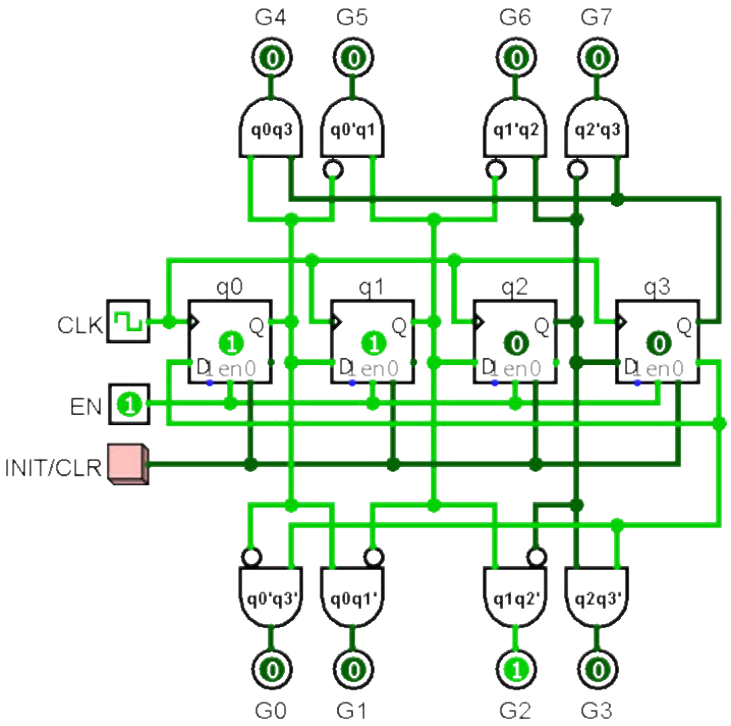


Figura 8.26: Contador Johnson de 4 bits con salida decodificada.

Puesto que este tipo de contadores perpetúan el desplazamiento de la secuencia en los biestables, para evitar la posible corrupción de la secuencia, es frecuente añadir alguna puerta adicional que permita «autocorregir» la secuencia tras algunos ciclos de reloj (como en el circuito de la figura 8.27). El circuito resultante se conoce como contador *Johnson autocorregido*.

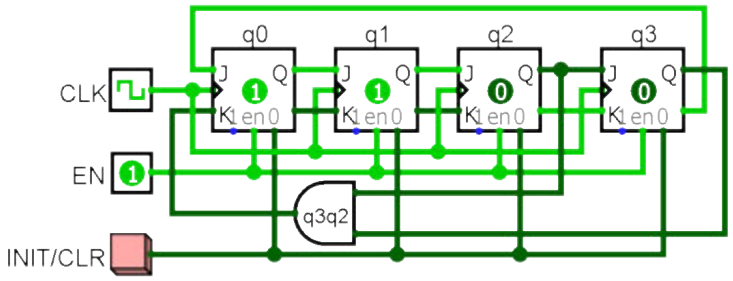


Figura 8.27: Circuito lógico de contador Johnson autocorregido de 4 bits.

Aplicaciones de contadores

En el control secuencial de sistemas físicos dirigidos por eventos, los contadores se pueden utilizar para contabilizar eventos ocurridos en el sistema. Por ejemplo, vehículos que entran en un aparcamiento, usuarios que entran en un edificio, piezas que han llegado a una célula automatizada de embalaje mediante una cinta transportadora, etcétera. En estos casos un detector —sensor todo (1)/nada (0)— se encarga de generar la transición de una señal que determina la ocurrencia del evento contabilizado por el contador.

> Ten cuidado para no confundir el control secuencial de sistemas físicos dirigidos por eventos, con el paradigma de programación dirigida por eventos (*event-driven programming*). Aunque en ambos casos los eventos «dirigen» la evolución del sistema, en los primeros se controla una secuencia definida de etapas cuyo inicio/fin está marcado por la ocurrencia de eventos discretos como la activación de detectores o finalización de acciones temporizadas. Sin embargo, en el segundo se realiza la programación de un sistema que responde a eventos no deterministas como por ejemplo, las acciones de un usuario en una interfaz gráfica. Aunque en ambos tipos de sistemas los eventos son de naturaleza asíncrona, el valor de dichos eventos tanto en uno como otro, se puede capturar de modo síncrono. Esto permite tomar decisiones síncronas sobre el control de ambos tipos de sistemas a través de la ejecución de acciones.

La estrategia más sencilla para contabilizar eventos consiste en sustituir la señal de sincronismo *CLK* de un contador por la señal procedente del detector. De este modo las transiciones generadas en el detector disparan el cambio de estado del contador que cuenta el número de transiciones en la señal generada por el evento. En la figura 8.28 se muestra el diagrama lógico del circuito para esta solución mediante empleo de un contador asíncrono. Por tanto, la salida del contador es asíncrona respecto al reloj del sistema.

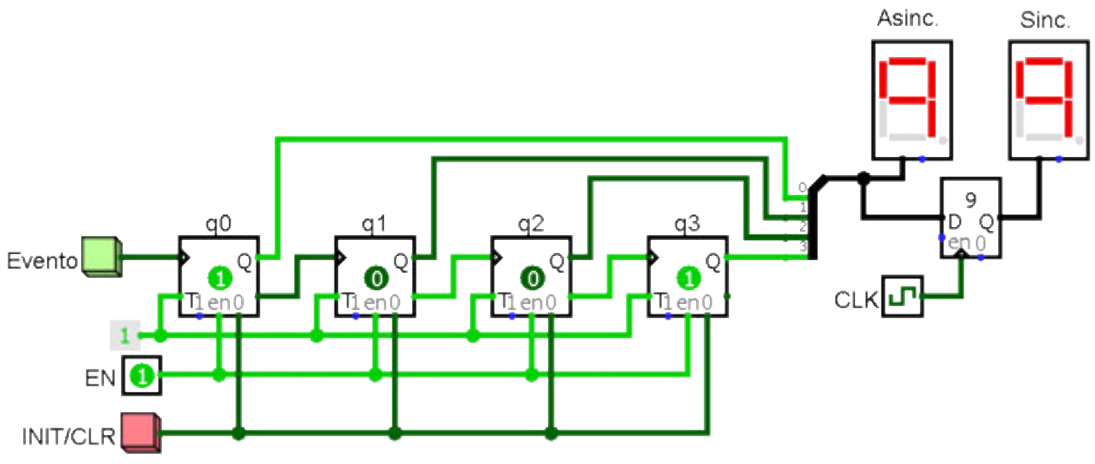


Figura 8.28: Contador asíncrono de eventos con registro de sicronización a la salida.

Para sincronizar la salida del contador con el reloj del sistema se emplea un registro a la salida. En estos casos la pérdida de sincronismo —saltos en la cuenta de salida— se evita ajustando la frecuencia del reloj de sincronismo para que sea superior a la frecuencia de la ocurrencia de los eventos. De este modo la salida síncrona es fiel reflejo de la evolución del contador. En caso contrario, la salida sincronizada avanza a saltos porque entre dos flancos consecutivos de la señal de reloj se produce varios eventos. De modo similar se podría utilizar un contador síncrono con la señal de reloj sustituida por la señal del evento.

> Recuerda que la estrategia de conteo de eventos sustituyendo la señal de disparo por la señal del evento no es aplicable en todos los sistemas digitales. Por ejemplo, en ciertos diseños de contadores sintetizados mediante FPGA se obliga a que los biestables empleen como señal de sincronismo una señal definida como reloj del sistema.

Una estrategia alternativa para obtener un contador de eventos síncrono, consiste en el diseño de un autómata finito —explicados en el capítulo próximo— para generar un pulso de duración de un ciclo de reloj por cada transición ocurrida en la señal del evento. En la figura 8.29 se ilustra la implementación de un contador síncrono de eventos implementando un autómata finito de tipo Mealy en el que la entrada es el evento asíncrono a contar y la salida es un evento de duración apropiada para el contador. En este caso el autómata es muy sencillo pues consiste en un biestable D y una puerta AND.

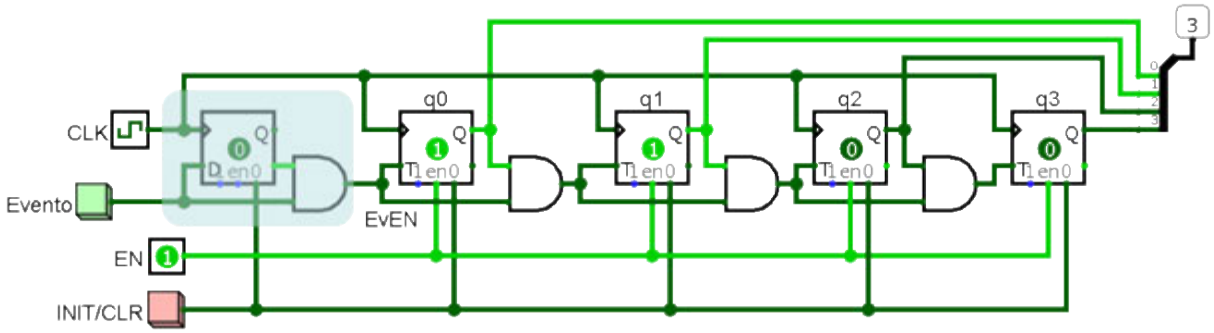


Figura 8.29: Circuito de contador síncrono de eventos asíncronos mediante autómata de Mealy.

Las dos estrategias comentadas anteriormente se presentan en los circuitos alternativos de la figura 8.30. En ellos se emplea el elemento *Counter* incorporado en la librería *Memory* de Logisim.

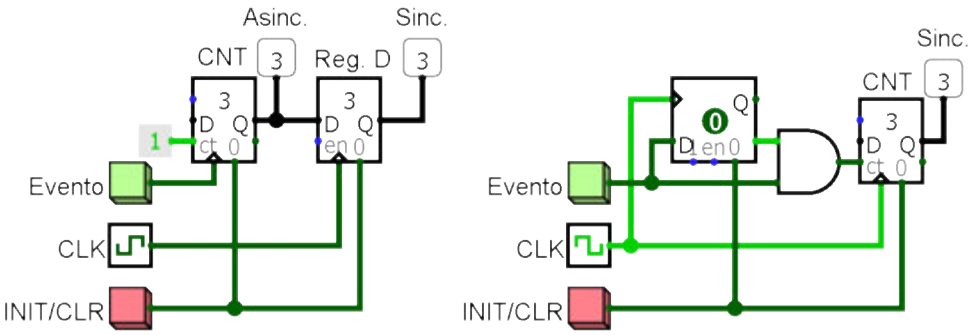


Figura 8.30: Circuitos para cuenta de eventos en Logisim: a) mediante cuenta asíncrona empleando evento como señal de disparo en el contador (izda.), y b) usando autómata finito de tipo Mealy para generar evento síncrono (dcha.).

Otra aplicación importante de los contadores en el control secuencial, es la temporización de acciones. En este caso el contador actúa como un *temporizador* (*timer*) para mantener activa una señal durante un tiempo determinado (*desactivación retardada*) o para activarla transcurrido un tiempo (*activación retardada*).

La figura 8.31 presenta una posible implementación con Logisim para simulación lógica de temporizadores de retardo a la (des)activación de señales de salida. Los circuitos constan de un contador disparado por una señal de reloj de 1 Hz (*tick* de simulación de 2 Hz) con ajuste de su valor máximo al número de segundos deseado para el retardo. En este caso se especifica el comportamiento de los contadores con enclavamiento en el final de cuenta para que una vez alcanzado el valor máximo permanezcan en dicho valor hasta que se cumpla la condición de reinicio del contador.

El enclavamiento mencionado se activa mediante el bit *TC* (*terminal count*) del contador. Este comportamiento se obtiene de modo inmediato en los elementos *Counter* incluidos en la librería *Memory*

de Logisim, mediante ajuste del atributo *'Action On Overflow'* (acción al desbordamiento) al valor *'Stay at value'* (mantener el valor).

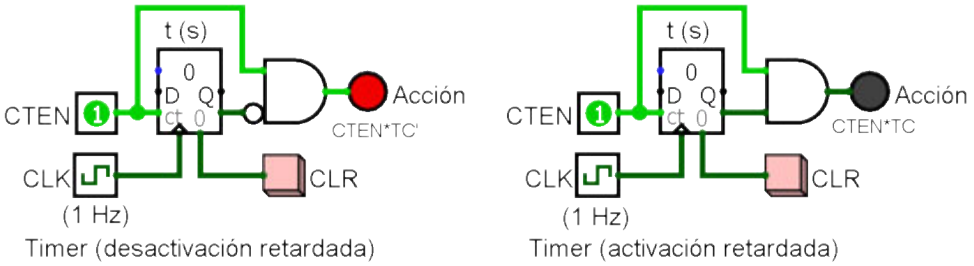


Figura 8.31: Circuitos lógicos de (des)activación retardada mediante temporizador.

Para diseñar un contador con enclavamiento en fin de cuenta se debe determinar la condición que activa la señal *TC*. Esta condición se obtiene habitualmente mediante decodificación del valor final de cuenta, con una puerta AND. El enclavamiento del contador se realiza mediante la inhibición de los biestables al activarse *TC* y su rehabilitación con la activación de la señal de reinicio del contador. La figura 8.32 muestra un ejemplo de implementación de este comportamiento para el contador síncrono de décadas.

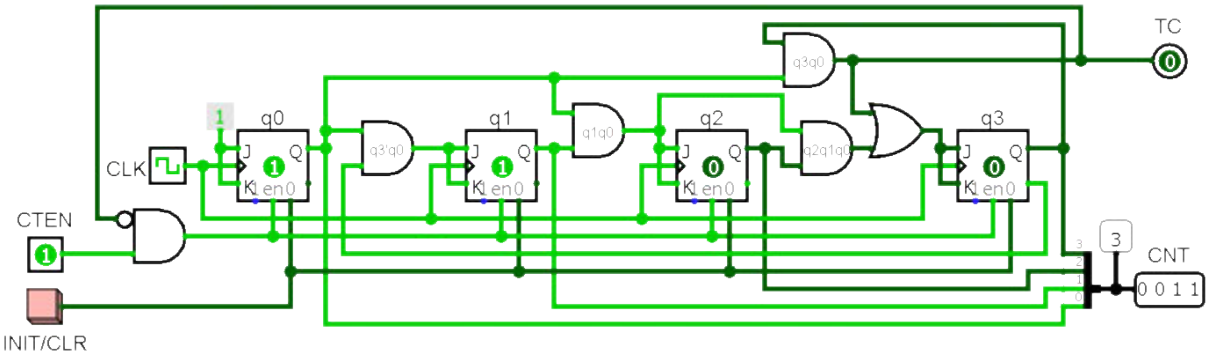


Figura 8.32: Circuito lógico de contador de décadas con enclavamiento en fin de cuenta.

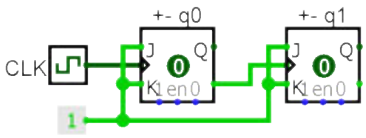
Conceptos clave

- ☑ Funcionamiento de registros de almacenamiento de E/S en paralelo.
- ☑ Funcionamiento de registros de desplazamiento de *n* etapas.
- ☑ Módulo de un contador y caracterización de su secuencia.
- ☑ Diferenciación entre contadores síncronos y asíncronos.
- ☑ Cálculo de la excitación de los biestables en contadores síncronos.
- ☑ Truncamiento asíncrono y síncrono de contadores.
- ☑ Contadores basados en registros de desplazamiento: contador en anillo y contador Johnson.
- ☑ Aplicaciones de contadores: activación secuencial de sistemas, contadores de eventos discretos, y temporizadores.

Problemas propuestos

Problema 8.1 Dado el circuito de la figura adjunta, contesta razonadamente las cuestiones siguientes:

1.- ¿Es un circuito síncrono o asíncrono?
2.- ¿Qué tipo de biestable equivalente se emplea?
3.- ¿Qué tipo de sincronización se emplea en los biestables, si es que emplea algún tipo?
4.- ¿Qué tipo de comportamiento se obtiene en dicho circuito?
5.- Realiza el cronograma de las señales q_1 y q_0, si el estado inicial es $q_1 = 1$, $q_0 = 0$. Comprueba que el comportamiento es el esperado y simula el circuito con Logisim.



Problema 8.2 Dado un contador asíncrono ascendente que cuenta de 0 a $11_{(10}$ implementado con biestables sincronizados por flanco de subida, contesta razonadamente las cuestiones siguientes:

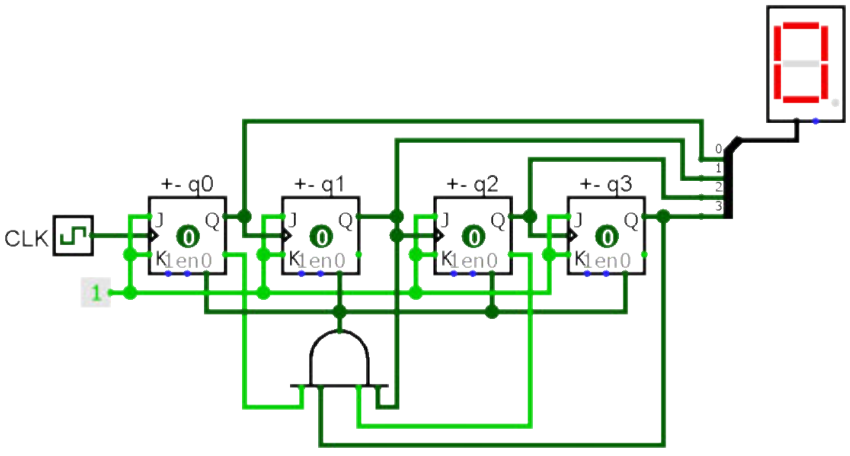
1.- ¿Cuántos estados tiene el contador?
2.- ¿Cuántos biestables son precisos para su implementación?
3.- Realiza una implementación del circuito para simulación lógica con Logisim.

Problema 8.3 Dado un contador asíncrono descendente que cuenta de 15 a 4 implementado con biestables sincronizados por flanco de subida, contesta razonadamente a las cuestiones siguientes:

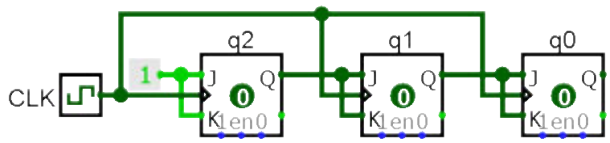
1.- ¿Cuántos estados tiene el contador?
2.- ¿Cuántos biestables se requieren para su implementación?
3.- Realiza una implementación del circuito para su simulación con Logisim.

Problema 8.4 Dado el circuito, con biestables sincronizados por flanco de bajada de la figura adjunta, contesta razonadamente a las cuestiones siguientes:

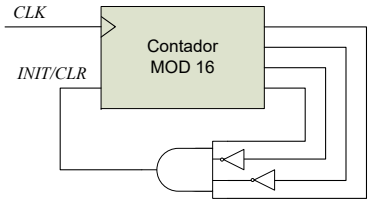
1.- ¿Es un sistema síncrono o asíncrono?
2.- ¿Qué tipo de biestable equivalente se emplea?
3.- ¿Qué tipo de sincronización se emplea en los biestables, si es que emplea algún tipo?
4.- ¿Describe la función de la puerta AND?
5.- ¿Qué tipo de comportamiento se obtiene en dicho circuito?
6.- ¿Cuál es la salida del circuito si en la puerta AND se eliminan las entradas q_2' y q_0'? Comprueba el comportamiento mediante simulación con Logisim.



Problema 8.5 ¿Cuál es la secuencia de salida del circuito de la figura adjunta si inicialmente los biestables son inicializados a 0? ¿Y si todos los biestables se inicializan a 1? Comprueba el resultado mediante simulación con Logisim.

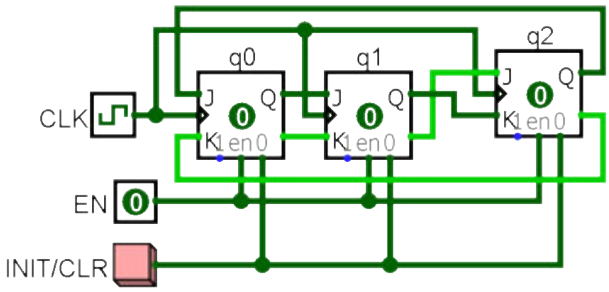


Problema 8.6 Describe razonadamente la funcionalidad del sistema secuencial correspondiente al diagrama de bloques de la figura adjunta.



Problema 8.7 Dado el circuito de la figura adjunta, contesta razonadamente a las cuestiones siguientes:

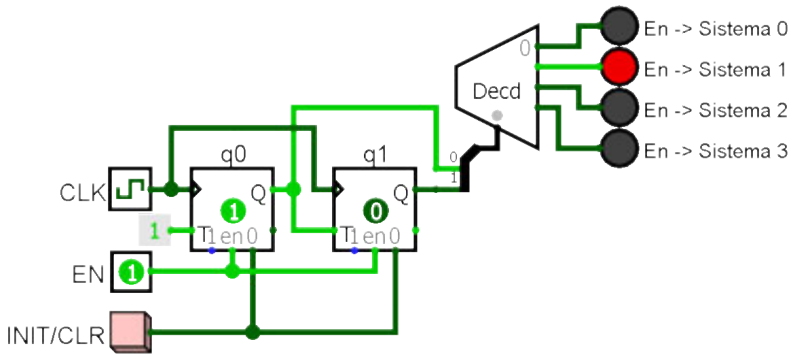
1.- Determina la secuencia de salida del circuito si se parte del estado inicial obtenido mediante activación de la señal INIT/CLR. ¿Se puede considerar dicho circuito un contador?
2.- Dibuja el cronograma de las señales q_0, q_1 y q_2. Comprueba el comportamiento del circuito empleando Logisim.



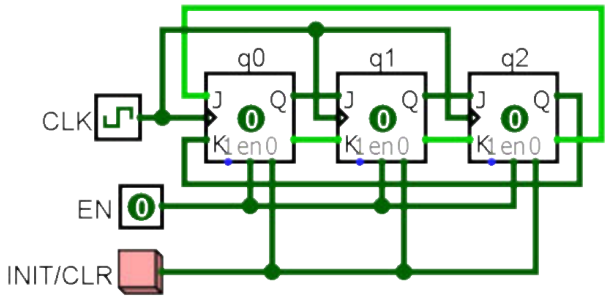
Problema 8.8 Calcula las funciones de excitación de los biestables T de un contador síncrono de décadas (BCD) para que su reinicio sea síncrono. Realiza una implementación para simular su comportamiento lógico con Logisim empleando biestables y las puertas lógicas necesarias.

Problema 8.9 Responde razonadamente: ¿Qué tipo de biestable se emplea habitualmente para implementar un registro de desplazamiento? ¿Y para un contador? ¿El tipo de biestable depende en algún caso del tipo de contador?

Problema 8.10 Explica razonadamente el funcionamiento del circuito proporcionado en la figura adjunta. Proporciona una implementación alternativa mediante el uso de un registro de desplazamiento.



Problema 8.11 Para el circuito de la figura adjunta, determina la secuencia de salida si se parte del estado inicial obtenido mediante la activación de la señal INIT/CLR. ¿Se puede considerar dicho circuito un contador? Comprueba el comportamiento lógico del circuito empleando Logisim.



Problema 8.12 Cuantos biestables y estados tienen los contadores de 5 bits que se señalan a continuación: a) Johnson, b) En anillo, c) Síncrono decreciente MOD 25, y d) Síncrono ascendente completo.

Problema 8.13 Describe detalladamente cómo se implementa un contador en anillo de 3 bits. ¿Cómo se puede implementar un contador en anillo de 3 bits en el que dependiendo de una señal de control H'/C se genere un tipo de secuencia determinado: *one-hot encoding* $[001, 010, 100]$ o *one-cold encoding* $[110, 101, 011]$? Realiza una implementación para simulación lógica con Logisim.

Problema 8.14 Implementa con Logisim los contadores de 4 bits: (A) en anillo con *one-hot enconding*, y (B) contador Johnson. Utiliza como elemento base el registro de desplazamiento (*shift register*) de la biblioteca *memory*.

9. Sistemas secuenciales síncronos

Las funciones de los *sistemas de lógica combinacional* se implementan mediante circuitos constituidos por puertas lógicas sin realimentación. La salida de dichos circuitos cambia en cuanto su entrada sufre una modificación. Por tanto, en los circuitos de lógica combinacional, la salida es el resultado de aplicar una función lógica a las señales de entrada (ver figura 9.1 izda.). Por el contrario, en los *sistemas de lógica secuencial*, el cambio en la salida depende tanto del valor instantáneo de las entradas como de su historia previa registrada en el estado almacenado en sus elementos de memoria (ver figura 9.1 dcha.).

La diferencia entre los sistemas combinacionales y secuenciales condiciona el tipo de formalismo matemático para modelar su comportamiento y abordar tanto su análisis como su diseño. En este capítulo se aborda el estudio práctico de los métodos para realizar el análisis y el diseño de los *sistemas secuenciales síncronos*.

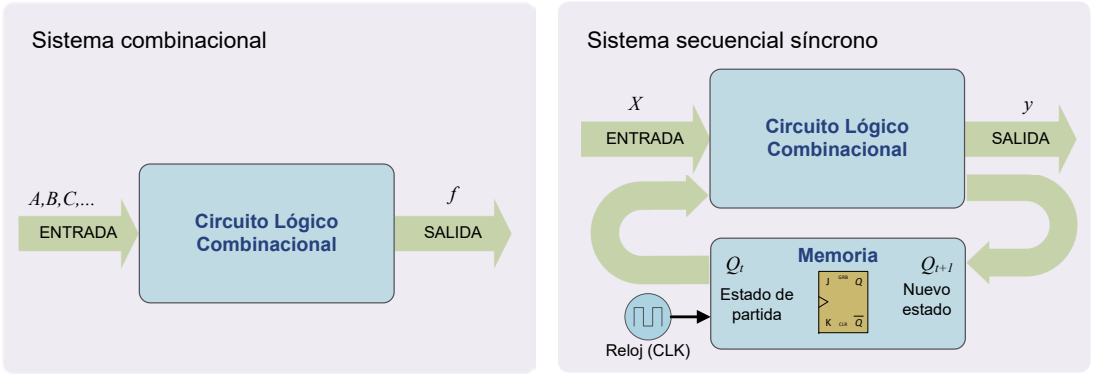


Figura 9.1: Diagramas de los sistemas de lógica combinacional (izda.) y secuencial (dcha.).

9.1 Sistemas asíncronos vs. síncronos

Dependiendo del instante en que se produce el cambio en la salida de los sistemas secuenciales, estos se pueden clasificar en:

- **Asíncronos o dirigidos por eventos**. Su salida y su estado reflejan de modo inmediato los cambios que se producen en las señales de entrada. Su diseño es complejo por la dificultad para establecer la coordinación entre subsistemas que funcionan de modo concurrente. Para el modelado de este tipo de sistemas se emplean herramientas como las *Redes de Petri*.

- **Síncronos**. Su salida y su estado reflejan los cambios que se producen en las señales de entrada en instantes determinados por una señal de sincronización o reloj (*CLK*). Para el modelado de este tipo de sistemas se emplean los *Autómatas de Estados Finitos* o *Autómatas Finitos* (*Finite-State Machine*, FSM).

9.2 Estados de un sistema secuencial

En el análisis y diseño de sistemas lógicos secuenciales es fundamental el concepto de *estado* pues es el responsable de sus diferencias con los sistemas combinacionales.

> **Definición 9.1 — Estado de un sistema secuencial.** El estado de un sistema secuencial es el valor instantáneo almacenado en los elementos de memoria (biestables) del sistema. De un modo más conceptual, cada estado del sistema está relacionado con las condiciones que satisface el funcionamiento del sistema en un momento dado. ■

El número máximo de estados posibles para un sistema secuencial queda determinado por el número de combinaciones aceptables de los valores almacenados en sus biestables. Para obtener dicho número se tiene en cuenta que cada biestable puede almacenar un bit de información. Por tanto, si el número de biestables del sistema es n, entonces la cantidad de estados posibles para el mismo es 2^n. Este es el número máximo de combinaciones posibles con n bits de información. La figura 9.2 presenta de un modo intuitivo los valores posibles del estado obtenidos a partir de la información almacenada en cada biestable (p. ej., de tipo J-K) cuando el sistema secuencial se simula con Logisim.

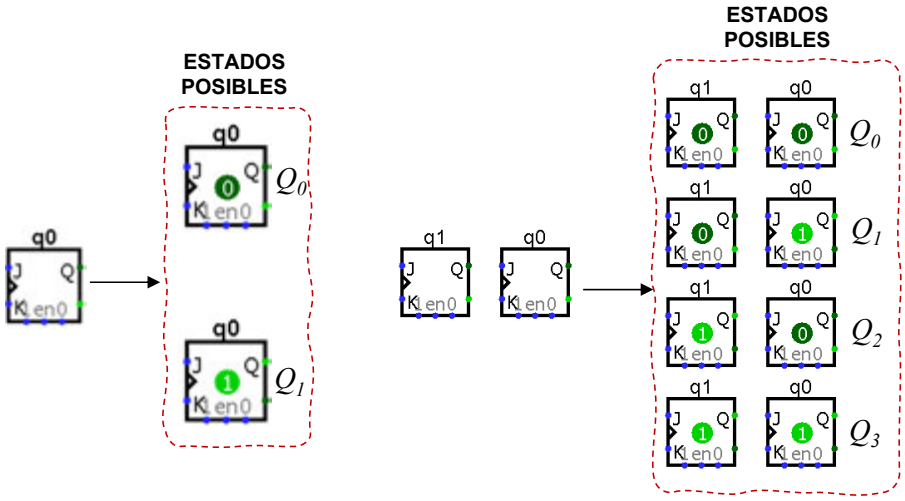


Figura 9.2: Estados posibles en un sistema secuencial cuando está compuesto por un biestable (izda.) y cuando está compuesto por dos biestables (dcha.).

A cada estado del sistema se asocia una variable multibit, en la que cada bit corresponde al estado individual de un biestable. La denominación del estado del sistema es arbitraria, pero resulta apropiado que la etiqueta asociada facilite la identificación de la codificación asociada al estado. En esta obra se emplea la denominación Q_i, con $i = 0, 1, \ldots, (p-1)$, siendo p el número total de estados del sistema, e i el valor decimal correspondiente al valor binario almacenado en los biestables. Con esta codificación, cada estado Q_i está compuesto por una combinación de n bits q_j, con $j = 0, 1, \ldots, (n-1)$, donde el valor de i corresponde a la representación en binario natural de los valores q_j que codifican el estado Q_i (ver figura 9.2). De este modo el subíndice i identifica al estado y j al biestable.

> **Ejemplo 9.1 — Codificación de estados en binario natural.** Se tiene un sistema cuyo estado Q_i está almacenado en 3 biestables. De este modo, empleando una codificación en binario natural, se tiene que $Q_i = [q_2\, q_1\, q_0]_i$ para todo estado del sistema. Indica el valor de cada uno de los biestables para el estado Q_5.

Solución:

El código binario correspondiente al subíndice del estado Q_i ofrece el valor binario de cada uno de los biestables que componen el estado. De este modo, cuando $i = 5$, el código binario asociado de tres bits es 101. Por tanto, el valor de los biestables en el estado Q_5 es: $[q_2\, q_1\, q_0] = [1\,0\,1]$. Esto es: $q_2 = 1$, $q_1 = 0$, $q_0 = 1$. ■

> Recuerda que el procedimiento para nombrar a las variables lógicas que intervienen en el análisis y diseño de sistemas lógicos combinacionales y secuenciales son meras convenciones. En consecuencia, es posible encontrar distintas obras de consulta en las que se siguen criterios diferentes a los expuestos en este texto. Por ejemplo, empleando la letra Z con subíndices para el estado, o las letras del abecedario (A,B,C,...).

9.3 Autómatas finitos

La evolución temporal de los sistemas secuenciales síncronos admite un formalismo matemático mediante un modelo denominado *Autómata de Estados Finitos* o *Máquina de Estados Finitos*, o simplemente *Autómata Finito* (FSM, *Finite State Machine*). Un autómata finito consta de:

- Un conjunto X de *entradas* al sistema.
- Un conjunto y de *salidas* generadas desde el sistema.
- Un conjunto Q de p *estados* entre los que puede evolucionar el sistema, con $Q = \{Q_0, Q_1, ...Q_{p-1}\}$.
- Una *función de transferencia o de transición de estado*, tal que

  $$\delta : Q \times X \rightarrow Q$$

  Esto es, en todo instante de sincronismo t_s, que desde el estado de partida Q_t con una entrada X aplicada al sistema, este evoluciona a un estado Q_{t+1}. Esto es: $Q_t \xrightarrow{t_s} Q_{t+1}$
- Una *función de salida*, tal que

  $$\lambda : Q \times X \rightarrow y$$

  Es decir, para un estado del sistema Q y una entrada X, se obtiene la salida y en el sistema. La salida y es una función combinacional que cambia de valor en cuanto lo hace el estado Q o la entrada X al sistema.

Dependiendo del tipo de autómata modelado y su implementación para un sistema secuencial síncrono, se puede hablar de dos modelos posibles: (1) *autómata de Moore* y (2) *autómata de Mealy*.

9.3.1 Autómata de Moore

Debe su nombre al profesor de matemáticas y ciencias de la computación estadounidense Edward F. Moore (1925-2003).[1] Los autómatas de Moore se caracterizan porque el efecto de la entrada siempre se transfiere a la salida del sistema a través del estado del mismo. Dicho con otras palabras, *el sistema se modela de forma que la salida del sistema dependa exclusivamente del estado del sistema*: $y = \lambda(Q)$

La figura 9.3 muestra el diagrama de bloques de un sistema implementado como un autómata de Moore. En dicho diagrama se observa como existe un sistema secuencial (con memoria) interpuesto entre la entrada (X) y la salida (y). Por tanto, la entrada no afecta directamente a la salida del sistema.

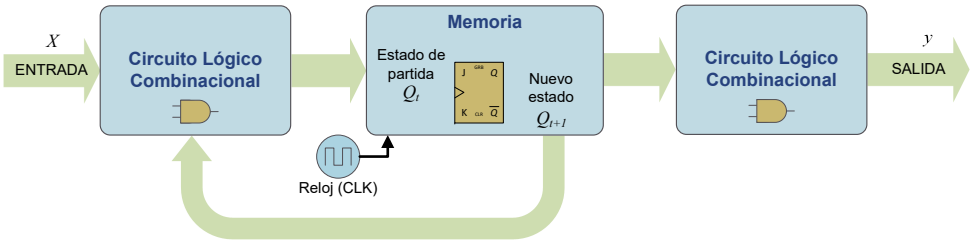


Figura 9.3: Diagrama de bloques de un autómata de Moore.

[1] Trabajó junto a Claude E. Shannon en un trabajo sobre *Teoría de la Computabilidad*.

En los autómatas de Moore, aunque la entrada cambie en cualquier instante (asíncrona), su efecto en la salida se produce en instantes concretos determinados por la sincronización del estado del sistema. En consecuencia, en cada instante de sincronismo, con el sistema en el estado Q_t para una entrada X, se obtiene un nuevo estado Q_{t+1}. En el nuevo estado del sistema se calcula inmediatamente —de modo combinacional— el valor de la salida.

9.3.2 Autómata de Mealy

Su origen se debe al matemático estadounidense George H. Mealy (1927-2010) que ejerció como profesor en Harvard. A diferencia de los autómatas de Moore, el efecto de la entrada es instantáneo en la salida del sistema. La salida es el resultado de una función combinacional del estado del sistema y la entrada al mismo (ver figura 9.4). Este hecho provoca que la salida del sistema pueda cambiar en cualquier instante, independientemente del cambio de estado del sistema. Por tanto, ante una entrada asíncrona, la salida sería también asíncrona.

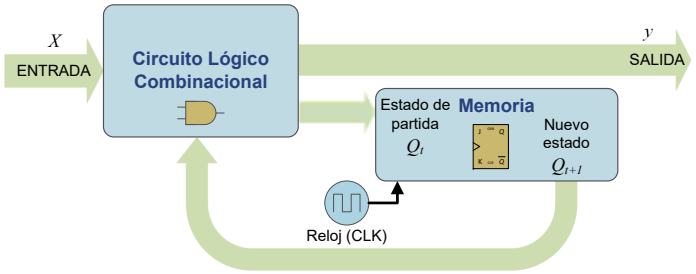


Figura 9.4: Diagrama de bloques de autómata de Mealy.

Para evitar que el sistema en su conjunto deje de ser síncrono, es preciso incorporar sincronismo bien a su entrada o bien a su salida. Por ejemplo, mediante el empleo de un registro de almacenamiento. De este modo se garantiza que la salida mantenga su valor al menos durante un ciclo de reloj. Esto es, hasta la actualización del registro de salida en el siguiente flanco de sincronismo.

9.4 Diagrama de transición de estado

El *diagrama de transición de estado* (DTE) de un sistema secuencial es una representación gráfica mediante un grafo orientado que describe el comportamiento del sistema mediante un autómata finito. Esta herramienta resulta muy útil tanto para el diseño como para el análisis de sistemas secuenciales. Un DTE se compone de distintos los elementos que se muestran en la figura 9.5:

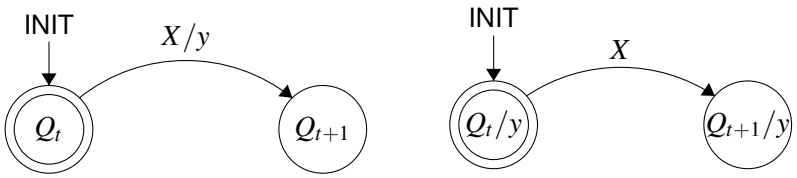
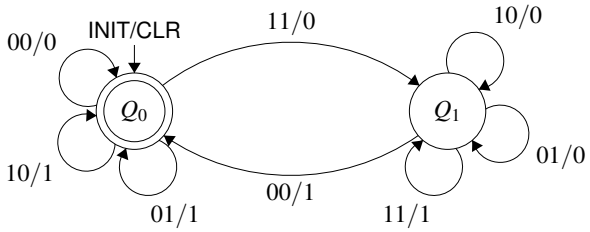


Figura 9.5: Elementos del diagrama de estados en los autómatas de Mealy (izda.) y Moore (dcha.).

- *Estados*. Se representan por círculos o elipses asociados a cada uno de los estados (Q_i) o condiciones de funcionamiento del sistema. Dentro del círculo se añade una etiqueta que identifica al estado. En el DTE de los autómatas de Moore, junto a la etiqueta del estado se indica el valor de la salida (y) para dicho estado, ya que en estos autómatas la salida depende únicamente del estado. Cuando la salida y el estado coinciden, como sucede en los contadores, la salida se omite. El estado inicial del sistema se señala habitualmente con un doble círculo. Una flecha de entrada con la etiqueta de inicio (INIT) marca el estado forzado en el reinicio del sistema.
- *Transiciones*. Son arcos orientados que parten de un estado y llegan a otro que incluso puede ser el mismo estado de partida. Representan la transición del estado en cada instante de sincronismo para una entrada (X). Están etiquetados con el valor de la entrada (X) que provoca la transición desde

el estado de partida hasta el estado alcanzado un instante posterior. Además, en los autómatas de Mealy, junto a la etiqueta de la entrada, se indica el valor de la salida del sistema (y), ya que esta se calcula de modo combinacional ante cualquier cambio de la entrada o del estado.

Ejemplo 9.2 — DTE de un autómata de Mealy. Se tiene el diagrama de transición de estado de la figura adjunta correspondiente a un sumador serie de un bit, con acarreo. Analiza el DTE proporcionado.

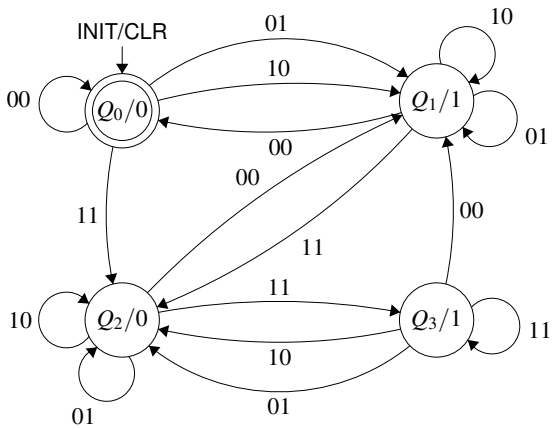


Solución:

El DTE de la figura adjunta corresponde a un autómata de Mealy, ya que el valor de la entrada y la salida del sistema aparecen en sus arcos o transiciones. La entrada al sistema está constituida por los dos bits que se van sumando y la salida es el resultado de dicha suma.

Como el resultado de sumas previas puede producir acarreo, es preciso almacenar el acarreo previo para obtener la suma total, teniendo en cuenta no solo los bits de entrada, sino también el posible acarreo. En este caso existen dos estados: Q_1 y Q_0 correspondientes a la existencia o ausencia de acarreo previo, respectivamente. Se parte del estado sin acarreo (Q_0) y en este caso, solo una entrada 11 de dos bits a uno produce un cambio de estado por el acarreo generado (Q_1). La vuelta al estado inicial, sin acarreo, se produce con entrada 00. ■

Ejemplo 9.3 — DTE de un autómata de Moore. Se tiene el diagrama de transición de estado de la figura adjunta correspondiente a un sumador serie de un bit con acarreo. Analiza el DTE proporcionado.



Solución:

El DTE de la figura adjunta corresponde a un autómata de Moore, ya que la salida del sistema se indica en cada estado y las transiciones solo están etiquetadas con la entrada que las provoca. Al igual que en el ejemplo anterior, la salida del sistema es el resultado de la suma, pero en este caso existen más estados pues las condiciones que pueden aparecer son: suma 0 con/sin acarreo generado y suma 1 con/sin acarreo generado. Por tanto, los estados correspondientes son:

Q_0 : suma 0 sin acarreo
Q_1 : suma 1 sin acarreo
Q_2 : suma 0 con acarreo
Q_3 : suma 1 con acarreo

■

Al igual que para los ejemplos previos, cualquier sistema secuencial se puede implementar mediante un autómata de Moore o uno equivalente de Mealy. La idea subyacente es que para el autómata de Mealy es posible adelantar la salida del sistema a la transición, ya que si se conoce el estado y la entrada, es posible determinar tanto el estado al que evolucionará el sistema como su salida. De este modo se pueden eliminar estados porque la salida se obtiene antes de llegar al estado. Además, es posible simplificar estados con idéntica salida. Por tanto, se pueden extraer las conclusiones siguientes:

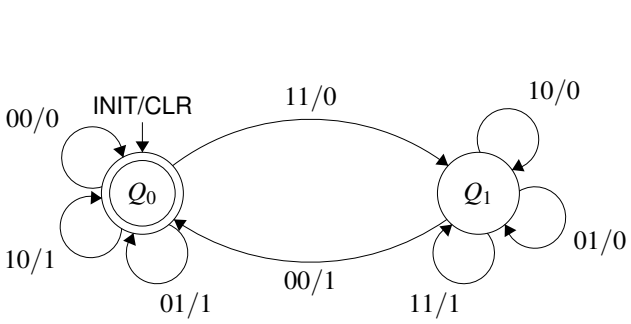
- Todo sistema secuencial admite un modelado alternativo con dos autómatas equivalentes, uno de Moore y otro de Mealy. De hecho, la diferencia entre utilizar uno u otro, no radica en la naturaleza del sistema sino en la decisión de diseño para su implementación.
- Para modelar un sistema dado, un autómata de Mealy precisa menos estados que equivalente de Moore.
- Para sincronizar la salida correspondiente a un autómata de Mealy se puede emplear un registro.
- En un autómata de Moore el efecto de las entradas en la salida del sistema se produce a través de su cambio de estado. Por tanto, el efecto de la entrada en la salida del sistema puede estar retrasado hasta que se actualiza el estado del que depende la salida.

> Recuerda, *«los sistemas secuenciales no son de Moore o de Mealy»*, es su modelado o implementación el que emplea un autómata de Moore o de Mealy, siendo esta una decisión de diseño, no una característica propia del sistema.

9.5 Tabla de transición de estado y tabla de salida

La *tabla de transición de estado* es una representación equivalente al DTE que describe el comportamiento del sistema y cómo afectan las entradas tanto en su estado como en su salida. La equivalencia entre el DTE y la tabla de transición de estado hace posible que el paso de uno a otro sea sistemático.

La figura 9.6 muestra la tabla de transición de estado para un sistema modelado como un autómata de Mealy cuyo DTE se adjunta en la misma figura (izda). La tabla de transición indica el estado resultante (Q_{t+1}) y la salida del sistema (y) desde un estado de partida (Q_t) en el instante de sincronismo t_s para una entrada determinada ($X = [X_1\, X_0]$).

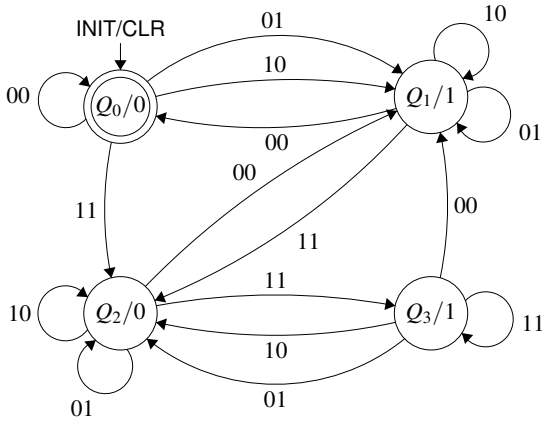


t				t+1		
Q	q_0	X_1	X_0	Q	q_0	y
Q_0	0	0	0	Q_0	0	0
Q_0	0	0	1	Q_0	0	1
Q_0	0	1	0	Q_0	0	1
Q_0	0	1	1	Q_1	1	0
Q_1	1	0	0	Q_0	0	1
Q_1	1	0	1	Q_1	1	0
Q_1	1	1	0	Q_1	1	0
Q_1	1	1	1	Q_1	1	1

Figura 9.6: DTE de un autómata de Mealy y tabla de transición correspondiente.

En la tabla que muestra el ejemplo se muestra el criterio adoptado en esta obra, colocando primero el estado y después todas las posibles entradas para el mismo. De este modo se facilita su construcción desde el DTE y la comprobación de la equivalencia entre este y la tabla de transición de estados. En este ejemplo se requiere solo un biestable (q_0) en el sistema, ya que posee dos estados (Q_0, Q_1) que se pueden codificar con el biestable mencionado, de modo que: $Q_0 \Rightarrow q_0 = 0, \quad Q_1 \Rightarrow q_0 = 1$

Para un autómata de Moore, en el que la salida depende solo del estado, la columna correspondiente a la salida repetiría los valores para el mismo estado de partida. Por esta razón es conveniente separar los valores de la salida en una tabla auxiliar denominada *tabla de salida* como se muestra en la figura 9.7.



t					$t+1$		
Q	q_1	q_0	X_1	X_0	Q	q_1	q_0
Q_0	0	0	0	0	Q_0	0	0
Q_0	0	0	0	1	Q_1	0	1
Q_0	0	0	1	0	Q_1	0	1
Q_0	0	0	1	1	Q_2	1	0
Q_1	0	1	0	0	Q_0	0	0
Q_1	0	1	0	1	Q_1	0	1
Q_1	0	1	1	0	Q_1	0	1
Q_1	0	1	1	1	Q_2	1	0
Q_2	1	0	0	0	Q_1	0	1
Q_2	1	0	0	1	Q_2	1	0
Q_2	1	0	1	0	Q_2	1	0
Q_2	1	0	1	1	Q_3	1	1
Q_3	1	1	0	0	Q_1	0	1
Q_3	1	1	0	1	Q_2	1	0
Q_3	1	1	1	0	Q_2	1	0
Q_3	1	1	1	1	Q_3	1	1

Q	q_1	q_0	y
Q_0	0	0	0
Q_1	0	1	1
Q_2	1	0	0
Q_3	1	1	1

Figura 9.7: DTE de un autómata de Moore junto a las tablas correspondientes de transición y salida.

En el ejemplo de la figura 9.7, se muestra como el sistema evoluciona entre 4 estados. Por tanto, para codificar los cuatro estados de este sistema son necesarios dos biestables (q_1, q_0).

9.6 Método de análisis de circuitos secuenciales

El objetivo del análisis de circuitos lógicos secuenciales es describir el comportamiento del sistema implementado por un circuito dado. Con dicho propósito se debe construir la tabla de transición de estado y el DTE correspondientes al circuito proporcionado. Los pasos a seguir son los siguientes:

1. **Obtención de funciones de excitación de los biestables**. Consiste en determinar las expresiones lógicas de las variables lógicas de entrada a los biestables. Dichas funciones pueden depender tanto de la entrada (X) al circuito, como del estado de partida Q_t en el instante de sincronismo. Por tanto, para su obtención se recorre el circuito desde cada entrada a los biestables hasta llegar a las entradas del circuito y/o las salidas de los biestables. Estas funciones determinan cómo cambia el estado del sistema. Esto es, el estado alcanzado Q_{t+1} desde un estado de partida en Q_t en el instante de sincronismo t_s para una entrada dada X.

2. **Obtención de la función de salida**. Se obtiene recorriendo el circuito hacia atrás, desde su salida (y) hasta alcanzar la entrada (X) y/o la salida de los biestables (Q). Si en este recorrido solo se alcanzan las salidas de los biestables y no hay caminos directos a las entradas del circuito, la expresión de la salida (y) solo depende del estado del sistema (Q). En este caso, el circuito implementa un autómata de Moore. Pero si existe algún recorrido directo que une la salida (y) del circuito con la entrada al mismo (X) sin atravesar los biestables, la salida depende tanto del estado (Q) como de la entrada al circuito (X). En este caso la implementación del sistema corresponde a un autómata de Mealy.

3. **Construcción de la tabla de transición de estados**. La tabla está ordenada con todas las combinaciones posibles del estado del sistema (Q) y sus entradas (X). Para cada valor del estado Q_t de partida se decodifica el valor del estado individual de cada biestable q_j que queda reflejado en la tabla. Para cada una de estas combinaciones, la tabla se completa con los valores de excitación de cada biestable obtenidos a partir de sus expresiones lógicas derivadas de la inspección inicial del circuito (paso 1).

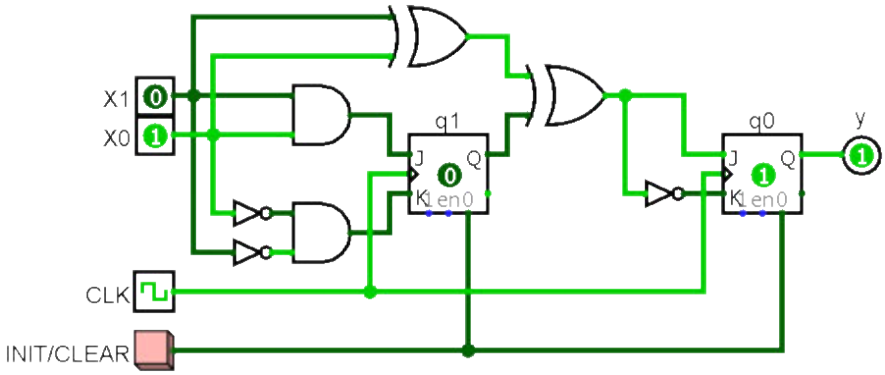
4. **Obtención de los estados de evolución del sistema**. En este paso, se obtiene el estado resultante de cada biestable q_j a partir de su estado de partida y el valor de la entrada al sistema, en el instante de sincronismo. De este modo se obtiene el estado resultante Q_{t+1} al que evoluciona el sistema en el instante de sincronismo desde un estado de partida Q_t para una determinada entrada X.

5. **Obtención del valor de la salida del sistema**. Consiste en calcular el valor de la salida (y) para cada estado de partida (Q_i) y cada entrada (X). La expresión lógica de la función de salida se deriva de la inspección inicial del circuito (paso 2). Si la función de salida tiene una expresión

de tipo $y = \lambda(Q,X)$ que depende tanto del estado como de la entrada, el circuito implementa un autómata de Mealy. En este caso, la salida se puede añadir como una columna al final de la tabla de transición de estado. Cuando la salida solo depende del estado con una expresión de tipo $y = \lambda(Q)$, la implementación corresponde a un autómata de Moore. En esta situación, es recomendable construir una tabla de salida independiente, ya que si la salida se añade a la tabla de transición todas las filas de la tabla correspondientes al mismo estado, tendrán idéntico valor de salida.

6. **Elaboración del DTE**. Este se puede dibujar trasladando la información de la tabla de transición de estado y la tabla de salida obtenidas en los pasos previos.

Ejemplo 9.4 — Análisis de un circuito lógico secuencial. En la figura adjunta se proporciona un circuito lógico con biestables J-K (disparados por flanco de subida). Analiza el circuito y determina las características del sistema al que corresponde.



Solución:

Este circuito corresponde a un sistema secuencial síncrono, ya que entre sus elementos hay dos biestables síncronos que comparten la señal de sincronismo *CLK*. Por simple inspección visual se puede afirmar que corresponde a la implementación de un autómata de Moore, puesto que la salida (y) no está conectada directamente con la entrada (X_1, X_0), sino únicamente con uno de los biestables (q_0). Dicho de otro modo, los cambios en la entrada se transfieren a la salida a través del estado del sistema.

A partir del circuito se obtienen tanto las expresiones lógicas de la excitación de los biestables (J_1, K_1, J_0, K_0) como de la salida (y), haciendo el recorrido desde dichas señales hacia la entrada (X) del circuito y/o las salidas de los biestables (q_1, q_0). De este modo se obtienen las expresiones algebraicas siguientes:

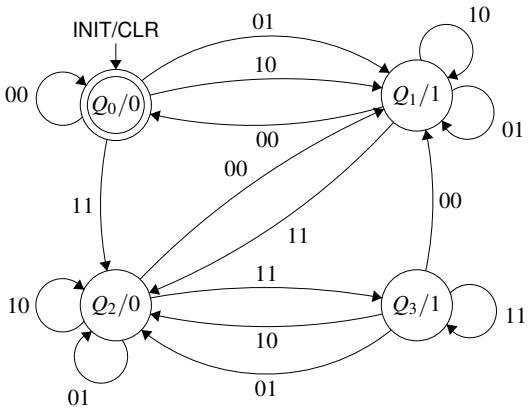
$$J_1 = X_1X_0 \qquad J_0 = (X_1 \oplus X_0) \oplus q_1 \qquad y = q_0$$
$$K_1 = X_1'X_0' \qquad K_0 = ((X_1 \oplus X_0) \oplus q_1)'$$

Con estas expresiones es posible elaborar una tabla de verdad cuyas entradas son el estado y las entradas al circuito (q_1, q_0, X_1, X_0), con salidas las excitaciones de los biestables (J_1, K_1, J_0, K_0). Una vez que se tiene el estado de partida para cada biestable q_j y el valor de sus entradas, es posible calcular el estado resultante al que evoluciona dicho biestable en el instante de sincronismo. De este modo se completa la tabla elaborada con el estado resultante de la evolución del estado de los biestables. La tabla de salida es la tabla de verdad correspondiente a la expresión de y como una función del estado. Ambas tablas se pueden trasladar al DTE para obtener una representación gráfica del comportamiento del sistema (ver figura adjunta).

La simulación con Logisim del circuito inicial es muy ilustrativa del comportamiento del sistema como un autómata. En la simulación se pueden cambiar a voluntad los valores de las entradas para observar cómo evoluciona el estado del mismo inspeccionando los valores de cada biestable.

	t								t+1		
Q	q_1	q_0	X_1	X_0	J_1	K_1	J_0	K_0	Q	q_1	q_0
Q_0	0	0	0	0	0	1	0	1	Q_0	0	0
Q_0	0	0	0	1	0	0	1	0	Q_1	0	1
Q_0	0	0	1	0	0	0	1	0	Q_1	0	1
Q_0	0	0	1	1	1	0	0	1	Q_2	1	0
Q_1	0	1	0	0	0	1	0	1	Q_0	0	0
Q_1	0	1	0	1	0	0	1	0	Q_1	0	1
Q_1	0	1	1	0	0	0	1	0	Q_1	0	1
Q_1	0	1	1	1	1	0	0	1	Q_2	1	0
Q_2	1	0	0	0	0	1	1	0	Q_1	0	1
Q_2	1	0	0	1	0	0	0	1	Q_2	1	0
Q_2	1	0	1	0	0	0	0	1	Q_2	1	0
Q_2	1	0	1	1	1	0	1	0	Q_3	1	1
Q_3	1	1	0	0	0	1	1	0	Q_1	0	1
Q_3	1	1	0	1	0	0	0	1	Q_2	1	0
Q_3	1	1	1	0	0	0	0	1	Q_2	1	0
Q_3	1	1	1	1	1	0	1	0	Q_3	1	1

	q_1	q_0	y
Q_0	0	0	0
Q_1	0	1	1
Q_2	1	0	0
Q_3	1	1	1



Observa que en el diagrama del circuito secuencial anterior, el orden de ponderación de los bits de estado $[q_1\, q_0]$ se ha colocado en el orden habitual (q_1 a la izda. y q_0 a la dcha.) para facilitar la identificación del estado mediante la inspección visual del circuito durante la simulación lógica con Logisim. ■

9.7 Diseño de sistemas secuenciales síncronos

Para diseñar un sistema secuencial síncrono, se parte de una especificación funcional o descripción textual del comportamiento deseado para el sistema. En este proceso, se utilizan las herramientas utilizadas en el análisis, pero —por así decirlo— en sentido inverso. El objetivo del diseño es implementar mediante un circuito lógico la funcionalidad que satisface la especificación proporcionada. Esto es, el circuito a diseñar debe implementar las funciones de excitación de los biestables del sistema y su función de salida. Los pasos a seguir en el diseño de sistemas secuenciales síncronos son los siguientes:

1. **Identificación de las entradas, salidas y estados del sistema**. Para llevar a cabo este paso, se tiene en cuenta que los estados representan condiciones sobre el comportamiento del sistema para determinar el valor de sus salidas. En esta fase se debe decidir la codificación de estados que facilite la implementación del sistema. Por ejemplo, intentando que la salida del sistema coincida con el estado o —al menos— simplifique la obtención de la función de salida.

2. **Transformación de la especificación en un DTE**. Una vez elegidas, las entradas, salidas y los estados, en el DTE se expresa el comportamiento del sistema de forma gráfica.

3. **Obtención de la tabla de transición de estados desde el DTE**. Para cada estado de partida Q_i en el instante de sincronismo (t) y cada posible entrada X, la tabla de transición indica el estado resultante al que evoluciona el sistema Q_i en un instante posterior $t+1$.

4. **Determinación de entradas a los biestables para obtener la transición de estado requerida**. Se calculan las entradas de los biestables elegidos J_j-K_j, D_j o T_j, para obtener las transiciones requeridas. Los valores se obtiene de las tablas de excitación de cada biestable (ver figura 7.17, pág. 197). Las entradas de los biestables especificados se añaden a la tabla de transición de estado. Cuando el valor requerido de la entrada de un biestable es indiferente (0/1), en la tabla se marca con un símbolo 'X' (*don't care*).

5. **Cálculo de la salida en cada transición de estado**. Dependiendo del tipo de autómata elegido para diseñar el sistema, la salida del mismo se añade en la tabla de transición (autómata de Mealy) o se crea una tabla independiente con la salida asociada a cada estado (autómata de Moore).

6. **Cálculo de funciones combinacionales para la excitación de los biestables**. Empleando la tabla de transición de estado como una tabla de verdad, se obtienen las funciones de excitación de cada biestable (J_j-K_j, D_j o T_j) dependiendo de su estado de partida q_j en el instante de sincronismo y

la entrada X al sistema. El método es similar al empleado para los sistemas combinacionales, de modo que se pueden obtener tanto las formas canónicas de las funciones, como sus expresiones simplificadas SOP y POS. La expresión más conveniente dependerá del tipo de implementación deseado para cada función combinacional.

7. **Cálculo de la función de salida**. Dependiendo del tipo de autómata elegido para modelar el comportamiento del sistema, la tabla de transición de estado se emplea para obtener la expresión de la salida (y) como una función del estado de partida Q y la entrada al sistema X, cuando se trata de un autómata de Mealy. En el caso del autómata de Moore la expresión de la salida (y) se obtiene desde la tabla de salida como una función del estado de partida Q.

8. **Implementación de las funciones calculadas**. La implementación consiste en la elaboración de los circuitos lógicos correspondientes a las expresiones de excitación de los biestables y la función de salida. Los circuitos se puede realizar cumpliendo requisitos en cuanto al tipo de puertas lógicas empleadas (p. ej., NAND de 2 entradas) o módulos combinacionales requeridos (p. ej., con MUX 4×1).

En las secciones siguientes se ilustrará el método de diseño sistemático de sistemas secuenciales síncronos, descrito anteriormente, aplicado a distintos tipos de problemas.

> Recuerda que debido a la naturaleza sistemática del proceso de diseño lógico de sistemas secuenciales síncronos descrito en los pasos anteriores, existen programas informáticos capaces de abordar el diseño a partir de su formalización mediante un autómata finito. Sin embargo, aunque Logisim permite la simulación lógica de circuitos secuenciales, no proporciona herramientas para el diseño automatizado de los mismos.

9.7.1 Diseño de un reconocedor de secuencia

En este tipo de problemas el sistema a diseñar debe reconocer un patrón concreto en una sucesión de bits de entrada. Dicho patrón puede indicar el comienzo o final de un mensaje, alertar de que está próxima a llegar una secuencia de control, etcétera. Cada bit entrante se sincroniza con la recepción de un pulso de la señal de reloj. A continuación, se ilustra la resolución de este tipo de problemas en un ejemplo práctico.

Ejemplo 9.5 — Diseño de un reconocedor de secuencia. Diseña un circuito lógico secuencial síncrono que reconozca la entrada serie de tres unos seguidos o más. El reconocimiento del patrón se indicará mediante la activación de un bit de salida.

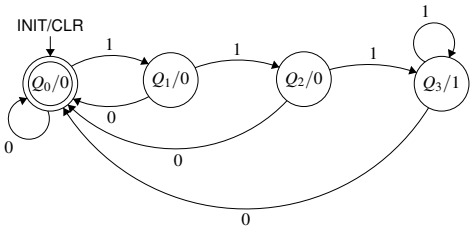
Solución:

Para reconocer el patrón se debe memorizar el valor de las entradas previas para saber si ha llegado la secuencia requerida compuesta por tres unos o más, consecutivos.

El sistema tiene una entrada serie por donde llegan uno a uno los bits en sucesión. La salida indica si se ha reconocido la llegada del al menos tres unos seguidos en la entrada. Los estados del sistema se pueden definir teniendo en cuenta las condiciones de funcionamiento:

- Q_0: El último bit entrante ha sido un 0.
- Q_1: El último bit entrante ha sido un 1.
- Q_2: Los últimos dos bits han sido 1.
- Q_3: Los últimos tres bits han sido 1.

Una vez identificada la entrada, la salida y los estados del sistema, es posible construir el DTE que describe el comportamiento del sistema. La figura adjunta muestra el DTE planteado como un autómata de Moore, ya que la salida del sistema solo depende de su estado. Dicha figura muestra también la tabla de transición en la que se indica la evolución del sistema desde un estado de partida Q dada una entrada X. Esta tabla se completa con el valor correspondiente de la excitación de biestables J-K requerida para conseguir la transición de estado deseada $Q_t \xrightarrow{t_s} Q_{t+1}$. Observa en la tabla mencionada como algunos de los valores son indiferentes y quedan marcados con un símbolo 'x'. La tabla de salida queda separada, ya que la salida solo depende del estado $Q = [q_1\, q_0]$.



	t			$t+1$			q_1		q_0	
Q	q_1	q_0	X	Q	q_1	q_0	J_1	K_1	J_0	K_0
Q_0	0	0	0	Q_0	0	0	0	x	0	x
Q_0	0	0	1	Q_1	0	1	0	x	1	x
Q_1	0	1	0	Q_0	0	0	0	x	x	1
Q_1	0	1	1	Q_2	1	0	1	x	x	1
Q_2	1	0	0	Q_0	0	0	x	1	0	x
Q_2	1	0	1	Q_3	1	1	x	0	1	x
Q_3	1	1	0	Q_0	0	0	x	1	x	1
Q_3	1	1	1	Q_3	1	1	x	0	x	0

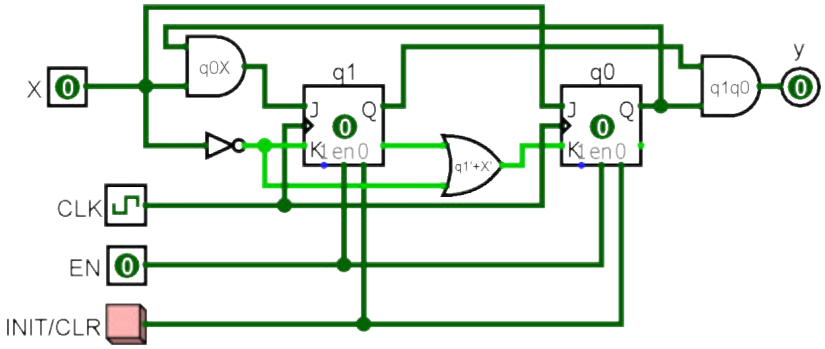
Q	q_1	q_0	y
Q_0	0	0	0
Q_1	0	1	0
Q_2	1	0	0
Q_3	1	1	1

A partir de la tabla de transición de estado y la tabla de salida se pueden obtener las expresiones booleanas de las funciones de excitación de los biestables y de la función de salida:

q1	q0	X	J1	K1	J0	K0
0	0	0	0	x	0	x
0	0	1	0	x	1	x
0	1	0	0	x	x	1
0	1	1	1	x	x	1
1	0	0	x	1	0	x
1	0	1	x	0	1	x
1	1	0	x	1	x	1
1	1	1	x	0	x	0

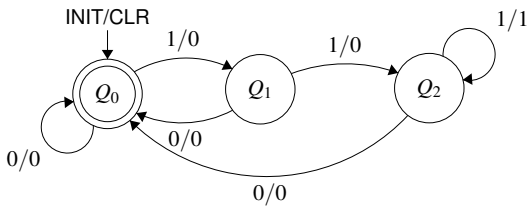
$$J_1 = q_0 X \qquad J_0 = X \qquad y = q_1 q_0$$
$$K_1 = X' \qquad K_0 = q_1' + X'$$

Las funciones de excitación anteriores se obtienen a partir de sus tablas de verdad extraídas de la tabla de transición de estado. Una vez obtenidas las funciones mencionadas se puede implementar el circuito correspondiente como se muestra en la figura adjunta:



El mismo ejemplo se puede modelar con un número menor de estados mediante un autómata de Mealy, ya que la salida se puede determinar en cuanto se ha reconocido la entrada de dos 1 consecutivos y la siguiente entrada es un 1. Las salidas se indican en las transiciones del DTE, puesto que la salida no solo depende del estado sino también de la entrada.

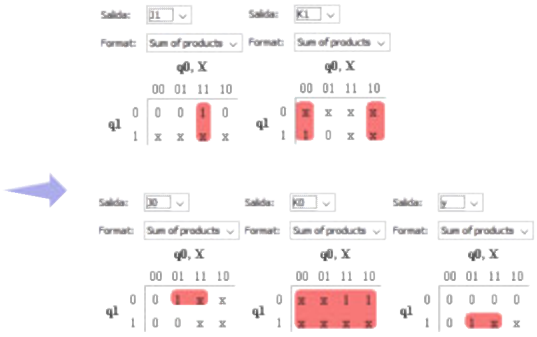
En este nuevo planteamiento se precisan únicamente tres estados que se pueden codificar mediante dos biestables, quedando un estado sin utilizar. Por este motivo en la tabla de transición de estado las filas correspondientes al estado no utilizado aparecen como condiciones libres (*don't care*).



	t			$t+1$			q_1		q_0		
Q	q_1	q_0	X	Q	q_1	q_0	J_1	K_1	J_0	K_0	y
Q_0	0	0	0	Q_0	0	0	0	x	0	x	0
Q_0	0	0	1	Q_1	0	1	0	x	1	x	0
Q_1	0	1	0	Q_0	0	0	0	x	x	1	0
Q_1	0	1	1	Q_2	1	0	1	x	x	1	0
Q_2	1	0	0	Q_0	0	0	x	1	0	x	0
Q_2	1	0	1	Q_2	1	0	x	0	0	x	1
Q_3	1	1	0	x	x	x	x	x	x	x	x
Q_3	1	1	1	x	x	x	x	x	x	x	x

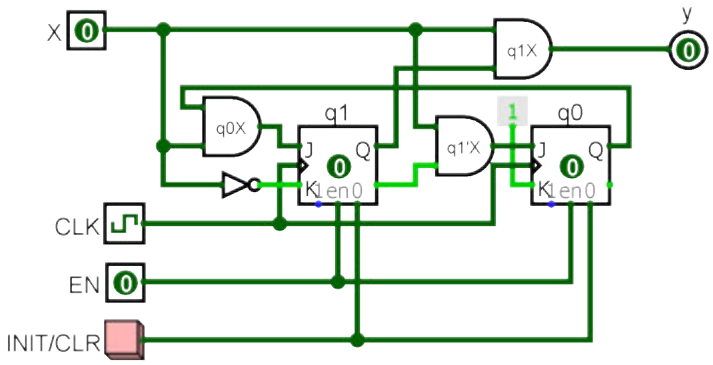
La resolución continúa con el cálculo de las funciones de excitación de biestables y la salida del sistema:

q1	q0	X	J1	K1	J0	K0	y
0	0	0	0	x	0	x	0
0	0	1	0	x	1	x	0
0	1	0	0	x	x	1	0
0	1	1	1	x	x	1	0
1	0	0	x	1	0	x	0
1	0	1	x	0	0	x	1
1	1	0	x	x	x	x	x
1	1	1	x	x	x	x	x



$$J_1 = q_0 X \qquad J_0 = q_1' X \qquad y = q_1 X$$
$$K_1 = X' \qquad K_0 = 1$$

Y el circuito resultante para simulación con Logisim:



En la inspección visual del circuito que implementa el autómata de Mealy para reconocer la secuencia, se observa que existe un camino que conecta directamente la entrada del circuito con la salida del mismo sin «atravesar» ninguno de los biestables del sistema. De este modo, un cambio en la entrada al sistema puede provocar un cambio inmediato en su salida. Dicho cambio tiene naturaleza asíncrona. Para evitar problemas es posible imponer sincronismo en la salida mediante empleo de un registro en dicha salida. En este caso concreto bastaría con un biestable D, ya que la salida tiene solo un bit. ■

> Recuerda que en los circuitos lógicos secuenciales diseñados para simulación lógica con Logisim es posible conocer el estado global del sistema mediante inspección visual del estado local de cada biestable. Para una correcta interpretación de los valores codificados hay que tener presente la ponderación de cada biestable y su ubicación física en el circuito, que no tiene por qué corresponder con el orden intuitivo en el que el bit más significativo se coloca a la izquierda.

■ **Ejercicio 9.1** Modifica los circuitos reconocedores de secuencia para simulación con Logisim, de modo que su entrada proceda de un registro de desplazamiento de 10 etapas. Realiza la carga en paralelo del registro con la secuencia de 10 bits que alimenta al circuito. ■

9.7.2 Diseño de contadores

Los contadores constituyen un tipo de módulo secuencial muy versátil. Son sistemas que generan una sucesión cíclica de valores correspondientes a una secuencia sin valores repetidos. Su modelado presenta grandes similitudes con un autómata de Moore y algunas particularidades que se señalan a continuación:

- Son sistemas sin entrada (X), salvo la señal de sincronismo (CLK) y las de control (p. ej., $INIT/CLR$, EN). El sistema evoluciona de un estado al siguiente con cada flanco de disparo de los biestables, repitiendo un ciclo de estados, sin necesidad de la intervención de una señal de entrada.

- La salida del sistema coincide con el estado. Esto es, $y = Q$. Por tanto, no es preciso aplicar ninguna función al estado de los biestables, ya que la salida se obtiene agrupando las salidas individuales de los biestables.

Cuando se trata de obtener un contador de secuencia ordenada completa o truncada, no es preciso recurrir a métodos de diseño de sistemas secuenciales, ya que este tipo de contadores responden a una estructura fácil de recordar. En el caso de contadores desordenados u ordenados de secuencia incompleta con saltos —no obtenidos mediante truncamiento—, se puede aplicar un método de diseño general consistente en los pasos siguientes:

1. **Obtención del DTE**. Consiste en un ciclo de estados coincidentes con los valores del contador. Se debe tener en cuenta que el sistema no tiene entradas, ya que la secuencia de estados se genera en sincronía con la señal de reloj (*CLK*).
2. **Determinación del número de biestables necesarios**. Su número debe ser suficiente para codificar los estados o valores del contador.
3. **Construcción de la tabla de transición de estado**. En la tabla se indica la evolución desde cada estado de partida Q_i en cada instante t_s de sincronismo al estado resultante un instante posterior teniendo en cuenta que la tabla debe estar ordenada respecto a la codificación en binario del estado de partida. De esta forma la tabla obtenida se emplea como tabla de verdad para obtener la excitación de los biestables en función del estado de partida.
4. **Inclusión de la excitación de biestables**. A la tabla construida en el paso anterior se añade el valor de las entradas a cada biestable según su tipo (J, K, T o D), de modo que el estado evolucione en cada instante de sincronismo según los valores especificados en la tabla de transición.
5. **Cálculo de las funciones de excitación de los biestables**. Estas funciones se obtienen a partir de las tablas de verdad correspondientes que aparecen en la tabla de transición de estado. Recuerda que se busca el valor de las funciones de excitación de cada biestable, dependientes de su estado de partida q_j en el instante de sincronismo.
6. **Implementación de las funciones de excitación**. Se emplean las expresiones calculadas en el paso previo y el método de implementación podrá ser similar al empleado con cualquier función lógica: con puertas lógicas elementales, con puertas lógicas universales (NAND o NOR), con módulos combinacionales MUX, con DEC, etcétera.

En las secciones siguientes se particulariza el método de diseño presentado según el tipo de contador deseado:

1. Contadores de secuencia completa desordenada, y
2. Contadores de secuencia incompleta.

Diseño de contadores de secuencia completa desordenada

Para ilustrar el método de diseño de contadores síncronos de secuencia completa desordenada se plantea la implementación de un contador de código Gray.[2] Este código tiene numerosas aplicaciones en los sistemas digitales. En este texto se introdujo en la sección 2.5.4 (pág. 23), en el caso de la minimización de funciones lógicas mediante K-maps.

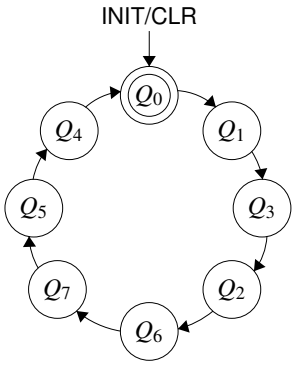
Ejemplo 9.6 — Diseño de un contador de código Gray de 3 bits. Realiza el diseño de un contador de código Gray de 3 bits.

Solución:

La figura adjunta muestra a la derecha la tabla de transición de estado del contador que se corresponde con el DTE de la izquierda. Es muy importante que la tabla de transición de estado se construya de

[2] El físico norteamericano Frank Gray (1887-1969) propuso esta codificación como fruto de su trabajo en los laboratorios Bell. Esta codificación también se conoce como *binaria reflejada* (RBC, *reflected binary code*) y tiene la particularidad de que entre un código y el siguiente solo cambia un bit.

modo ordenado respecto del estado de partida Q_i (en el instante de sincronismo). De este modo al completarla con los valores apropiados de la excitación (entrada) de cada biestable se obtienen las tablas de verdad para el cálculo de cada función de excitación.



	t		
Q	q_2	q_1	q_0
Q_0	0	0	0
Q_1	0	0	1
Q_2	0	1	0
Q_3	0	1	1
Q_4	1	0	0
Q_5	1	0	1
Q_6	1	1	0
Q_7	1	1	1

	$t+1$		
Q	q_2	q_1	q_0
Q_1	0	0	1
Q_3	0	1	1
Q_6	1	1	0
Q_2	0	1	0
Q_0	0	0	0
Q_4	1	0	0
Q_7	1	1	1
Q_5	1	0	1

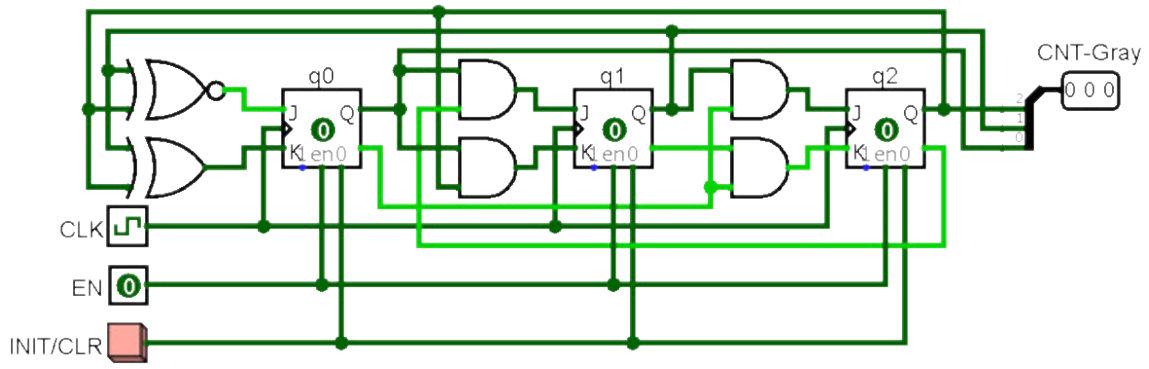
A continuación, se completa la tabla con la excitación apropiada de cada biestable para obtener la transición de estado requerida. Los valores añadidos dependerán del tipo de biestable empleado para codificar el estado. El resultado se muestra en la tabla siguiente:

	t			$t+1$				q_2		q_1		q_0	
Q	q_2	q_1	q_0	Q	q_2	q_1	q_0	J_2	K_2	J_1	K_1	J_0	K_0
Q_0	0	0	0	Q_1	0	0	1	0	x	0	x	1	x
Q_1	0	0	1	Q_3	0	1	1	0	x	1	x	x	0
Q_2	0	1	0	Q_6	1	1	0	1	x	x	0	0	x
Q_3	0	1	1	Q_2	0	1	0	0	x	x	0	x	1
Q_4	1	0	0	Q_0	0	0	0	x	1	0	x	0	x
Q_5	1	0	1	Q_4	1	0	0	x	0	0	x	x	1
Q_6	1	1	0	Q_7	1	1	1	x	0	x	0	1	x
Q_7	1	1	1	Q_5	1	0	1	x	0	x	1	x	0

Al completar la tabla con las excitaciones de los biestables, se obtiene la tabla de verdad para las funciones de excitación. Para cada función se busca la expresión (p. ej., canónica/simplificada, SOP y POS) más conveniente dependiendo del tipo de implementación que se desee. Las funciones obtenidas en este ejemplo son:

$$J_0 = (q_2 \oplus q_1)' \qquad J_1 = q_2' q_0 \qquad J_2 = q_1 q_0'$$
$$K_0 = q_2 \oplus q_1 \qquad K_1 = q_2 q_0 \qquad K_2 = q_1' q_0'$$

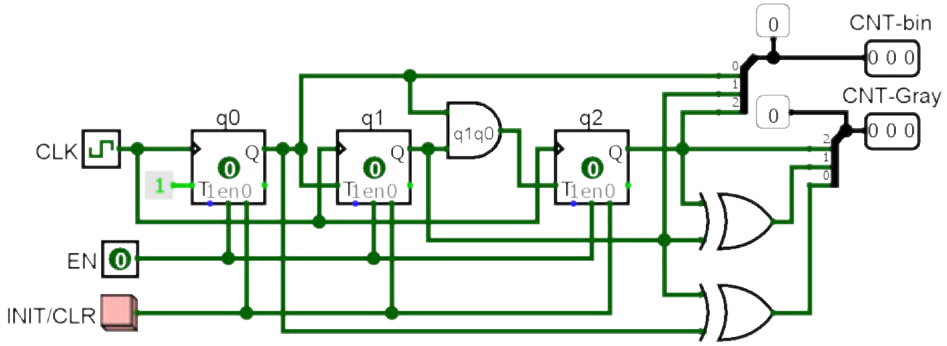
Cuyo circuito mediante puertas lógicas para simulación lógica con Logisim es:



Este ejemplo sirve para ilustrar el proceso de diseño de contadores de secuencia completa desordenada, aunque el caso concreto del contador Gray admite una solución alternativa más rápida partiendo del

contador ascendente ordenado MOD 8 para generar un contador convencional y tener en cuenta las ecuaciones lógicas para pasar de código binario $[B_2\,B_1\,B_0]$ a código Gray $[G_2\,G_1\,G_0]$:

$$G_2 = B_2$$
$$G_1 = B_2 \oplus B_1$$
$$G_0 = B_1 \oplus B_0$$



Diseño de contadores de secuencia incompleta

Los contadores de secuencia incompleta son aquellos que poseen un número de estados menor que el máximo permitido por los biestables disponibles. Cuando dicha secuencia está ordenada y no tiene saltos se dice también que su secuencia está truncada. El truncamiento de secuencia se puede conseguir mediante dos técnicas alternativas:[3]

- *Truncamiento asíncrono*. Se aplica tanto a contadores asíncronos como síncronos. Consiste en decodificar el valor de la secuencia que debe reiniciar inmediatamente el contador mediante la activación de la entrada *CLR* (borrado asíncrono) de los biestables. Por ejemplo, en un contador de décadas se decodifica el valor 10 para reiniciar la cuenta.
- *Truncamiento síncrono*. Se obtiene como parte del diseño de un contador síncrono. También se puede conseguir mediante activación de la señal de carga (*LD*) —en contadores con dicha posibilidad— para reiniciar el contador cuando se alcanza el valor de truncamiento.

Si la secuencia del contador es desordenada (p. ej., [0,2,1,3,5]) o es ordenada, pero tiene saltos (p. ej., [0,1,3,4]) entonces el contador se diseña mediante el método general descrito en la sección anterior. Los estados no utilizados se pueden considerar condiciones libres para obtener un diseño más simple, pero también es habitual forzar en dichos estados la evolución hacia el estado inicial, para evitar estados incongruentes del sistema.

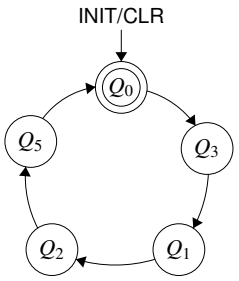
Ejemplo 9.7 — Diseño de contador de secuencia incompleta desordenada. Diseña un contador síncrono que genere la secuencia desordenada incompleta [0,3,1,2,5].

Solución:

La secuencia es incompleta porque consta de 5 estados y no existe ningún número n entero igual al número de biestables del sistema, tal que $2^n - 1 = 5$. Puesto que el número de biestables requeridos es 3 y el número de estados total representables sería de 8, el sistema tiene 3 estados inalcanzables. El diseño del contador se resume en los pasos siguientes:

1. Dibujar DTE.
2. Crear la tabla de transición de estados.
3. Incluir en la tabla la excitación de cada biestable para provocar la transición de estado requerida.
4. Calcular las funciones de excitación de biestables dependientes del estado de partida.
5. Implementar el circuito con los biestables elegidos y las excitaciones obtenidas en el paso previo.

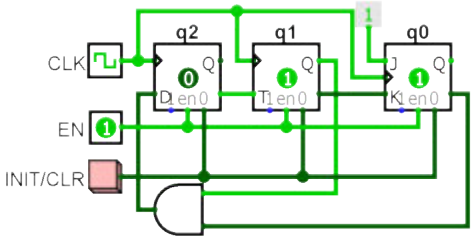
[3] Consulta más información sobre truncamiento síncrono y asíncrono en las secs. 8.2.2 y 8.2.3.



	t		
Q	q_2	q_1	q_0
Q_0	0	0	0
Q_1	0	0	1
Q_2	0	1	0
Q_3	0	1	1
Q_4	1	0	0
Q_5	1	0	1
Q_6	1	1	0
Q_7	1	1	1

	$t+1$			q_2	q_1	q_0		
Q	q_2	q_1	q_0	D_2	T_1	J_0	K_0
Q_3	0	1	1	0	1	1	x
Q_2	0	1	0	0	1	x	1
Q_5	1	0	1	1	1	1	x
Q_1	0	0	1	0	1	x	0
x	x	x	x	x	x	x	x
Q_0	0	0	0	0	0	x	1
x	x	x	x	x	x	x	x
x	x	x	x	x	x	x	x

En este ejemplo se utilizan tres tipos de biestables que ilustran la obtención de las funciones de excitación para cada uno de ellos. Los estados Q_4, Q_6 y Q_7 que no se emplean en el contador se consideran condiciones libres en la tabla de transición de estados. Observa que no existe tabla de salida, ya que la salida y el estado coinciden. Una vez calculadas las funciones de excitación de los biestables se obtiene: $D_2 = q_1 q_0'$, $T_1 = q_2'$, $J_0 = 1$, $K_0 = q_1'$. El circuito asociado resulta:



En los ejemplos anteriores se ha asumido que el valor inicial del contador se alcanza mediante activación de la señal de reinicio (*INIT/CLR*), incluso en condiciones de encendido o arranque del sistema (*power on*). La situación de arranque en Logisim determina el estado inicial de los sistemas una vez que se carga el fichero correspondiente al circuito deseado. Si se analiza detenidamente esta situación, se asume que en el arranque todos los biestables se inician en estado 0, sin precisar de una activación de la señal *INIT/CLR*. Sin embargo, se puede plantear la siguiente cuestión práctica: *¿es posible la simulación de un contador que «arranque» en un valor distinto de 0 sin precisar la activación manual de las señales CLR o PRE requeridas en los biestables?*

El ajuste automático en el arranque de un circuito secuencial se puede conseguir mediante un biestable D que memorice el estado de inicio. Si inicialmente este biestable está a 0 y su entrada se fija a 1, la salida será 0 hasta el primer pulso de la señal de sincronismo. Tomando el valor negado de su salida se puede generar una señal de activación de *CLR* o *PRE* para los biestables del circuito para iniciar asíncronamente su valor hasta que el biestable D de encendido se pone a 1, momento en que se desactiva la señal de inicio de los biestables. Esta estrategia se puede emplear con Logisim para simular el inicio automático de contadores en el arranque a valores distintos de 0 sin necesidad de activación manual de la señal de inicio.

En la figura 9.8 se muestra un contador que genera la secuencia del ejemplo 9.7, en el que se incluye un biestable D para que en el arranque del sistema el valor inicial sea distinto de 0, en este caso 3. De este modo es posible generar la secuencia [3,1,2,5,0] sin necesidad de activación manual de la señal *INIT*.

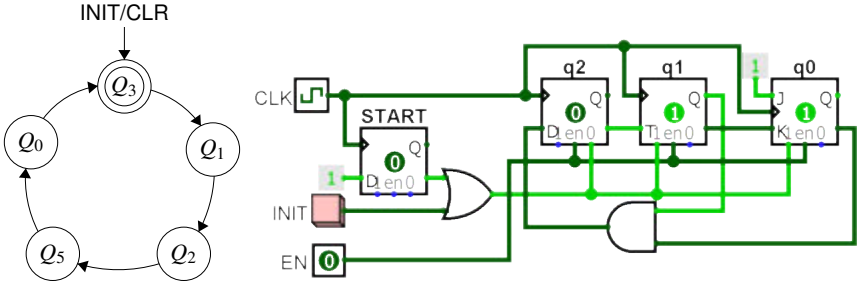


Figura 9.8: Diseño de contador incompleto e inicio automático mediante biestable D.

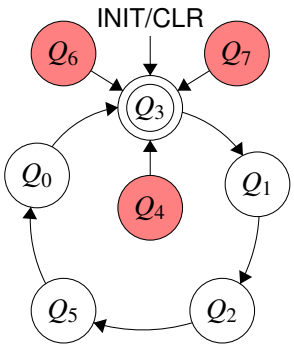
> En sistemas reales el inicio automático en el arranque se realiza mediante circuitos POR (*power-on reset*) que consisten en un periférico que genera una señal de *reset* en el arranque del sistema. De este modo se asegura que el sistema inicie su operación en un estado conocido.

Ejemplo 9.8 — Diseño de contador de secuencia incompleta con forzado de estados. Diseña un contador que genere la secuencia [3,2,1,5,0] con forzado de transición de los estados inalcanzables hacia el estado inicial.

Solución:

El procedimiento de diseño que se sigue es idéntico al descrito anteriormente en el texto. Sin embargo, en la tabla de transición, los estados inalcanzables no dan lugar a condiciones libres (*don't care*), ya que en los estados inalcanzables se fuerza la evolución del sistema hacia el estado inicial.

Con el forzado de estados se garantiza que el sistema no queda incontrolado si por cualquier causa accidental se llega a un estado inalcanzable. En este caso se habla también del diseño de un contador «autocorregido». La figura siguiente muestra el DTE (izda.) y tabla de transición correspondiente (dcha.).

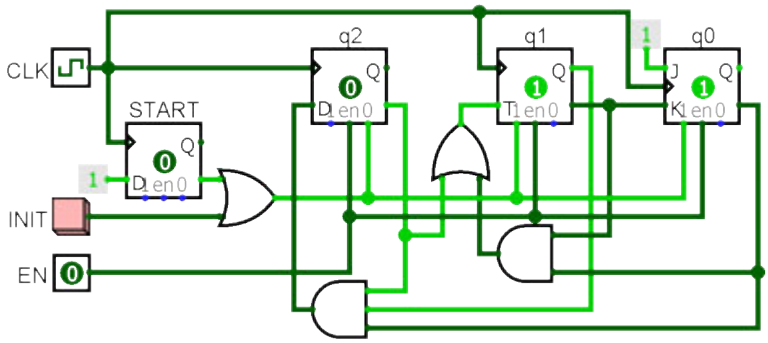


	t			$t+1$				q_2	q_1	q_0	
Q	q_2	q_1	q_0	Q	q_2	q_1	q_0	D_2	T_1	J_0	K_0
Q_0	0	0	0	Q_3	0	1	1	0	1	1	x
Q_1	0	0	1	Q_2	0	1	0	0	1	x	1
Q_2	0	1	0	Q_5	1	0	1	1	1	1	x
Q_3	0	1	1	Q_1	0	0	1	0	1	x	0
Q_4	1	0	0	Q_3	0	1	1	0	1	x	0
Q_5	1	0	1	Q_0	0	0	0	0	0	x	1
Q_6	1	1	0	Q_3	0	1	1	0	0	1	x
Q_7	1	1	1	Q_3	0	1	1	0	0	x	0

A partir de la tabla de transición, las ecuaciones de excitación que se obtienen para los biestables son:

$$D_2 = q_2' q_1 q_0' \qquad J_0 = 1$$
$$T_1 = q_2' + q_1' q_0' \qquad K_0 = q_1'$$

El circuito resultante del sistema para simulación lógica con Logisim queda:



■

9.7.3 Diseño de sistemas temporizados

Por sistema temporizado síncrono se entiende aquel que genera una secuencia con al menos un valor duplicado (p. ej., [0,3,2,1,2,1,3]). Estos sistemas se pueden modelar mediante un autómata finito con las propiedades siguientes:

- No posee entrada. El estado evoluciona en sincronía con la señal de reloj.
- ➲ *La salida no coincide con el estado.*

Puesto que en los sistemas temporizados la salida no coincide con su estado, la salida correspondiente a distintos estados puede coincidir. Esto es, la secuencia de salida puede tener valores repetidos. Este tipo de modelo también se puede emplear para secuencias sin valores repetidos cuando estos no coinciden con el estado. Por ejemplo, para generar la secuencia [12,1,15,6] se precisan 4 estados (2 biestables), sin embargo, la salida exige 4 bits. Este tipo de sistemas también se puede contemplar como aplicación de un conversor de código a la salida de un contador.

Los sistemas temporizados se emplean como *generadores de secuencia* que poseen multitud de aplicaciones en la automatización de sistemas secuenciales controlados por eventos (p. ej., semáforos, electrodomésticos, etc.). En dichos sistemas automatizados, cada valor de la secuencia se interpreta como un conjunto de acciones que se repiten en el tiempo.

El diseño de los sistemas temporizados se basa en el empleo de un contador ascendente o descendente ordenado, con tantos estados como valores tenga la secuencia a generar. Si el contador es ascendente permite saber cuántos estados se han generado en la secuencia y si es descendente cuántos quedan para completarla. La salida se obtiene como una función combinacional del estado al que se asigna el valor deseado de la secuencia.

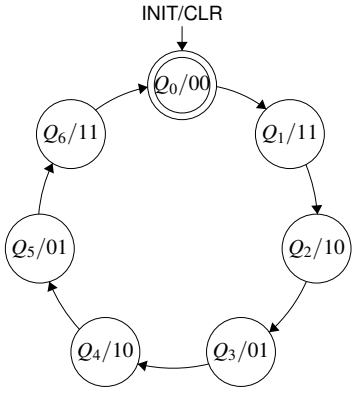
El aspecto temporal de este tipo de sistemas es muy importante, ya que además de generar una secuencia, lo hacen con una cadencia determinada. Para conseguir establecer dicha cadencia, la señal de sincronismo se ajusta a la frecuencia requerida. Para establecer la frecuencia de sincronismo es habitual el uso de divisores de la frecuencia base del reloj del sistema que emplea escalas del orden del MHz o incluso GHz.

Ejemplo 9.9 — Diseño de un generador de secuencia. Diseña un sistema secuencial síncrono que genere la secuencia [0,3,2,1,2,1,3] sin una frecuencia específica para el reloj de sincronismo. Esto es, la secuencia se puede generar a un ritmo de cambio arbitrario.

Solución:

Puesto que la secuencia posee valores repetidos, no se puede obtener directamente como la salida de un contador. Como la secuencia posee 7 valores, la primera alternativa de diseño parte de un contador síncrono ordenado de 3 bits, truncado a 7 estados: de Q_0 a Q_6. El truncamiento asíncrono del contador se consigue mediante la decodificación del valor 7 para activar la señal de borrado (*CLR*) de los biestables. Es decir, con $[q_2\, q_1\, q_0] = [1\,1\,1]$.

Para codificar el valor de salida se necesitan 2 bits (y_1, y_0), ya que en la salida es preciso representar hasta el valor $3_{(10} = 11_{(2}$. En la figura siguiente se muestra el DTE y la tabla de salida para el sistema. Observa que la salida en cada estado se representa en binario con dos bits. No se incluye la tabla de transición pues el circuito empleado es un contador ascendente MOD 7 para el que solo es necesario calcular la salida que corresponde a cada estado. En la tabla de salida el estado no empleado Q_7 da lugar a condiciones libres para el cálculo de la función de salida.



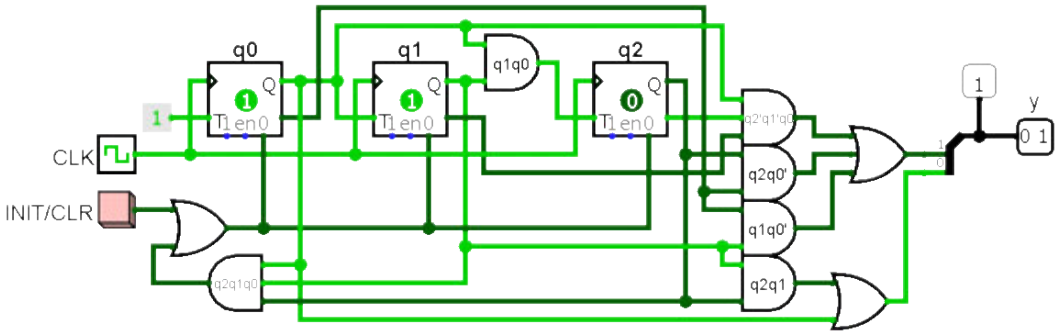
q2	q1	q0	y1	y0
0	0	0	0	0
0	0	1	1	1
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	1	1
1	1	1	x	x

Las formas SOP minimizadas para las funciones de salida que se obtienen son:

$$y_0 = q_0 + q_2 q_1 \qquad y_1 = q_2' q_1' q_0 + q_1 q_0' + q_2 q_0'$$

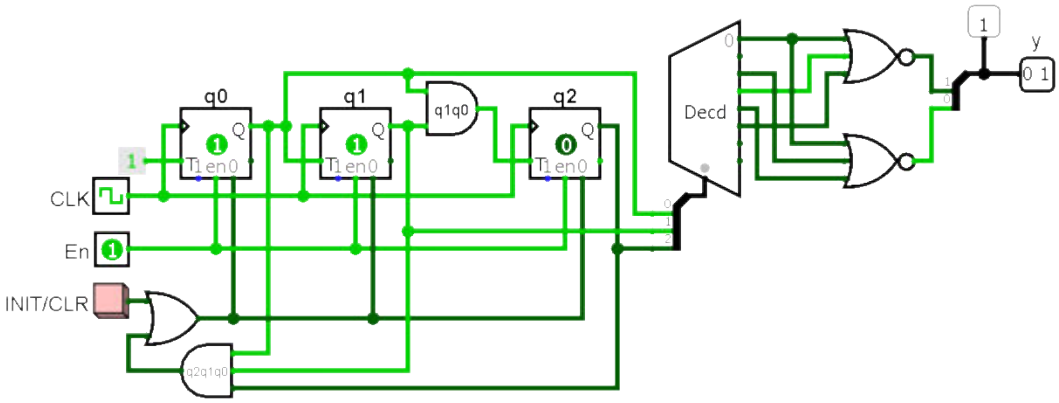
Para la implementación de las funciones de salida vamos a considerar varias alternativas:

▷ Implementación mediante puertas lógicas.



▷ Empleo de un DEC 3×8. En este caso se utilizan las expresiones canónicas de y_1 e y_0. Para ambas funciones se toman los maxitérminos, ya que su número es menos que el de minitérminos. De este modo, en lugar de emplear puertas OR a la salida del DEC, se emplean puertas NOR.

$$y_1 = \prod M(0,3,5) \qquad y_0 = \prod M(0,2,4)$$

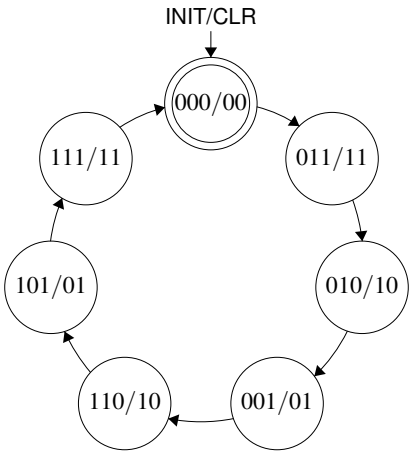


En los problemas de diseño secuencial síncrono siempre es posible plantear la recodificación o reordenación de estados. El propósito es obtener la salida del modo más sencillo posible a partir del estado. Puesto que en este texto se recomienda una codificación de estado en binario natural, cuando hablamos de recodificación del estado realmente nos referimos a una reordenación del estado. Esto queda más claro mediante la aplicación en el ejemplo siguiente.

Ejemplo 9.10 — Diseño de un sistema temporizado con reordenación de estados. Diseña un sistema secuencial síncrono que genere la secuencia [0,3,2,1,2,1,3].

Solución:

Como ya se ha explicado anteriormente en este sistema el estado y la salida no pueden coincidir, ya que existen valores repetidos en la secuencia. Pero en este caso en lugar de recurrir a un contador ordenado, se emplea un contador con secuencia de estados: [Q_0, Q_3, Q_2, Q_1, Q_6, Q_5, Q_7]. Esta elección no es caprichosa, sino motivada por una obtención más simple de la función de salida. Con la reordenación del estado se consigue que la salida coincida con los dos bits menos significativos que codifican el estado (ver figura adjunta).



	t		
Q	q_2	q_1	q_0
Q_0	0	0	0
Q_1	0	0	1
Q_2	0	1	0
Q_3	0	1	1
x	1	0	0
Q_5	1	0	1
Q_6	1	1	0
Q_7	1	1	1

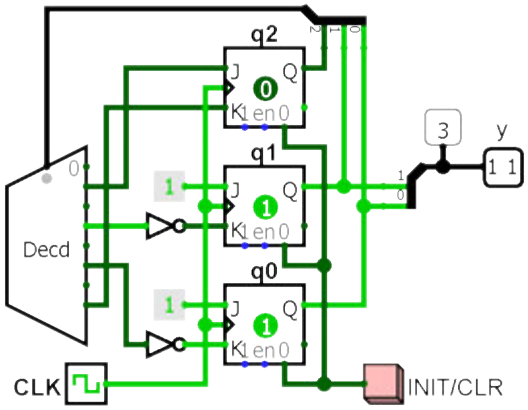
	$t+1$			q_2		q_1		q_0		y	
Q	q_1	q_1	q_0	J_2	K_2	J_1	K_1	J_0	K_0	y_1	y_0
Q_3	0	1	1	0	x	x	1	1	x	0	0
Q_6	1	1	0	1	x	1	x	x	1	0	1
Q_1	0	0	1	0	x	x	1	1	x	1	0
Q_2	0	1	0	0	x	x	0	x	1	1	1
x	x	x	x	x	x	x	x	x	x	x	x
Q_7	1	1	1	x	0	1	x	x	0	0	1
Q_5	1	0	1	x	0	x	1	1	x	1	0
Q_0	0	0	0	x	1	x	1	x	1	1	1

En este tipo de sistema aunque la salida solo depende del estado, la salida se puede añadir a la tabla de transición de estado porque no existe entrada. Por tanto, se tiene directamente la tabla de verdad para la función de salida. A partir de las tablas de transición de estado y salida se obtienen las expresiones de las funciones de excitación de los biestables (J-K):

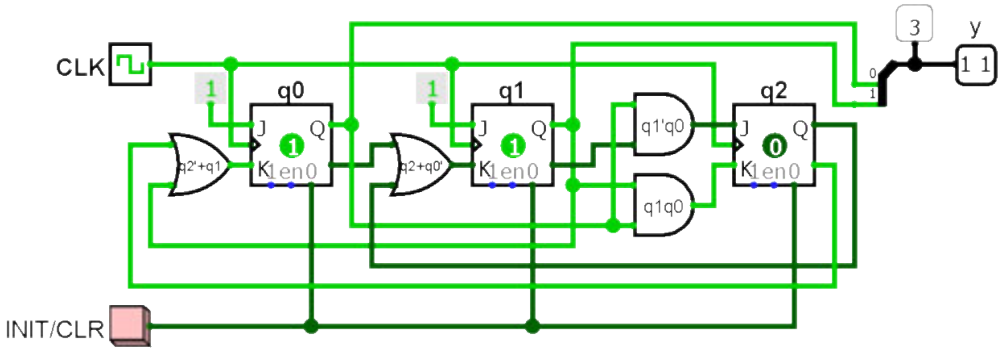
$$J_2 = m_1 = q_1' q_0 \qquad J_1 = 1 \qquad J_0 = 1$$
$$K_2 = m_7 = q_1' q_0 \qquad K_1 = m_3' = q_2 + q_0' \qquad K_0 = m_5' = q_2' + q_1$$

Como se ha indicado previamente, la expresión las funciones de salida coincide con los dos bit menos significativos del estado: $y_1 = q_1,\ y_0 = q_0$

El circuito resultante emplea un DEC 3×8, en el que tomando las expresiones canónicas de las funciones de excitación la implementación resultante queda:



Si en el mismo circuito se emplean con puertas lógicas elementales para las funciones de excitación de biestables, el circuito resultante queda:



Partiendo del empleo de un contador truncado para generar los estados de un sistema temporizado, la obtención de la salida deseada se puede abordar por métodos alternativos de implementación. Por ejemplo, basados en el uso de multiplexores multibit, memorias e incluso registros multibit en anillo. Estos métodos tienen en común su flexibilidad y sencillez de diseño. Su «desventaja» reside en el uso de módulos que emplean un mayor número de componentes elementales como: puertas, lógicas, biestables, etc.

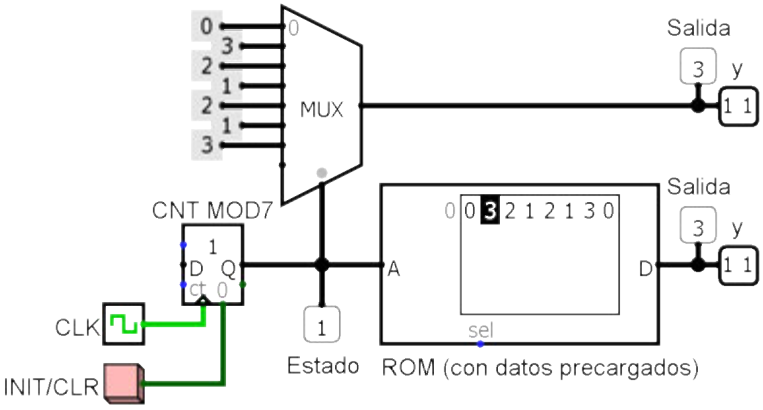
Ejemplo 9.11 — Implementación alternativa de un generador de secuencia. Diseña un sistema secuencial síncrono que genere la secuencia [0,3,2,1,2,1,3].

Solución:

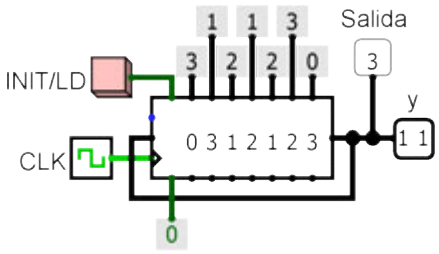
Los estados de la secuencia se pueden generar mediante un contador MOD 7 obtenido por truncamiento de un contador MOD 8. La salida del contador (estado) será utilizada para el direccionamiento del elemento que proporcione el valor de salida. Veamos dos opciones para simulación lógica con Logisim:

▷ **Direccionamiento de un multiplexor**. La salida del contador se emplea como entrada de selección a un MUX cuyas entradas de datos están conectadas a valores constantes de tipo multibit con los valores correspondientes de la secuencia a generar. En un sistema genérico serían precisos tantos multiplexores en paralelo como bits para codificar los valores de salida. En la implementación de sistemas físicos reales, los valores constantes para la entrada de estos multiplexores se pueden obtener mediante conexión de celdas de memoria SRAM cargadas con el valor binario adecuado.

▷ **Direccionamiento de una memoria ROM**. Se emplea una memoria en la que se almacenan previamente los valores de la secuencia a generar. De nuevo el contador se emplea para obtener sucesivamente las direcciones de acceso a la memoria.



Un método alternativo de implementación permite la sustitución del contador y la memoria por un registro multibit de desplazamiento en anillo. La ausencia del contador impide conocer el estado en que se encuentra el sistema. Además, se precisa un ciclo de carga de los valores iniciales del registro.



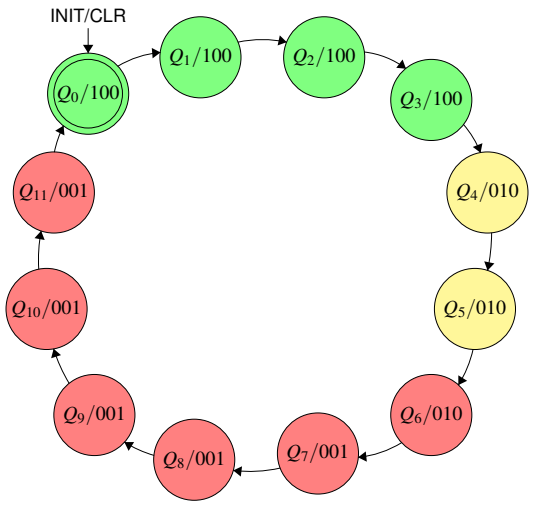
La aplicación práctica de los sistemas temporizados es el control de dispositivos mediante la activación de determinadas acciones por un tiempo determinado. Por tanto, los valores de la secuencia de salida corresponden con la activación o desactivación de cada salida representada por un bit del valor de la secuencia. Para este tipo de sistema el elemento clave de control es el contador, cuya misión fundamental es controlar la cadencia de generación de la salida.

Ejemplo 9.12 — Control de un sistema temporizado. Diseña un sistema secuencial síncrono que controle de modo cíclico los tiempos de activación de las luces de un semáforo cuyas duraciones son: verde (4 s), ámbar (2 s) y rojo (6 s).

Solución:

Este ejemplo consiste en el diseño de un sistema sin entrada que está gobernado por un contador que cambia de estado a una cadencia concreta. La solución se aborda considerando un generador de secuencia con 12 estados correspondientes a los 12 s de duración total del ciclo del semáforo. A cada estado se asocia como salida, el valor de activación de cada una de las luces (V=verde, A=ámbar y R=rojo). Por comodidad se emplean letras mayúsculas para las funciones de salida.

A continuación se muestra el DTE con los estados marcados con el color de la luz activa del semáforo para facilitar su comprensión. En cada estado se indica también el valor binario de la salida. Junto al DTE se detalla la tabla de transición de estados y los valores correspondientes de excitación para biestables de tipo T. Como el número de estados es 12, se necesitan 4 biestables, con 4 estados inalcanzables que se tratan como condiciones libres.



	t					*t+1*					q_3	q_2	q_1	q_0	y		
Q	q_3	q_2	q_1	q_0	Q	q_3	q_2	q_1	q_0	**T_3**	**T_2**	**T_1**	**T_0**	**V**	**A**	**R**
Q_0	0	0	0	0	Q_1	0	0	0	1	0	0	0	1	1	0	0
Q_1	0	0	0	1	Q_2	0	0	1	0	0	0	1	1	1	0	0
Q_2	0	0	1	0	Q_3	0	0	1	1	0	0	0	1	1	0	0
Q_3	0	0	1	1	Q_4	0	1	0	0	0	1	1	1	1	0	0
Q_4	0	1	0	0	Q_5	0	1	0	1	0	0	0	1	0	1	0
Q_5	0	1	0	1	Q_6	0	1	1	0	0	0	1	1	0	1	0
Q_6	0	1	1	0	Q_7	0	1	1	1	0	0	0	1	0	0	1
Q_7	0	1	1	1	Q_8	1	0	0	0	1	1	1	1	0	0	1
Q_8	1	0	0	0	Q_9	1	0	0	1	0	0	0	1	0	0	1
Q_9	1	0	0	1	Q_{10}	1	0	1	0	0	0	1	1	0	0	1
Q_{10}	1	0	1	0	Q_{11}	1	0	1	1	0	0	0	1	0	0	1
Q_{11}	1	0	1	1	Q_0	0	0	0	0	1	0	1	1	0	0	1
x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x

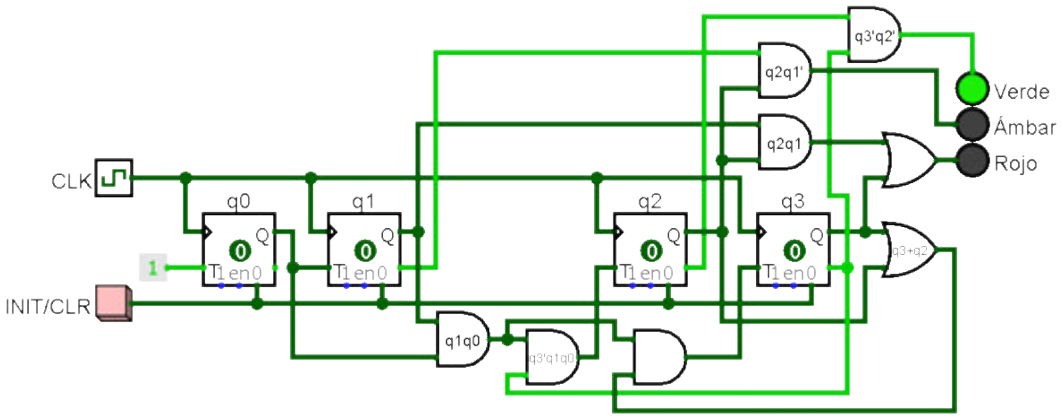
A partir de la tabla de transición se obtienen (mediante K-maps) las funciones de excitación de los biestables que se indica:

$$T_0 = 1 \qquad T_2 = q_3' q_1 q_0$$
$$T_1 = q_0 \qquad T_3 = (q_2 + q_3) q_1 q_0$$

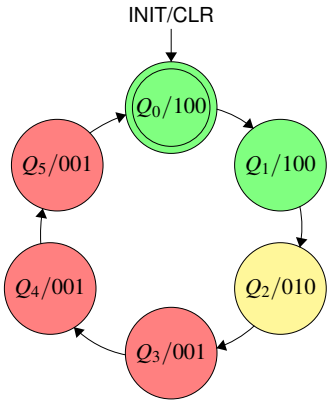
En la tabla anterior también se incluyen las funciones de salida correspondientes a la activación de cada una de las luces del semáforo:

$$\text{VERDE} = q_3' q_2' \qquad \text{AMBAR} = q_2 q_1' \qquad \text{ROJO} = q_2 q_1 + q_3$$

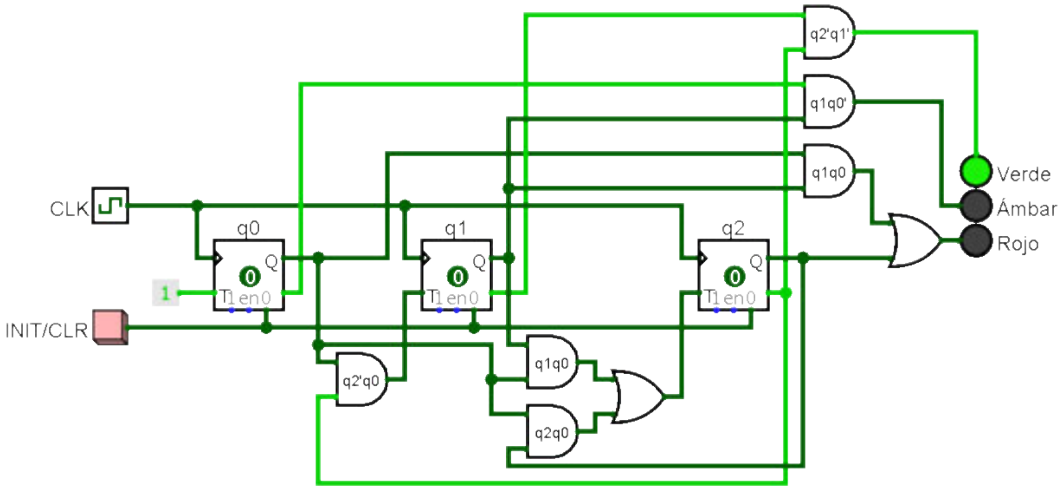
La figura adjunta muestra el circuito lógico resultante para simulación lógica con Logisim. Para conseguir la temporización adecuada en dicho circuito, es preciso elegir una frecuencia de *tick* de 2 Hz, para que la frecuencia de la señal *CLK* sea de 1 Hz (1 s de duración por estado).



Este sistema admite una simplificación del número de estados si la duración de estos se extiende a 2 s. Para conseguirlo se ajusta la frecuencia de *tick* a 0.5 Hz. De este modo se precisa la mitad de estados, ya que estos tienen el doble de duración. Esta simplificación solo es posible si existe un divisor común de la duración correspondiente a cada una de las luces. En este ejemplo, dicho factor común es 2 que da lugar a un DTE con 6 estados y una tabla de transición más sencilla (ver figura siguiente):



t				$t+1$				$q_2\ q_1\ q_0$			y		
Q	q_2	q_1	q_0	Q	q_2	q_1	q_0	T_2	T_1	T_0	V	A	R
Q_0	0	0	0	Q_1	0	0	1	0	0	1	1	0	0
Q_1	0	0	1	Q_2	0	1	0	0	1	1	1	0	0
Q_2	0	1	0	Q_3	0	1	1	0	0	1	0	1	0
Q_3	0	1	1	Q_4	1	0	0	1	1	1	0	0	1
Q_4	1	0	0	Q_5	1	0	1	0	0	1	0	0	1
Q_5	1	0	1	Q_0	0	0	0	1	0	1	0	0	1
x	x	x	x	x	x	x	x	x	x	x	x	x	x
x	x	x	x	x	x	x	x	x	x	x	x	x	x



Puesto que no es posible elegir una frecuencia de *tick* de 1 Hz, para obtener una frecuencia de 0.5 Hz para *CLK*, se recurre a una señal *CLK* con un periodo de 4 *ticks*, siendo la frecuencia de *tick* de 2 Hz. ■

El diseño basado en autómatas de estados finitos es ineficiente para sistemas en los que se repite un valor de la secuencia durante un número elevado de estados, como en el ejemplo anterior. Una alternativa de diseño más conveniente es la que emplea una estrategia de control basada en el *modelo de etapa-transición*.[4] Con este modelo el sistema se divide en etapas durante las que se activan unas

[4] El modelo etapa-transición es una herramienta empleada en el control asíncrono de sistemas secuenciales dirigidos por eventos. En este caso los eventos que «dirigen» el sistema están constituidos por el final de los temporizadores.

salidas y la transición de una etapa a la siguiente está determinada por una condición (p. ej., el final de un temporizador).

La figura 9.9 muestra el diagrama etapa-transición para el ejemplo 9.12. Dicho diagrama se compone de las tres etapas correspondientes a la activación de cada una de las luces del semáforo. Con el inicio de cada etapa se activa un temporizador ajustado al tiempo que está activada la etapa. La transición de una etapa a la siguiente está marcada por el final de la temporización de cada etapa.

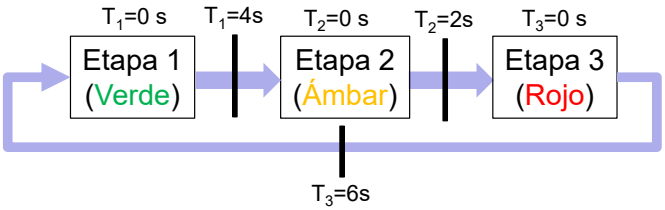


Figura 9.9: Diagrama etapa-transición correspondiente al control de las luces de un semáforo.

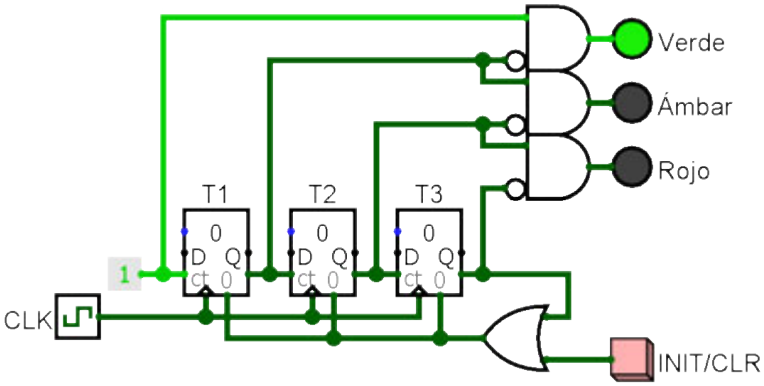
Definición 9.2 — Temporizador. Es un contador ordenado completo, o de secuencia truncada, cuya señal de sincronismo está ajustada a una medida estándar de tiempo como el segundo o una fracción de este. ■

Ejemplo 9.13 — Control etapa-transición de sistema temporizado. Diseña un sistema secuencial síncrono que controle de modo cíclico los tiempos de activación de las luces de un semáforo cuyos tiempos de duración de cada luz son: verde (4 s), ámbar (2 s) y rojo (6 s).

Solución:

Aplicando un modelo de control basado en el diagrama etapa-transición (ver figura 9.9), se precisan tres temporizadores T_1, T_2 y T_3. Estos temporizadores emplean una frecuencia de sincronismo de 1 Hz de modo que su valor de cuenta representa el valor en segundos de la temporización.

Para la simulación lógica de temporizadores con Logisim, se emplea el módulo contador ascendente con disparo por flanco de subida y enclavamiento en el final de cuenta (atributo *Action on Overflow* ajustado a *Stay at value*). El bit de fin de cuenta del temporizador activa la cuenta del siguiente temporizador y así sucesivamente hasta alcanzar la última etapa. El final de la última etapa pone a cero todos los contadores quedando activo solo el primero de ellos. Siguiendo este esquema, cada luz del semáforo se activa cuando el temporizador correspondiente está activado (iniciado) y no finalizado.



■

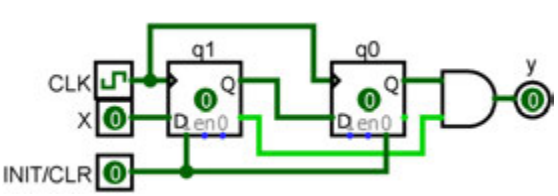
La ventaja principal del control temporizado mediante el modelo etapa-transición frente al diseño mediante un autómata finito, es su flexibilidad e independencia de los tiempos asignados a cada etapa. Además, el uso de temporizadores facilita la identificación de las transiciones entre etapas.

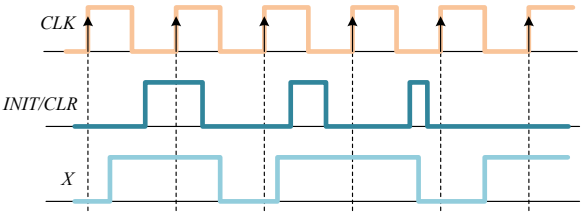
Conceptos clave

- ☑ Diferencias entre los sistemas combinacionales y secuenciales.
- ☑ Cómo reconocer el circuito asociado a un sistema secuencial.
- ☑ Diferencias entre los sistemas secuenciales síncronos y asíncronos.
- ☑ Cómo reconocer si un circuito secuencial es síncrono o asíncrono.
- ☑ Definición de autómata finito y tipos.
- ☑ Cómo reconocer el tipo de autómata implementado por un circuito.
- ☑ Análisis y elaboración de tablas de transición de estado de un sistema secuencial síncrono y DTE correspondiente.
- ☑ Pasos de análisis de sistemas secuenciales síncronos.
- ☑ Pasos de diseño de sistemas secuenciales síncronos.
- ☑ Diferencias entre contadores y sistemas temporizados.
- ☑ Diseño de contadores y de sistemas temporizados.

Problemas propuestos

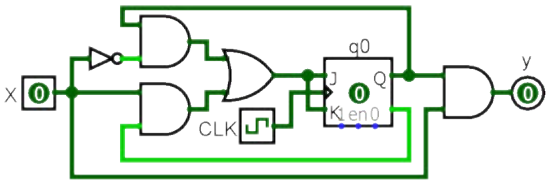
Problema 9.1 Dibuja el cronograma de la señal de salida del circuito secuencial de la figura adjunta teniendo en cuenta el valor de las señales indicadas. Observa que ambos biestables parten del estado $q_1 = q_0 = 0$.



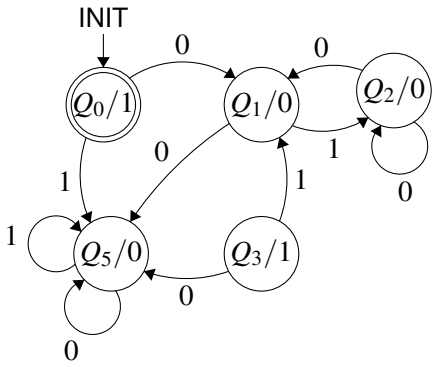


Problema 9.2 Para el circuito lógico de la figura adjunta, contesta de modo razonado a las cuestiones siguientes:

1.- ¿El circuito implementa un sistema combinacional o secuencial?
2.- En caso de implementar un sistema modelado con un autómata finito. ¿Cuántos estados tendría el sistema, como máximo? ¿Sería un autómata de Moore o de Mealy?
3.- Elabora la tabla de transición de estado del sistema.
4.- Dibuja el DTE correspondiente.

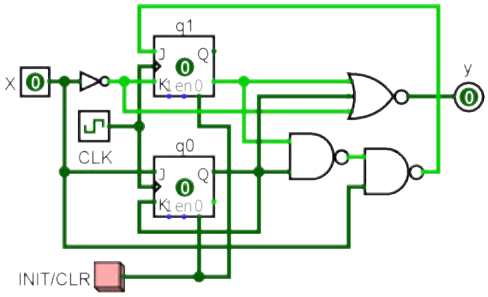


Problema 9.3 Dado el DTE de la figura adjunta, identifica los errores presentes en el mismo.



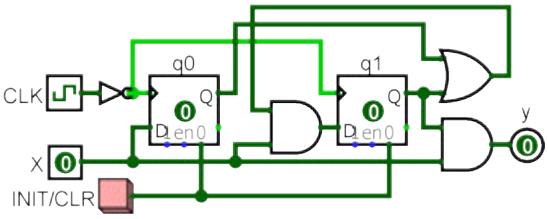
Problema 9.4 Para el circuito de la figura adjunta contesta razonadamente a las cuestiones siguientes:

1.- Identifica la/s entrada/s y salida/s del circuito.
2.- ¿Qué tipo de sistema representa: combinacional o secuencial, síncrono o asíncrono?
3.- Si el circuito correspondiese a un autómata finito, ¿de qué tipo sería este?
4.- Calcula las expresiones lógicas de las excitaciones de los biestables y la función de salida.
5.- Elabora la tabla de transición de estado y salida del sistema.
6.- Dibuja el DTE correspondiente.

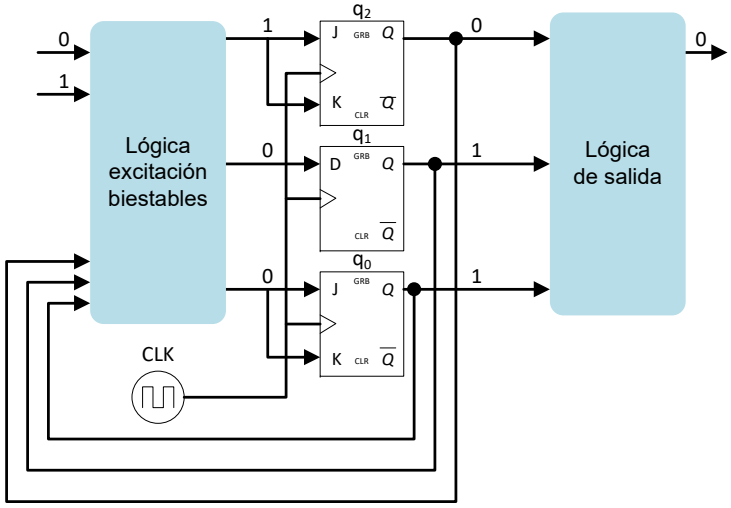


Problema 9.5 Para el circuito de la figura adjunta contesta razonadamente a las cuestiones siguientes:

1.- Identifica entradas y salidas e indica el tipo de autómata que se implementa con el circuito.
2.- Plantea las expresiones lógicas de las excitaciones de los biestables y de la salida del circuito.
3.- Determina la tabla de transición de estado incluida la salida. Para el estado emplea la codificación $Q = [q_1\, q_0]$.
4.- Dibuja el DTE correspondiente.
5.- Dibuja el cronograma de las señales q_1, q_0 para los 5 primeros ciclos de reloj con $X = 1$, si se considera el estado inicial $Q_0 = [00]$. Explica la funcionalidad del circuito.



Problema 9.6 El diagrama de bloques de la figura adjunta representa un autómata finito.



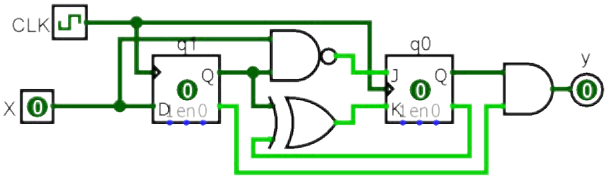
Dados los componentes del sistema y los valores concretos de las señales indicados en la figura, contesta razonadamente a las cuestiones siguientes:

1.- ¿Cuál es el número máximo de estados posible del autómata?
2.- ¿Es un autómata de Moore o de Mealy?
3.- ¿Cuántas filas tiene la tabla de verdad que define la salida del autómata?
4.- ¿Cuántas filas tiene la tabla de verdad que define el estado Q_{t+1} al que evoluciona el autómata en cada instante de sincronismo?
5.- ¿En qué estado se encuentra el sistema en el instante mostrado en la figura proporcionada?
6.- Si estuviese a punto de producirse un flanco positivo del reloj (CLK) ¿Cuál sería el estado del sistema después del sincronismo de los biestables?

Problema 9.7 Dado el circuito de la figura adjunta, contesta razonadamente a las cuestiones siguientes:

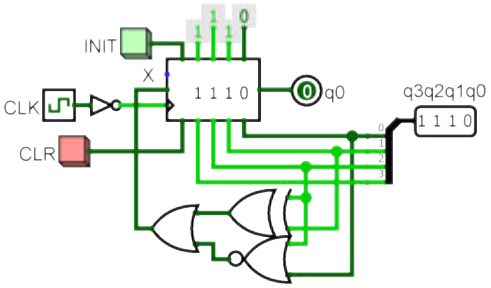
1.- Calcula las funciones de excitación de los biestables D_1, J_0, K_0 y la función de salida y.
2.- ¿Qué tipo de autómata implementa el circuito?
3.- Elabora la tabla de transición de estado y de salida correspondientes al circuito.
4.- Dibuja el DTE del sistema a partir de las tablas del apartado anterior.
5.- Construye una tabla con la evolución temporal en 5 flancos consecutivos de sincronismo ($t_s = 1$ a $t_s = 5$) para las entradas de excitación de los biestables (D_1, J_0 y K_0), la salida de los biestables (q_1, q_0) y la salida del circuito (y). El estado inicial de partida es $Q_0 = [00] \Rightarrow q_1 = 0\, q_0 = 0$ y los valores de la entrada en los instantes de sincronismo es $X = [0,1,1,1,0]$.

6.- Traslada la tabla obtenida en el apartado anterior a un cronograma de las señales de entrada (X), salidas de los biestables (q_1, q_0) y salida del sistema (y).



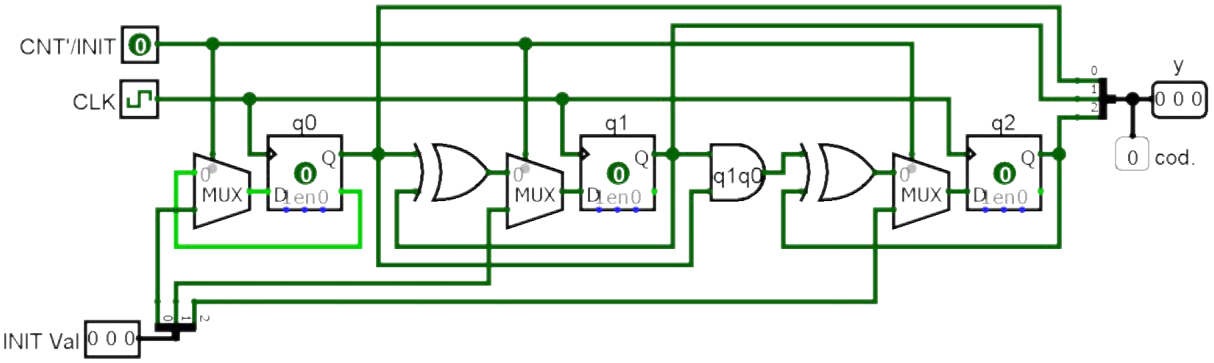
Problema 9.8 El circuito de la figura adjunta representa un registro de desplazamiento sincronizado por flanco de bajada, en el que los bits se desplazan de q_3 a q_0 (desplazamiento a la derecha) y $q_3 = X$. Es decir, el valor que entra en el registro se obtiene realimentando el valor resultante de un circuito combinacional aplicado al contenido del registro. Responde razonadamente a las cuestiones siguientes relacionadas con el circuito mencionado:

1.- Calcula la expresión lógica de la variable X de entrada al registro.
2.- Analiza la evolución del estado del registro mediante una tabla de evolución temporal, sabiendo que el estado de partida es $q_3 = 1$, $q_2 = 1$, $q_1 = 1$, $q_0 = 0 \Rightarrow Q_{14} = [1110]$. Realiza dicha tabla hasta que el estado se repita.
3.- Dibuja el cronograma del circuito partiendo del estado indicado en el apartado anterior.
4.- Implementa un circuito equivalente al proporcionado, pero compuesto con biestables individuales. Comprueba la equivalencia de los circuitos mediante simulación con Logisim.



Problema 9.9 Dado el circuito de la figura adjunta contesta razonadamente a las cuestiones siguientes:

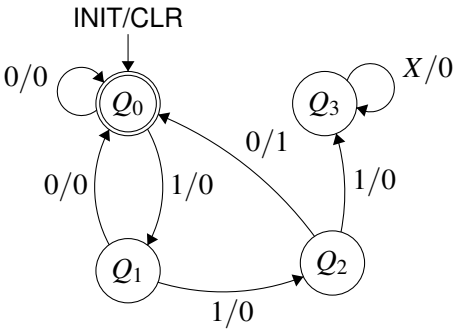
1.- ¿Cuál es el número máximo de estados que puede poseer?
2.- ¿Qué tipo de sincronismo presenta el circuito?
3.- ¿Cuál es el comportamiento del circuito cuando $CNT'/INIT = 1$?
4.- Elabora la tabla de transición de estado si $CNT'/INIT = 0$.
5.- Dibuja el DTE del sistema y describe detalladamente su funcionalidad.



Problema 9.10 Dado el DTE de la figura adjunta contesta razonadamente a las cuestiones siguientes:

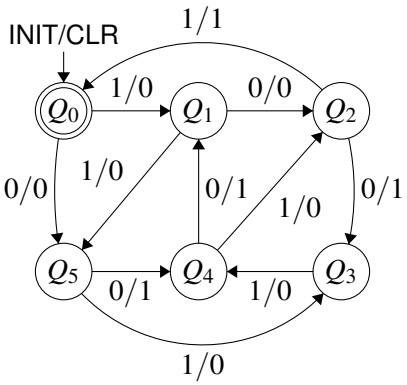
1.- ¿A qué tipo de autómata corresponde el DTE proporcionado?
2.- ¿Cuántos biestables se precisan, como mínimo, para implementar el sistema? Especifica la codificación elegida para los estados.
3.- Elabora la tabla de transición de estado del sistema.

4.- Implementa el autómata con un circuito lógico que conste de un biestable J-K (para el MSB del estado) y un biestable T (para el LSB del estado). Para ello implementa las funciones lógicas de excitación de los biestables y la función de salida mediante dos métodos: (A) solo con puertas lógicas, y (B) con un DEC 3×8 y las puertas lógicas necesarias.



Problema 9.11 Dado el DTE de la figura adjunta contesta razonadamente a las cuestiones siguientes:

1.- ¿A qué tipo de autómata (Moore o Mealy) corresponde el DTE proporcionado?
2.- ¿Cuántos estados presenta el sistema? ¿Qué número de biestables son necesarios para codificar los estados del sistema? ¿Qué codificación elegirías para los estados?
3.- ¿Cuántas entradas y salidas tiene el sistema?
4.- Elabora la tabla de transición de estado del sistema junto con los valores de las señales de excitación de los biestables. Para codificar el estado, emplea un biestable T como bit más significativo, un J-K para el menos significativo y considera el resto de biestables de tipo D.
5.- Implementa el sistema con un circuito lógico para simulación con Logisim siguiendo la alternativa más simple para las funciones de excitación y la función de salida.



Problema 9.12 Se desea diseñar un sistema síncrono que genere la secuencia: $[0,1,2,3,1,3,2,1]$. Contesta razonadamente a los apartados siguientes:

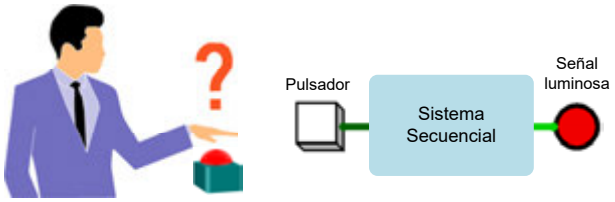
1.- ¿Cuántos estados tiene el sistema? ¿Cuántos biestables son necesarios, como mínimo, para su implementación?
2.- Indica si este sistema corresponde a un contador y en caso contrario indica de qué tipo de sistema se trata.
3.- Diseña el sistema con el tipo de biestable de tu elección.
4.- Implementa el diseño realizado para su simulación con Logisim.

Problema 9.13 Se desea diseñar un circuito secuencial síncrono de dos entradas $X = [X_1\, X_0]$ y una salida (y), de modo que $y = 1$ cuando la entrada sea $X = [01]$ y esté precedida de las entradas previas $X = [00]$ seguida de $X = [10]$. En cualquier otro caso la salida debe ser nula $y = 0$. Una vez que el sistema genera la salida $y = 1$, debe regresar al estado inicial. Completa las fases que se indican a continuación:

1.- Determina la definición y codificación de estados. Plantea tanto un autómata de Moore como uno de Mealy.
2.- Elabora el DTE del sistema para cada tipo de autómata.

3.- Confecciona la tabla de transición de estados correspondientes a cada tipo de autómata empleando tanto biestables de tipo J-K como D.
4.- Implementa al menos un circuito para cada tipo de autómata planteado.

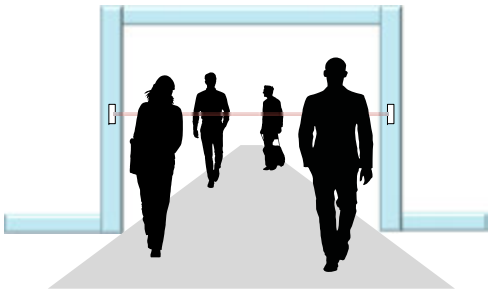
Problema 9.14 En un concurso de preguntas y respuestas cuando el jugador desea responder acciona un pulsador que activa inmediatamente una señal luminosa de color rojo.



Se desea diseñar un sistema secuencial síncrono mediante un autómata de Moore que controle la activación memorizada de la señal luminosa mediante un pulsador de acción momentánea. Esto es, el pulsador genera un nivel alto (1) solo cuando está pulsado. El sistema a diseñar debe memorizar la acción que enciende la señal luminosa, de modo que esta permanezca encendida cuando se libera el pulsador. Para apagar la señal luminosa será precisa una nueva pulsación. En este nuevo estado, la señal permanece apagada hasta nueva activación. Completa los pasos de diseño indicados a continuación:

1.- Define las señales de entrada, salida y los estados que tendrá el sistema cuando se emplee un autómata de Moore para su diseño.
2.- Determina el número de biestables de tipo J-K necesarios para implementar el sistema y dibuja un DTE apropiado.
3.- Traslada el DTE a una tabla de transición de estado y una tabla de salida.
4.- Calcula las funciones de excitación para los biestables de tipo J-K y la función de salida.
5.- Implementa el circuito resultante con Logisim. Para la entrada emplea un pulsador y para la salida un elemento LED.
6.- Explica la influencia de la frecuencia de reloj elegida en el funcionamiento del sistema.

Problema 9.15 En la entrada de un edificio inteligente se instala un sistema sensorial para determinar su grado de ocupación siguiendo el esquema que muestra la figura adjunta.



El sistema sensorial consiste en una barrera fotoeléctrica que activa una señal de nivel alto (1) cada vez que una persona entra en el edificio y bloquea dicha barrera. Se desea diseñar un sistema secuencial síncrono que genere un único pulso durante un ciclo de la señal de reloj cada vez que se bloquea la barrera fotoeléctrica (transición $0 \rightarrow 1$ ⎍). De este modo, se pretende evitar que el sistema sensorial genere de modo continuo una señal a nivel alto mientras la barrera está bloqueada, contabilizando repetidamente a una persona detectada. Completa las fases de diseño del sistema que se indican a continuación:

1.- Define las señales de entrada y salida del sistema junto a los estados que tendrá este para su diseño con un autómata de Moore.
2.- Determina el número de biestables J-K requerido para implementar el sistema y dibuja el DTE correspondiente.
3.- Traslada el DTE a una tabla de transición de estado y una tabla de salida. Calcula las funciones de excitación para los biestables J-K y la función de salida.

4.- Implementa el circuito resultante con Logisim para su simulación lógica y explica la influencia de la frecuencia de reloj elegida en la simulación. Emplea el elemento contador incluido con Logisim para contabilizar los pulsos producidos por el sistema sensorial.
5.- Diseña un sistema análogo teniendo en cuenta que el pulso generado por la barrera sensorial se produce en la transición $1 \rightarrow 0$ ($\sqcap\!\!\sqcup$).

Problema 9.16 Diseña un sistema secuencial síncrono para avisar a un vigilante de seguridad (g) e incluso a la policía (p) de los intentos de robo en una sucursal bancaria.

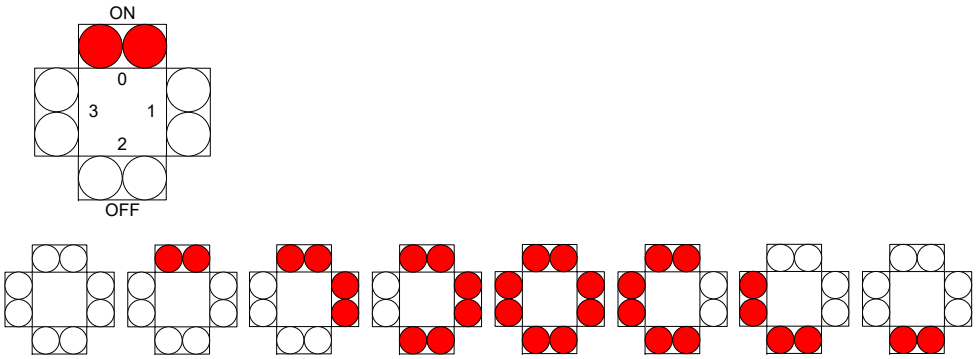
Para acceder a la caja fuerte hay que pasar necesariamente por un hall con un detector de presencia (H) que se activa a nivel alto (1) cuando una persona se mueve en su rango de detección. En la cámara acorazada en la que se encuentra la caja fuerte hay instalado otro detector de presencia (C) con funcionamiento similar. Las condiciones que describen el comportamiento del sistema son las siguientes:

- SI no entra nadie, ENTONCES sistema en reposo sin avisar ni al vigilante ni a la policía.
- SI un ladrón accede al hall, ENTONCES avisar al vigilante, pero no a la policía.
  - SI el intruso entra en la cámara acorazada, ENTONCES avisar a la policía.
  - SI el intruso abandona el hall antes de acceder a la cámara acorazada, ENTONCES interrumpir el aviso al vigilante, pero «recordar» que alguien estuvo en el hall.
  - SI se vuelve a activar el detector de presencia del hall, ENTONCES avisar tanto al vigilante como a la policía.
  - SI se activa el aviso a la policía, ENTONCES permanecer en este estado, retornando al estado inicial solo mediante reinicio del sistema.

Considera que nunca se activan simultáneamente el detector del hall y de la cámara acorazada. Además, el sensor de la cámara acorazada solo se activa en caso de que antes se haya activado el del hall. Completa las fases de diseño indicadas a continuación:

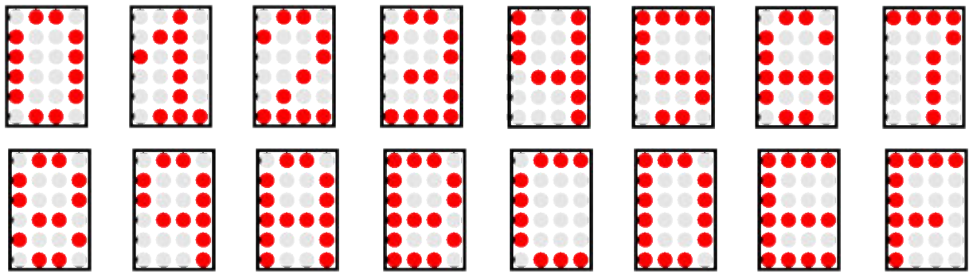
1.- Define las entradas y salidas del sistema, así como los estados para un autómata de Moore.
2.- Elabora el DTE correspondiente al sistema y la tabla de transición de estados. Considera el empleo de un biestable D para el bit de estado más significativo, un biestable T para el menos significativo y biestables tipo J-K para el resto, si es preciso.
3.- Implementa las funciones de excitación de los biestables y la función de salida mediante puertas lógicas en dos niveles de la forma más sencilla posible.

Problema 9.17 Se dispone de una luminaria compuesta por 4 segmentos, cada uno formado por 2 LED según se muestra en la figura adjunta.

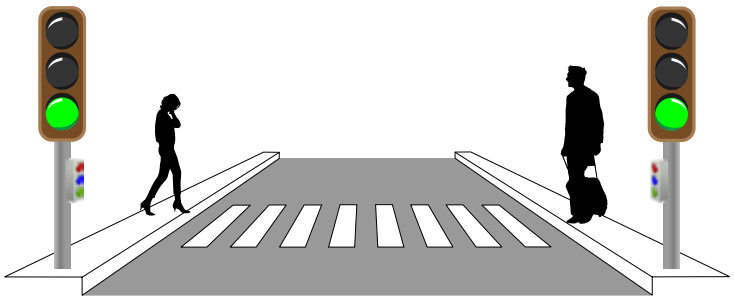


Diseña un circuito secuencial síncrono de control que permita obtener la secuencia de iluminación indicada en la figura (parte inferior). Considera que el control de la cadencia de cambio está marcada por un reloj de 1 Hz. Realiza la implementación del circuito para simulación con Logisim empleando biestables de tipo T.

Problema 9.18 Diseña un sistema secuencial síncrono capaz de generar una secuencia temporizada de 8 valores hexadecimales programados y mostrarlos en una matriz de leds de tamaño 6×4 (filas $\times$ columnas). La matriz tiene 6 entradas de 4 bits correspondientes a sus filas y columnas. Emplea los elementos disponibles en Logisim para la simulación lógica del sistema cuando la secuencia generada es [1,3,6,2,0,2,2,E]. La figura adjunta muestra los valores posibles mostrados por la matriz de leds.



Problema 9.19 Se desea controlar un paso de peatones constituido por el semáforo bimodal esquematizado en la figura adjunta:



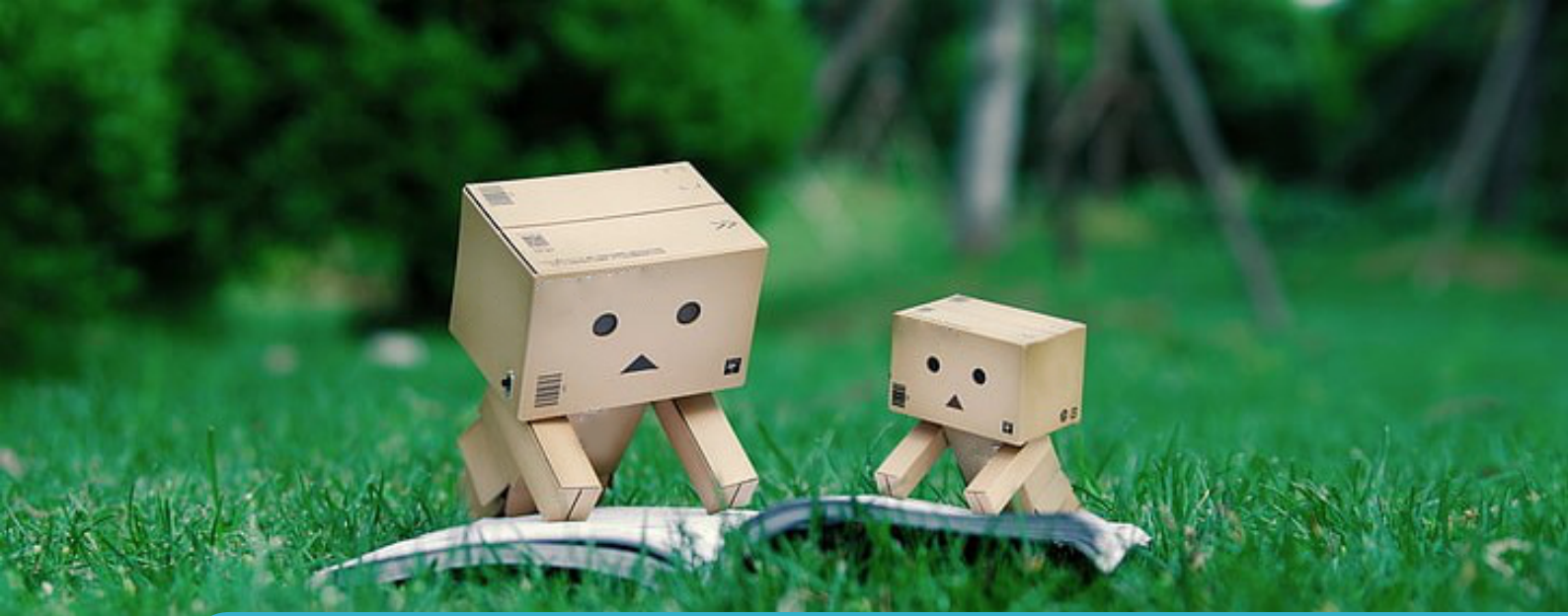
El semáforo se compone de tres luces (amarilla, roja y verde) que controlan el paso de vehículos con el significado estandarizado. Junto a estas se ubican otras tres luces en el semáforo encargadas de proporcionar indicaciones a los peatones: *pase* (verde), *espere* (rojo) y *pulse* (azul). El ciclo de funcionamiento del semáforo sigue las fases que se indican a continuación:

(1) En el estado inicial, el semáforo de vehículos está en verde y el de peatones en azul (*pulse*). Si ningún peatón solicita paso mediante pulsación, el sistema permanece en este estado de manera indefinida (*vehículos circulando*).
(2) Cuando un peatón solicita paso mediante pulsación, se enciende la luz roja (*espere*), mientras que la luz sigue en verde para los vehículos durante T_1 segundos.
(3) Al finalizar los T_1 segundos se enciende la luz amarilla para vehículos durante T_2 segundos.
(4) Transcurridos los T_2 segundos se enciende la luz roja para vehículos. Durante las fases anteriores, la luz roja para los peatones permanece encendida (*espere*) y por precaución continua encendida durante T_3 segundos antes de encenderse la luz verde (*pase*).
(5) La luz roja para los vehículos permanece encendida $T_3 + T_4$ segundos, transcurridos los cuales se enciende la luz verde para vehículos y la roja (*espere*) para peatones.
(6) Cuando pasan T_5 segundos el sistema regresa de nuevo al estado inicial con la luz azul (*pulse*) encendida. En este momento se vuelven a aceptar nuevas solicitudes de paso de los peatones.

Diseña un circuito secuencial síncrono que implemente la lógica de control del sistema descrita y permita su simulación con Logisim, considerando: $T_1 = 3$ s, $T_2 = 5$ s, $T_3 = 3$ s, $T_4 = 18$ s y $T_5 = 30$ s. Además, el sistema debe contar con una señal de reinicio que permita forzar —en cualquier instante— la evolución del sistema a su estado inicial.

IV

Obras de consulta e Índice Temático



Obras de consulta

[1] Carl Burch. Logisim: a graphical system for logic circuit design and simulation. *Journal on Educational Resources in Computing*, 2(1):5–16. Association for Computing Machinery (ACM), March 2002. DOI: 10.1145/545197.545199.

[2] Thomas L. Floyd. *Fundamentos de sistemas digitales*. Pearson Educación, Madrid, 11th edition, 2016. ISBN: 978-84-9035-300-4.

[3] David Harris and Sarah L. Harris. *Digital Design and Computer Architecture*. Elsevier Science & Technology, August 2012. ISBN: 978-01239-442-45.

[4] M. Morris Mano and Michael Ciletti. *Digital Design, Global Edition*. Pearson Education Limited, July 2018. ISBN: 978-12922-311-81.

[5] George Self. *Exploring digital Login with Logisim-Evolution*. Autoedición, 7th. Ed., 2019. Disponible online: https://github.com/grself/

[6] Javier Garcia Zubia, I. Angulo, y J. Angulo. *Sistemas digitales y tecnología de computadores*. Thomson, Madrid, 2 edition, 2007. ISBN: 978-84-9732-486-1.

[7] Javier García Zubia. *Problemas resueltos de electrónica digital*. Paraninfo, Madrid, España, 2012. ISBN: 978-84-9732-195-2.

Índice Temático

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Esta obra está concebida como manual docente para la asignatura de primer curso Tecnología de Computadores impartida en el grado en Ingeniería Informática de la Universidad de Castilla-La Mancha, aunque puede ser empleado en otras titulaciones para materias relacionadas con el Diseño y Análisis de Sistemas Digitales. El texto se divide en tres partes:

I.- Representación de información y lógica booleana (Caps. 1 a 3),
II.- Sistemas Combinacionales (Caps. 4 a 6), y
III.- Sistemas Secuenciales (Caps. 7 a 9).

El rasgo diferenciador de esta obra reside en su enfoque práctico mediante utilización intensiva de simulación lógica de circuitos digitales. De este modo se facilita la experimentación inmediata e intuitiva de los circuitos digitales, independientemente de su complejidad. En una primera aproximación, la simulación lógica de los circuitos permite obviar los aspectos prácticos de su realización física, otorgando más relevancia a los aspectos formales de su análisis y diseño lógico. La herramienta de simulación opensource elegida en esta obra ha sido Logisim [1] (www.cburch.com/logisim), un programa multiplataforma de software libre.

En los nueve capítulos de esta obra se incluyen 122 ejemplos prácticos explicados que cubren tanto el análisis como diseño de sistemas lógicos, y alrededor de 170 circuitos lógicos elaborados con Logisim, que se pueden recrear fácilmente. Cada capítulo finaliza con una sección de problemas propuestos, hasta un total de 174, cuya resolución se aborda en una obra complementaria —en preparación—.

Ediciones de la Universidad
de Castilla-La Mancha