#99

SERIE
COMPUTACIÓN

Elisa Viso Gurovich

Circuitos digitales

Manual de prácticas de laboratorio

AÑO
2004

VÍNCULOS
MATEMÁTICOS

Circuitos digitales

Elisa Viso G.

Departamento de Matemáticas
Facultad de Ciencias, UNAM. Marzo, 2013

Nº 99

Impreso en la Coordinación de Servicios Editoriales de la Facultad de Ciencias, UNAM

Índice general

Circuitos digitales

Elisa Viso G.
Facultad de Ciencias, UNAM

Aplicaciones del álgebra booleana

1.1. Implementación de fórmulas con circuitos digitales

Hoy en día nos encontramos en todos lados dispositivos electrónicos llamados *microprocesadores* que controlan automóviles, lavadoras, refrigeradores, I-pods, relojes digitales y un sin fin de aparatos eléctricos y electrónicos que usamos ya casi sin pensar. El cómo funcionan está determinado por los circuitos que contienen y que están modelados con lo que se conoce como *álgebra booleana*, la cual responde a la lógica proposicional.

Como en la lógica proposicional tenemos variables que pueden tomar únicamente dos valores, 0 y 1 –¿recuerdan el uso de interpretaciones?–, con tres funciones básicas:

- $\overline{v}$ que corresponde a $\neg v$.
- $v_1 + v_2$ que corresponde a $v_1 \vee v_2$.
- $v_1 \cdot v_2$ (o simplemente $v_1 v_2$) que corresponde a $v_1 \wedge v_2$.

Como cualquiera de los operadores restantes de la lógica proposicional ($\rightarrow, \leftrightarrow$) los podemos representar usando únicamente estos tres, tenemos todo lo que podemos necesitar[1].

Veamos algunos ejemplos de circuitos.

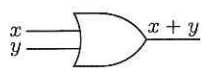
[1] A esta forma de construir fórmulas se le denomina *forma normal conjuntiva*.

Ejemplo 1.1.1 Construyamos un controlador para el aire acondicionado de una oficina. Queremos que si la temperatura ambiental sube a más de 28 grados el aire acondicionado se active; lo mismo si la humedad está por arriba del 50 %. El dispositivo tiene dos *entradas*, el estado de la temperatura (en realidad, sólo si la temperatura excede 28 grados) y si la humedad está por arriba o por abajo de 50 %. Como se ve, podemos representar ambas entradas con variables booleanas (lógicas) y cómo debe reaccionar el aire acondicionado con una tabla de verdad:

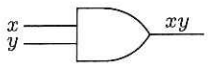
¿Temp > 28°?	¿Humedad >50 %?	¿Aire encendido?
no	no	no
no	sí	sí
sí	no	sí
sí	sí	sí

Podemos observar que esta tabla de verdad corresponde –si asociamos 1 con "sí" y 0 con "no"– a la tabla de verdad del conectivo lógico $\vee$, por lo que si asignamos la variable c a que la temperatura sea mayor a 28°, h a que la humedad sea mayor a 50 % y p a si el aire acondicionado está o no encendido, podemos pensar que la expresión booleana que modela nuestro controlador está dada por $\boldsymbol{p \equiv c \vee h}$, o bien, en términos de operaciones en el álgebra booleana $\boldsymbol{p = c + h}$.

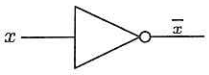
A las tres operaciones básicas del álgebra booleana les corresponden *compuertas lógicas*, que tienen, en general, una o más entradas y una única salida[2]. Las tres compuertas básicas se representan como sigue:



$x \vee y$

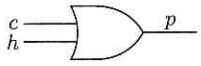


$x \wedge y$



$\neg x$

Para el ejemplo anterior, todo lo que necesitamos es una compuerta *or* que queda como sigue:



$p = c + h$

Ejemplo 1.1.2 Una impresora va a imprimir sólo si la impresora está en línea y el sensor de papel dice que hay papel.

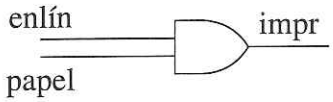
$$\text{en-línea} \wedge \text{papel} = \text{imprimir}$$

[2]Muchos autores utilizan compuertas con k entradas, $k \geqslant 2$, y una salida, aprovechando la propiedad de asociatividad de $\wedge$ y $\vee$.

La tabla que corresponde a esto es:

en línea	papel	imprimir
sí	sí	sí
sí	no	no
no	sí	no
no	no	no

Una compuerta *and* nos resuelve fácilmente el problema:

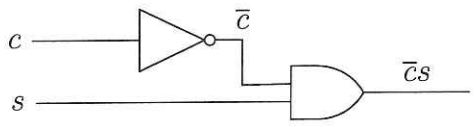


Ejemplo 1.1.3 El calentador de gas de una casa está conectado a dos termostatos, uno en la sala y el otro donde está el calentador. Si la temperatura de la sala baja a 17° o menos, el calentador se prende. Pero si la temperatura en el cuarto del calentador sube a más de 40°, el calentador se apaga.

$(\overline{c}\ s)$

cuarto >de 40	sala <17°	encender
0	0	0
0	1	1
1	0	0
1	1	0

Las compuertas correspondientes son:

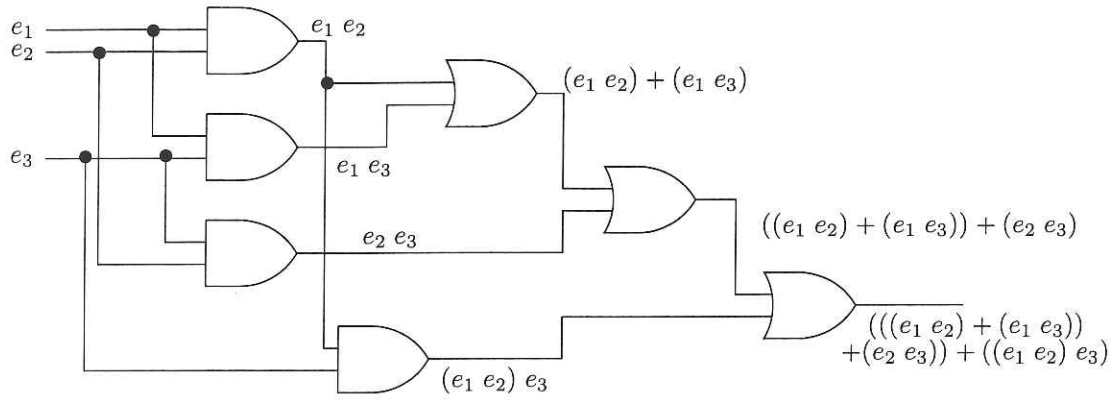


Ejemplo 1.1.4 Entre tres electores, la ley pasa si hay dos o más votando a favor.

La tabla queda como sigue:

elector 1	elector 2	elector 3	se aprueba
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

La fórmula que representa a esta tabla es $(e_1e_2) + (e_1e_3) + (e_2e_3) + (e_1e_2e_3)$ y el circuito correspondiente:



Cuando una línea muestra un punto, quiere decir que se saca la señal directamente del cable después de alimentarla, para garantizar que todas son la misma señal.

Ejemplo 1.1.5 Queremos construir un sumador de números de tres bits. El algoritmo para sumar números binarios es igual al que usamos para sumar números en base 10, por lo que procedemos de derecha a izquierda. Tenemos un acarreo cuando la suma de los números iguala o pasa la base. En el caso de base 2, la suma de dos bits responde a la siguiente tabla:

bit_1	bit_2	$suma$	$acarreo$
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

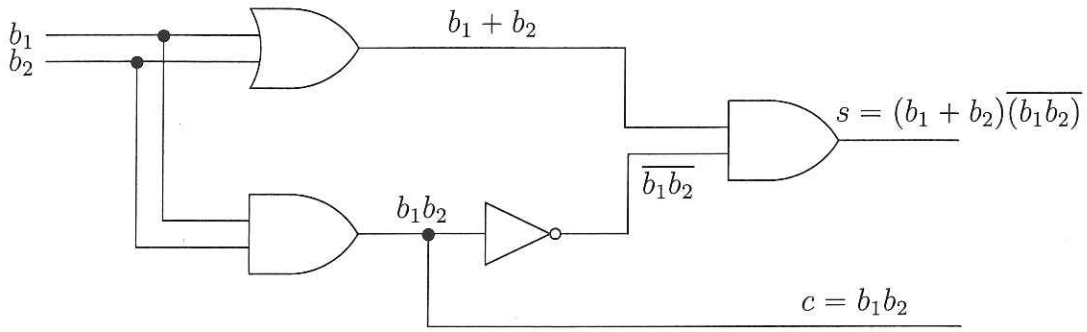
Es fácil construir un circuito lógico que produzca la suma y el acarreo de dos números de un solo bit. La tabla correspondiente es la que acabamos de dar.

Estamos frente a un dispositivo con dos entradas y dos salidas. Las entradas las constituyen los bits a sumar (B_1 y b_2), mientras que las salidas son la suma (s_1) y el acarreo (c_1). En general, cuando estamos haciendo la suma de dos cantidades de k bits podemos esperar una salida con $k+1$ bits, k de ellos las sumas correspondientes y el último bit que corresponderá al acarreo.

Si vemos la tabla notaremos que la suma es 1 cuando algunos de los dos bits es 1, pero no ambos (a esto se le conoce como el *o exclusivo* y se representa con $\oplus$). De lo anterior, la columna de la suma corresponde a la fórmula $s = (b_1 \wedge \neg b_2) \vee (\neg b_1 \wedge b_2)$ que en la notación que estamos siguiendo para los circuitos digitales, con $+$ en lugar de $\vee$, la multiplicación implícita en lugar de $\wedge$ y $\overline{x}$ en lugar de $\neg x$, queda como $s = (b_1\overline{b_2}) + (\overline{b_1}b_2)$ –que es equivalente a $(b_1 + b_2)(\overline{b_1 b_2})$. En cuanto al acarreo,

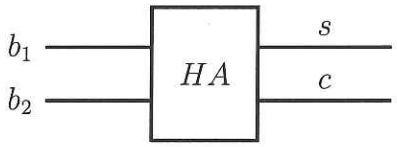
vemos que hay acarreo cuando ambos bits están en 1, o sea $b_1 \wedge b_2$ que corresponde a la expresión $c = b_1 b_2$. El circuito correspondiente se encuentra en la figura 1.1.

Figura 1.1. Circuito correspondiente a la suma de dos bits



A este circuito se le conoce como medio sumador (*half adder*) y lo podemos representar en forma compacta como una caja que recibe dos entradas y produce dos salidas, como se muestra en la figura 1.2.

Figura 1.2. Diagrama de bloque de un medio sumador



Si queremos sumar dos números de dos bits cada uno, tomamos el acarreo producido por la primera pareja de bits para sumarlo a la segunda pareja de bits:

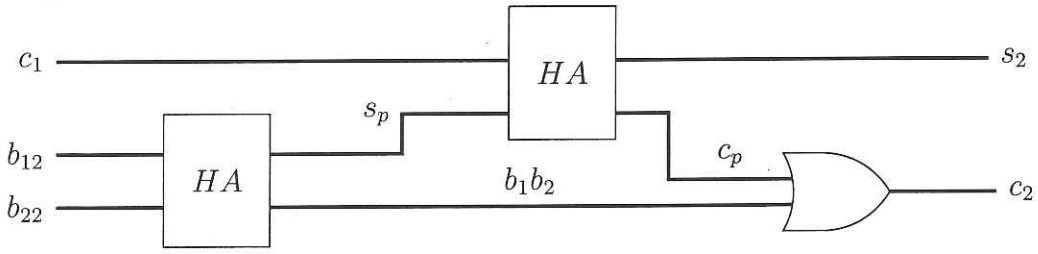
b_{12}	b_{22}	c_1	s_2	c_2
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Sin embargo, como nuestros medio sumadores sólo reciben dos entradas (no tres, como las que tenemos) tenemos que asociar y sumar los dos bits y obtener una suma

parcial y un acarreo parcial; usamos la suma parcial para sumarla al acarreo anterior y obtener la suma de la segunda posición. Para el acarreo, si hubo acarreo en la suma de los dos bits, ya no puede haber más acarreo al sumarle el acarreo parcial y viceversa: si no hubo acarreo en la suma de los dos bits, podrá haber acarreo al sumar el acarreo parcial. La tabla para este sumador queda como sigue:

b_{12}	b_{22}	s_p	c_1	s_2	c_2
0	0	0	0	0	0
0	0	0	1	1	0
0	1	1	0	1	0
0	1	1	1	0	1
1	0	1	0	1	0
1	0	1	1	0	1
1	1	0	0	0	1
1	1	0	1	1	1

Los únicos renglones un poco más complicados son los dos últimos, pues tienen acarreo provocado por los dos bits pero que no se refleja en la suma parcial, sino que hay que mandarlo al acarreo total de la suma. Hay que notar que cuando los dos bits a sumar están en 1, el acarreo se va a producir independientemente del valor del acarreo anterior; asimismo, el acarreo anterior ya no puede provocar un segundo acarreo, por lo que la suma de los dos bits produce directamente el segundo acarreo en este caso. El segundo acarreo también puede ser provocado, como se ve en la tabla, si la suma parcial es 1 y el acarreo anterior es 1, $s_p c_1$. Este último resultado lo obtenemos si a un medio sumador le alimentamos la suma parcial y el acarreo anterior. De esta discusión podemos diseñar nuestro circuito usando medios sumadores y una compuerta *or*:

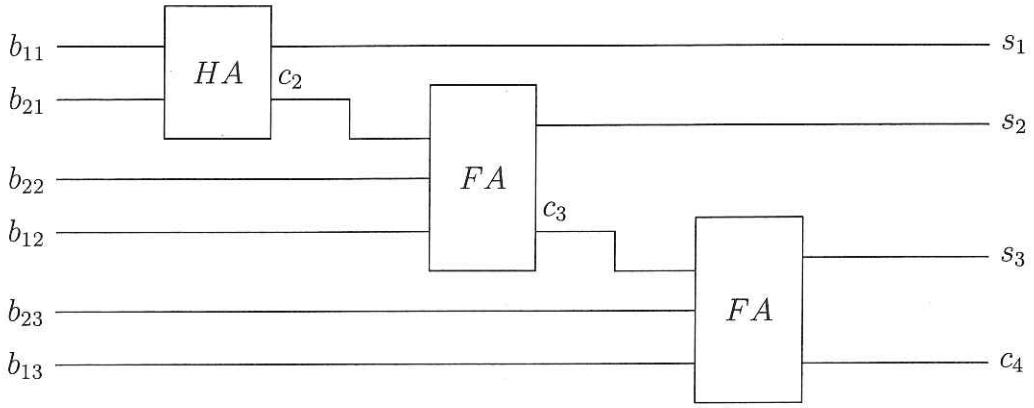


A la figura anterior es a lo que se conoce como un sumador completo (*full adder*) que recibe tres entradas y produce dos salidas, la suma y el acarreo.

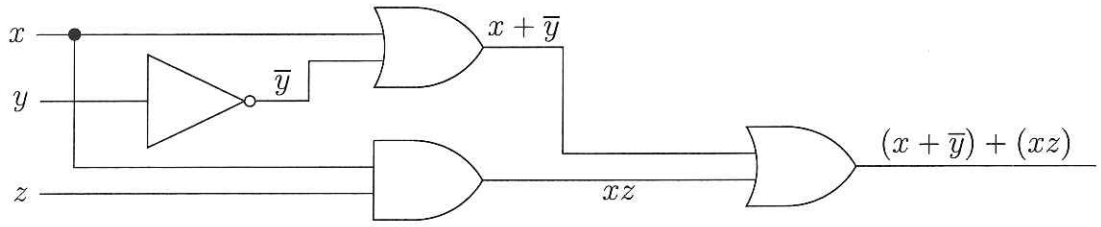
Para cumplir con nuestro objetivo original, donde queremos sumar cantidades de tres bits, vamos a usar un medio sumador, ya que la posición 0 no tiene acarreo, y dos sumadores completos para las otras dos posiciones, quedando nuestro circuito como se muestra en la figura 1.3 y que corresponde a la siguiente suma:

$$\begin{array}{cccc} & c_3 & c_2 & 0 \\ & b_{13} & b_{12} & b_{11} \\ & b_{23} & b_{22} & b_{21} \\ \hline c_4 & s_3 & s_2 & s_1 \end{array}$$

Figura 1.3. Sumador para cantidades de tres bits



Ejemplo 1.1.6 Tenemos el siguiente circuito lógico para la fórmula $(x + \overline{y}) + (xz)$.



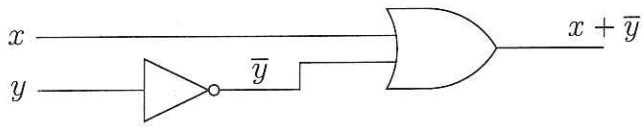
A este circuito le corresponde la siguiente tabla de verdad:

x	y	z	$\overline{y}$	$x + \overline{y}$	xz	$(x + \overline{y}) + (xz)$
0	0	0	1	1	0	1
0	0	1	1	1	0	1
0	1	0	0	0	0	0
0	1	1	0	0	0	0
1	0	0	1	1	0	1
1	0	1	1	1	1	1
1	1	0	0	1	0	1
1	1	1	0	1	1	1

En el último ejemplo nos preguntamos si hay alguna expresión más sencilla para este circuito. Observando la tabla de verdad vemos que la fórmula se evalúa a 1 cuando $\overline{y}$ es verdadera o cuando x es verdadera. Por lo tanto, podríamos "simplificar" esta fórmula para tener $x + \overline{y}$, que es mucho más sencilla y nos daría el mismo resultado –véase figura 1.4.

Figura 1.4. Fórmula simplificada $x + \overline{y} = (x + \overline{y}) + xz$

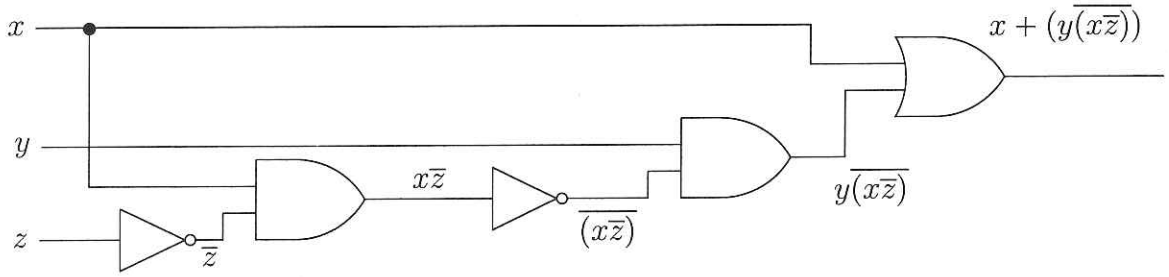
x	y	$\overline{y}$	$x + \overline{y}$
0	0	1	1
0	1	0	0
1	0	1	1
1	1	0	1



Una manera de encontrar esta equivalencia es utilizando derivaciones o lógica ecuacional:

$$
\begin{aligned}
(x + \overline{y}) + (xz) \equiv & (\overline{y} + x) + (xz) && \text{Conmutatividad} \\
\equiv & \overline{y} + (x + (xz)) && \text{Asociatividad} \\
\equiv & \overline{y} + x && \text{Eliminación de } \wedge \\
\equiv & \mathbf{x + \overline{y}}
\end{aligned}
$$

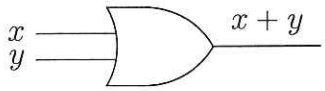
Ejemplo 1.1.7 Veamos el circuito que corresponde a la fórmula $x + (y\overline{(x\overline{z})})$.



Simplifiquemos esta fórmula:

$$
\begin{aligned}
x + (y(\overline{(x\overline{z})}) \equiv & x + (y(\overline{x} + z) && \text{De Morgan} \\
\equiv & (x + y)(x + (\overline{x} + z)) && \text{Distributividad} \\
\equiv & (x + y)((x + \overline{x}) + z) && \text{Asociatividad} \\
\equiv & (x + y)(true + z) && \text{Tercero excluido} \\
\equiv & (x + y)(true) && \text{Dominancia de la suma} \\
\equiv & (x + y) && \text{Identidad del producto}
\end{aligned}
$$

Lo que deja un circuito mucho más económico y sencillo que el anterior:



El problema con este método es que tenemos que tener una idea de a dónde queremos llegar (los dos lados de la equivalencia), lo que cuando queremos simplificar fórmulas no siempre es claro. Para ello tenemos un mecanismo para simplificar circuitos digitales que veremos a continuación.

Ejercicios

1.1.1.- Dada la siguiente descripción, construye la tabla de verdad que representa a esa función y el circuito digital correspondiente.

(a) Un circuito que implemente votación mayoritaria de cinco individuos.

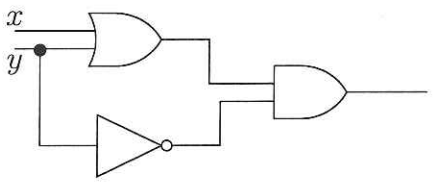
(b) Un circuito que compare dos números binarios de dos bits cada uno –(x_1, x_0), (y_1, y_0)– y regrese 0 si el segundo es mayor que el primero y 1 si el primero es mayor que el segundo.

(c) Un circuito que diga que una torta de jamón puede tener queso pero una torta de huevo no.

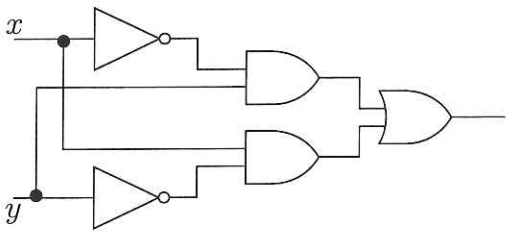
(d) Los refrescos que hay tienen gas si son de cola o fresa y no tienen gas si son de fresa o naranja. Un refresco no puede ser más que de un sabor.

1.1.2.- Describe la fórmula que corresponde a los siguientes circuitos:

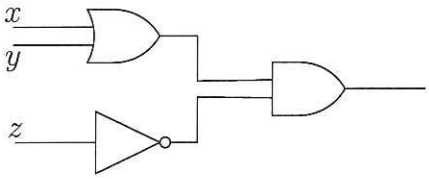
(a)



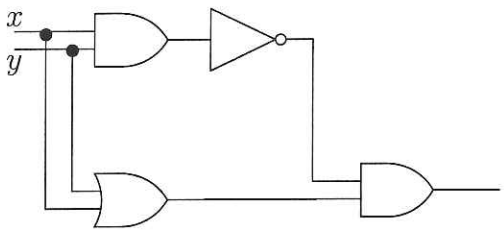
(c)



(b)



(d)



1.1.3.- Construye los circuitos digitales con compuertas AND, NOT y OR que correspondan a las siguientes fórmulas:

(a) $xyz + \overline{x}\,\overline{y}\,\overline{z}$

(b) $(x + y)\overline{x}\,\overline{y}$

(c) $(x + \overline{y} + z)(\overline{x} + y + \overline{z}$

(d) $\big((xy) + (\overline{x}\,\overline{y})\big)\overline{z}$

1.1.4.- Usando las propiedades del álgebra booleana, simplifica las siguientes fórmulas:

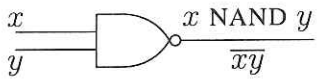
(a) $xyz + x\overline{y}z$

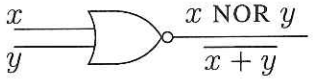
(b) $xy + x\overline{y}$

(c) $xy\overline{z} + x\overline{y}\,\overline{z}$

(d) $\overline{x} + \overline{y} + xyz$

1.1.5.- Otras dos compuertas disponibles son la compuerta NAND y la compuerta NOR representadas por los siguientes diagramas:





Para las siguientes fórmulas:

(a) $\overline{x}$ (b) $x + y$ (c) xy (d) $x \oplus y$

construye los circuitos utilizando únicamente

a) Compuertas NAND.

b) Compuertas NOR.

1.1.6.- Construye un medio restador (*half substracter*). Un medio sustractor tiene como entrada dos bits y produce como salida un bit para la diferencia y un bit para lo que se lleva.

1.1.7.- Construye un circuito para un restador completo (*full substractor*). Un restador completo tiene como entrada dos bits y un bit para lo que se lleva y produce como salida un bit para la diferencia y un bit de lo que se lleva.

1.2. Mapas de Karnaugh

La manera como hemos diseñado los circuitos lógicos hasta ahora ha sido, fundamentalmente, partiendo de la tabla de verdad. También hemos usado algunas equivalencias lógicas para simplificar estos circuitos. Sin embargo, esta manera de reducir las fórmulas no es mecanizable ni algorítmica, por lo que no nos garantiza que lleguemos a una buena solución.

En 1953 Maurice Karnaugh introdujo un método gráfico para minimizar funciones expresadas como disyunciones de conjunciones (sumas de productos en el álgebra booleana). A cada una de las conjunciones se le conoce como *mintérmino*. Empezaremos a trabajar con fórmulas de dos variables, para extender después a más variables, aunque el método introducido por Karnaugh y conocido como de *mapas de Karnaugh* no se presta para trabajar con más de seis variables.

1.2.1. Mapas de Karnaugh de dos variables

Si tenemos dos variables, digamos x e y, tenemos cuatro posibles *mintérminos*: xy, $\overline{x}y$, $x\overline{y}$ y $\overline{xy}$. La fórmula para representar este circuito utilizará aquellos mintérminos que sean 1 en la tabla de verdad. El mapa de Karnaugh de dos variables consiste de una retícula de dos por dos, donde cada columna está etiquetada con una de las variables y su negación y cada renglón con la otra variable y su negación; la celda que corresponde al renglón x columna y –(x, y)– va a contener un 1 si el mintérmino xy aparece en la fórmula.

	y	$\overline{y}$
x	xy	$x\overline{y}$
$\overline{x}$	$\overline{x}y$	$\overline{xy}$

En cada una de las celdas va a aparecer un 1 si ese mintérmino aparece en la expresión y nada si no aparece. Veamos algunos ejemplos:

	y	$\overline{y}$
x	1	
$\overline{x}$	1	

$xy + \overline{x}y$

	y	$\overline{y}$
x		1
$\overline{x}$	1	

$\overline{x}y + x\overline{y}$

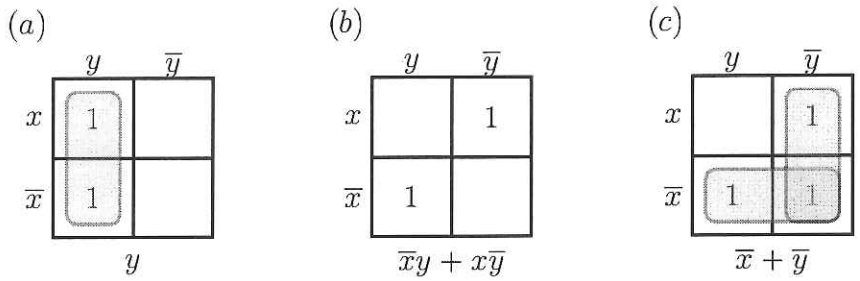
	y	$\overline{y}$
x		1
$\overline{x}$	1	1

$x\overline{y} + \overline{x}y + \overline{xy}$

Siempre que se encuentran dos mintérminos podemos combinar y eliminar a la variable que aparece también negada:

$$xy + \overline{x}y = (x + \overline{x})y = \mathsf{true}\, y = y$$

Gráficamente, esto se hace enmarcando aquellas celdas contiguas (vertical u horizontalmente). Un recuadro en una columna o renglón indica la eliminación de la variable que aparece tal cual y negada en esa columna o renglón. En los tres ejemplos que acabamos de dar, la simplificación se logra de la siguiente manera:



Si procedemos a simplificar usando equivalencias lógicas, obtenemos las siguientes secuencias:

(a) Para este mapa tenemos la siguiente sucesión de pasos:

$$\begin{aligned} xy + \overline{x}y &= (x + \overline{x})y && \text{Distributividad} \\ &= (true)y && \text{Tercero excluido} \\ &= y && \text{Identidad del producto} \end{aligned}$$

(b) Para el segundo mapa no podemos hacer nada gráficamente y tampoco algebraicamente.

(c) Para el tercer mapa tenemos los siguientes pasos:

$$\begin{aligned} x\overline{y} + \overline{x}y + \overline{x}\,\overline{y} &= x\overline{y} + \overline{x}\,\overline{y} + \overline{x}y + \overline{x}\,\overline{y} && \text{Usamos idempotencia para repetir } \overline{x}\,\overline{y} \\ &= (\overline{x} + x)\overline{y} + \overline{x}(y + \overline{y}) && \text{Distributividad} \\ &= (true)\overline{y} + \overline{x}(true) && \text{Tercero excluido} \\ &= \overline{y} + \overline{x} && \text{Identidad del producto} \\ &= \overline{x} + \overline{y} && \text{Conmutatividad} \end{aligned}$$

1.2.2. Mapas de Karnaugh con n variables

Como pudieron observar, cada celda en un mapa de Karnaugh representa a un estado para evaluar la fórmula. En el caso de dos variables el número de celdas es de $2^2 = 4$; si

tenemos tres variables tenemos que acomodar $2^3 = 8$ celdas, $2^4 = 16$ celdas para cuatro variables; y así sucesivamente. La regla para armar los mapas de Karnaugh es que el cambio entre dos celdas consecutivas, ya sea horizontales o verticales, no puede ser más que de una variable.

Veamos el caso de un mapa de Karnaugh para tres variables, x, y y z. Tenemos que representar en las celdas del mapa todas las combinaciones de estas tres variables: xyz,$xy\overline{z}$, $x\overline{y}z$, etcétera. Si vemos a las variables negadas como 0 y a las no negadas como 1, tendríamos que tener todos los estados que van del 000 al 111. Podemos acomodarlos en una retícula de dos renglones por cuatro columnas de la siguiente forma:

	yz	$y\overline{z}$	$\overline{y}\,\overline{z}$	$\overline{y}z$	
x	111 xyz	110 $xy\overline{z}$	100 $x\overline{y}\,\overline{z}$	101 $x\overline{y}z$	(1)
$\overline{x}$	011 $\overline{x}yz$	010 $\overline{x}y\overline{z}$	000 $\overline{x}\,\overline{y}\,\overline{z}$	001 $\overline{x}\,\overline{y}z$	(0)
	(11)	(10)	(00)	(01)	

Si observamos el mapa de Karnaugh anterior, en el que colocamos en cada celda a cuál mintérmino corresponde, podemos verificar que dos celdas contiguas, ya sea vertical u horizontalmente, sólo tienen un dígito de diferencia. Por ejemplo, a la izquierda de $xy\overline{z}$ (110) se encuentra la celda xyz (111) que únicamente tienen distinto el último dígito, mientras que abajo se encuentra el mintérmino $\overline{x}y\overline{z}$ (010) del que sólo se distingue por la primera posición. Esto es muy importante, pues cuando queramos cubrir celdas contiguas, tenemos que garantizar que dos celdas contiguas una de ellas tiene a alguna de las variables en 0 mientras que la otra celda tiene a esa misma variable en 1 y esa es la única diferencia.

Si hacemos el mapa de Karnaugh para la fórmula $xyz + xy\overline{z} + x\overline{y}\,\overline{z} + xy\overline{z} + \overline{x}yz + \overline{x}y\overline{z}$, el mapa (y su simplificación) queda como sigue:

	yz	$y\overline{z}$	$\overline{y}\,\overline{z}$	$\overline{y}z$
x	1	1	1	1
$\overline{x}$	1	1		

$= x + y$

El término x resulta de todo el renglón superior, ya que va a estar multiplicando a $(y+\overline{y})$ y $(z + \overline{z})$ que al distribuirlo queda sólo el término x.

$$
\begin{aligned}
xyz + xy\overline{z} + x\overline{y}\,\overline{z} + xy\overline{z} &= x(yz + y\overline{z} + \overline{y}z + \overline{y}\,\overline{z}) && \text{Distributividad}\\
&= x(y(z + \overline{z}) + \overline{y}(z + \overline{z})) && \text{Distributividad}\\
&= x(y(1) + \overline{y}(1)) && \text{Tercero excluido}\\
&= x(y + \overline{y}) && \text{Identidad de suma}\\
&= x(1) && \text{Tercero excluido}\\
&= x && \text{Identidad de suma}
\end{aligned}
$$

El término y resulta de que está multiplicando a $(x+\overline{x})$ y a $(z+\overline{z})$. Tenemos que repetir algunos de los sumandos, lo cual está permitido por la ley de idempotencia: $x + x = x$, $xx = x$.

$$
\begin{aligned}
xyz + xy\overline{z} + \overline{x}yz + \overline{x}y\overline{z} &= x(yz + y\overline{z}) + \overline{x}(yz + y\overline{z}) && \text{Distributividad}\\
&= x(y(z + \overline{z})) + \overline{x}(y(z + \overline{z}) && \text{Distributividad}\\
&= x(y(1)) + \overline{x}(y(1)) && \text{Tercero excluido}\\
&= xy + \overline{x}y && \text{Identidad de producto}\\
&= (x + \overline{x})y && \text{Distributividad}\\
&= 1y && \text{Tercero excluido}\\
&= y && \text{Identidad de producto}
\end{aligned}
$$

Como se puede ver, gráficamente se minimizó la expresión con mucho menos trabajo. Una expresión que tenía cuatro variables y seis mintérminos, cada uno de ellos involucrando a tres variables (dos compuertas por mintérmino) se redujo a dos variables y una sola compuerta.

Mapas de Karnaugh de cuatro variables

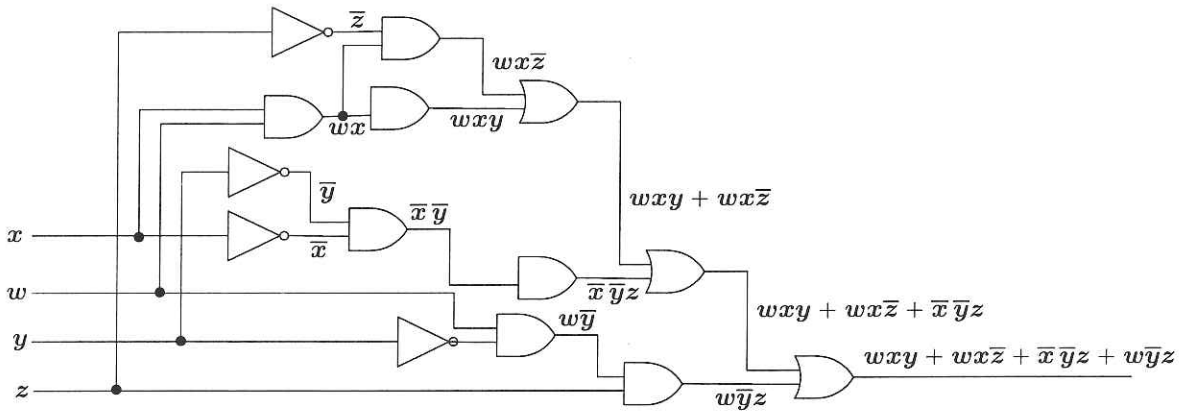
En el caso de cuatro variables tenemos 16 posibles estados ($2^4 = 16$), por lo que acomodaremos el mapa de Karnaugh en una retícula de 4×4, cuidando que entre celdas contiguas, ya sean verticales u horizontales, cambie exactamente una variable de positiva a negada o viceversa. El modelo para mapas de 4×4 se da en la figura 1.5.

Figura 1.5. Mapas de Karnaugh para cuatro variables

	$\overline{y}\,\overline{z}$ 00	$\overline{y}z$ 01	yz 11	$y\overline{z}$ 10
$w\overline{x}$ 10	$w\overline{x}\,\overline{y}\,\overline{z}$	$w\overline{x}\,\overline{y}z$	$w\overline{x}yz$	$w\overline{x}y\overline{z}$
wx 11	$wx\overline{y}\,\overline{z}$	$wx\overline{y}z$	$wxyz$	$wxy\overline{z}$
$\overline{w}x$ 01	$\overline{w}x\overline{y}\,\overline{z}$	$\overline{w}x\overline{y}z$	$\overline{w}xyz$	$\overline{w}xy\overline{z}$
$\overline{w}\,\overline{x}$ 00	$\overline{w}\,\overline{x}\,\overline{y}\,\overline{z}$	$\overline{w}\,\overline{x}yz$	$\overline{w}\,\overline{x}yz$	$\overline{w}\,\overline{x}y\overline{z}$

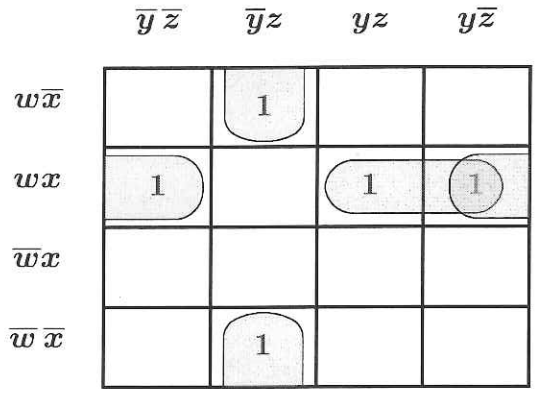
Supongamos que queremos simplificar la expresión $(wxy) + (wx\overline{z}) + (\overline{x}\,\overline{y}z) + (w\overline{y}z)$, cuyo circuito digital se encuentra en la figura 1.6.

Figura 1.6. Circuito correspondiente a $(wxy) + (wx\overline{z}) + (\overline{x}\,\overline{y}z) + (w\overline{y}z)$



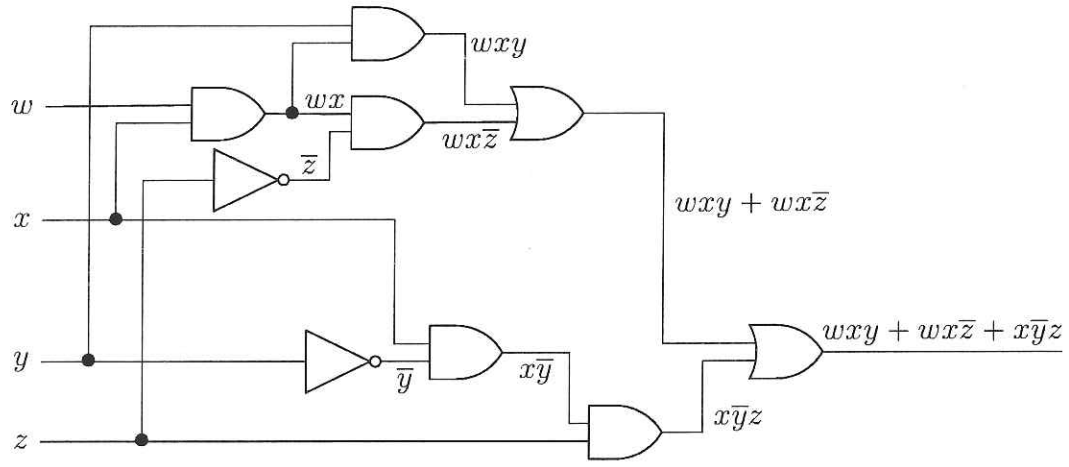
Para hacer los mapas de Karnaugh, aquellas variables que no aparecen las marcamos en 1 y en 0, pues quiere decir que el mintérmino está multiplicando por $(var + \overline{var})$. Procedemos a llenar el mapa de Karnaugh correspondiente:

$$\begin{aligned} wxy + wx\overline{z} + \overline{x}\,\overline{y}x &= wxy(z + \overline{z}) + wx(y + \overline{y})\,\overline{z} + (w + \overline{w})\overline{x}\,\overline{y}z \\ &= wxyz + wxy\overline{z} + wxy\overline{z} + wx\overline{y}\,\overline{z} + w\overline{x}\,\overline{y}z + \overline{w}\,\overline{x}\,\overline{y}z \\ &= wxyz + wxy\overline{z} + wx\overline{y}\,\overline{z} + w\overline{x}\,\overline{y}z + \overline{w}\,\overline{x}\,\overline{y}z \end{aligned}$$



$$wxyz + wxy\overline{z} + wx\overline{y}\,\overline{z} + w\overline{x}\,\overline{y}z + \overline{w}\,\overline{x}\,\overline{y}z =$$
$$= wxy + wx\overline{z} + x\overline{y}z$$

El circuito digital queda como se muestra a continuación. Vale la pena notar que en lugar de 14 compuertas se utilizaron 9, que representa un 36 % de reducción en el número de compuertas.



Ejercicios

1.2.1.- Encuentra las sumas de productos representados por los siguientes mapas de Karnaugh de dos variables (sin minimizar):

(a)

	y	$\overline{y}$
x	1	
$\overline{x}$	1	1

(b)

	y	$\overline{y}$
x	1	1
$\overline{x}$		

(c)

	y	$\overline{y}$
x	1	
$\overline{x}$		1

(d)

	y	$\overline{y}$
x	1	1
$\overline{x}$	1	1

1.2.2.- Encuentra las sumas de productos representados por los siguientes Mapas de Karnaugh de tres variables (sin minimizar):

(a)

	yz	$y\overline{z}$	$\overline{y}\,\overline{z}$	$\overline{y}z$
x	1		1	
$\overline{x}$		1		1

(b)

	yz	$y\overline{z}$	$\overline{y}\,\overline{z}$	$\overline{y}z$
x	1	1		
$\overline{x}$	1		1	1

(c)

	yz	$y\overline{z}$	$\overline{y}\,\overline{z}$	$\overline{y}z$
x	1	1	1	1
$\overline{x}$	1			1

(d)

	yz	$y\overline{z}$	$\overline{y}\,\overline{z}$	$\overline{y}z$
x		1	1	
$\overline{x}$			1	1

1.2.3.- Minimiza los Mapas de Karnaugh de los ejercicios (1.2.1) y (1.2.2).

1.2.4.- Usa Mapas de Karnaugh para minimizar las siguientes funciones de dos variables:

a) $\overline{x}y + \overline{x}\,\overline{y}$

b) $xy + x\overline{y}$

c) $x\overline{y} + xy + \overline{x}y$

d) $\overline{x}\,\overline{y} + \overline{x}y + xy$

1.2.5.- Usando Mapas de Karnaugh minimiza las siguientes funciones de tres variables. Dibuja los circuitos correspondientes a las funciones originales y a las funciones minimizadas.

a) $xyz + \overline{x}\,yz$

b) $xy\overline{z} + x\overline{y}\,\overline{z} + \overline{x}y\overline{z} + \overline{x}\,\overline{y}\,\overline{z}$

c) $\overline{x}yz\big((x + \overline{z}) + (\overline{y} + \overline{z})\big)$

d) $\overline{x}yz + \overline{x}\,\overline{y}\,\overline{z}$

1.3. Método de Quine-McCluskey para minimización

El problema con los mapas de Karnaugh es que trabajar con más de seis variables de manera gráfica es imposible. Lo es para cinco o seis variables, pues el representar la retícula con 32 o 64 celdas no es fácil de manejar.

El método de Quine-McCluskey fue diseñado en la década de 1950. Consiste en manejar los mintérminos como cadenas de bits (ceros y unos). Todas las cadenas tienen el mismo número de posiciones y a cada variable se le asigna la misma posición en cada cadena. Si

la literal aparece tal cual se le asigna un 1 en esa posición y si aparece negada se le asigna un 0. Cuando la variable no aparece, que significa que aparecía tanto en positivo como negada, se marca la posición con un guión. Veamos la codificación de un ejemplo con cuatro variables que corresponde a la siguiente tabla de verdad:

w	x	y	z	$f(w,x,y,z)$
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1

w	x	y	z	$f(w,x,y,z)$
1	0	0	0	1
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	0
1	1	1	0	1
1	1	1	1	0

Para el método de Quine-McCluskey podemos observar que la codificación de los mintérminos consiste, precisamente, de las cadenas de ceros y unos en la primera columna.

A continuación clasificamos las cadenas que correspondan a un mintérmino (donde $f(w,x,y,z) = 1$) por el número de unos que tengan, quedando en la misma clase aquellas cadenas que tengan el mismo número de unos. Los listamos en orden descendente por el número de unos:

Núm.	**mintérmino**	**# de unos**	
(1)	0111	3	✓
(2)	1011		✓
(3)	1110		✓
(4)	0101	2	✓
(5)	0110		✓
(6)	1010		✓
(7)	1100		✓
(8)	0010	1	✓
(9)	0100		✓
(10)	1000		✓
(11)	0000	0	✓

A continuación procedemos a la minimización. La presencia de una variable y su negación quiere decir que hay dos mintérminos en los cuales la posición correspondiente a la variable aparece con 0 en una y con 1 en la otra. Aquella en la que aparece esa posición con 1 tiene un 1 más que la otra, por lo que debemos comparar parejas de mintérminos que tienen un 1 de diferencia. Por lo tanto comparamos cada uno de los que tienen tres unos con los que tienen dos unos, los de dos unos con los de un uno, y los de un uno con los de cero unos. Aquellos que sean distintos por una única posición y que tengan en esa posición 0 y 1, se sustituyen por un mintérmino que tenga un guión en esa posición:

Combinación	Mintérm.	Cadena	Id.	
(1) y (4)	$\overline{w}xz$	01-1	(12)	✓
(1) y (5)	$\overline{w}xy$	011-	(13)	✓
(2) y (6)	$w\overline{x}y$	101-	(14)	
(3) y (5)	xyz	-111	(15)	
(3) y (6)	$wy\overline{z}$	1-10	(16)	✓
(3) y (7)	$wx\overline{z}$	11-0	(17)	✓
(4) y (9)	$\overline{w}xy$	011-	(18)=(13)	
(5) y (8)	$\overline{w}y\overline{z}$	0-10	(19)	✓
(5) y (9)	$\overline{w}x\overline{z}$	01-0	(20)	✓
(6) y (8)	$\overline{x}y\overline{z}$	-010	(21)	✓
(6) y (10)	$w\overline{x}\,\overline{z}$	10-0	(22)	✓
(7) y (9)	$x\overline{y}\,\overline{z}$	-100	(23)	✓
(7) y (10)	$w\overline{y}\,\overline{z}$	1-00	(24)	✓
(8) y (11)	$\overline{w}\,\overline{x}\,\overline{z}$	00-0	(25)	✓
(9) y (11)	$\overline{w}\,\overline{y}\,\overline{z}$	0-00	(26)	✓
(10) y (11)	$\overline{x}\,\overline{y}\,\overline{z}$	-000	(27)	✓
(12) y (20)	$\overline{w}x$	01--	(28)	
(16) y (19)	$y\overline{z}$	--10	(29)	✓
(16) y (24)	$w\overline{z}$	1--0	(30)	✓
(17) y (20)	$x\overline{z}$	1--0	(31)=(30)	✓
(17) y (22)	$w\overline{z}$	1--0	(32)=(31)	✓
(19) Y (26)	$\overline{w}\,\overline{z}$	0--0	(33)	✓
(20) y (25)	$\overline{w}\,\overline{z}$	0--0	(34)=(33)	✓
(21) y (27)	$\overline{x}\,\overline{z}$	-0-0	(35)	
(23) y (27)	$\overline{y}\,\overline{z}$	--00	(36)	✓
(24) y (26)	$\overline{y}\,\overline{z}$	--00	(37)=(36)	✓

Continúa en la siguiente página...)

Continúa de la página anterior...

Combinación	Mintérm.	Cadena	Id.	
(29) y (36)	$\overline{z}$	---0	(38)	✓
(29) y (37)	$\overline{z}$	---0	(39)=(38)	✓
(32) y (34)	$\overline{z}$	---0	(40)=(39)	✓
(33) y (36)	$\overline{z}$	---0	(41)=(40)	

De la tabla anterior tenemos la siguiente fórmula, que representa al circuito dado:

$$f(w,x,y,z) = w\overline{x}y + xyz + \overline{w}xy + \overline{w}x + \overline{x}\,\overline{z} + \overline{z}$$

La fórmula original requiere de 37 compuertas para implementarla (sin considerar ningún tipo de reutilización de productos ya calculados, ni uso de factorización) mientras que la simplificación obtenida requiere de únicamente 11 compuertas (el lector interesado lo puede verificar).

Hay que notar que el algoritmo utilizado para la minimización presupone la comparación dos a dos de todos los mintérminos que tienen k unos contra todos los términos que tienen $k-1$ unos, lo que hace que la complejidad del algoritmo lo convierta en un algoritmo *intratable*, algo de lo que no había conciencia sino hasta 1976, en que se definió la teoría de complejidad de algoritmos. Por ello podemos decir que no hay algoritmo razonable para minimizar fórmulas booleanas.

Ejercicios

1.3.1.- Usa el método Quine-McCluskey para minimizar la función de la figura 1.6.

1.3.2.- Usa el método Quine-McCluskey para minimizar las funciones del ejercicio (1.3.5).

1.4. Tabletas (*chips*) de compuertas

Para fabricar circuitos lógicos se dispone de conjuntos de compuertas con un número fijo de compuertas iguales, presentadas en una tableta (*chip*), por lo que muchas veces se usan las equivalencias lógicas para utilizar más eficientemente las compuertas lógicas. En la figura 1.7 se muestran distintos tipos de tabletas disponibles.

Figura 1.7. Algunas tabletas disponibles con compuertas lógicas (1/2)

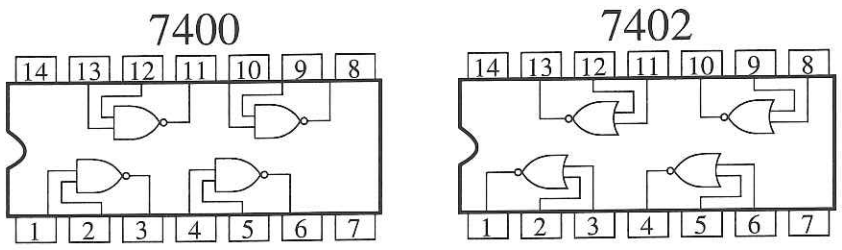
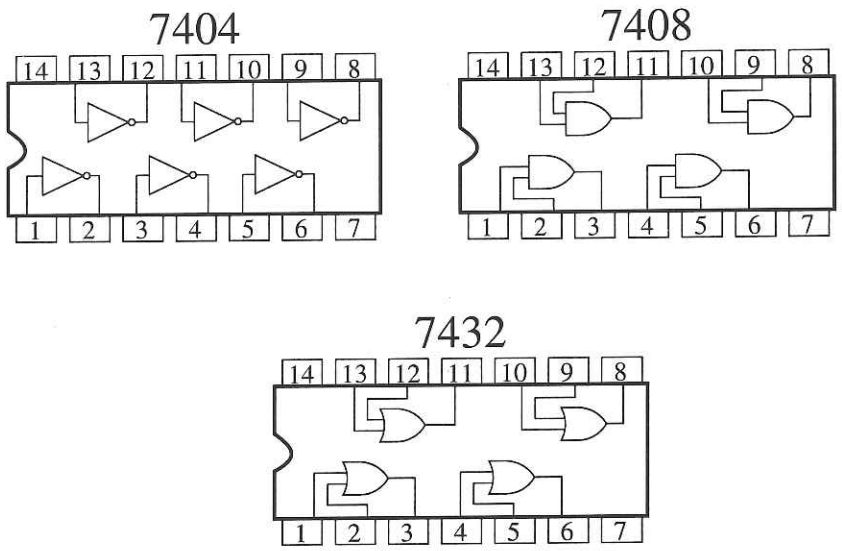
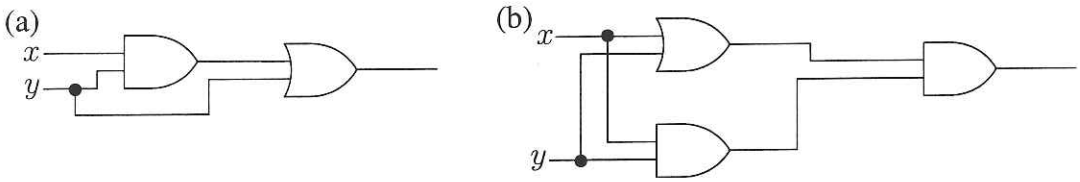


Figura 1.7. Algunas tabletas disponibles con compuertas lógicas (2/2)

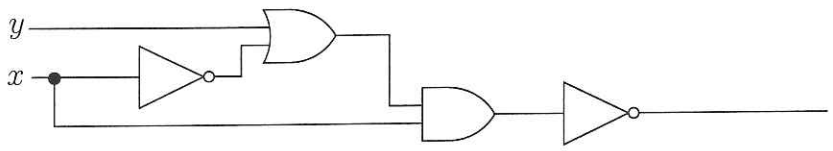


Ejercicios

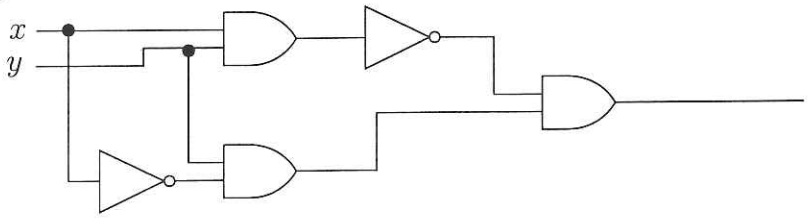
1.4.1.- Escribe las expresiones booleanas asociadas a los siguientes circuitos:



(c)



(d)



1.4.2.- Dibuja un circuito que represente a cada una de las siguientes expresiones booleanas.

(a) $(xy) + (\overline{x} + y)$

(b) $\overline{x}y + \big(x(yz)\big)$

(c) $(\overline{y}\,\overline{z}) + \overline{\big((w\overline{x})\overline{y}\big)}$

(d) $\big(x(yz)\big)\big((\overline{x}\,\overline{y}) + (z\overline{w})\big)$

1.4.3.- Para las siguientes expresiones booleanas, da el resultado que se presenta en el estado dado:

(a) $(x + y)(\overline{x} + z)$; estado: $(x = 1, y = 1, z = 0)$.

(b) $\big((xy) + z\big)\big(x + (\overline{y}\,z)\big)$; estado: $x = 0, y = 1, z = 1$.

(c) $\overline{x(yz)}$; estado: $x = 0, y = 1, z = 0$.

(d) $\big(x(y + \overline{z})(\overline{x} + \overline{z})\big)$; estado: $x = 0, y = 1, z = 1$.

1.4.4.- Construye una tabla de verdad para las siguientes expresiones booleanas:

(a) $x(y + \overline{x})$

(b) $\overline{x + \overline{y}} + x$

(c) $(x + \overline{y}) + (x\,\overline{z})$

(d) $\big((xy)z\big) + \big(x(y\overline{z})\big)$

1.4.5.- La luz de una escalera se controla con tres apagadores, una fuera de la casa, una al pie de la escalera y otra al final de la escalera. Debe ser posible prender o apagar la luz desde cualquiera de estos apagadores, sin importar cuál es la posición de cualquiera de los apagadores. Diseña un circuito que haga esto posible. (Pista: siempre que el número de apagadores prendidos sea impar, la luz estará prendida.)

1.4.6.- Usando mapas de Karnaugh simplifica el circuito correspondiente a las siguientes tablas de verdad.

(a)

x	y	z	$x+(y(x\overline{z}))$
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

(b)

x	y	z	$\overline{x}yz+x(\overline{y}z+y\overline{z}+yz)$
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

(c)

x	y	z	$x+\big(y(\overline{x}+\overline{z})\big)$
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

(d)

x	y	$x(x+y)$
0	0	0
0	1	0
1	0	1
1	1	1

1.4.7.- Usa mapas de Karnaugh para simplificar la siguiente expresión booleana:

$$(wxy)+(wx\overline{z})+(w\overline{y}z)+(\overline{x}\overline{y}z)$$

1.4.8.- Escribe las expresiones que corresponden a los óvalos marcados en los siguientes mapas de Karnaugh.

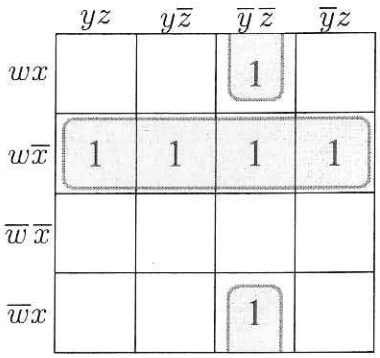
(a)

	y	$\overline{y}$
x	1	1
$\overline{x}$	1	

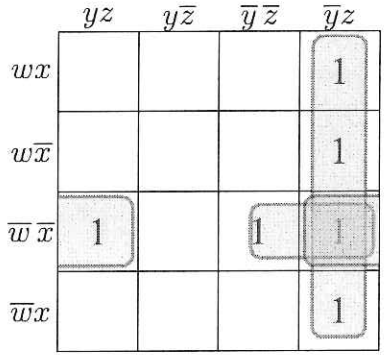
(b)

	yz	$y\overline{z}$	$\overline{y}\,\overline{z}$	$\overline{y}z$
x	1	1	1	
$\overline{x}$	1	1		

(c)



(d)



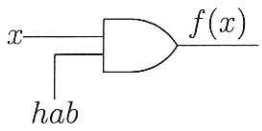
1.5. Dispositivos digitales combinatorios

A partir de circuitos lógicos se pueden construir dispositivos digitales que son característicos de las computadoras actuales. Los circuitos lógicos pueden ser de naturaleza *combinatoria*, que consisten de un conjunto de compuertas lógicas cuyas salidas, en todo momento, se calculan usando operaciones lógicas directamente de la combinación de sus valores de entrada. Ya vimos uno de estos dispositivos que es el sumador (tanto el medio como el completo).

Por otro lado tenemos circuitos lógicos que tienen elementos cuya función es almacenar algunos de los valores producidos por el mismo circuito y realimentados al mismo. En este tipo de circuitos se debe tomar en cuenta lo que se conoce como el retardo (*delay*) del dispositivo, ya que el paso de una señal por una compuerta se lleva alrededor de 10 nanosegundos. Esto quiere decir, por ejemplo, que si la salida de una compuerta se alimenta a la entrada de otra, esto va sucediendo en orden y con un cierto retardo. Si además la compuerta a la que se retroalimenta es ella misma o alguna que recibe sus primeras entradas al mismo tiempo que ésta, se debe tomar en cuenta dicho retardo. Es esta característica la que da el nombre de *secuencial* a este tipo de circuitos o dispositivos. Continuaremos por el momento trabajando con circuitos combinatorios.

Muchos de los dispositivos que presentaremos cuentan con una entrada que *habilita* al dispositivo; funciona como un interruptor que puede estar prendido o apagado y que cuando está prendido permite que el circuito funcione. Por ejemplo, si queremos un dispositivo que deje pasar la señal cuando esté habilitado pero la inhiba cuando no, podemos pensar en una compuerta *and* donde una de las entradas es la señal de habilitación y la otra es la señal que se desea transmitir –véase figura 1.8.

Figura 1.8. Circuito que inhibe o deja pasar una señal



$habilitado = 0$	
x	$f(x)$
0	0
1	0

Dispositivo apagado

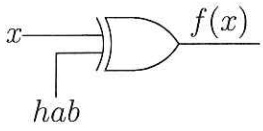
$habilitado = 1$	
x	$f(x)$
0	0
1	1

Dispositivo prendido

En este caso se puede pensar en la entrada que corresponde a x como la línea de entrada, mientras que la que corresponde a *habilitar* como una línea de control.

Otro dispositivo común es lo que se conoce como inversor. En este caso el papel del dispositivo es, simplemente, complementar la señal, en caso de que esté habilitado. Una compuerta xor funciona como un inversor si consideramos a una de las entradas como la línea de dato y la otra entrada como la señal de control –véase figura 1.9.

Figura 1.9. Inversor usando una compuerta xor



$Invertir = 0$	
x	$f(x)$
0	0
1	1

Dispositivo apagado

$Invertir = 1$	
x	$f(x)$
0	1
1	0

Dispositivo prendido

1.5.1. Multiplexores

Un multiplexor es un dispositivo que permite elegir una de varias entradas para ser emitida como salida del dispositivo. Consiste de k líneas de entrada, $\lceil \log_2 k \rceil$ líneas de control –que permiten elegir alguna de las k entradas– y una línea de salida que corresponde a la línea de entrada elegida.

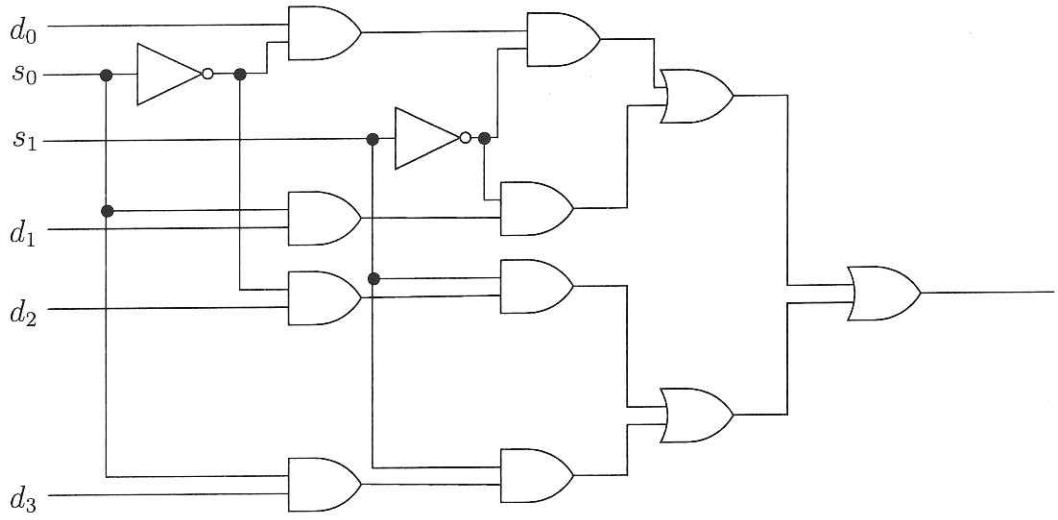
Si suponemos que tenemos cuatro líneas de entrada a elegir, d_0, d_1, d_2 y d_3, requerimos de dos líneas de control para elegir, s_0 y s_1. La tabla de cuál línea elegir se encuentra a

continuación.

s_1	s_0	se elige la línea
0	0	0
0	1	1
1	0	2
1	1	3

El dispositivo digital que selecciona una de cuatro entradas se puede ver en la figura 1.10.

Figura 1.10. Multiplexor de cuatro entradas



La tabla de verdad para este circuito se encuentra a continuación.

s_0	s_1	d_0	d_1	d_2	d_3	Out
0	0	1	–	–	–	1
0	1	–	1	–	–	1
1	0	–	–	1	–	1
1	1	–	–	–	1	1

Esta tabla muestra particularidades que no hemos revisado hasta ahora. La combinatoria nos dice que la tabla debería tener $2^6 = 64$ renglones y, sin embargo, esta tabla tiene únicamente cuatro renglones, uno por cada posible combinación de s_0 y s_1. Los guiones en la tabla es lo que se conoce como condiciones que no importan (*don't care*, ya que el resultado no va a tomar en cuenta esos valores. En los otros 60 casos el circuito entregará el valor

de falso. De esta tabla podemos dar una expresión booleana relativamente sencilla para el multiplexor con cuatro entradas:

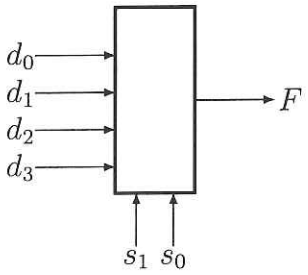
$$\overline{s_1}\,(\overline{s_0}d_0) + \overline{s_1}(s_0d_1) + s_1(\overline{s_0}d_2) + s_1(s_0d_3)$$

Los paréntesis en esta expresión se usan para seguir usando compuertas de dos entradas únicamente, aunque podríamos usar compuertas de tres o cuatro entradas y reducir el nivel del circuito a dos, lo que representa un menor tiempo de ejecución con un costo mayor por compuerta.

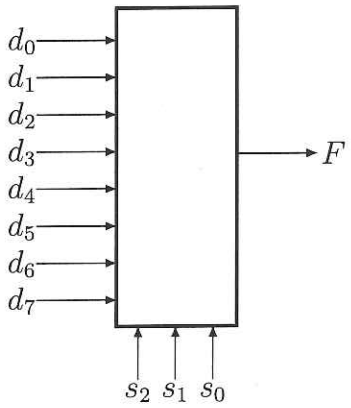
Se utilizan diagramas de bloque para este tipo de dispositivos. Al multiplexor de 4 entradas, por ejemplo, le corresponde el diagrama de la figura 1.11(a) y a un multiplexor de 8 entradas le corresponde el diagrama de bloque de la figura 1.11(b).

Figura 1.11. Diagramas de bloque para multiplexores

(a) Con 4 entradas



(b) Con 8 entradas



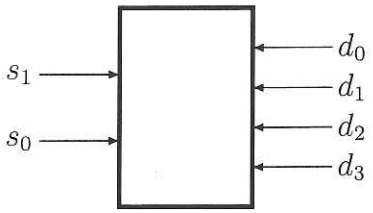
Como se puede ver en estas figuras, las líneas que seleccionan funcionan en notación binaria: para elegir entre cuatro posibles entradas se requieren dos líneas (posiciones binarias), de $00 = 0$ a $11 = 3$, pero para elegir entre 8 posibles datos se requiere de tres líneas (bits) que van de $000 = 0$ a $111 = 7$.

1.5.2. Decodificadores binarios

Un *decodificador* es un dispositivo que recibe un número binario de m bits –que puede ser visto como m líneas de señal– y produce como salida otro número binario de n bits –m y n pueden ser iguales. El diagrama de bloque de un decodificador con dos líneas de entrada y cuatro líneas de salida se puede ver en la figura 1.12.

Figura 1.12. Decodificador binario de 2×4

(a) Diagrama de bloque



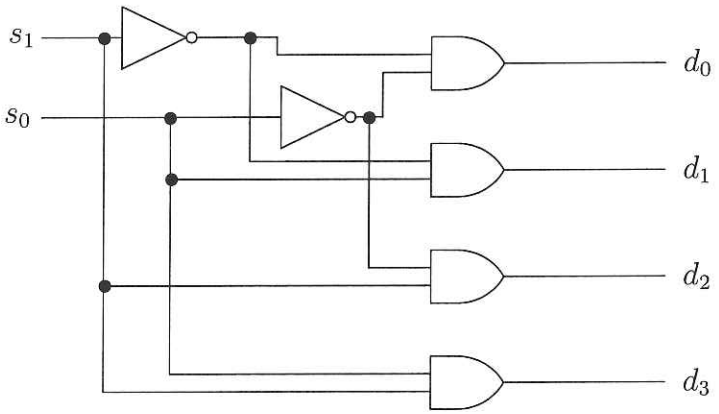
(b) Tabla de verdad

s_1	s_0	d_0	d_1	d_2	d_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

El decodificador ilustrado recibe un número binario entre cero y tres y elige la línea que corresponde a ese número para que valga 1 mientras el resto de las líneas valen 0.

El circuito digital que corresponde a la tabla de verdad se encuentra en la figura 1.13.

Figura 1.13. Implementación de un decodificador de 2×4



Ejercicios

1.5.1.- Diseña un multiplexor $32 : 1$ usando dos diagramas de bloque de multiplexores $16 : 1$ (puedes utilizar compuertas adicionales).

1.5.2.- Utiliza dos decodificadores de $2 : 4$ (los diagramas de bloque) para diseñar un decodificador de $4 : 8$ (puedes utilizar compuertas adicionales).

1.5.3.- Diseña la tabla de verdad de un decodificador $3 : 10$ de binario a decimal.

1.5.4.- Diseña la tabla de verdad para un decodificador $4 : 16$.

1.5.5.- Diseña un circuito de codificador con cuatro entradas para la tabla de verdad que se encuentra a continuación (la X significa que no importa si es 0 o 1):

Entradas				Salidas		
D_3	D_2	D_1	D_0	A_1	A_0	V
0	0	0	0	X	X	0
0	0	0	1	0	0	1
0	0	1	X	0	1	1
0	1	X	X	1	0	1
1	X	X	X	1	1	1

A este circuito se le conoce como un codificador de prioridad. Noten que, a diferencia de los decodificadores, este tipo de circuitos combinatorios tienen más entradas que salidas.

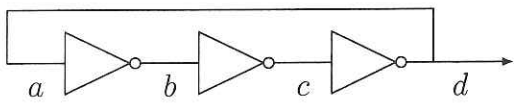
1.6. Circuitos digitales secuenciales

Para construir componentes de computadoras es necesario contar con circuitos capaces de "recordar", de representar un *estado*. Esto se logra haciendo que la respuesta del circuito se rija tanto por las entradas actuales como por el resultado de entradas anteriores; a esto último se le conoce como retroalimentación (*feedback*).

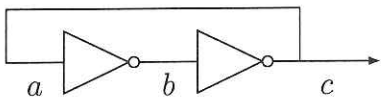
Las figuras 1.14(a) y 1.14(b) presentan dos circuitos secuenciales simples.

Figura 1.14. Circuitos simples con retroalimentación

(a) Circuito inestable



(b) Circuito estable



Tratemos de analizar el comportamiento de estos circuitos secuenciales. El circuito de la figura 1.14(a) tiene tres compuertas *not* que "voltean" el valor de la entrada. Si suponemos que el valor de d es 1 en algún momento, se invierte al pasar p[ro la primer compuerta teniendo 0 como valor de b, se vuelve a invertir sal pasar p;or la segunda compuerta teniendo 1 como el valor de c y la tercera compuerta vuelve a invertir el valor, quedando d con el valor de 0; si observamos el circuito, al cabo de [poco tiempo el valor se invierte nuevamente, valiendo 1; y así sucesivamente. Si nos olvidamos un poco del retardo entre una

compuerta y otra, obtendremos valor distinto dependiendo en el momento en que observemos el valor de d. Si se desea saber el valor de la señal d en un momento dado, ésta puede valer 0 o 1. A esto se le llama un circuito *inestable*, ya que no conserva el mismo valor en ausencia de nuevas entradas. En cambio, en el circuito de la figura 1.14(b), si suponemos un valor de 1 para c, al pasar por la primera compuerta se invierte con b valiendo 0 y al pasar por la segunda compuerta se vuelve a invertir, dando a c el valor de 1. En cualquier momento en que se pruebe el valor de c éste seguirá siendo 1 –similarmente si se inicia con un valor de 0–. Decimos entonces que el circuito es *estable*.

Como las compuertas digitales requieren de señal eléctrica para funcionar suponemos que al encenderse el primer valor detectado en la salida es aleatorio en el caso de los dos circuitos secuenciales que acabamos de revisar.

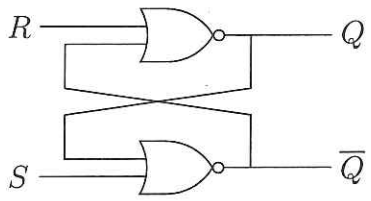
Los circuitos combinatorios están construidos a partir de compuertas, mientras que los circuitos secuenciales tienen como componente fundamental a los circuitos de pestillos –en adelante *latches*– y a los biestables –en adelante *flip flop*–.

Revisaremos primero los elementos básicos de los circuitos secuenciales, los latches y flip flops.

1.6.1. *Latches*

Un latch es el circuito secuencial más sencillo que tiene retroalimentación y se usa para establecer un estado en un circuito secuencial. Mientras que en los circuitos combinatorios las salidas de las compuertas se conectan siempre a compuertas posteriores, en los circuitos secuenciales algunas de las salidas se conectan a compuertas anteriores. Los latches tienen dos señales de entrada –conocidas como S (*set*) y R (*reset*), y producen dos salidas[3] generalmente denotadas con Q y $\overline{Q}$. En la figura 1.15 vemos el diagrama de un latch.

Figura 1.15. Diagrama de un latch básico



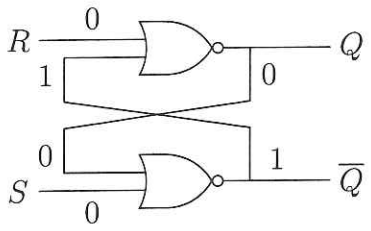
Este latch recibe el nombre de latch–SR por los nombres de sus entradas.

El funcionamiento de un circuito secuencial no se puede describir directamente con una tabla de verdad, pues una de las entradas de cada compuerta es la salida de la otra compuerta, que se lleva a cabo en el siguiente instante de tiempo. El nombre de las salidas es Q y $\overline{Q}$, ya que letra Q se usa generalmente para representar los estados de un sistema. Si

[3] Algunos autores usan únicamente una salida, Q, tomando como dado que $\overline{Q}$ se puede obtener a partir de aquella.

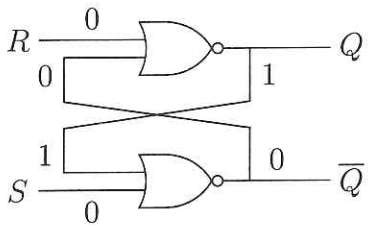
la retroalimentación es positiva el circuito tenderá a ser estable mientras que si es negativa tenderá a oscilar, a ser inestable.

Un latch, para tener un comportamiento estable, debe recibir a lo más una de sus entradas con valor 1. Cuando tanto S como R son 0 tiene dos estados estables posibles. Si Q es 0 tenemos los siguientes valores para las distintas señales:

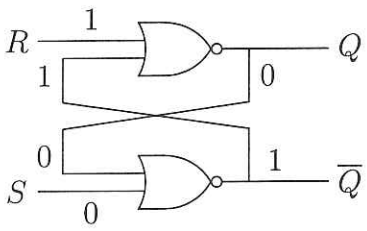


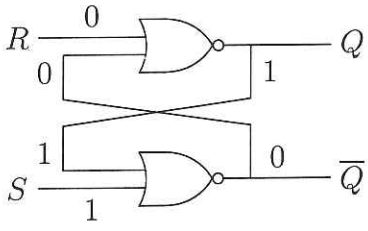
Con estos valores el sistema tendrá un comportamiento estable, ya que los valores, a través del tiempo, se mantendrán.

Si $Q = 1$ entonces tendremos el siguiente esquema, que también presenta un comportamiento estable.



También conseguimos comportamientos estables con $SR = 10$ y $SR = 01$, como se muestra en los esquemas a continuación:





Para representar los estados de un circuito secuencial tenemos que considerar el transcurrir del tiempo, ya que la señal se retroalimenta en el siguiente instante en el cual se produce. Por lo tanto, para revisar el comportamiento del circuito hay que hacer una tabla que tome en cuenta el paso del tiempo. Veamos un ejemplo, donde T_s representa el tiempo de retraso de una compuerta:

Tiempo	**S**	**R**	**Q**	$\overline{\mathbf{Q}}$	**Estabilidad**
Inicial	0	0	0	1	Estable
T_s	0	0	0	1	Estable
$2T_s$	0	0	0	1	Estable

Tiempo	**S**	**R**	**Q**	$\overline{\mathbf{Q}}$	**Estabilidad**
Inicial	0	0	1	0	Estable
T_s	0	0	1	0	Estable
$2T_s$	0	0	1	0	Estable

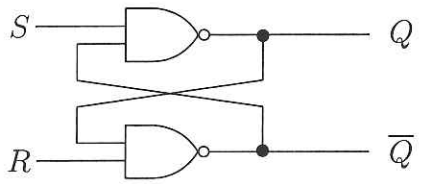
Como se ve en estas tablas y en los distintos esquemas de los latches-SR, se puede observar que estos circuitos secuenciales son capaces de recordar el último estado de las entradas en el sentido de determinar cuál de las dos entradas, S o R fue la última en tener el valor 1.

Un latch-SR funciona de manera muy similar a la de un apagador de luz. Si se desea prender el foco se cambia la entrada en S a 1 y de regreso a 0. Si se desea apagar el foco se cambia la entrada R a 1 y de regreso a 0. Si el foco está prendido no pasa nada si se cambia la entrada S como acabamos de describir; similarmente si el foco está apagado –el foco está representado por la salida Q.

La condición normal de entrada para el latch-SR es $SR = 00$. Si se desea cambiar la salida se ingresa a R o a S un 1 seguido de un 0. Normalmente, S y R no reciben simultáneamente 1, ya que entonces $Q = \overline{Q} = 0$ y $\overline{Q}$ no va a ser el complemento de Q. Adicionalmente, si se cambia la señal $SR = 11$ a $SR = 00$ el estado del latch es impredecible. La mitad de las veces se obtendrá $Q\overline{Q} = 01$ y la otra mitad $Q\overline{Q} = 10$. En la práctica no debería suceder que el latch reciba la entrada $SR = 11$.

Se deja al lector verificar que el latch-SR se puede implementar usando compuertas *nand*, como se muestra en la figura 1.16.

Figura 1.16. Implementación del latch-SR con compuertas *nand*



1.6.2. Circuitos de flip flop

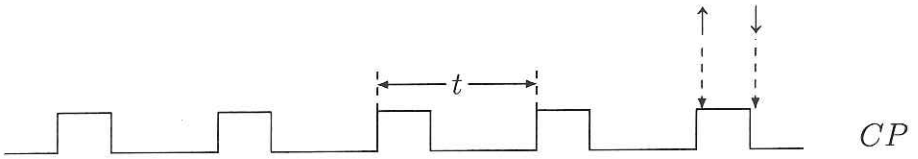
Los circuitos de latch son asíncronos, lo que quiere decir que la salida queda determinada poco después de que se cambia la entrada. Sin embargo, la mayoría de las computadoras hoy en día son síncronas, que significa que la salida de los circuitos secuenciales cambian simultáneamente con el pulso de la señal del reloj del procesador.

Un circuito de flip flop es una versión síncrona de un circuito latch. El objetivo de un flip flop es almacenar un bit de datos. Existen cuatro tipos básicos de circuitos de flip flop. Empezaremos por el más sencillo.

Dado que los circuitos de flip flop son síncronos, además de las entradas S y R de los latches, tienen una entrada que corresponde a la señal del reloj, que trabaja con pulsos a una cierta velocidad. Se acostumbra un diagrama como el de la figura 1.17, donde t representa a un ciclo completo del reloj. En general, lo que se considera como entrada es cuando la señal *cambia* de 0 a 1 o cuando cambia de 1 a 0, que es cuando se registra esta entrada.

Denotaremos con ↑ al cambio del pulso de 0 a 1 y con ↓ al cambio del pulso de 1 a 0−; llamamos CP a la señal dada por el reloj (*Clock Pulse*).

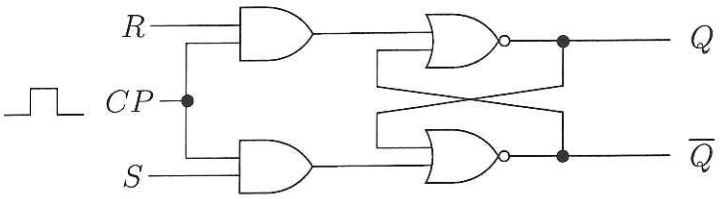
Figura 1.17. Diagrama de los pulsos de un reloj



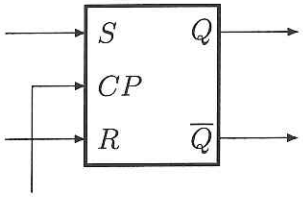
Veremos primero el flip flop-SR, que es el más sencillo de este tipo de circuitos. Consiste de dos compuertas *and* que, de alguna manera, filtran las entradas a un circuito latch, combinándolas con la señal del reloj. El diagrama se encuentra en la figura 1.18.

Figura 1.18. Flip flop SR síncrono

(a) Diagrama lógico



(b) Diagrama de bloque



La tabla de verdad para el flip flop es la siguiente:

Q	S	R	Q_{t+1}
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	*indeterminado*
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	*indeterminado*

El flip flop-SR consiste de un latch básico y dos compuertas and. Estas últimas producen como salida 0 cuando el pulso del reloj está en 0, independientemente de los valores que se alimenten a S y R. Cuando el pulso del reloj *cambia* a 1, estas compuertas dejan pasar los

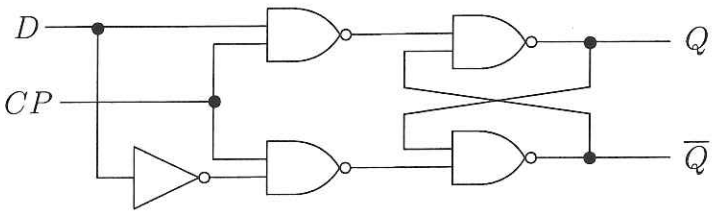
valores de S y R al latch original. Si $SR = 11$ los valores de $Q\overline{Q}$ valen momentáneamente 0. Al eliminar el pulso del reloj el estado del flip flop es indeterminado, ya que dependiendo de si SR permanecen valiendo 1 más tiempo del que le lleva al pulso cambiar a 0.

Circuito flip flop D

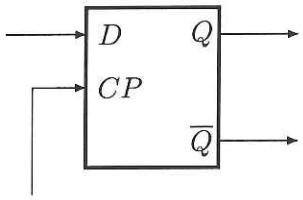
Un circuito de flip flop muy útil es el tipo D, que presenta una pequeña modificación al flip flop-SR, obligando a las señales S y R a ser complementarias entre sí. El diagrama lógico y de bloque se encuentra en la figura 1.19.

Figura 1.19. Circuito flip flop D síncrono

(a) Diagrama lógico



(b) Diagrama de bloque



La tabla de verdad para el flip flop-D se encuentra a continuación:

Q_t	D	Q_{t+1}
0	0	0
0	1	1
1	0	0
1	1	1

Según esta tabla, cuando el pulso del reloj cambia de 0 a 1 el dispositivo se fija en el valor de la entrada D, y si es 1, en el siguiente ciclo del reloj la salida será 1 y así permanecerá hasta que, con el cambio del reloj de 0 a 1 la entrada que se detecte sea 0. En resumen, el flip flop-D se usa para capturar la señal de datos (D) presente en la línea en el momento en que el pulso del reloj cambia a 1.

Dispositivo de flip flop tipo JK

Este es un circuito secuencial muy versátil y es una extensión del flip flop-RS, excepto que todos sus estado están perfectamente determinados sin la inestabilidad que tiene el flip flop-RS cuando $RS = 11$. Como el flip flop-RS tiene dos entradas, J y K, y un reloj. Cambia de estado cuando el pulso del reloj cambia de 1 a 0 ($\downarrow$). El flip flop-JK tiene las siguientes características:

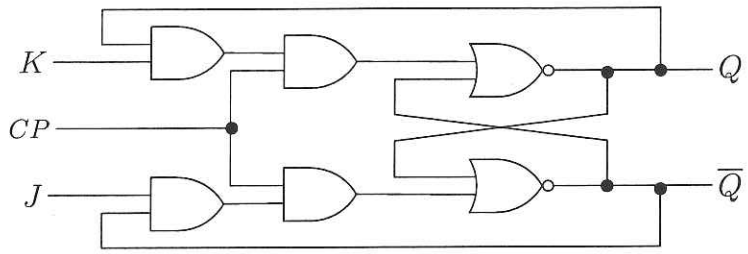
i. Si $JK = 01$ o $JK = 10$, la salida es $Q\overline{Q} = 01$ o $JK = 10$ respectivamente.

ii. Si $JK = 00$, la salida permanece igual.

iii. Si $JK = 1$, la salida se invierte con $\downarrow$.

Para conseguir que cuando $JK = 11$ la respuesta sea estable, se retroalimenta la señal de Q a una compuerta *and* junto con la señal de K (similarmente $\overline{Q}$ con J). El diagrama de este tipo de flip flop se encuentra en la figura 1.20.

Figura 1.20. Circuito secuencial flip flop tipo JK



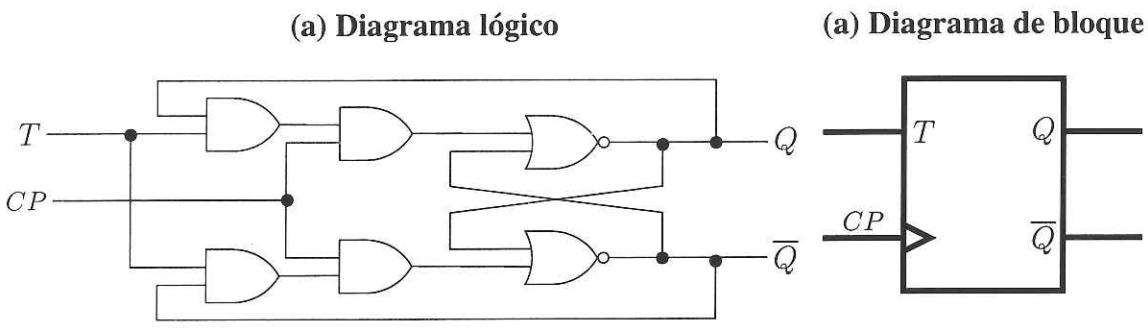
La tabla de verdad para este flip flop se encuentra a continuación.

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

Dispositivo de flip flop tipo T

El flip flop tipo T es una versión simplificada del flip flop tipo JK. Su única función es cambiar de estado en cada ciclo del reloj (**Toggle**) y lo consigue juntando las dos entradas en una sola. Los diagramas lógico y de bloque se pueden ver en la figura 1.21.

Figura 1.21. Flip flop tipo T



Podemos pensar en un flip flop tipo T como un contador de un bit, por lo que en cada pulso del reloj simplemente invierte su entrada. Es muy útil para construir contadores o divisores del ciclo del reloj. Por ejemplo, cuando T se mantiene en 1, si la frecuencia del reloj es de, por ejemplo, 4MHz, la frecuencia que se obtiene de la salida del flip flop T es de 2MHz. La tabla de verdad para este circuito se encuentra a continuación (recordar que la señal T se detecta únicamente en el cambio del pulso del reloj de 0 a 1, aunque pudiera estar construido para detectar el cambio del pulso del reloj de 1 a 0):

T	Q_t	Q_{t+1}
0	0	0
0	1	1
1	0	1
1	1	0

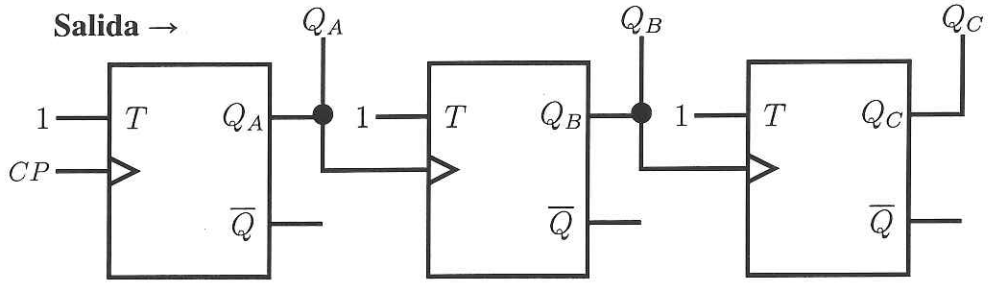
Veremos a continuación algunos de los usos de los flip flops en la construcción de computadoras.

Contadores

Por la característica que tienen los circuitos secuenciales de poder almacenar información, se usan tanto para distintos tipos de registros como para contadores. Únicamente revisaremos los contadores.

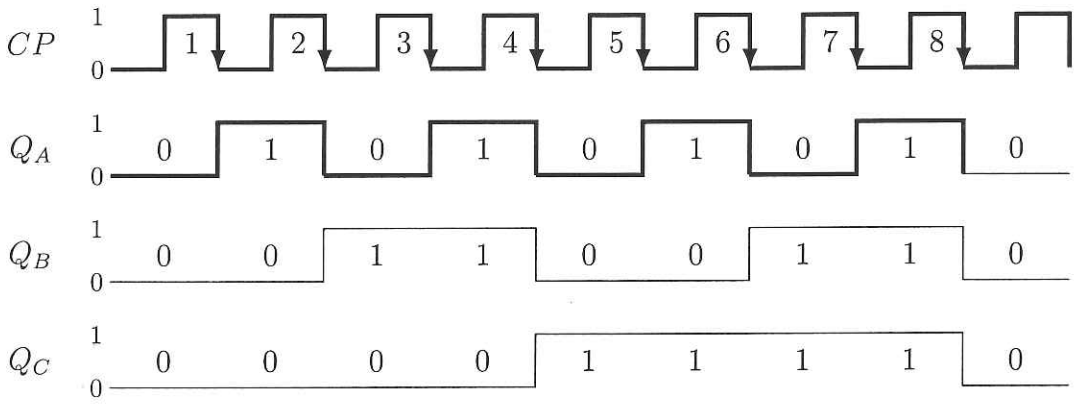
Los contadores son, en sí mismos, circuitos secuenciales que se construyen con grupos de circuitos de flip flop síncronos. Por ejemplo, un contador binario puede contar el número de pulsos que se le dan de entrada. Por ejemplo, podemos conectar dos circuitos de flip flop T, uno que está controlado por el reloj y el segundo está controlado por la salida del primero. Los cambios en el flip flop se llevarán a cabo en la caída de la señal (de 1 a 0). El diagrama de bloque de un contador de tres bits se muestra en la figura 1.24.

Figura 1.22. Contador binario ascendente de tres bits



En la figura anterior la señal de T se mantiene en 1 para permitir los cambios en la caída del ciclo de reloj. Como la salida de cada flip flop actúa como el reloj del siguiente flip flop, el siguiente flip flop invierte su señal cuando la del flip flop anterior pasa de 1 a 0. En la figura 1.23 se muestra el comportamiento a través del tiempo de este contador. La salida del flip flop A, que es el dígito menos relevante del contador, es el que cambia más frecuentemente y el flip flop C es el que lo hace menos frecuentemente. La salida Q del flip flop A actúa como reloj del flip flop B, por lo que cuando la salida de A cambia de 1 a 0 es cuando se invierte la salida Q de B, que a su vez, cuando cambia de 1 a 0 es cuando se invierte la salida del flip flop C.

Figura 1.23. Comportamiento de contador ascendente asíncrono de tres bits



A continuación mostramos la salida del contador a través del tiempo, observando 9 ciclos.

T	t_i	2^2 Q_C	2^1 Q_B	2^0 Q_A	N
1	t_0	0	0	0	0
1	t_1	0	0	1	1
1	t_2	0	1	0	2
1	t_3	0	1	1	3
1	t_4	1	0	0	4
1	t_5	1	0	1	5
1	t_6	1	1	0	6
1	t_7	1	1	1	7
1	t_8	0	0	0	0

De manera similar podemos construir contadores de cuatro, cinco o el número de bits que queramos (siempre y cuando quede fijo antes de construir).

Es fácil ver que la capacidad N de un contador –el máximo entero que puede representar– está relacionada con el número de flip flops utilizados, mediante la siguiente fórmula:

$$N = 2^n - 1$$

donde n representa el número de flip flops utilizados. Por ejemplo, si $n = 8$, tendremos $N = 2^8 - 1 = 255$.

Por otro lado, si queremos saber cuántos flip flops necesitamos dada la capacidad que requerimos, usamos la fórmula

$$n = \lceil log_2 N \rceil = \lceil log_2 10 \cdot log_{10} N \rceil$$

donde los símbolos $\lceil$ y $\rceil$ se refieren al primer entero mayor o igual que la cantidad que se está obteniendo. Por ejemplo, si $N = 5000$, entonces obtenemos

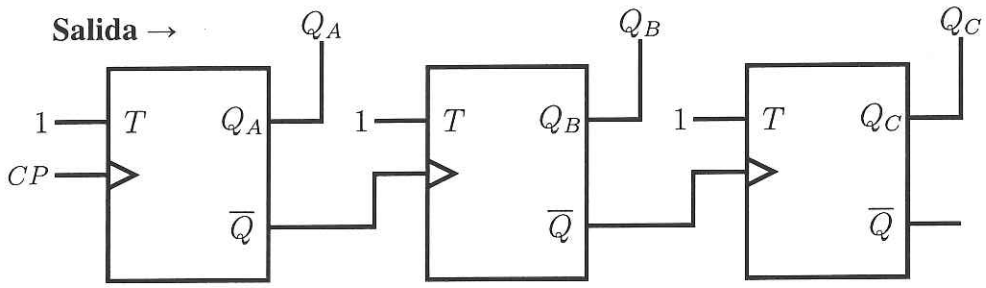
$$n = \lceil log_2 5000 \rceil = \lceil 12.28 \rceil = 13.$$

Con 12 flip flops podremos representar hasta $2^{12} - 1 = 4095 < 5000$, mientras que con 13 flip flops podremos representar hasta $2^{13} - 1 = 8191$.

La principal desventaja de este tipo de contadores es que cada flip flop se activa a continuación del anterior, teniéndose un retardo por cada uno,. por lo que el tiempo total de propagación que se requiere es proporcional al número de flip flops. Esto puede causar problemas cuando todos los flip flops cambian al mismo tiempo, ya que se va a generar un retraso hasta que el último flip flop cambie de estado.

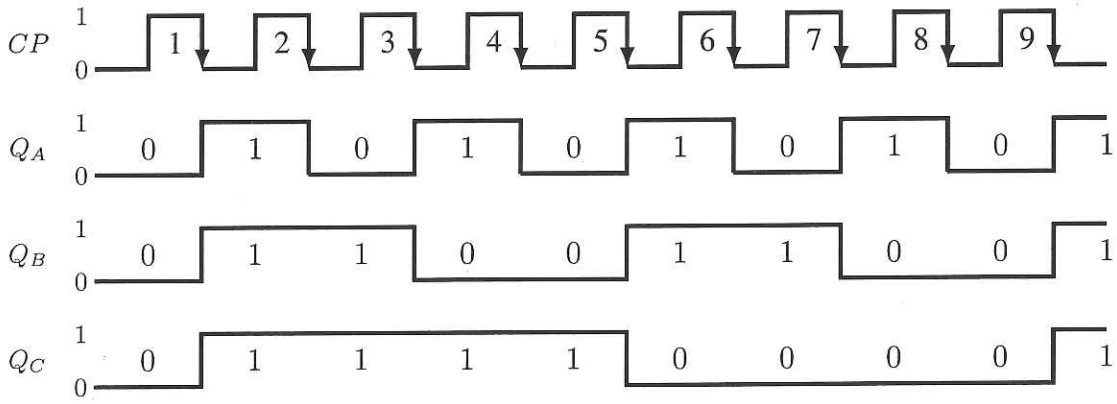
De manera similar a cómo construimos un contador ascendente de tres bits, podemos construir uno descendente. La idea es sencilla: la salida complementada la alimentamos como si fuera el pulso del reloj. El diagrama de bloques se encuentra en la figura 1.25.

Figura 1.24. Contador binario descendente de tres bits



Si suponemos que el contador empieza en 0 (000_2), con el primer pulso del reloj Q_A cambia a 1 teniendo $\overline{Q} = 0$, por lo que el pulso del flip flop B pasa de 1 a 0, lo que hace que $Q_B = 1$. Similarmente entre el flip flop B y el C. A continuación mostramos el comportamiento del circuito al paso de 9 ciclos de reloj.

Figura 1.25. Comportamiento de contador descendente asíncrono de tres bits



La tabla de verdad para este circuito se encuentra a continuación. Cada pulso del reloj decrementa el contador en una unidad.

T	t_i	2^2 Q_C	2^1 Q_B	2^0 Q_A	N
1	t_0	0	0	0	0
1	t_1	1	1	1	7
1	t_2	1	1	0	6
1	t_3	1	0	1	5
1	t_4	1	0	0	4
1	t_5	0	1	1	3
1	t_6	0	1	0	2
1	t_7	0	0	1	1
1	t_8	0	0	0	0

Es fácil diseñar circuitos que cuenten hasta una cantidad arbitraria (pero fija), pero hay que tomar en cuenta que, como estos contadores son asíncronos, los pulsos se van propagando de flip flop en flip flop, por lo que la acumulación de retardos puede ser significativa. Para evitar estos retrasos se deben diseñar circuitos secuenciales síncronos, en los que todos los flip flops reciban la señal de manera sincronizada con el reloj. Dejamos este tema al lector interesado.

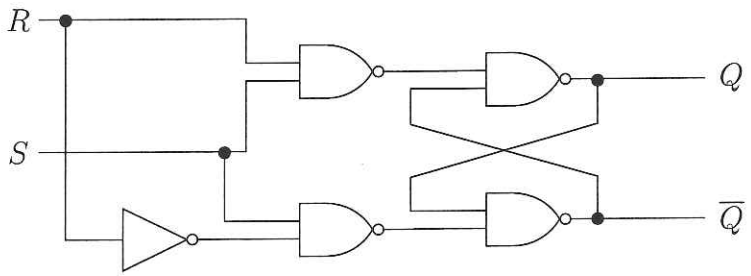
Ejercicios

1.6.1.- Da las salidas (Q y $\overline{Q}$) en cada uno de los instantes de tiempo $T_0, \ldots, t_5$, que produce un latch básico (página 30) frente a la siguiente entrada:

$R \rightarrow$ 0 1 1 0 1
$S \rightarrow$ 1 0 1 0 1

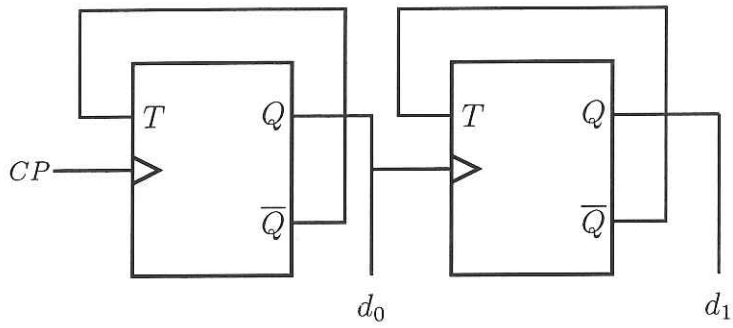
1.6.2.- Construye un flip flop SR usando un latch que esté construido con compuertas *nand* en lugar de *nor*.

1.6.3.- Describe el comportamiento (la tabla de verdad) del siguiente circuito digital:



1.6.4.- Observemos el flip flop tipo T. Si se ejecutan cuatro ciclos de reloj (en un ciclo el reloj pasa de 0 a 1 y de regreso a 0), ¿por cuántos ciclos pasa la salida del flip flop?

1.6.5.- Supongamos que conectamos dos flip flops tipo T de la siguiente manera:



Describe el comportamiento de este circuito a lo largo de cuatro ciclos de reloj.

1.6.6.- Si en el circuito anterior interpretas la salida del circuito como un número binario, donde el bit más significativo está a la derecha, ¿cuál es el número binario con el que empieza el circuito?

Bibliografía

[1] K. Doets and J. van Eijck. *The Haskell Road to Logic, Maths and Programming*. King's Coll. Pub., London, 2004.

[2] John A. Dossey, Albert D. Otto, Lawrence E. Spence, and Charles Vanden Eynden. *Discrete Mathematics*. Pearson/Addison-Wesley, 5-th edition, 2006.

[3] Judith L. Gersting. *Mathematical Structures for Computer Science*. Computer Science Press, W.H. Freeman and Company, third edition, 1993.

[4] Winifried Karl Grassman and Jean-Paul Tremblay. *LOGIC AND DISCRETE MATHEMATICS, A Computer Science Perspective*. Prentice-Hall Inc., 1996.

[5] David Gries and Fred B. Schneider. *A Logical Approach to Discrete Mathematics*. Springer-Verlag, 1994.

[6] Jerold W. Grossman. *DISCRETE MATHEMATICS, an introduction to concepts, methods and applications*. Macmillan Publishing Company, 1990.

[7] James L. Hein. *Discrete Structures, Logic, and Computability, Third Edition*. JONES AND BARTLETT PUBLISHERS, 2010.

[8] Thomas Koshy. *Discrete Mathematics with Applications*. Elsevier Academoc Press, 2004.

[9] K.H. Rossen. *Discrete Mathematics and its Applications*. McGraw Hill, 6-th edition, 2006.