

3 Circuitos lógicos combinacionales

Juan J. Navarro
Toni Juan

Primera versión: 07-2007; versión actual: 09-2010

En este capítulo estudiamos lo que consideramos imprescindible sobre circuitos combinacionales para entender el diseño del computador que vamos a crear paso a paso en este curso.

Como introducción al tema, en la sección 3.1 definimos qué es un circuito combinacional y formulamos un modelo para facilitar su estudio. Aquí aparecen los conceptos de variable lógica, función lógica y tabla de verdad. También presentamos las puertas lógicas Not, And y Or, que son los circuitos combinacionales más simples con los que vamos a construir el computador y, finalmente, damos las reglas de interconexión de dispositivos combinacionales para que el resultado sea un circuito combinacional válido. En la sección 3.2 obtenemos la especificación funcional de circuitos combinacionales formados por puertas lógicas (análisis) y en la sección 3.3 tratamos el problema inverso, dada una especificación funcional deseada obtenemos el esquema del circuito que la implementa (síntesis). En la 3.4 completamos el modelo de circuito combinacional, haciéndolo más realista: consideramos el tiempo de propagación de las señales al atravesar los dispositivos del circuito. Por último, en el Apéndice I definimos el álgebra de Boole, una herramienta matemática que nos puede ayudar en el análisis y sobre todo la síntesis eficiente de circuitos combinacionales (este apéndice no está disponible todavía).

3.1 Introducción

3.1.1 Circuito Lógico Combinacional

Como vimos en el tema 1, un computador es capaz de comunicar, almacenar y procesar información digital. Los circuitos lógicos combinacionales son los circuitos encargados de procesar las señales eléctricas binarias, que son las que codifican la información.

Definición

3.1

Denominamos **circuito lógico combinacional (CLC)** a un circuito lógico (las señales que entran al circuito y las que salen de él son binarias) en el que el valor de cada una de sus señales de salida en cada momento es función únicamente de lo que valen sus entradas en ese preciso momento (respuesta instantánea -sin retardo^a- y sin memoria). Esto es, a cada combinación de valores concretos de las entradas le corresponden siempre unos mismos valores concretos de las salidas, con independencia del instante de tiempo en el que se den estos valores en las entradas y con independencia de los valores que hayan tenido las entradas en momentos anteriores.

- a. Como veremos en la sección 3.4, en un circuito combinacional real sí que hay un retardo entre el cambio de una entrada y lo que tarda en reaccionar la salida, o las salidas, a ese cambio. Pero de momento vamos a considerar que el retardo es 0.

El elemento característico de los circuitos combinacionales es que no tienen memoria. Por ejemplo, el circuito interno del dispositivo o bloque f , que se muestra en la figura 3.1, está diseñado de forma que para cualquier instante de tiempo, t_1 , en el que señales de entrada $x(t_1)$ e $y(t_1)$ valgan respectivamente 5 y 0 voltios, la señal de salida en ese mismo instante, $w(t_1)$ valga 5 voltios; y esto es siempre así, independientemente del valor de las entradas en cualquier momento anterior a t_1 .

Cronograma

Esto se muestra en la figura 3.1, mediante un **cronograma**: un gráfico que indica el voltaje (eje vertical) que toma la señal de salida a lo largo del tiempo (eje horizontal) como respuesta a variaciones temporales concretas de los valores de las señales de entrada. Este cronograma también nos indica que, para este circuito concreto, cuando las dos entradas valen 0 voltios la salida también vale 0 voltios, lo que ocurre al principio y al final del intervalo de tiempo mostrado. También se ve que cuando las dos señales de entrada tienen 5 voltios la salida es de 0 voltios, sin importar los valores anteriores de las entradas.

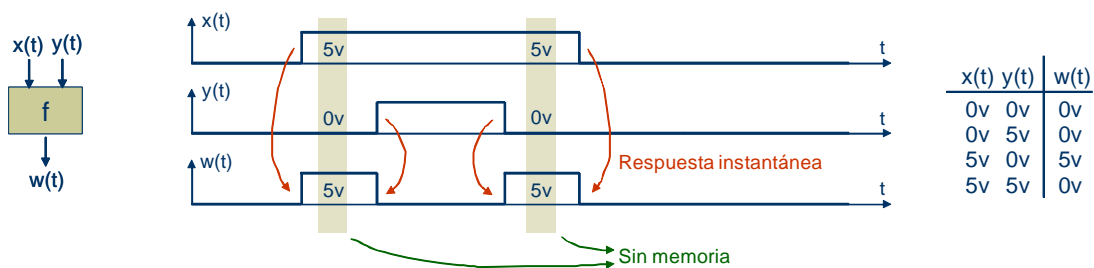


Fig. 3.1 Circuito combinacional f con dos entradas (señales $x(t)$ e $y(t)$) y una salida ($w(t)$). Cronograma y tabla que define el funcionamiento del circuito.

Lo que no muestra el cronograma es el comportamiento del circuito cuando la entrada $x(t)$ vale 0 voltios y la $y(t)$ vale 5. Un cronograma muestra el comportamiento del circuito ante un comportamiento temporal concreto de las señales de entrada, pero no tiene porqué mostrarlo para todas las posibles combinaciones de los valores de las entradas (no tiene porqué ser una descripción exhaustiva, completa, del funcionamiento del circuito).

Por todo lo dicho, el funcionamiento de un circuito (dispositivo o bloque) combinacional se puede especificar mediante una tabla que indique, para cada combinación de los posibles valores de las señales de entrada, qué valor toman cada una de las salidas. La tabla junto al cronograma de la figura 3.1 indica el comportamiento completo del circuito combinacional f incluido el valor de la salida para $x = 0$ voltios e $y = 5$, que no aparecía en el cronograma.

Modelo matemático. Variables y funciones lógicas

Para comprender, analizar, diseñar,... sistemas complejos como son los computadores, los ingenieros idean modelos matemáticos que simplifican la realidad para hacerla más manejable. Según el aspecto de la realidad que se desee trabajar se usa un modelo u otro. Aquí presentamos el modelo usado para facilitar el análisis y la síntesis de circuitos combinacionales.

Como la respuesta de los circuitos combinacionales es instantánea en el tiempo y sin memoria, cuando trabajamos con este tipo de circuitos sólo nos importa el valor de todas las señales eléctricas del circuito en un determinado instante, pero no nos importará cuál es ese instante. Así, en vez de hablar de señales eléctricas que varían en el tiempo, como $x(t)$, hablamos de variables matemáticas, como x , en la que desaparece la idea de tiempo. Además, como los dos niveles de voltaje que pueden tomar las señales digitales binarias dependen de la tecnología usada en la implementación del circuito (por ejemplo, 0 voltios y 5 voltios en TTL y 0 voltios y 3.1 en algunas tecnologías CMOS), para ser independientes de la tecnología decimos que las variables matemáticas que modelan a las señales eléctricas binarias pueden tomar únicamente dos posibles valores, denominados 0 y 1. Por ello, a cada una de estas variables matemáticas se la denomina **variable lógica binaria** o simplemente **variable lógica**. Así que en muchos casos no hablaremos de la señal eléctrica $x(t)$ sino de la variable lógica x .

Cada señal eléctrica de salida del circuito tiene, en cada momento, uno de los dos posibles valores de voltaje, dependiendo del voltaje que tienen las entradas en ese mismo momento. Esto mismo, dicho con la nomenclatura del modelo matemático que estamos construyendo se dice así: cada variable lógica de salida de un circuito lógico combinacional es función de las variables lógicas de entrada. Este tipo de función se denomina **función lógica**. Así, el comportamiento de un circuito combinacional se define mediante tantas funciones lógicas como señales de salida tiene el circuito. Cada función define el valor (0 o 1) que toma una variable de salida para cada una de las posibles combinaciones de valores de las variables de entrada. La figura 3.2 muestra estos conceptos para un circuito combinacional muy simple, con sólo dos entradas y una salida. Ahora, a la tabla que define el comportamiento del circuito se la denomina **tabla de verdad** y formula tantas funciones lógicas como salidas tiene el circuito. La tabla de verdad indica el comportamiento lógico de un circuito combinacional¹.

1. No obstante, la tabla de verdad no es un modelo completo para un circuito combinacional real. Hay otras características que no quedan plasmadas en la tabla de verdad, como por ejemplo los tiempos de retardo de cada entrada a cada salida (que estudiamos en la sección 3.4), la potencia consumida por el circuito, etc.

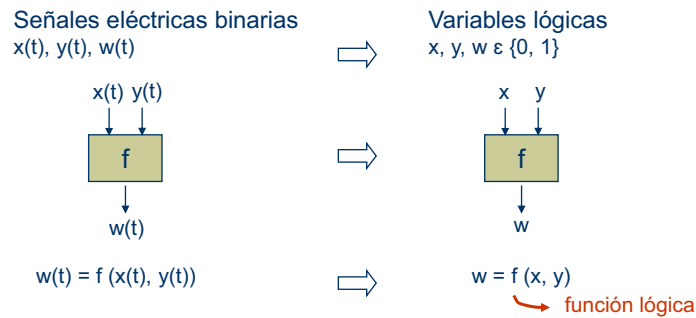


Fig. 3.2 Modelo lógico de un circuito combinacional: variable lógica, función lógica.

Tabla de verdad

3.1

La tabla de verdad indica el valor que toma cada variable lógica de salida de un circuito combinacional para cada uno de los posibles valores de las variables de entrada. Una tabla de verdad tiene una columna para cada entrada al circuito (para cada variable lógica de entrada) y una columna para cada salida (para cada variable lógica de salida). La tabla de verdad tiene una primera fila de cabecera, donde se especifica el nombre de las variables de cada columna. Además de la cabecera, tiene una fila para cada una de las posibles combinaciones de valores que pueden tomar las entradas.^a

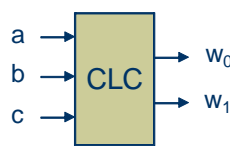
En general, si tenemos un circuito combinacional con n variables de entrada ordenadas de alguna forma, por ejemplo: $x_{n-1}, x_{n-2}, \dots, x_2, x_1, x_0$, la tabla de verdad tiene 2^n filas, que son las posibles combinaciones distintas de n elementos pudiendo tomar cada uno de ellos 2 valores distintos.

Por convenio, las filas de una tabla de verdad se ordenan de arriba abajo, comenzando por la fila 0, como sigue. La fila i es la que corresponde a la combinación de valores de las entradas que representan el valor i , suponiendo que las variables de entrada son los dígitos de un vector de bits que representa el número natural i codificado en binario. Esto es así aunque cada variable de entrada represente algo que nada tenga que ver con el dígito de un número binario^b.

- Por ejemplo, si el circuito tiene solamente 1 entrada, la tabla de verdad tiene 2 filas, una para el valor 0 de la entrada y otra para el 1. Si el circuito tiene dos entradas denominadas, por ejemplo x e y , la tabla de verdad tiene 4 filas: una para la combinación de entradas $x=0$ e $y=0$, otra para la combinación $x=0$ e $y=1$, otra para $x=1$ e $y=0$ y la última para $x=1$ e $y=1$.
- Por ejemplo, si tenemos 3 variables de entrada ordenadas como a , b y c , la fila 6 corresponde a los valores de las variables de entrada $a=1$, $b=1$ y $c=0$, ya que el 6 se codifica en binario como 110.

La figura 3.3 muestra, a modo de ejemplo, el símbolo de un circuito con 3 entradas (a , b , c) y 2 salidas (w_1 y w_0) y su tabla de verdad, que define las funciones lógicas $w_1(a, b, c)$ y $w_0(a, b, c)$.

✍ 1



	a	b	c	w_1	w_0
Fila 0 →	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
Fila 7 →	1	1	1	1	1

Fig. 3.3 Tabla de verdad de un circuito lógico combinacional, CLC, con 3 entradas y 2 salidas.

3.1.2 De la descripción funcional a la tabla de verdad

A veces tenemos que resolver un problema mediante un circuito combinacional del que no nos dan su tabla de verdad, sino que nos dan una **descripción funcional** de su comportamiento. Esta descripción relaciona, de manera más o menos formal, las salidas y las entradas mediante un texto explicativo y/o expresiones algebraicas y/o un fragmento de código en un pseudolenguaje de alto nivel, etc. La descripción puede completarse con figuras aclaratorias. En estos casos primero debemos encontrar la tabla de verdad del circuito a partir de la descripción funcional y después pasamos de la tabla de verdad al esquema lógico del circuito mediante alguna de las técnicas que veremos en la sección 3.3. Veamos dos ejemplos de circuitos combinacionales que hacen cosas muy distintas.

3.2

Para pasar de la descripción funcional de un circuito a su tabla de verdad no hay un método sistemático de hacerlo. Depende de la funcionalidad concreta, de cómo está expresada, etc. Se trata de cambiar la manera de como se representa el funcionamiento del circuito, de un texto más o menos informal, por ejemplo, a una tabla de verdad, que es una representación formal que indica de manera exhaustiva el valor lógico de cada salida para cada uno de los posibles valores de las entradas. Esto lo aprenderemos haciendo ejemplos.

Ejemplo 1 Control de llenado de un depósito

En la figura 3.4 se muestra la caja que representa un circuito combinacional con 3 entradas y 1 salida para controlar el llenado automático de un depósito para el consumo de agua. La entrada x viene de un reloj y vale 1 durante el día y 0 durante la noche. Las entradas y y z vienen de dos sensores colocados dentro del depósito, en la parte superior e inferior respectivamente. Cada señal indica con el valor 1 que el sensor está cubierto de agua y con 0 que no lo está. La salida del circuito controla una bomba de agua, cuando w vale 1 se bombea agua de un pozo para llenar el depósito y

cuando vale 0 no. El agua del depósito se va consumiendo de forma irregular durante las 24 horas del día (la salida de agua no se muestra en la figura). Incluso en caso de máximo consumo, el flujo de salida de agua es menor que el de entrada si la bomba está accionada. Se desea que siempre haya agua en el depósito para poderla consumir, pero a poder ser, accionando la bomba por la noche para ahorrar dinero, ya que la tarifa nocturna de electricidad es más barata que la diurna.

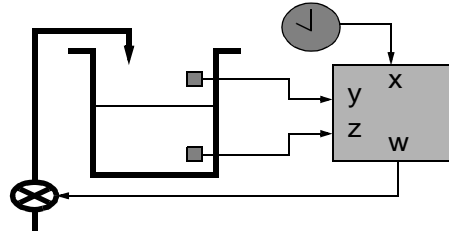


Fig. 3.4 Sistema de control del llenado de un depósito.

A partir de esta descripción del sistema debemos obtener la tabla de verdad del circuito. Primero creamos la estructura de la tabla. Ponemos la fila de cabecera con las variables de entrada, ordenadas sin ningún criterio específico y después la variable de salida, como se ve en la figura 3.5a.

Ahora podemos ir dando valor a la salida para cada una de las filas de la tabla. Podemos plantearnos las situaciones por las que puede pasar el sistema y razonar qué tiene que valer la salida para cada caso. Una situación trivial es que cuando el depósito este completamente lleno, hay que dejar de bombear agua para evitar que se derrame, y esto independientemente de que sea de día o de noche. Esto es cuando (x, y, z) valen $(1, 1, 1)$ y $(0, 1, 1)$ la salida w debe valer 0. Otra situación cuya respuesta también es independiente de que sea de día o de noche (la respuesta es la misma tanto si x vale 1 como si vale 0) es cuando no queda apenas agua y las dos entradas y y z valen 0: en este caso hay que bombear agua y para ello w ha de valer 1. Las 4 filas de la tabla que cubren las dos situaciones que acabamos de tratar se han marcado en la tabla de verdad de la figura 3.5b.

¿Qué debe hacerse cuando el depósito no esté en ninguna de las dos situaciones límite que acabamos de tratar (por ejemplo como se ve en la figura 3.4)? Esta situación se da cuando y vale 0 y z vale 1. En este caso, como no hay peligro inmediato de quedarnos sin agua, si es de día ($x = 1$) no vamos a dar la orden de bombear agua ($w = 0$), para ahorrar electricidad. Pero si es de noche ($x = 0$) sí vamos a bombear agua ($w = 1$), ya que la electricidad es más barata. Las dos filas de la tabla para esta situación se ha sombreado en la figura 3.5c.

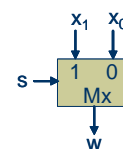
a)	b)	c)	d)
x y z w	x y z w	x y z w	x y z w
0 0 0	0 0 0 1	0 0 0 1	0 0 0 1
0 0 1	0 0 1	0 0 1 1	0 0 1 1
0 1 0	0 1 0	0 1 0	0 1 0 x
0 1 1	0 1 1 0	0 1 1 0	0 1 1 0
1 0 0	1 0 0 1	1 0 0 1	1 0 0 1
1 0 1	1 0 1	1 0 1 0	1 0 1 0
1 1 0	1 1 0	1 1 0	1 1 0 x
1 1 1	1 1 1 0	1 1 1 1	1 1 1 0

Fig. 3.5 Proceso de construcción de la tabla de verdad del ejemplo 1.

Ya solo quedan dos filas por completar, una para el día y otra para la noche. En este caso los sensores nos dicen que hay agua en la parte superior del depósito y que no la hay en la parte inferior. Pero, ¡esta situación es imposible! (a no ser que los sensores se hayan estropeado). En este caso, como en el enunciado no nos dicen nada de qué hacer en situaciones anómalas, considereamos que es imposible que las entradas (x , y , z) valgan (0, 1, 0) o (1, 1, 0). En estos casos decimos que no importa lo que valga la salida, ya que nunca se darán estas combinaciones de valores de las entradas. Al rellenar la tabla de verdad nosotros pondremos una cruz (x) en las salidas cuyo valor puede ser tanto 0 como 1 (también se suelen usar el guión, -, o la letra d para estas salidas). La tabla final completa se ve en la figura 3.5d, con estas dos filas marcadas. Cuando se implemente el circuito (ver ejemplo 9) se hará de forma que la salida sea 0 cuando la tabla dice que puede ser 0 o 1, porque esto es lo más económico en el tipo de implementación que vemos en la sección 3.3.

Ejemplo 2 El multiplexor

Vamos a obtener la tabla de verdad de un dispositivo (bloque combinacional) que más adelante usaremos para formar dispositivos más complejos, llamado multiplexor. Lo denotamos como Mx-2-1 porque tiene dos entradas de datos (x_1 , x_0) y una salida, w . Además tiene una entrada de selección s . El símbolo del Mx-2-1 se muestra al margen.



La especificación del comportamiento del circuito puede hacerse mediante el siguiente texto. La entrada de selección s indica qué entrada, x_0 o x_1 , se conecta en cada momento a la salida w . Cuando s vale 0 la entrada x_0 se conecta a la salida, mientras que cuando s vale 1 se conecta la entrada x_1 . Este comportamiento se puede ver de forma gráfica en los dibujos de la figura 3.6.

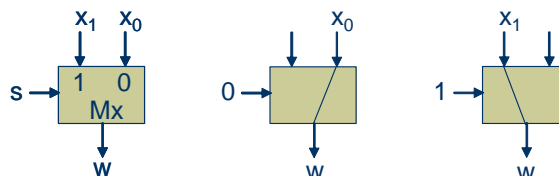


Fig. 3.6 Dos posibles situaciones en las que se puede encontrar un Mx-2-1 en función de s .

De una manera más formal se puede especificar el comportamiento mediante un lenguaje de programación de alto nivel, como el C, por ejemplo, (o cualquier otro pseudolenguaje, en el sentido de que no hace falta que tenga todo el formalismo de un lenguaje de programación):

```
if (s == 0) w = x0;
else w = x1;
```

Para encontrar la tabla de verdad primero ponemos la fila de cabecera con las variables de entrada, ordenadas según nos parezca más razonable (o sin ningún criterio) y después la variable de salida. Ordenamos las variables como muestra la figura 3.7.a: la variable s a la izquierda, después x_1 y por último x_0 . La figura 3.7.b muestra cómo rellenamos las filas correspondientes a $s=0$, copiando las 4 filas correspondientes de x_0 en la salida w . En la parte c) de la figura mostramos el caso de $s=1$ y en la d) la tabla resultante.

a)	b)	c)	d)
s	s	s	s
x ₁	x ₁	x ₁	x ₁
x ₀	x ₀	x ₀	x ₀
w	w	w	w
0 0 0	0 0 0	0 0 0	0 0 0
0 0 1	0 0 1	0 0 1	0 0 1
0 1 0	0 1 0	0 1 0	0 1 0
0 1 1	0 1 1	0 1 1	0 1 1
1 0 0	1 0 0	1 0 0	1 0 0
1 0 1	1 0 1	1 0 1	1 0 1
1 1 0	1 1 0	1 1 0	1 1 0
1 1 1	1 1 1	1 1 1	1 1 1

Fig. 3.7 Construcción, paso a paso, de la tabla de verdad del multiplexor Mx-2-1.

Es interesante observar que partiendo de la sentencia *if*, que indica la salida en función de *s*, ha resultado muy cómoda la creación de la tabla de verdad, en la que la variable *s* ocupa la posición de más a la izquierda. Si probamos a obtener la tabla de verdad ordenando las variables de entrada con *s* en otra posición nos hubiera costado más tiempo. No obstante, cualquier orden es correcto.

10..13

3.1.3 Puertas Lógicas

Definimos ahora los circuitos lógicos más simples con los que vamos a trabajar, que se denominan **puertas lógicas**. Las puertas lógicas tienen muy pocas entradas y sólo una salida. Como sólo tiene una salida, podemos decir que una puerta lógica es una implementación física de una función lógica.

Usaremos las puertas lógicas como “ladrillos” para construir poco a poco un gran edificio como es el computador. Usaremos solamente tres tipos distintos de puertas, las denominadas *Not*, *And* y *Or*, ya que con ellas se puede implementar cualquier función lógica (se dice que es un conjunto completo - universal- de puertas). No entraremos en cómo se construyen estas puertas con una tecnología determinada, pues esto forma parte de un curso de electrónica o de diseño digital más avanzado. Definiremos su funcionamiento y asociaremos un símbolo a cada puerta. Usaremos las puertas como dispositivos para, conectándolos entre sí, formar otros circuitos combinatoriales más complejos.

Puertas de una entrada. ¿Cuántas puertas lógicas o funciones lógicas distintas podemos definir que tengan una sola entrada? La tabla de verdad de una de estas puertas tiene solamente dos filas, una para especificar el valor de la salida cuando la entrada toma el valor 0 y otra para el valor 1. El número de funciones lógicas distintas de una entrada es el número de columnas distintas que sepamos poner con dos bits por columna, (un bit en cada fila). La respuesta es cuatro ya que cuatro es el número de combinaciones diferentes que pueden tomar dos bits (00, 01, 10 y 11). La figura 3.8 muestra las cuatro posibles funciones que hemos denominado *g*₃, *g*₂, *g*₁ y *g*₀.

$x \rightarrow g_i \rightarrow w$	x	g_3	g_2	g_1	g_0	$x \rightarrow \neg x \rightarrow w$
	0	1	1	0	0	Not
	1	1	0	1	0	$w = !x$

$w = g_i(x)$

Fig. 3.8 Tabla con las 4 posibles funciones de una entrada. Símbolo usado para la puerta **Not** y notación algebraica de la operación **Not** (que es la función *g*₂ de la tabla).

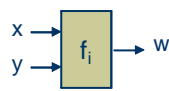
De entre ellas sólo la función g_2 tiene interés, ya que el resto son funciones “triviales”. La función g_0 tiene la salida siempre igual a 0 independientemente del valor que tome la entrada. Es la función constante 0, que podemos decir que no tiene entrada. Se construye conectando la salida a masa (si el 0 se corresponde con 0 voltios). Lo mismo podemos decir de la g_3 , la función constante 1, que se construye conectando la salida a la tensión de alimentación (si este voltaje coincide con la implementación del 1 lógico). Por último, la g_1 es la igualdad, la salida w es igual a la entrada x , que se construye con una simple línea que une la entrada con la salida.

La función lógica g_2 se denomina función **Not**, al igual que la puerta que la implementa, puerta **Not**. Lo que hace es muy simple: el valor de la salida es el contrario del de la entrada. Esto es, invierte la señal de entrada, si le llega un 0 saca un 1 y si le llega un 1 saca un 0. El símbolo que usamos para referirnos a esta puerta, cuando forma parte de un circuito más complejo, se muestra en la figura 3.8, junto con la notación matemática para expresar la función como operación lógica de una variable ($w = !x$) que implementa la puerta.

Puertas de dos entradas. ¿Cuántas funciones lógicas distintas podemos definir que tengan únicamente dos entradas? La tabla de verdad de una de estas puertas tiene cuatro filas, una para cada una de las posibles combinaciones de los dos bits de entrada. El número de funciones lógicas distintas de dos entradas es 16 ya que este es el número de columnas distintas que podemos poner con cuatro bits por columna, (un bit en cada fila) (0000, 0001, ... y 1111). La figura 3.9 muestra las 16 posibles funciones que hemos denominado f_{15}, \dots, f_0 . De entre las 16 hay algunas que son funciones “triviales”, como la f_0, f_{10}, f_{12} y f_{15} . La función f_3 es la **Not** de x y la f_5 es la **Not** de y , que realmente son funciones de una entrada, y que ya hemos encontrado antes.


El resto de funciones tienen interés y se usan en el diseño de circuitos lógicos. No obstante, nosotros usaremos por ahora solamente dos de ellas, la f_8 y la f_{14} , que se denominan funciones **And** y **Or** como las puertas de dos entradas que las implementan: puertas **And** y **Or**. El símbolo que asignamos a estas puertas junto con la notación matemática (\cdot y $+$) empleada para expresar las operaciones lógicas sobre dos variables que implementan, se muestra también en la figura 3.9.

Las operaciones lógicas **Not**, **And** y **Or** forman un álgebra de Boole, como se formula en el Apéndice I.




$w = f_i(x, y)$

x	y	f_{15}	f_{14}	f_{13}	f_{12}	f_{11}	f_{10}	f_9	f_8	f_7	f_6	f_5	f_4	f_3	f_2	f_1	f_0
0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
0	1	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0
1	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0
1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0



Or

$w = x + y$



And

$w = x \cdot y$

Fig. 3.9 Tabla con las 16 posibles funciones lógicas de dos entradas y una salida. Símbolos usados para las puertas **Or** y **And** y notación algebraica de las funciones **Or** y **And**.

3.3

Las tres puertas básicas. La figura 3.10 muestra, para cada una de las tres puertas básicas: el nombre y el símbolo que le asignamos en nuestra librería de dispositivos Digit@Lib, la notación matemática usada para la operación lógica que implementa la puerta y su tabla de verdad. Además, definimos las puertas con las siguientes frases.

Not: La salida vale 1 cuando la entrada vale 0.
(la salida vale 0 cuando la entrada vale 1).

And: La salida vale 1 cuando todas las entradas valen 1.
(la salida vale 0 cuando alguna entrada vale 0).

Or: La salida vale 1 cuando alguna entrada vale 1.
(la salida vale 0 cuando todas las entradas valen 0).




Nombre	Símbolo	Expresión lógica	Tabla de verdad															
Not		$w = !x$ $(w = \bar{x})$ $(w = x')$	<table><tr><th>x</th><th>w</th></tr><tr><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td></tr></table>	x	w	0	1	1	0									
x	w																	
0	1																	
1	0																	
And-2		$w = x \cdot y$ $(w = x \cdot y)$ $(w = x \&\& y)$	<table><tr><th>x</th><th>y</th><th>w</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	x	y	w	0	0	0	0	1	0	1	0	0	1	1	1
x	y	w																
0	0	0																
0	1	0																
1	0	0																
1	1	1																
Or-2		$w = x + y$ $(w = x + y)$ $(w = x y)$	<table><tr><th>x</th><th>y</th><th>w</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	x	y	w	0	0	0	0	1	1	1	0	1	1	1	1
x	y	w																
0	0	0																
0	1	1																
1	0	1																
1	1	1																

Fig. 3.10 Definición de las tres puertas básicas: Not, And y Or.

Notación. En la figura 3.10 se indica entre paréntesis alguna notación alternativa para cada operación. Nosotros usamos normalmente la notación $w = !x$ para referirnos a la Not porque por un lado la notación que lleva una línea encima no se puede hacer con algunos editores y simuladores de circuitos y por otro, la notación con un apóstrofo se ve poco. Además, cuando esta decisión no induzca a errores, no pondremos el punto entre las variables para indicar la operación And. Igualmente, usaremos el símbolo $+$ para la operación Or, y lo indicaremos en negrita cuando pueda inducir a errores, para diferenciarlo de la suma aritmética sobre números naturales, enteros o reales. La operación And se denomina también producto lógico y la Or suma lógica

3.1.4 Conectando circuitos combinacionales entre si

Diseño modular

Vamos a construir un computador sencillo, pero completo, utilizando muy pocos elementos distintos entre los que se encuentran las tres puertas básicas Not, And-2 y Or-2, con las que construiremos todos los circuitos combinacionales. Construir estos circuitos, que pueden llegar a ser muy complejos, puede ser muy complicado y muy fácil cometer errores. Para evitar estas dificultades seguimos un método de diseño modular (a bloques), que además facilita la comprensión del funcionamiento del computador.

Primero diseñamos circuitos sencillos, aunque un poco más complejos que las puertas lógicas básicas. Estos circuitos sencillos los construimos directamente conectando puertas lógicas. Para cada uno de estos circuitos asociamos un símbolo, con sus entradas y salidas. Después, podremos usar estos **símbolos (dispositivos o bloques)** para, conectándolos entre sí, construir circuitos un poco más complejos. A su vez, a estos circuitos más complejos, también les asociaremos otros símbolos, que podrán ser usados como dispositivos en otros diseños más complejos todavía.

En un diseño modular como el que seguimos en este curso, es fácil detectar y corregir los errores. Cada dispositivo se construye con pocos dispositivos de los niveles inferiores. La sencillez del circuito hace que no sea muy probable que cometamos errores y si los cometemos, es fácil que los detectemos. Además, cada dispositivo nuevo se prueba exhaustivamente antes de ser incorporado a la librería de dispositivos, de forma que cuando es usado para construir otros dispositivos más complejos, es seguro que funciona correctamente.

Esquema lógico

La figura 3.11 (parte de la izquierda) muestra el esquema lógico de un circuito combinacional construido a partir de varias puertas lógicas conectadas entre sí. Un **esquema lógico** es un dibujo que representa un circuito lógico. Consiste en varios símbolos de dispositivos lógicos unidos por líneas que especifican las conexiones entre las entradas y salidas de los dispositivos. Por convenio, dos líneas que se cruzan no están conectadas excepto que se dibuje un punto en la intersección. Este esquema, también tiene tres conectores, dos de entrada y uno de salida que hemos etiquetado como x, y y w (aunque en algunos esquemas no dibujaremos los conectores). A la derecha de la figura se muestra el símbolo que asociamos al circuito de la izquierda cuando lo usamos como dispositivo en un circuito más complejo.

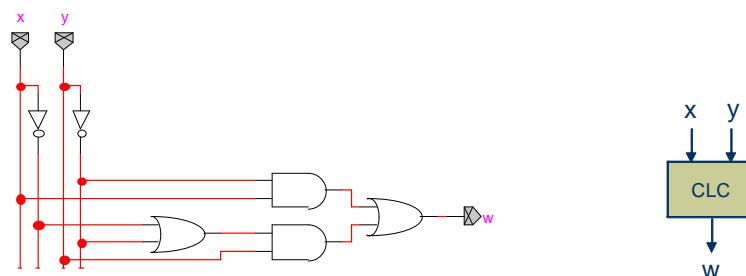


Fig. 3.11 Esquema lógico interno de un dispositivo combinacional formado mediante la interconexión de varias puertas lógicas y símbolo del dispositivo.

Reglas de interconexión de dispositivos para formar circuitos combinacionales

Para construir un circuito combinacional grande a partir de la interconexión de circuitos combinacionales más pequeños (dispositivos o bloques) hay que seguir algunas reglas, si queremos asegurar que el resultado sea un circuito combinacional y que funcione correctamente. Las reglas de interconexión prohíben tres situaciones que se resumen en la figura 3.12.

1. No se pueden conectar directamente (con una línea en el esquema lógico) dos o más salidas de dispositivos (puertas lógicas u otros dispositivos combinacionales). A nivel eléctrico, cuando se conectan distintos puntos de un circuito, todos estos puntos y las líneas que los conectan quedan con el mismo voltaje (a nivel lógico decimos que esos puntos tienen el mismo valor, 0 o 1, pero todos el mismo). Si se conectan directamente las salidas de, por ejemplo, dos puertas puede aparecer un problema eléctrico. Si una puerta está forzando en su salida el valor 0 (por ejemplo 0 voltios) y la otra está forzando un 1 (por ejemplo 5 voltios) ¿Qué voltaje aparece en la conexión cuando estas salidas se unen? A nivel eléctrico se está produciendo un cortocircuito y los dispositivos podrían dejar de funcionar. A nivel lógico diremos que hay un conflicto.

En el simulador de circuitos LogicWorks, que usamos en las prácticas de este curso, cuando hay un conflicto en una señal de un circuito, el simulador le asigna el valor lógico **C**, que no es ni 0 ni 1, con lo que podemos detectar el error fácilmente.

Tampoco se pueden conectar directamente dos o más entradas del circuito que estamos considerando, ya que las entradas del circuito, cuando este esté inmerso en un sistema más grande, vendrán de las salidas de otros circuitos y por lo tanto de las salidas de otros dispositivos. Así, conectar dos entradas de nuestro circuito supone conectar las salidas de dos dispositivos de otros circuitos y eso provoca un conflicto.

2. Todas las entradas de todos los dispositivos del circuito deben tener un valor lógico, 0 o 1, para que los dispositivos funcionen correctamente y puedan producir en sus salidas también valores lógicos 0 o 1. Por ello, cada entrada de cada dispositivo que forma el circuito tiene que estar conectada:

- a una entrada del circuito, o
- a la salida de otro dispositivo del circuito, o por último
- tener un valor constante fijado a 0 o a 1.

Si la entrada de un dispositivo está “al aire”, a nivel eléctrico se dice que está en alta impedancia, y el valor lógico que tiene, 0 o 1, depende de la tecnología. Como estamos usando un modelo independiente de la tecnología, consideramos prohibido dejar una entrada en alta impedancia. Si por descuido en un esquema dejamos la entrada de un dispositivo al aire, no podemos saber el valor lógico de la entrada y por lo tanto tampoco podemos saber el valor lógico de las salidas del dispositivo (excepto en algunos casos, como se indica más adelante).

En el simulador LogicWorks se denota con una **Z** el valor de una entrada al aire y con una **X** el de la salida de un dispositivo que no puede calcular su valor lógico ya que tiene al menos una entrada con valor **C**, **Z** o **X**.

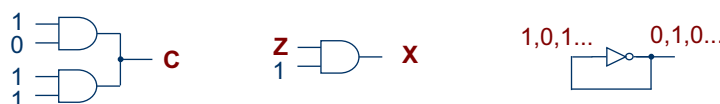


Fig. 3.12 Tres situaciones prohibidas al interconectar dispositivos combinacionales para formar otros circuitos combinacionales más complejos y notación **C**, **Z** y **X**.

3. No pueden aparecer ciclos, caminos cerrados, en el esquema del circuito. No se debe poder crear un camino que pasando a través de dispositivos llegue al mismo sitio de donde salió. Pasar a través de un dispositivo quiere decir entrar al dispositivo por una entrada y salir por una salida. Si en un circuito hay un camino cerrado puede ocurrir que los valores lógicos en el camino,
- no se estabilicen, sino que oscilen y que vayan cambiando de valor ininterrumpidamente (lo que se muestra en la figura 3.12), o bien que
 - se estabilicen y entonces se forma un elemento de memoria¹.

En la figura 3.13 se muestran los esquemas lógicos de algunos circuitos. Para dar valor lógico a algunas entradas hemos usado un conmutador binario (*binary switch*) que simula un conmutador que pone a su salida un 0 o un 1 y lo mantiene mientras no se pulse. Para ver el valor de algunos puntos del circuito y de las salidas hemos conectado un visor binario (*binary probe*), que nos muestra un 0, 1, Z, C o X, según sea el caso.

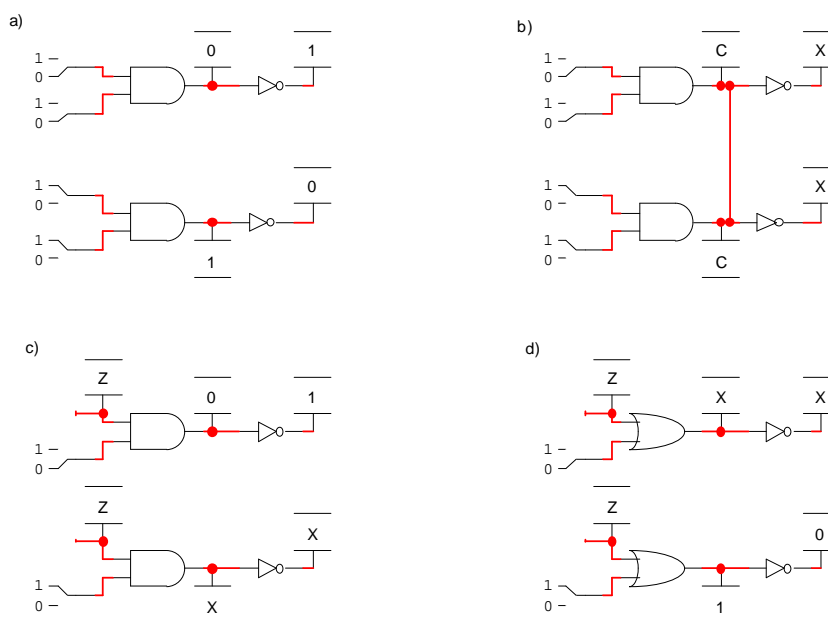


Fig. 3.13 Esquemas lógicos de cuatro circuitos. El a) es el único correcto.

El circuito a) es válido, pues cumple con las tres reglas. El b) no es válido, se viola la regla 1, ya que se han conectado las dos salidas de las puertas And. Se produce un conflicto en este punto, por lo que el visor muestra una C. A la salida de las puertas Not parece una X pues no podemos saber el valor de la

1. En general, si hay memoria en el circuito las salidas en un momento dado ya no dependen de las entradas al circuito en ese momento sino que dependen de la secuencia de valores que llegaron por las entradas desde que empezó a funcionar el circuito. Esto ya no es un circuito combinacional, es un circuito secuencial, que estudiaremos en el capítulo 5.

entrada de la puerta. El circuito c) no es válido, se viola la regla 2, ya que no se ha dado valor a una de las entradas de la puerta And. No obstante, en la situación primera, aunque una entrada no tenga valor se puede saber el valor de la salida de la puerta And, ya que la otra entrada tiene valor 0 y la And produce un 0 en la salida siempre que una entrada vale 0 independientemente del valor de la otra entrada. No podemos decir lo mismo del segundo caso, en que la entrada con valor lógico tiene un 1. El circuito d) tampoco es válido por el mismo motivo que el c).

3.4

Un **esquema lógico** formado por la interconexión de puertas lógicas y otros dispositivos combinacionales es **válido** para especificar un **circuito combinacional** si se cumplen las tres reglas siguientes.

1. No existe ninguna conexión directa entre dos o más salidas de dispositivos (puertas lógicas u otros dispositivos combinacionales). Tampoco se pueden conectar directamente dos o más de las entradas del circuito.
2. Cada entrada de cada dispositivo que forma el circuito está conectada:
 - a una entrada del circuito, o
 - a la salida de otro dispositivo del circuito, o por último
 - a un valor constante fijado a 0 o a 1.
3. No hay ciclos, caminos cerrados, en el esquema del circuito.

Ejemplo 3

La figura 3.14 muestra, a modo de ejemplo, el esquema lógico de un CLC con 3 entradas (x, y, z) y dos salidas (w1, w0). El circuito está formado por la interconexión válida de varias puertas Not, And y Or. Observad que se cumplen las tres reglas que producen un circuito combinacional válido.

3

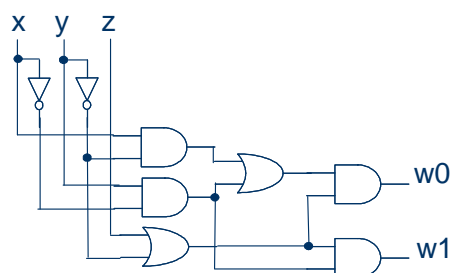


Fig. 3.14 Ejemplo de circuito combinacional correcto formado mediante la interconexión de puertas lógicas.

3.2 Análisis de circuitos combinacionales

Ya sabemos qué no tiene que ocurrir en el esquema de un circuito formado por puertas u otros dispositivos conectados entre sí para que resulte un combinacional válido. Pero, una cosa es saber esto y otra es saber qué dispositivos hay que conectar y cómo para que el circuito implemente la función que deseamos. El proceso de pasar de una descripción funcional al esquema del circuito se denomina **síntesis** y lo estudiamos más adelante en este capítulo. El proceso contrario es el **análisis**, que vemos a continuación. El análisis es importante para entender un circuito diseñado por otros y es útil para comprobar que el circuito que hemos diseñado hace lo que queríamos.

3.2.1 Análisis: creando la tabla de verdad por filas

El **análisis lógico**¹ de un circuito combinacional consiste en saber qué hace el circuito. Una forma de saberlo consiste en obtener su tabla de verdad. Esto es lo que hacemos cuando el circuito tiene pocas entradas² y una forma de hacerlo es la siguiente.

3.5

Del esquema lógico a la tabla de verdad: por filas. Para cada combinación de las n variables de entrada del circuito, debemos obtener el valor que toman cada una de las salidas de cada dispositivo/puerta del circuito (algunas de estas salidas serán las salidas del circuito, que son las que realmente nos interesan). Para saber el valor de una salida de un dispositivo hay que conocer, además de su tabla de verdad, el valor de todas sus entradas. Esta regla nos marca el orden en que podemos conocer los valores de las salidas de los dispositivos. Por ello, atravesamos 2^n veces el circuito (una por cada combinación de entrada, por cada fila de la tabla de verdad) desde las entradas hasta las salidas, calculando el valor de las salidas de cada dispositivo por el que pasamos, en el orden que nos indica la regla anterior.

Vamos a obtener la tabla de verdad del circuito de la figura 3.15. En la figura se indica, sobre un fondo gris, el valor lógico de las salidas de cada puerta para $x=0$ e $y=0$. El orden de avance por el circuito es: primero las puertas **Not**, luego las **And** y por último la **Or**, cuya salida es la del circuito, que vale $w=0$. Este valor 0 se escribe en la fila 0 de la tabla de verdad (que corresponde al valor de la salida para los valores de las entradas $x=0$ e $y=0$).

Procediendo así para cada una de las filas se completa la tabla. Sobre las líneas/conexiones del esquema de la figura 3.16, a la derecha de los valores con fondo gris que acabamos de ver se indican los valores de las variables para las entradas $x=0$ e $y=1$, a la derecha de estos los de las entradas $x=1$ e $y=0$ y por último los de $x=1$ e $y=1$.

4

1. Pueden analizarse otros aspectos del circuito, como la potencia consumida, el coste del circuito etc. No obstante, los únicos análisis que hacemos en este curso son el análisis lógico, en esta sección, y el análisis temporal, que veremos en la sección 3.4. Si hablamos de analizar el circuito, omitiendo el calificativo, nos referimos a análisis lógico.
2. En el tema 5 consideramos el caso de circuitos combinacionales de muchas entradas y salidas (por ejemplo, 32 entradas) donde no es posible escribir su tabla de verdad, ya que tendría muchas filas (2^{32} filas, en el ejemplo).

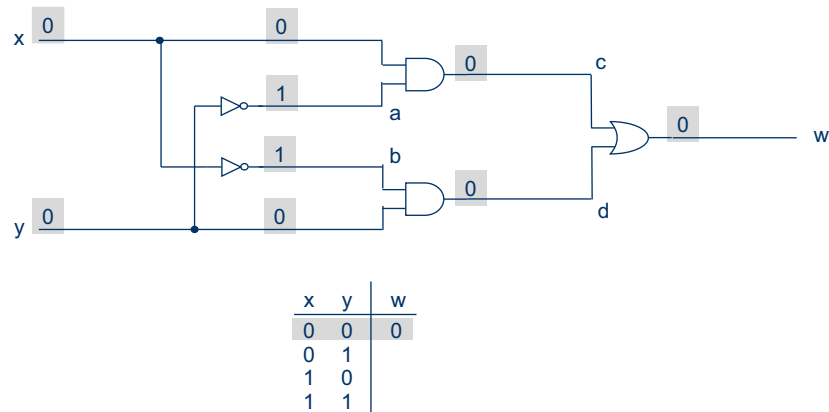


Fig. 3.15 Análisis de un circuito para obtener la fila 0 de la tabla de verdad.

La tabla de verdad resultante es la función f_0 de la figura 3.9 que se denomina Xor y este esquema es una posible implementación interna de la puerta con dos entradas Xor-2, cuyo símbolo se muestra en la figura 3.16. La salida de la Xor vale 1 cuando las dos entradas tienen distinto valor lógico. Esta puerta es muy útil en el diseño del sumador y restador que usaremos en la construcción del computador.

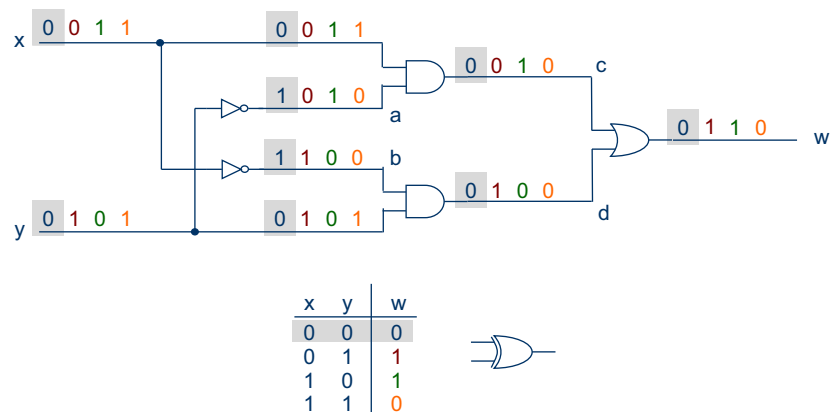


Fig. 3.16 Análisis de un circuito creando la tabla de verdad por filas. Puerta Xor-2

3.2.2 Análisis: creando la tabla de verdad por columnas

El análisis que acabamos de hacer es un poco pesado: simplemente con 5 entradas, en vez de las 2 del ejemplo anterior, el trabajo se multiplica por 8 (hay que dar 32 pasadas sobre el esquema del circuito). Veamos otra forma más rápida de encontrar la tabla de verdad, usando como ejemplo el de la figura 3.17.

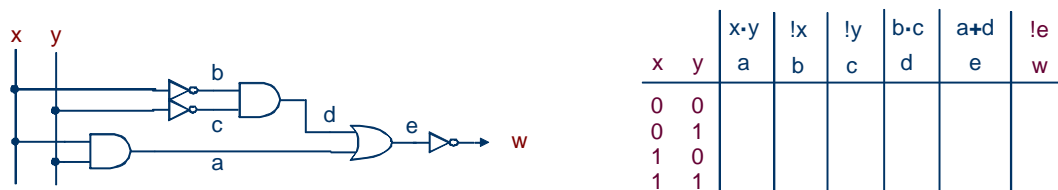


Fig. 3.17 Circuito a analizar y estructura de la tabla de verdad para el análisis por columnas.

Primero creamos las cabeceras de una tabla de verdad con las entradas del circuito, x e y , y rellenamos las dos columnas con las combinaciones de 2 bits en el orden estándar. A continuación ponemos la cabecera de las variables internas (las salidas de los dispositivos/puertas que no son salidas del circuito) a , b , etc. y de la salida del circuito, w (una columna para cada variable lógica). En la figura 3.17 se muestra esto y además, para mayor claridad, para cada variable se indica su expresión lógica en función de las variables de entrada de la puerta que la genera. Por ejemplo, la variable a es igual a $x \cdot y$ y la variable de salida w es $!e$.

Vamos a completar esta tabla por columnas. Para ello solamente hay que saber la tabla de verdad de las puertas/dispositivos que aparecen en el esquema: Not, And-2 y Or-2. Podemos comenzar por la columna a , b o c , ya que estas variables sólo dependen de las entradas x e y ($a = x \cdot y$, $b = !x$ y $c = !y$), cuyos valores conocemos a priori. Pero no podemos completar la columna d , por ejemplo, hasta que no tengamos completa la b y la c , ya que $d = b \cdot c$.

Por ejemplo, para completar la columna b sólo tenemos que copiar la columna x de la tabla de verdad pero cambiando ceros por unos y unos por ceros, ya que $b = !x$. Lo mismo ocurre para la columna c , pero negando la columna y . La columna d , que ya podemos completar después de tener la b y la c , se rellena avanzando hacia abajo haciendo la operación And lógica de cada pareja de bits de las columnas b y c (de arriba abajo: $1 \cdot 1 = 1$, $1 \cdot 0 = 0$, $0 \cdot 1 = 0$ y $0 \cdot 0 = 0$).

Así se procede, columna a columna, hasta obtener la columna de la salida. La tabla completa y el resultado del análisis (con sólo la columna de salida, w) se muestran en la figura 3.18.

x	y	x·y a	!x b	!y c	b·c d	a+d e	!e w
0	0	0	1	1	1	1	0
0	1	0	1	0	0	0	1
1	0	0	0	1	0	0	1
1	1	1	0	0	0	1	0

x	y	w
0	0	0
0	1	1
1	0	1
1	1	0

Fig. 3.18 Tablas de verdad resultantes del análisis por columnas.

Es interesante observar que aunque el circuito de la figura 3.17 es distinto del de la figura 3.16, ambos tienen la misma tabla de verdad. Se dice que son circuitos equivalentes: tienen el mismo comportamiento lógico, pero están contruidos de distinta forma. En el Apéndice I se habla de esto.

3.5

Del esquema lógico a la tabla de verdad: por columnas. Primero creamos el esqueleto de una tabla de verdad que luego rellenaremos:

- Damos un nombre distinto a cada una de las variables internas del circuito (cada salida de cada dispositivo/puerta que no sea salida del circuito).
- Creamos las cabeceras de una tabla de verdad con una columna por cada entrada y por cada variable interna y de salida del circuito.
- Rellenamos las columnas de entrada con los valores estándar.

Ahora ya podemos rellenar de una en una cada columna de la tabla. Rellenar una columna, que representa una variable de salida de un dispositivo, consiste en encontrar el valor de la variable para cada uno de los valores de las entradas del circuito (cada fila de la tabla de verdad que estamos construyendo). Para ello, simplemente hay que conocer, además de la tabla de verdad del dispositivo, el valor lógico de todas las entradas del dispositivo para todas las filas de la tabla que estamos construyendo. Esto nos marca un cierto orden a la hora de ir rellenando las columnas: antes de rellenar una columna de salida de un dispositivo/puerta hay que haber rellenado las columnas de todas sus entradas. Así se procede hasta obtener todas las columnas de salida del circuito, que son el resultado del análisis.

Ejemplo 4 Análisis lógico por columnas

A partir del esquema del circuito combinacional de la figura 3.19 obtenemos su tabla de verdad procediendo por columnas. La solución se muestra en la misma figura. Cada variable interna se ha nombrado con su expresión lógica en función de las entradas del circuito. Las columnas se han dispuesto para que puedan rellenarse de izquierda a derecha. La tabla de verdad resultante coincide con la del análisis por filas que se hizo de este mismo circuito en la sección 3.2.1: puerta Xor-2

5



Fig. 3.19 Circuito y tabla de verdad resultante del análisis por columnas del ejemplo 4.

3.3 Síntesis de circuitos combinacionales

3.3.1 Definición

La síntesis de un circuito combinacional consiste en obtener el esquema lógico del circuito a partir de una especificación de su comportamiento. De momento partimos de la especificación del circuito mediante su tabla de verdad y obtenemos un circuito formado por puertas lógicas Not, And y Or que sigue una estructura general que se denomina “en suma de minterms”.

Más adelante ampliamos las posibilidades de síntesis. Por un lado, podremos partir de una especificación del comportamiento del circuito diferente de la tabla de verdad: mediante un texto explicativo, un pseudo-lenguaje de alto nivel, etc. (sección 3.1.2). Por otro lado, podremos obtener el circuito resultante usando bloques combinacionales: mediante un decodificador y puertas Or (sección 3.3.5) o mediante una única memoria ROM (sección 3.3.6).

3.3.2 Caso sencillo de síntesis en suma de minterms

Para empezar vamos a encontrar el esquema lógico de un circuito muy sencillo, con sólo dos entradas y una salida, como el que se especifica en la figura 3.20. El método de síntesis que vamos a seguir tiene dos pasos.

x	y	w
0	0	1
0	1	0
1	0	0
1	1	1

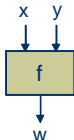


Fig. 3.20 Tabla de verdad de un circuito muy sencillo, con sólo dos entradas y una salida.

El primer paso consiste en sintetizar las funciones minterm necesarias para implementar la tabla de verdad: **nivel de puertas And**. Una función **minterm** es una función lógica que vale 1 solamente para una única combinación concreta de las entradas, para todas las demás combinaciones vale 0. Dicho de otra forma, una función minterm sólo tiene un 1 en la columna de salida de su tabla de verdad. Dependiendo de en qué fila esté el 1 se definen las distintas funciones minterm. La función minterm 0, que denominamos m_0 , tiene el 1 en la fila 0 de la tabla de verdad, la m_1 en la fila 1, y así sucesivamente.

Así, para circuitos de dos entradas, por ejemplo x e y , existen 4 posibles funciones minterm. El minterm más trivial de sintetizar es el minterm 3, que como se ve en la tabla de verdad de la figura 3.21 es igual que la tabla de verdad de la puerta And (la salida sólo vale 1 cuando las dos entradas valen 1, lo que corresponde a la fila 3 de la tabla de verdad).

La puerta And parece muy interesante para implementar una función minterm, ya que su tabla de verdad sólo tiene un 1. El problema nos aparece cuando queremos que el 1 no esté en la última fila de la tabla de verdad. Por ejemplo ¿cómo implementamos el minterm 2, $m_2(x,y)$? La respuesta se ve en la figura 3.22: al negar la entrada y de la puerta And, la salida de la puerta vale 1 sólo cuando la entrada x

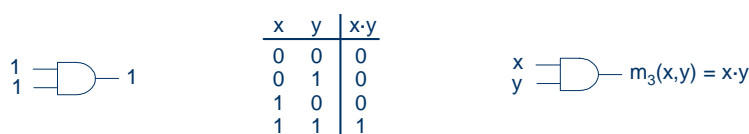


Fig. 3.21 Tabla de verdad e implementación de la función minterm 3 de dos entradas.

vale 1 y la y vale 0. Esta combinación es la única que hace que las dos entradas de la And valgan 1 a la vez y por tanto que la salida valga 1. En la derecha de la figura se muestra el esquema que implementa la función m_2 y sobre él se indica como se obtiene su **expresión lógica** (expresión algebraica donde las variables son lógicas (pueden valer solamente 0 o 1) y los operadores son la suma, +, el producto, ·, y la negación, !, lógicas): $m_2(x, y) = x \cdot !y$. Puede decirse que el esquema de la derecha es una implementación directa de la expresión lógica $x \cdot !y$.

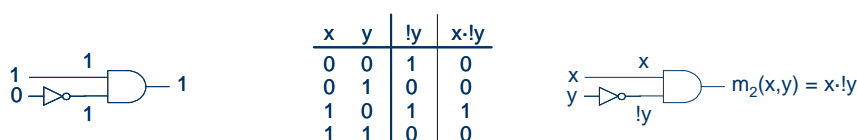


Fig. 3.22 Tabla de verdad e implementación de la función minterm 2 de dos entradas.

La tabla de verdad de las 4 funciones minterm de dos entradas se ve en la figura 3.23, donde se indica, también, la expresión lógica de cada minterm. Cualquier minterm se implementa con una puerta And y las correspondientes puertas Not según el minterm de que se trate.

x	y	!x	!y	m ₃ x·y	m ₂ x·!y	m ₁ !x·y	m ₀ !x·!y
0	0	1	1	0	0	0	1
0	1	1	0	0	0	1	0
1	0	0	1	0	1	0	0
1	1	0	0	1	0	0	0

Fig. 3.23 Tabla de verdad de las 4 funciones minterm de dos entradas.

Volvamos al caso particular de la síntesis de la tabla de verdad de la figura 3.20. Tiene 2 unos en la columna de salida, que corresponden a los minterms 0 y 3. Ya sabemos cómo implementar cada uno de estos minterms pero ¿cómo implementar la función que nos piden, que tiene 2 unos?

El **segundo paso** consiste en hacer la Or de los minterms del circuito: **nivel Or**. Fijémonos en la tabla de verdad de la Or. Cuando una de sus entradas vale 1 y la otra 0 su salida vale 1. Si conectamos cada una de las salidas de las puertas And que implementan los minterms m_0 y m_3 a las dos entradas de una puerta Or, la salida valdrá 1 tanto cuando m_1 valga 1 como cuando lo valga m_2 . Así obtenemos la expresión lógica de la función w que buscamos, $w(x,y) = m_0(x,y) + m_3(x,y) = !x \cdot !y + x \cdot y$.

La figura 3.24 muestra las tablas de verdad de la función objetivo y de las funciones internas usadas para su implementación directa a partir de la expresión lógica en suma de minterms y el esquema del circuito resultante.

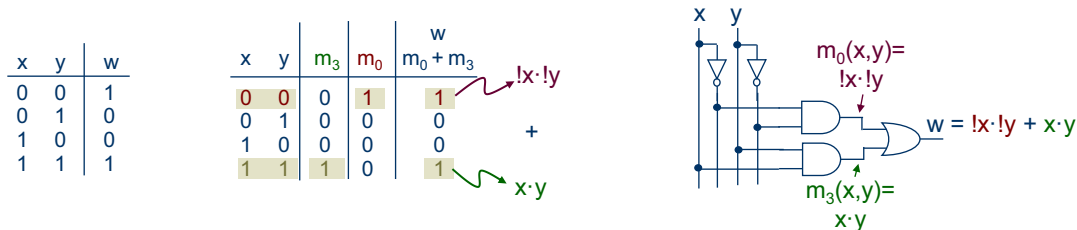


Fig. 3.24 Caso sencillo de síntesis en suma de minterms.

Ejemplo 5 Puerta Xor-2

En la última columna de la tabla de verdad de la figura 3.25 se muestra la función $!x \cdot y + x \cdot !y$, que es la suma de las funciones minterm 1 y 2 de (x, y) . Esta función es precisamente la operación lógica Xor de x e y . Esta expresión lógica de la función Xor nos puede servir para implementarla directamente con dos puertas Not para obtener $!x$ e $!y$, dos puertas And-2 para implementar los minterms m_1 y m_2 y una Or-2 para sumar los dos minterms. Así pues, el esquema lógico del circuito que analizamos en la sección 3.2.1 (figura 3.16) es precisamente la implementación en suma de minterms de la puerta Xor-2.

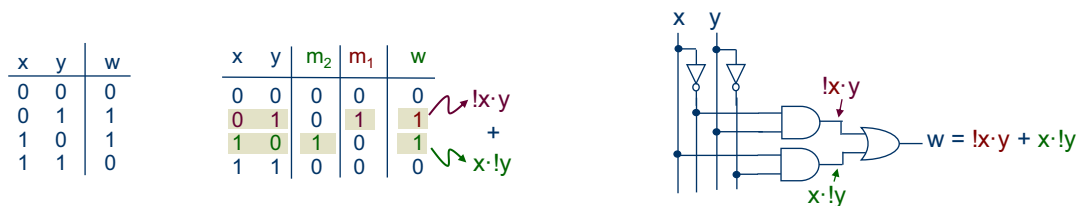


Fig. 3.25 Tabla de verdad de la función Xor de dos variables. Tabla de verdad de los dos minterms de la función. Esquema lógico resultante de la síntesis en suma de minterms.

¿Qué ocurre con funciones más complejas? Este método se nos complica un poco cuando el circuito a sintetizar tiene más de dos entradas (pues la puerta And usada para implementar cada función minterm sólo tiene dos entradas) o cuando la función objetivo tiene más de dos unos en la columna de salida de su tabla de verdad (ya que la puerta Or sólo tiene dos entradas). Para generalizar este método de síntesis necesitamos generalizar primero el concepto de puerta/función And y Or al caso de n entradas, para un n mayor que dos. Para ello veamos primero algunas propiedades básicas de las funciones (operaciones) lógicas And (\cdot) y Or ($+$) que nos ayudarán en la definición de las funciones (operaciones) And y Or para el caso general de n entradas¹.

1. Las operaciones lógicas And y Or forman una estructura matemática denominada álgebra de Boole, que tiene más propiedades (axiomas y teoremas) de las que vamos a estudiar en la siguiente sección. En el Apéndice I se define formalmente el álgebra de Boole.

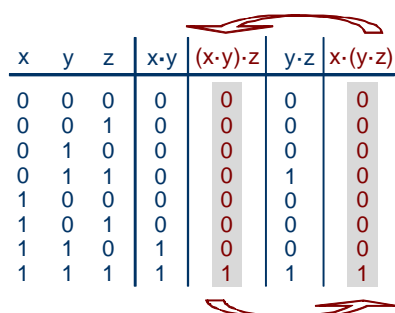
3.3.3 Algunas propiedades de la And y la Or. Puertas de n entradas

Veamos primero el caso del **producto lógico**, la función And. Las dos primeras propiedades que vamos a considerar son:

- **Conmutativa:** $x \cdot y = y \cdot x$
- **Asociativa:** $(x \cdot y) \cdot z = x \cdot (y \cdot z)$

La demostración de la primera propiedad es trivial, ya que existe una simetría de la tabla de verdad de la And respecto de x e y . Hay que probar que $x \cdot y$ es igual que $y \cdot x$ para las 4 combinaciones posibles de valores de x e y . Podemos hacer la tabla de verdad de $y \cdot x$ y observar que es idéntica a la de $x \cdot y$. No lo hacemos, ya que el único caso que hay que observar es que $0 \cdot 1$, que vale 0, es igual que $1 \cdot 0$, que también vale 0.

Para demostrar la propiedad asociativa sí que vamos a crear las tablas de verdad de la expresión de la izquierda de la igualdad, $(x \cdot y) \cdot z$, y de la expresión de la derecha, $x \cdot (y \cdot z)$, para observar que son iguales. Como aparecen 3 variables la demostración no es tan trivial como la anterior. La figura 3.26 muestra, en una sola tabla, las columnas de las expresiones que aparecen en la propiedad asociativa. Se ve que la columna de la expresión izquierda es igual a la de la expresión derecha, quedando demostrada la propiedad.



x	y	z	$x \cdot y$	$(x \cdot y) \cdot z$	$y \cdot z$	$x \cdot (y \cdot z)$
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	0	0	0	0
0	1	1	0	0	1	0
1	0	0	0	0	0	0
1	0	1	0	0	0	0
1	1	0	1	0	0	0
1	1	1	1	1	1	1

Fig. 3.26 Demostración de la propiedad asociativa de la And.

Dado que se cumplen estas dos propiedades podemos decir que:

$$\begin{aligned} (x \cdot y) \cdot z &= x \cdot (y \cdot z) = (x \cdot z) \cdot y = x \cdot (z \cdot y) = (y \cdot x) \cdot z = y \cdot (x \cdot z) = \\ &= (y \cdot z) \cdot x = y \cdot (z \cdot x) = (z \cdot x) \cdot y = z \cdot (x \cdot y) = (z \cdot y) \cdot x = z \cdot (y \cdot x) \end{aligned}$$

Y dado que el orden en que aparecen las tres variables en la expresión lógica y el orden de los paréntesis no importan podemos generalizar la idea de producto lógico a 3 variables y escribir:

$$x \cdot y \cdot z = x \cdot z \cdot y = y \cdot x \cdot z = y \cdot z \cdot x = z \cdot x \cdot y = z \cdot y \cdot x$$

Esto define la función lógica And de tres entradas y su correspondiente puerta, que denominamos And-3. La puerta And-3, su tabla de verdad y una de las posibles implementaciones internas usando puertas And-2, que proviene de la expresión $x \cdot y \cdot z = (x \cdot y) \cdot z$, se muestra en la figura 3.27. Generalizando a n entradas tenemos lo siguiente.

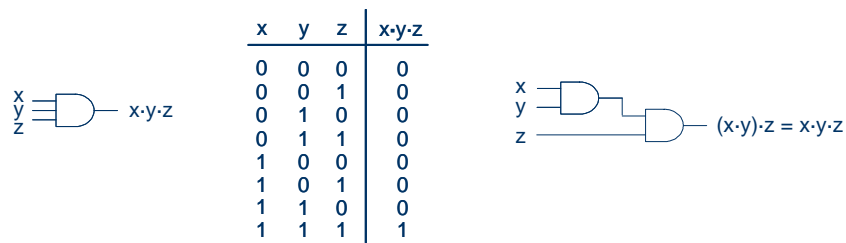


Fig. 3.27 Puerta And-3, tabla de verdad y posible implementación interna con puertas And-2.

3.3

La salida de una **puerta And de n entradas**, And- n , vale 1 cuando todas las entradas valen 1 (o también: la salida vale 0 cuando alguna entrada vale 0).

De forma equivalente se pueden demostrar las propiedades conmutativa y asociativa de la **suma lógica**, función Or:

- **Conmutativa:** $x + y = y + x$
- **Asociativa:** $(x + y) + z = x + (y + z)$

Igualmente podemos definir la función/puerta Or de n entradas. La puerta Or-3, su tabla de verdad y una de las posibles implementaciones internas usando puertas Or-2, que proviene de la expresión $x+y+z = (x+y)+z$, se muestra en la figura 3.28. También podemos generalizar a n entradas y decir que:

3.3

La salida de una **puerta Or de n entradas**, Or- n , vale 1 cuando alguna entrada vale 1 (o también: la salida vale 0 cuando todas las entradas valen 0).

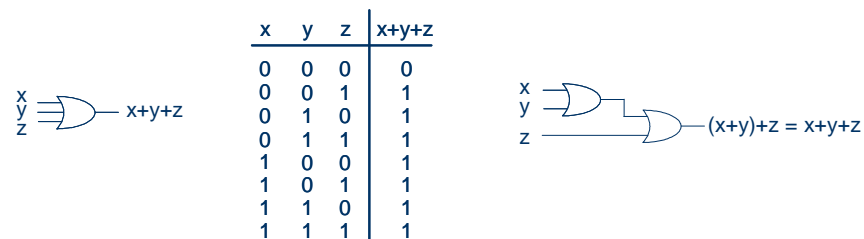


Fig. 3.28 Puerta Or-3, tabla de verdad y posible implementación interna con puertas Or-2.

Usando las propiedades asociativa y conmutativa podemos demostrar que hay muchas maneras de sintetizar una puerta de n entradas con puertas de 2 entradas. La estructura más usada es la estructura en árbol, cuya ventaja sobre una estructura lineal aparece cuando se considera el tiempo de retardo de las

puertas (en la sección 3.4). La parte superior de la figura 3.29 muestra el caso de la And-4. La estructura lineal se obtiene implementando directamente la expresión $((a \cdot b) \cdot c) \cdot d$ y la estructura en árbol se obtiene de $(a \cdot b) \cdot (c \cdot d)$. Esto puede generalizarse a n variables. De la misma forma, cambiando la operación lógica \cdot por la $+$ y la puerta And por la Or podemos construir las puertas Or de n entradas, como muestra la parte inferior de la figura 3.29 para la Or-4.

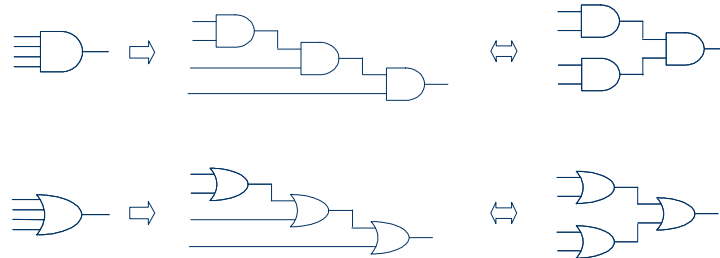


Fig. 3.29 Implementación interna de las puertas And-4 y Or-4 mediante una estructura lineal y en árbol de puertas And-2 y Or-2 respectivamente.

Más propiedades útiles de la suma y el producto lógicos. A partir de ahora, cuando necesitemos usar una puerta And u Or de n entradas dibujaremos su símbolo en el esquema del circuito que estemos sintetizando, sin preocuparnos por su construcción interna. No obstante, si sólo disponemos de puertas de 4 entradas y necesitamos construir una de 3 podemos usar una propiedad de las operaciones lógicas And y Or que dice que el producto neutro de la And es el 1 y el de la Or es el 0. Esto es:

- **Elemento neutro del producto:** $x \cdot 1 = x$
- **Elemento neutro de la suma:** $x + 0 = x$

Basta observar la tabla de verdad de la And para ver que la expresión lógica se cumple para los dos valores posibles de x , ya que $0 \cdot 1 = 0$ y $1 \cdot 1 = 1$. Lo mismo puede hacerse para la Or.

De estas propiedades se desprende que $x \cdot y \cdot z \cdot 1 = x \cdot y \cdot z$ y que $x + y + z + 0 = x + y + z$. Por ello decimos que los circuitos de la figura 3.30 son correctos. Un 1 o un 0 al lado de la entrada de una puerta en un esquema lógico indica que esa entrada está fijada con el valor 1 o 0 respectivamente.

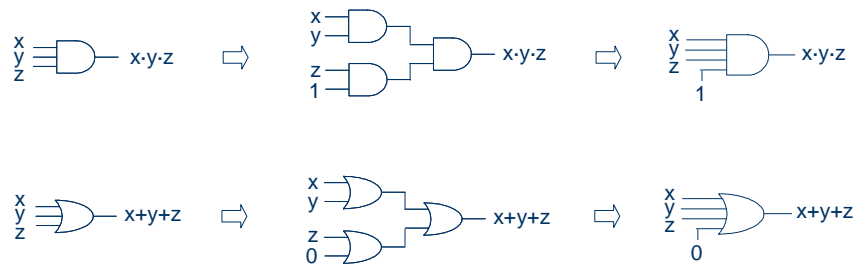


Fig. 3.30 Implementación de una puerta And-3 usando una And-4 y una Or-3 usando una Or-4.

3.3.4 Síntesis en suma de minterms. Caso general

Vamos a generalizar y formalizar, en primer lugar, la idea de función minterm de n variables lógicas y después la síntesis en suma de minterms de cualquier función lógica.

Las funciones minterm

3.6

Vamos a definir formalmente la función **minterm i** de n variables $m_i(x_{n-1}, \dots, x_1, x_0)$. Definimos primero el termino **literal k**, denotado L_k , como la variable lógica directa, x_k , o negada, $\neg x_k$. La función $m_i(x_{n-1}, \dots, x_1, x_0)$ se puede expresar como el producto lógico de los n literales L_k para $k = 0$ hasta $n-1$, donde la variable x_k del literal aparece directa o negada según que el bit k de la representación binaria con n bits del número natural i sea 1 o 0 respectivamente. Por ejemplo, $m_{13}(a,b,c,d,e) = \neg a \cdot b \cdot c \cdot \neg d \cdot e$ (ya que el 13 se codifica con 5 bits como 01101).

Así para implementar cualquier minterm se necesita una sola puerta And, de tantas entradas como variables tenga la función, y puertas Not para negar las entradas que corresponda, cosa que depende del minterm concreto que queramos implementar.

Es importante especificar el orden de las variables de entrada de una función minterm. No es la misma función, por ejemplo, la $m_3(x,y,z)$ que la $m_3(z,y,x)$, ya que $m_3(x,y,z) = \neg x \cdot y \cdot z$, mientras que $m_3(z,y,x) = \neg z \cdot y \cdot x$, y $\neg x \cdot y \cdot z$ es distinto de $\neg z \cdot y \cdot x$. Sin embargo, por ejemplo, $m_6(x,y,z)$ sí que es igual a $m_3(z,y,x)$ ya que por la propiedad conmutativa $x \cdot y \cdot \neg z = \neg z \cdot x \cdot y$.

Ejemplo 6

La figura 3.31 muestra las ocho funciones minterm de tres entradas (x_2, x_1, x_0). Por ejemplo, el minterm 3, $m_3(x_2, x_1, x_0)$, tiene un 1 en la fila 3 de la tabla de verdad, o lo que es lo mismo, la función vale 1 solamente cuando los valores de entrada valen: $x_2=0, x_1=1$ y $x_0=1$. La expresión lógica de $m_3(x_2, x_1, x_0)$ es $\neg x_2 \cdot x_1 \cdot x_0$ y su implementación se muestra también en la figura indicando la única combinación de valores de las entradas que produce un 1 en la salida

x_2	x_1	x_0	m_7	m_6	m_5	m_4	m_3	m_2	m_1	m_0
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0

$\Rightarrow m_3(x_2, x_1, x_0) = \neg x_2 \cdot x_1 \cdot x_0$

Fig. 3.31 Tabla de verdad con las 7 funciones minterm de 3 variables e implementación de m_3 .

Implementación en suma de minterms

3.6

Cualquier función lógica se puede expresar como suma lógica (Or) de tantas funciones minterm como combinaciones distintas de las entradas hacen que la función valga 1. A la **expresión** resultante se le denomina **en suma de minterms**.

Por ejemplo, la función $w(x_3, x_2, x_1, x_0)$ cuya tabla de verdad se muestra en la figura 3.32 se expresa como la suma de los minterms 2, 12 y 15, ya que w vale 1 para las filas 2, 12 y 15 de su tabla de verdad. La expresión lógica en suma de minterms es:

$$w = !x_3 \cdot !x_2 \cdot x_1 \cdot !x_0 + x_3 \cdot x_2 \cdot !x_1 \cdot !x_0 + x_3 \cdot x_2 \cdot x_1 \cdot x_0$$

También se puede expresar de forma compacta como:

$$w(x_3, x_2, x_1, x_0) = m_2 + m_{12} + m_{15}$$

6

3.6

La **síntesis en suma de minterms** de un circuito combinacional es una implementación directa de las expresiones en suma de minterms de cada una de las funciones/salidas del circuito y produce siempre esquemas lógicos con la misma estructura en dos niveles. Primero un **nivel de puertas And** con sus entradas conectadas adecuadamente a las entradas del circuito directamente o a través de puertas Not para implementar todos los minterms distintos que tiene el circuito y un segundo **nivel de puertas Or** formado por una puerta Or para cada salida del circuito que suma los minterms que tiene esa salida, esa función lógica.

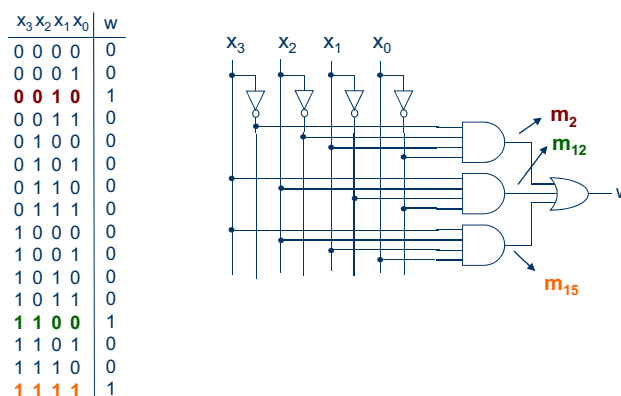


Fig. 3.32 Síntesis en suma de minterms de una función de 4 variables.

Ejemplo 7 Un circuito con una salida

En la figura 3.32 se muestra el esquema lógico resultante de la síntesis en suma de minterms de w . Como la función tiene 4 variables de entrada hacen falta puertas And-4 para implementar los minterms. Como hay 3 minterms en la función w hace falta una puerta Or-3 para sumarlos.

✍ 7

Ejemplo 8 Un circuito con dos salidas

Dada la tabla de verdad del circuito combinacional con tres entradas, (x, y, z) y dos salidas, (w_0, w_1) , que se muestra en la parte superior de la figura 3.33, vamos a obtener el esquema lógico de su implementación en suma de minterms. La solución se encuentra en la parte inferior de la figura donde también se indican las expresiones de los minterms involucrados y las expresiones en suma de minterms de cada salida del circuito. Como el minterm 3 aparece tanto en la expresión de w_1 como en la de w_0 , hemos implementado una vez solamente este minterm y lo hemos usado para las dos salidas.

✍ 8

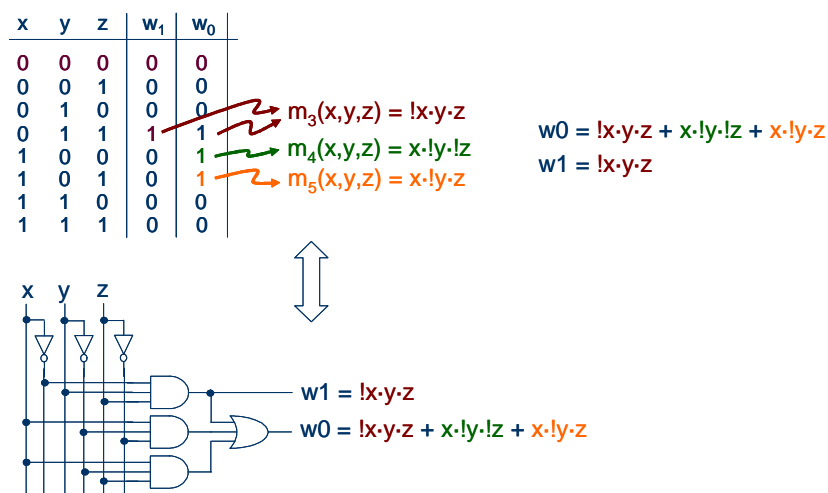


Fig. 3.33 Tabla de verdad de un circuito con 3 entradas y 2 salidas. Expresiones lógicas de los minterms involucrados en las dos funciones de salida. Esquema del circuito resultante de la síntesis en suma de minterms.

Ejemplo 9 El multiplexor en suma de minterms

Vamos a sintetizar el circuito interno del multiplexor, Mx-2-1, cuya tabla de verdad obtuvimos en el ejemplo 2 (ver la izquierda de la figura 3.34). De la tabla de verdad obtenemos primero la expresión¹ en suma de minterms de la salida w , como se ve en la figura:

- Una expresión equivalente se podría haber obtenido ordenando las variables de entrada de otra forma. En este caso los sumandos aparecerían en otro orden y las variables de cada minterm también, pero por las propiedades asociativa y conmutativa de la suma (Or) y del producto (And) lógicos sabemos que las dos expresiones son equivalentes, son la misma función.

$$w(s, x_1, x_0) = m_1 + m_3 + m_6 + m_7 = !s \cdot !x_1 \cdot x_0 + !s \cdot x_1 \cdot x_0 + s \cdot x_1 \cdot !x_0 + s \cdot x_1 \cdot x_0$$

(incluimos el orden de las entradas en la salida w , por no repetirlo en cada uno de los minterms).

Por último, en la figura 3.34 dibujamos el esquema lógico en suma de minterms. Como las variables de entrada negadas se usan en más de un minterm, negaremos sólo una vez cada variable.

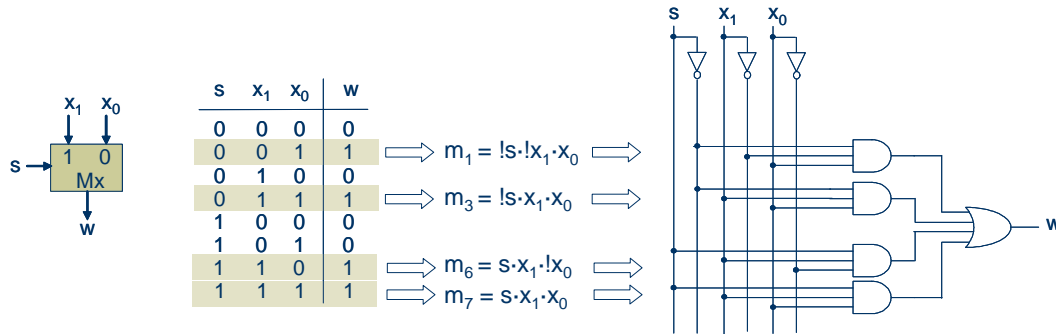


Fig. 3.34 Esquema lógico del multiplexor Mx-2-1 implementado en suma de minterms.

Se dice que las puertas (funciones lógicas) **Not**, **And** y **Or** son un conjunto de puertas (funciones) universales o completas porque con sólo ellas se puede implementar cualquier circuito combinacional. La implementación en suma de minterms es una prueba de ello¹.

3.3.5 Síntesis con decodificador y puertas Or

Otra forma de sintetizar un circuito a partir de la expresión en suma de minterms consiste en usar un bloque combinacional que se denomina **Decodificador** (*Decoder*). El decodificador implementa todas las funciones minterm del número de entradas que tiene el circuito, aunque algunos minterms no se usen porque no aparezcan en la expresión lógica que deseamos implementar. Definimos qué es un decodificador y después implementamos el circuito anterior con un decodificador y puertas Or.

Decodificador

3.7.2

Un decodificador de n entradas (a_{n-1}, \dots, a_1, a_0) y 2^n salidas ($d_{2^n-1}, \dots, d_1, d_0$), que denominamos **Dec-n-2ⁿ**, es un dispositivo lógico combinacional que implementa cada uno de los 2^n minterms de n entradas. La salida d_i es la función minterm i de las n entradas del decodificador,

$$d_i = m_i(a_{n-1}, \dots, a_1, a_0).$$

1. Existen otros conjuntos de puertas universales pero aquí, para simplificar, sólo estudiamos estas.

Veamos, a modo de ejemplo, la figura 3.35 que muestra la tabla de verdad del dispositivo decodificador de 3 entradas y 8 salidas, Dec-3-8 y su implementación interna usando una puerta And-3 para cada uno de los minterms que implementa. Por ejemplo, la salida d_3 implementa el minterm 3 de (a_2, a_1, a_0): $d_3 = m_3(a_2, a_1, a_0) = !a_2 \cdot a_1 \cdot a_0$.

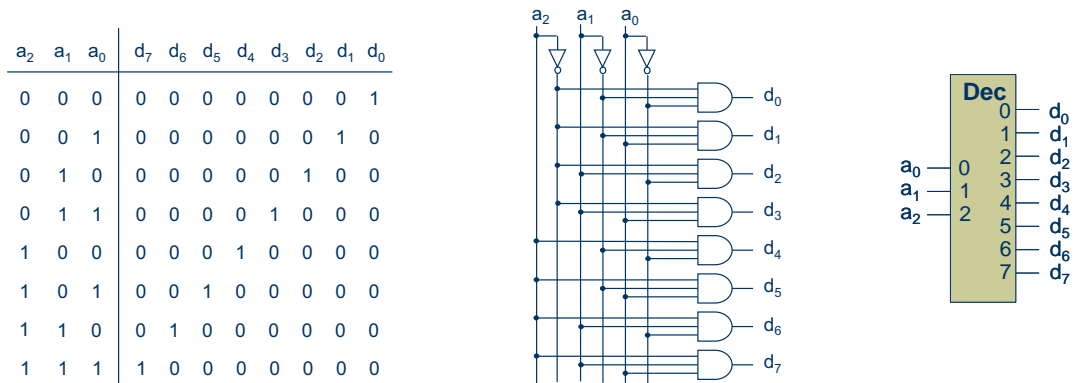


Fig. 3.35 Tabla de verdad del dispositivo Dec-3-8, su implementación interna y su símbolo.

La figura también muestra el símbolo Dec-3-8, que puede ser usado como dispositivo para formar un circuito más complejo. Es importante observar que en el interior del símbolo se han numerado las entradas de 0 a 2. A la entrada 2 se la denomina entrada de más peso y es la que se sitúa en la columna de más a la izquierda en la tabla de verdad, mientras que la entrada 0 es la de menor peso. Las salidas están numeradas de 0 a 7, para indicar que la entrada i implementa el minterm i de las entradas ordenadas como se ha dicho. Siempre hay que numerar las entradas y salidas del símbolo del decodificador ya que si no se hace así no queda definida la función que implementa cada salida.

9

14

Síntesis

3.7.2

Para sintetizar un circuito combinacional de n entradas y m salidas se debe usar un decodificador de n entradas y 2^n salidas (Dec- n - 2^n) para implementar los 2^n minterms de n entradas, y m puertas Or, una para cada salida del circuito. Cada puerta Or debe tener tantas entradas como minterms tiene la expresión en suma de minterms de la salida que implementa.

La interconexión de los dispositivos es la siguiente. Denominamos a las n entradas como entrada $n-1, n-2, \dots, 1$ y 0, siguiendo el mismo orden que usamos para crear la tabla de verdad y para denominar las funciones minterm (la entrada $n-1$ es la de la columna de más a la izquierda, la de más peso). Las n entradas del circuito se conectan a las n entradas del decodificador, la entrada k del circuito se conecta con la entrada k del decodificador. La salida de cada puerta Or es una salida del circuito. Las entradas de cada Or se conectan con las salidas del decodificador correspondientes a los minterms de la función que implementa esa puerta Or.

El multiplexor con decodificador y puertas Or como ejemplo

Veamos, en la figura 3.36, la síntesis de un multiplexor como el del ejemplo 9, Mx-2-1, pero ahora usando un decodificador para implementar los minterms y una puerta Or para obtener la única salida del dispositivo. Hemos dibujado el circuito interno del Mx-2-1 usando un decodificador 3-8, del que se muestra su circuito interno, y una puerta Or-4. Esta es la implementación en suma de minterms que hicimos en el ejemplo 2 excepto que ahora hemos dibujado todos los minterms de tres entradas y hay cuatro de ellos que no se usan. A la derecha se ha dibujado el esquema con el símbolo del Dec-3-8.

El esquema lógico con un decodificador y puertas Or es más compacto y más sencillo de dibujar que el esquema en suma de minterms. El único inconveniente de usar un decodificador es que no usamos todo el hardware interno del decodificador, pero esto no es muy importante para nosotros

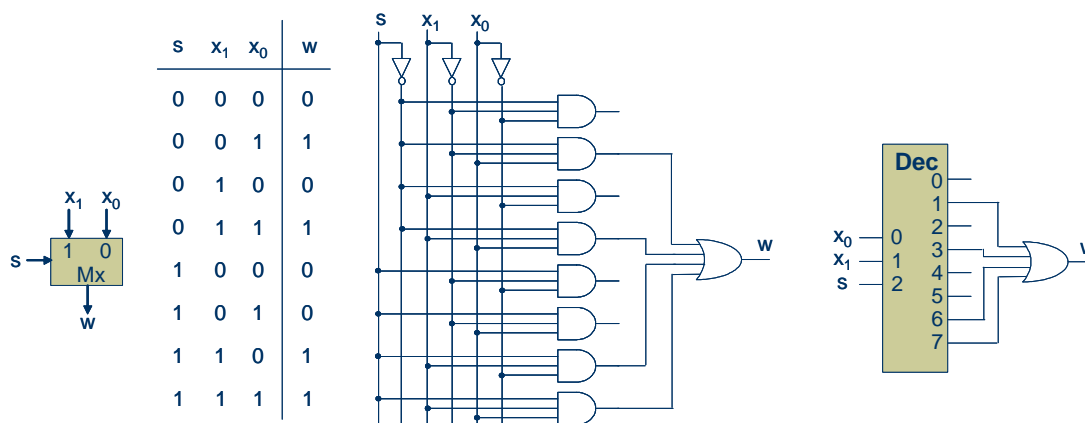


Fig. 3.36 Mx-2-1 implementado con un decodificador y una puerta Or.

Ejemplo 10 Circuito con dos salidas

Dibujamos, en la figura 3.37, el esquema lógico del circuito del ejemplo 8, pero usando un decodificador y puertas Or. Se requiere un decodificador 3-8, ya que el circuito tiene 3 entradas. En la tabla de verdad x es la variable de más peso, por lo que se conecta a la entrada 2 del decodificador. La entrada y se conecta a la entrada 1 del decodificador y la z a la 0. En este caso, aunque hay dos salidas no se requieren dos puertas Or ya que la salida w_1 sólo tiene un 1 en la columna de la tabla de verdad (su expresión lógica sólo tiene un minterm).

✍ 15

3.3.6 Síntesis con una ROM

Veamos cómo implementar cualquier circuito combinacional de la forma más compacta posible: mediante un único dispositivo denominado ROM (memoria de solo lectura, *Read Only Memory*)¹. Normalmente, usaremos una ROM cuando una implementación en suma de minterms resulte demasiado compleja (por ejemplo para más de 4 o 5 entradas)².

1. De momento olvidemos lo de memoria, que más bien puede confundirnos, ya que es un dispositivo combinacional)

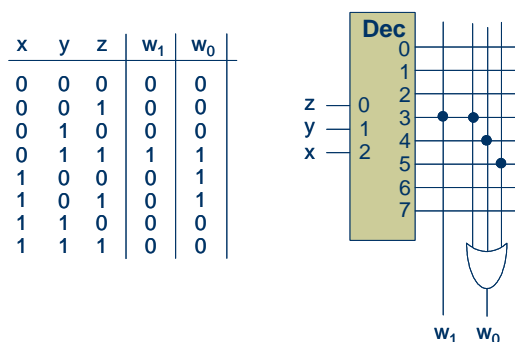


Fig. 3.37 Síntesis del circuito combinacional del ejemplo 8 con un decodificador y puertas Or.

ROM

3.9

ROM. La figura 3.38 muestra una pequeña ROM de 2 entradas y 3 salidas. El símbolo del dispositivo se muestra en a). En b) se dibuja el esquema interno, con:

- un decodificador para generar todos los posibles minterms de dos entradas,
- una matriz de puntos de conexión (cada punto concreto puede ser conectado por el fabricante), y
- 3 puertas Or de 4 entradas cada una para poder implementar tres funciones cualesquiera que sea la suma de, como máximo, 4 minterms cada una.

Conforme aumenta el número de entradas (n) de la ROM aumenta exponencialmente (2^n) el número de entradas de cada una de las puertas Or. Para evitar la complejidad de estas puertas Or con n líneas de entrada, los tecnólogos han inventado una forma de construir la Or de n entradas con una sola línea. Esta tecnología no la explicamos aquí, ya que es un tema de electrónica o de diseño VLSI. La figura 3.38.c) muestra un esquema interno de la ROM más próximo a la tecnología usada para su implementación, dibujando cada Or como una línea vertical con el símbolo de la Or en la salida.

La ROM se fabrica con cada punto concreto de la matriz de conexión conectado o no conectado de acuerdo con lo que haya especificado el diseñador al fabricante, que va a comprar miles o millones de estos dispositivos. Así, una ROM, una vez fabricada, implementa un circuito concreto y no puede volver a cambiarse¹.

2. No obstante, no implementaremos con ROM dispositivos con un número de entradas/salidas muy grande, como por ejemplo un sumador de dos números codificados con 16 bits cada uno, que tiene 32 entradas y 16 salidas, ya que haría falta una ROM excesivamente grande. En el siguiente capítulo vemos como implementar estos dispositivos grandes.

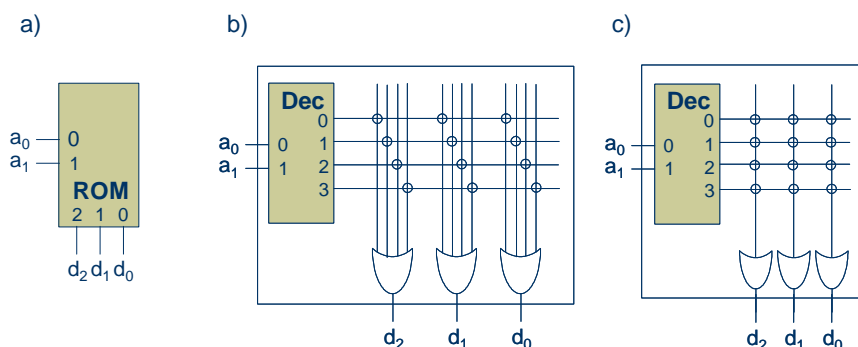


Fig. 3.38 ROM de 2 entradas y 3 salidas. a) Símbolo. b) Esquema lógico. c) Esquema tecnológico.

Síntesis

3.10

3.11

Para implementar un circuito combinacional con n entradas y m salidas hace falta una ROM de n entradas y m salidas. Las entradas de la ROM son directamente las entradas del circuito y las salidas de la ROM son las del circuito. Dentro de la ROM está todo el hardware para implementar el circuito aunque no se use todo eficientemente. La síntesis con este dispositivo es muy sencilla, solamente hay que especificar correctamente los nombres de las entradas y salidas, las etiquetas de la ROM y las conexiones (contenido) de la ROM, como se ve en las dos secciones siguientes.

17

Notación para especificar las conexiones internas (contenido de la ROM)

Cuando usamos el símbolo de una ROM, formando parte de un circuito más complejo, debemos definir claramente qué conexiones son efectivas dentro de la ROM y cuáles no lo son: el contenido de la ROM. Por ejemplo, si deseamos realizar el circuito cuya tabla de verdad se indica al margen, no dispondremos solamente del símbolo de una ROM como en la figura 3.38a). Para indicar qué puntos de la matriz de conexión están conectados y cuáles no lo están, lo que se denomina “el contenido de la ROM”, podemos indicar en algún sitio cerca del símbolo de la ROM una tabla con las funciones lógicas que implementa. Esto se muestra para nuestro ejemplo en la figura 3.39.

a_1	a_0	d_2	d_1	d_0
0	0	0	0	1
0	1	0	1	0
1	0	0	1	1
1	1	1	0	0

1. Como esto es muy rígido y sólo es económicamente viable si se van a usar miles o millones de ROMs con las mismas conexiones, los tecnólogos han creado un dispositivo sobre el cual el usuario puede programar qué conexiones son efectivas y cuáles no lo son. Esos dispositivos se denominan PROM, programable ROM. La programación se hace con un circuito sencillo. Una vez que se ha programado ya no puede volverse a reprogramar. No obstante, hay otras tecnologías que permiten borrar la programación hecha y reprogramar la PROM (EPROM). Además existen otros dispositivos programables para implementar circuitos como las PLAs y las FPGAs que tampoco vamos a tratar. Sólo nos interesa el concepto de ROM y saber que con este dispositivo podemos implementar circuitos lógicos combinacionales.

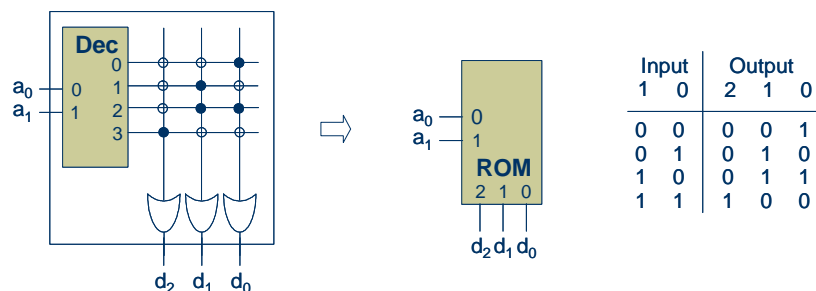


Fig. 3.39 Esquema interno de una ROM con algunas conexiones concretas efectivas y símbolo alternativo de la ROM indicando las conexiones internas mediante una tabla.

Una forma alternativa es la que se muestra a la derecha de la figura 3.40, en la que el propio símbolo de la ROM muestra su contenido. En este símbolo un 1 indica que la conexión de esa posición de la matriz de conexiones es efectiva y un 0 que no lo es. Como puede verse, esta matriz de ceros y unos es la misma que la parte de la derecha (las columnas de las salidas) de la tabla de verdad de la figura 3.39. Al usar este símbolo es muy importante indicar el orden de las filas de la matriz de conexiones. Lo usual es poner la fila 0 arriba y el resto hacia abajo, como en la tabla de verdad del circuito. En el símbolo de la figura 3.40 hemos puesto un 0 y un 3 indicando la fila 0 y la 3, ya que no hay espacio para numerar todas las filas. Es muy importante que no haya ninguna ambigüedad sobre qué funciones implementa la ROM y para ello hay que indicar el peso de las entradas, el orden de las salidas etc. Veamos esto con un ejemplo.

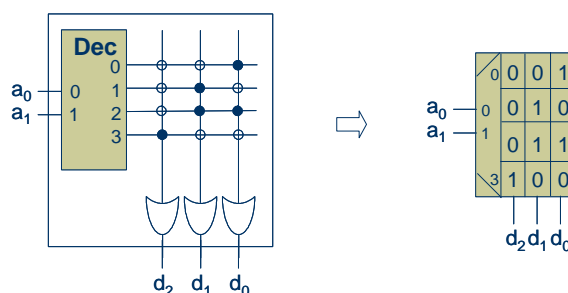


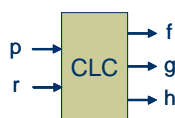
Fig. 3.40 Esquema interno de una ROM con algunas conexiones concretas efectivas y símbolo alternativo de la ROM indicando las conexiones internas.

Importancia del etiquetado de las entradas y salidas

Es muy importante resaltar que para que una ROM sea una implementación correcta de la tabla de verdad del circuito que queremos implementar hay que especificar correctamente

- los nombres de las líneas de entrada y salida del circuito (estos son los nombres de las variables de entrada y salida que nos dan en la tabla de verdad y las líneas de entrada y salida del circuito son las líneas de entrada y salida de la ROM),

- las etiquetas internas de la ROM (el orden de los bits de entrada a la ROM: $0, \dots, n$; las etiquetas de las filas de la ROM: $0, \dots, 2^n - 1$; y el orden de los bits de cada fila contenida en la ROM -las salidas de la ROM) y
- el contenido de la ROM (conexiones o ceros y unos).



Para ejemplificar esta importancia analizamos los circuitos dibujados en la parte superior de la figura 3.41 que consisten en una sola ROM cada circuito. Los circuitos tienen en común el nombre de sus entradas y de sus salidas (ver figura a la derecha de este párrafo). La solución del análisis, la tabla de verdad de cada uno de los circuitos se muestra en la parte inferior de la misma figura.

Se observa que el circuito b) y el c) son equivalentes, la tabla de verdad es la misma, aunque las columnas de salida en la tabla de verdad y de la ROM tienen distinto orden en b) que en c). Además, el orden de las entradas de cada circuito es diferente, pero el resultado es el mismo ya que se ha cambiado también de manera acorde el orden de las etiquetas de las entradas del símbolo de la ROM.

Los circuitos a) y d) también son equivalentes ya que su tabla de verdad es la misma. La diferencia en los dos circuitos es que en el símbolo de la ROM del circuito d) se han ordenado las filas con la fila 0 abajo y la 3 arriba.

Por otro lado, los circuitos a) y b) no son equivalentes a pesar de que el nombre de las entradas y salidas es el mismo en los dos esquemas y el contenido de la ROM es aparentemente el mismo. La diferencia estriba en el orden de las etiquetas de las entradas dentro del circuito. En el a) la entrada p es la de menor peso y en el b) es la de mayor peso. Este pequeño detalle hace que los circuitos no sean equivalentes. Por todo esto, es importante etiquetar correctamente las entradas de la ROM.

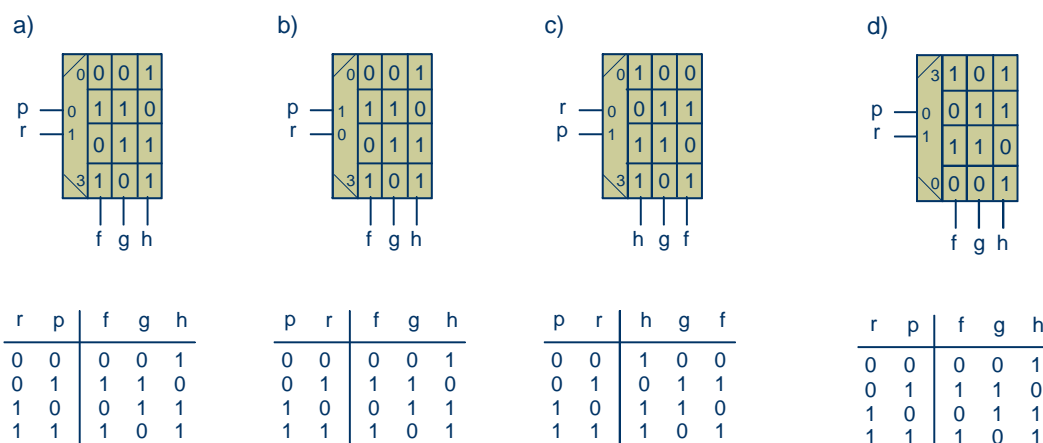


Fig. 3.41 Circuitos implementados con una sola ROM y sus respectivas tablas de verdad.

Ejemplo 11 Circuito con dos salidas

La figura 3.42 muestra la tabla de verdad del circuito que sintetizamos en suma de minterms en el ejemplo 8 y con un decodificador y puertas Or en el ejemplo 10 y a la derecha su implementación con una sola ROM. Puede observarse como cada vez es más sencillo el esquema lógico del circuito (aunque para ello se usan cada vez más puertas, que no se ven en los dibujos con bloques).

✍ 18

x	y	z	w ₁	w ₀
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	1	1
1	0	0	0	1
1	0	1	0	1
1	1	0	0	0
1	1	1	0	0

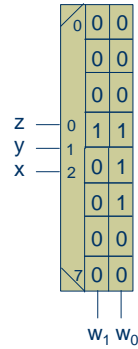


Fig. 3.42 Síntesis con una ROM del circuito combinacional del ejemplo 8.

Visión de la ROM como memoria de sólo lectura

El dispositivo ROM que hemos visto se denomina memoria ROM, *Read Only Memory*, memoria de sólo lectura. Hemos dejado la visión del dispositivo como memoria de sólo lectura para el final de la sección ya que de momento nos ha interesado más la visión de la estructura interna de la ROM con un decodificador y puertas Or, para que se vea más claro que una ROM permite implementar funciones de manera equivalente a la implementación en suma de minterms o a la implementación con un decodificador y puertas Or. Veamos ahora la ROM desde el punto de vista de memoria.

Las n entradas de una ROM se pueden ver como un vector de n bits que codifica en binario un número natural, denominado **dirección**. El rango de direcciones de una ROM de n entradas es de 0 a 2^n-1 . La etiqueta $n-1, n-2, \dots, 1, 0$, de cada entrada indica el peso, $2^{n-1}, 2^{n-2}, \dots, 2^1, 2^0$ de los bits que forman la dirección. Se suelen denominar $a_{n-1}, a_{n-2}, \dots, a_1, a_0$ (la *a* de dirección en inglés, *address*) a los n bits de las entradas de la ROM, de la dirección.

3.9

Una **memoria ROM** de n entradas y m salidas es un dispositivo que almacena 2^n palabras de m bits cada palabra. Las 2^n palabras fueron escritas, almacenadas, en el interior del dispositivo por el fabricante, cada palabra en una posición de la memoria. La memoria ROM puede verse como una cajonera (almacén) de 2^n cajones (posiciones de memoria) en los que se almacena una palabra de m bits en cada cajón (el contenido de la memoria). Los cajones están numerados, cada uno tiene una dirección, de 0 a 2^n-1 .^a

- a. Como el número máximo de palabras que puede almacenar una ROM es una potencia de 2 (2^n), para almacenar k palabras de m bits hace falta una ROM que tenga, como mínimo, un número de entradas (n) igual al logaritmo en base 2 de k , y un número de salidas igual a m . Esto es así, ya que se tiene que cumplir que:
- $$2^{n-1} < k \leq 2^n.$$

La memoria ROM mantiene siempre su contenido tal como lo creó el fabricante, aunque el dispositivo esté sin tensión de alimentación. Cuando el dispositivo está inmerso en un circuito alimentado el funcionamiento es el siguiente. Los m bits de salida de la ROM tienen los valores binarios de la palabra que se encuentra almacenada en la dirección que indican los n bits de dirección de la ROM. La figura 3.43 muestra una memoria de 8 palabras (3 bits de entrada) de 6 bits cada palabra. Por ejemplo, la posición con dirección 2, la posición 2 de memoria, contiene el vector de bits: 011111.

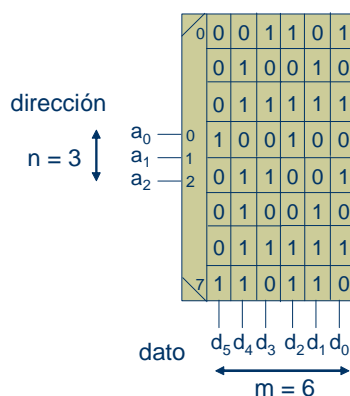


Fig. 3.43 Ejemplo de memoria ROM de 8 palabras de 6 bits cada una.

3.4 Completando el modelo de un circuito combinacional. Tiempos de propagación

Hasta el momento nos hemos familiarizado con el funcionamiento a nivel lógico de los circuitos combinacionales, que especificamos formalmente mediante su tabla de verdad¹. La tabla de verdad modela de forma incompleta el funcionamiento de un circuito combinacional real. Hay otras características de una puerta, o en general de un circuito combinacional, que también son importantes cuando se diseña un circuito. En este curso sólo consideraremos una de estas características: el tiempo de propagación² (en inglés *propagation delay*) desde cada entrada a cada salida del circuito; y no consideraremos, por ejemplo, el espacio que ocupa o la potencia que consume.

En el “modelo lógico” de una puerta, y por tanto de un circuito combinacional construido conectando puertas, se ha considerado que el valor de la salida en un instante concreto del tiempo es función del valor de las entradas en ese mismo instante. O sea, si un cambio de valor en una entrada ha de provocar un cambio en el valor de una salida, este cambio es instantáneo. La realidad es muy distinta y un cambio en una entrada tarda un tiempo en propagarse a la salida: este es el tiempo de propagación de la señal a través del circuito, desde esa entrada a esa salida. El porqué del tiempo de propagación y cómo construir la puerta para que el tiempo de propagación sea pequeño, es un tema de física, electrónica, o diseño digital avanzado, que no tratamos aquí.

3.4.1 Análisis temporal de las puertas básicas

Vamos a definir el comportamiento temporal de las tres puertas básicas con las que construimos los circuitos combinacionales y de esta forma formulamos el “**modelo temporal**” que usamos.

Not

Comenzamos por la puerta más simple, la que tiene sólo una entrada: la puerta Not. El cronograma de la figura 3.44 muestra cómo varía la señal de salida de la puerta Not ante una serie de cambios a lo largo del tiempo del valor lógico de la señal de entrada. En los primeros momentos que muestra el cronograma, la entrada vale 0 y la salida vale 1, como corresponde a su tabla de verdad. Pero en el instante en que la entrada pasa de 0 a 1 la salida debería cambiar de 1 a 0, según su tabla de verdad. La realidad es que el cambio de la entrada tarda un tiempo en propagarse a la salida. Usualmente, en la bibliografía de circuitos digitales, se denota este tiempo como T_{pHL} , tiempo de propagación de alto a bajo, ya que la salida ha pasado de 1 (valor alto, *High*) a 0 (valor bajo, *Low*). El cronograma de la figura muestra también cómo se comporta la puerta al pasar la entrada de 1 a 0. En este caso la salida pasa de 0 a 1 después del tiempo T_{pLH} .

El tiempo T_{pHL} tiene distinto valor que el T_{pLH} y las diferencias entre ellos dependen de la tecnología. No obstante, para simplificar el análisis temporal, vamos a considerar en nuestro modelo temporal que en las tres puertas básicas T_{pHL} tiene el mismo valor que T_{pLH} . Como construimos todos los circuitos

-
1. También se puede especificar mediante una expresión lógica. Un tipo de expresión lógica es la expresión en suma de min-terms, que hemos visto, pero, como veremos en el apéndice sobre las propiedades de las funciones Not, And y Or, hay muchas expresiones distintas que tienen la misma tabla de verdad y que se denominan expresiones equivalentes.
 2. También se le denomina **tiempo de retardo** (*delay time*) o simplemente **retardo**.

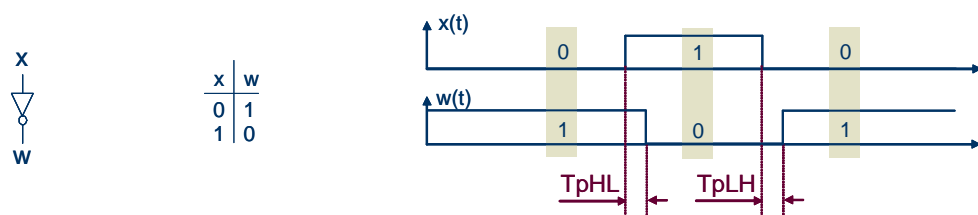


Fig. 3.44 Cronograma que muestra los tiempos de propagación de una puerta Not.

combinacionales con las puertas Not, And-2 y Or-2, podemos decir que en cualquiera de nuestros circuitos $T_{pHL} = T_{pLH}$. Por esto, de ahora en adelante, hablaremos simplemente de tiempo de propagación, T_p , sin importarnos si la salida pasa de 0 a 1 o de 1 a 0.

And-2

Veamos, en la figura 3.45, el cronograma del comportamiento de la puerta And-2. Inicialmente, las dos entradas valen 0 y la salida vale 0, como corresponde al comportamiento lógico. Cuando la entrada x pasa de 0 a 1 y la y se mantiene a 0, ocurre lo que indica la tabla de verdad: la salida no cambia. Pero más adelante, cuando la entrada y pasa de 0 a 1 y la entrada x sigue valiendo 1, la salida que debería cambiar de 0 a 1 inmediatamente, tarda un tiempo en hacerlo. Este es el tiempo de propagación de la entrada y a la salida w , y lo denotamos como $T_{p_{y-w}}$ (inicialmente vamos a diferenciar las dos entradas de la And-2). La figura también muestra el tiempo de propagación de la entrada x a la salida w , $T_{p_{x-w}}$. Sobre los cronogramas se indica cada cambio de la señal de salida con la punta de una flecha, indicando con el origen de la flecha el cambio en la señal de entrada que lo provocó.

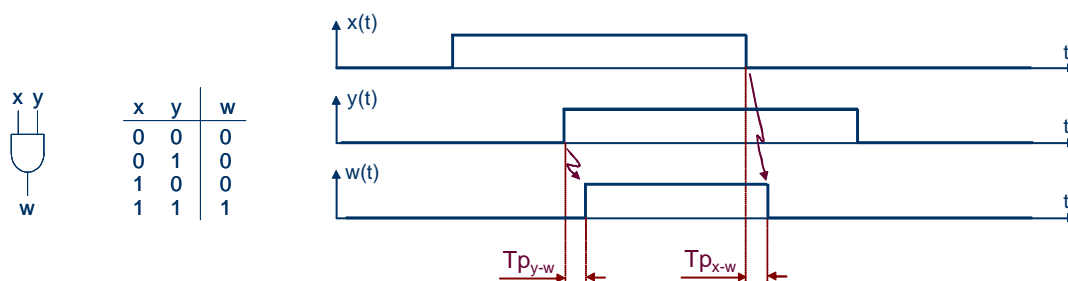


Fig. 3.45 Cronograma que muestra tiempo de propagación de una puerta And-2.

Or-2

El comportamiento temporal de la puerta Or-2 es equivalente al de la And-2 considerando, claro está, su distinto comportamiento lógico. El cronograma del comportamiento de la puerta Or-2, para las mismas variaciones temporales de las entradas que hemos considerado para la puerta And-2, se muestra en la figura 3.46.

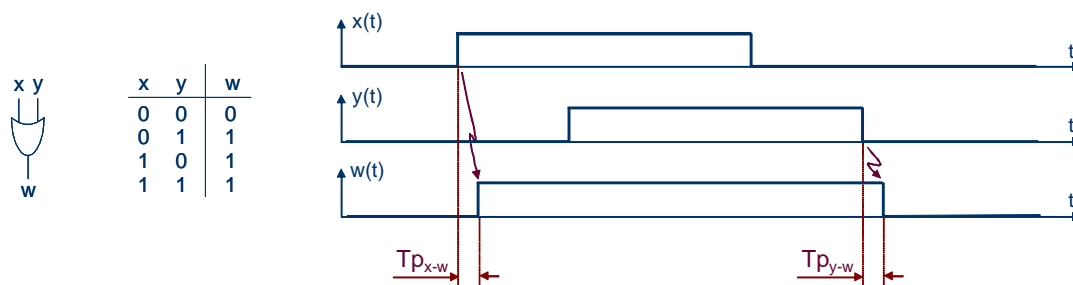


Fig. 3.46 Cronograma que muestra tiempo de propagación de una puerta Or-2.

Tiempo de propagación de las puertas básicas

Vamos a definir lo que entendemos por tiempo de propagación, T_p , de una puerta básica. Para ello, dos aclaraciones.

Por un lado, y desde este momento, vamos a considerar que al realizar una puerta And-2 u Or-2, con cualquier tecnología, el tiempo de propagación desde cualquiera de sus dos entradas resulta ser el mismo ($T_{p_{x-w}} = T_{p_{y-w}}$). Esto, junto a lo ya dicho al estudiar la puerta Not, (que para las tres puertas básicas consideramos que los tiempos de propagación al pasar la salida de 0 a 1 son los mismos que al pasar de 1 a 0, $T_{p_{LH}} = T_{p_{HL}}$), hace que para las tres puertas Not, And-2 y Or-2 hablemos simplemente de **tiempo de propagación de la puerta, T_p** (sin distinguir si la salida cambia de 0 a 1 o de 1 a 0, ni qué entrada produce el cambio).

Por otro lado, a la vista de los cronogramas de las puertas And-2 y Or-2 observamos que el comportamiento temporal (y lógico) de la salida provocado por un cambio en una entrada depende del valor de la otra entrada. Por ejemplo, en la Or-2 cuando una entrada vale 1 la salida vale 1 y no cambia, por muchos cambios que se produzcan en la otra entrada. Podríamos decir que en este caso el tiempo que pasa desde que se producen los cambios entre la entrada y la salida es nulo, mientras que si la entrada que no cambia vale 0 sí que se produce un cambio en la salida que tarda un tiempo distinto de cero en propagarse. A pesar de ello, por definición, el **tiempo de propagación de una puerta, T_p** , es el máximo de estos tiempos (al hablar de T_p no se dice ni el cambio concreto de la entrada que produce el cambio en la salida (de 0 a 1 o de 1 a 0) ni el valor concreto (0 o 1) que tiene la entrada que no varía. Así:

- En la puerta And-2 las entradas son intercambiables: a nivel lógico la And es conmutativa, $x \cdot y = y \cdot x$, y a nivel temporal también, $T_{p_{x-w}} = T_{p_{y-w}} (= T_p)$. Podemos decir lo mismo para la Or-2. (pero no para circuitos formados por varias puertas, como veremos enseguida).
- Para las tres puertas básicas, es seguro que pasado el tiempo de propagación de la puerta, la salida ha tomado el valor lógico que indica su tabla de verdad. También podemos decir que, según nuestro modelo temporal, el comportamiento de la señal de salida va “retardado” un tiempo T_p respecto a lo que sería su comportamiento considerando solamente el modelo lógico.

Tiempos en la librería de dispositivos Digit@Lib

Consideramos que las puertas básicas de nuestra librería, Not, And-2 y Or-2, con las que construiremos el computador, tienen los siguientes tiempos de propagación, que medimos en **unidades de tiempo, u.t.**, para ser independientes de la tecnología (no indicaremos cuántos nanosegundos, ns, corresponden a cada u.t.). Ponemos entre paréntesis el dispositivo a que hace referencia el tiempo:

$$Tp(\text{Not}) = 10 \text{ u.t.}, Tp(\text{And-2}) = 20 \text{ u.t.} \text{ y } Tp(\text{Or-2}) = 20 \text{ u.t.}$$

Con esto queda completado el modelo (lógico y temporal) de las puertas básicas. De ahora en adelante, si no decimos lo contrario, consideramos que las puertas básicas que usaremos tienen estos tiempos de propagación.

Sobre cronogramas. Por último, un comentario antes de pasar al siguiente punto. En general, un cronograma muestra el comportamiento del circuito para unos casos particulares de valores y cambios de las entradas, pero no suele representar todas las posibilidades, no es exhaustivo como lo es una tabla de verdad. El que dibuja el cronograma decide qué situación concreta quiere mostrar. Por ejemplo, en los cronogramas anteriores para las puertas And-2 y Or-2 no se muestra qué ocurre cuando la entrada x pasa de 0 a 1 mientras la y permanece con valor 1.

3.4.2 Análisis temporal de un circuito combinacional

¿Qué pasa con los tiempos de propagación de un circuito combinacional formado por puertas lógicas¹ conectadas entre sí? Veámoslo primero con un circuito muy simple y luego ya lo complicaremos.

Caso simple: circuitos con un solo camino de cada entrada a cada salida

Con un circuito tan sencillo como el de la figura 3.47, con dos entradas (x e y) y una salida (w), vamos a ver que el tiempo de propagación es distinto según que entrada consideremos ($Tp_{x-w} \neq Tp_{y-w}$). El circuito está construido con una puerta Not y una And-2, implementando la función $w = x \cdot !y$. La figura también muestra la tabla de verdad de la señal interna del circuito (!x) y de la salida del circuito (w).

Analizamos el circuito con un cronograma. Para ello usamos los mismos cambios a lo largo del tiempo de las señales de entrada que hemos usado en los cronogramas anteriores (figura 3.45 y figura 3.46), aunque sabemos que con estos cambios no se cubren todas las posibilidades.

Pero, ¿cómo se dibujan cronogramas para circuitos con varias puertas? Dibujar un cronograma consiste en encontrar los valores de cada salida del circuito a lo largo del tiempo, a partir de los valores concretos que nos dan de las señales de entrada a lo largo de un intervalo de tiempo (fuera de ese intervalo los valores de las entradas son desconocidos). Cada señal de salida del circuito es la señal de salida de una puerta² y para poder dibujar la señal de salida de una puerta necesitamos, además de saber la tabla de verdad de la puerta, conocer los valores a lo largo del tiempo de las entradas de la

1. Todos los circuitos combinacionales que estudiamos están contruidos con puertas lógicas Not, And y Or conectadas entre sí, aunque el esquema lógico del circuito sea un esquema a bloques y puede que en el nivel superior no se vean todas las puertas sino que aparezcan los símbolos de dispositivos/bloques combinacionales como una caja negra (no se ve su interior). Internamente estos bloques están contruidos, tras uno o varios niveles de bloques, con las puertas básicas.
2. Si no es así quiere decir que la salida está conectada directamente con una entrada y como las conexiones no producen retardo estamos en un caso trivial: la señal de salida a lo largo del tiempo es igual a la de la entrada.

puerta. Pero las señales de entrada de esa puerta, en general, también son salidas de otras puertas y por lo tanto necesitaremos saber el valor a lo largo del tiempo de las señales de entrada de esas puertas, y así hasta llegar a las señales de entrada del circuito, de las que conocemos sus valores a lo largo del tiempo.

3.12

Procedimiento para dibujar cronogramas para circuitos con varias puertas. Se dibuja el cronograma de la señal de salida de cada una de las puertas del circuito para el periodo de tiempo en que es posible hacerlo (antes y después el valor de la salida será *desconocido*), comenzando por las puertas conectadas directamente a las entradas del circuito y terminando con las puertas cuyas salidas son las salidas del circuito. El orden en que se pueden dibujar las señales viene dado por la regla: no se puede dibujar la señal de salida de una puerta hasta que no se conocen los valores a lo largo del tiempo de todas sus entradas. Con esto hemos reducido el problema a saber dibujar el cronograma de la señal de salida de una puerta conocidas sus señales de entrada a lo largo de un intervalo de tiempo.

¿Cómo se dibuja el cronograma de una puerta básica? Se puede hacer en dos pasos, para el primero hay que conocer la tabla de verdad de la puerta y para el segundo su tiempo de propagación, T_p . Suponemos que conocemos el valor de las entradas desde t_0 a t_f . Primero se dibuja el valor de la señal de salida de t_0 a t_f , suponiendo que el tiempo de propagación es nulo: aplicando instantáneamente los cambios en la salida que provocan los cambios en las entradas de acuerdo con la tabla de verdad de la puerta. En el segundo paso se retarda la señal que acabamos de obtener (se desplaza a la derecha) T_p unidades de tiempo. El valor de la señal de salida desde t_0 hasta $t_0 + T_p$ no se conoce y por lo tanto: no se dibuja, o se dibuja un valor *desconocido* (se ensombrece el espacio entre el 0 y el 1)^a.

Cronograma de una puerta básica en un solo paso. Dada la sencillez del proceso anterior, también podemos ir dibujando directamente la señal de salida de izquierda a derecha en un solo paso. Comenzamos dibujando un valor *desconocido* (sombras entre el 0 y el 1) desde t_0 hasta $t_0 + T_p$ y en ese momento pasamos a dibujar el valor de la salida que nos indica la tabla de verdad para las combinaciones concretas de las entradas en t_0 (T_p u.t. antes). Este valor de salida se mantiene durante un tiempo igual al que transcurre desde t_0 hasta el primer cambio de las entradas (de una o de varias entradas a la vez). Pasado este tiempo se cambia la salida de acuerdo con la tabla de verdad para el valor de las señales de entrada después del primer cambio (T_p u.t. antes). Este valor de salida se mantendrá durante un tiempo igual al que transcurre desde el primer cambio de las entradas hasta que se produce el segundo cambio,... y se repite el proceso hasta el final del cronograma (dibujando la salida hasta T_p u.t. después de t_f).^b

- En algunos cronogramas hemos dejado el valor que tiene la salida en T_p , ya que suponemos que al menos T_p u.t. antes del inicio del tiempo del cronograma las entradas valían lo mismo que al inicio.
- Cuando los valores de todas las entradas de la puerta no se conocen a la vez (en t_0 , como se ha supuesto) sino que una entrada de una puerta tiene valor desconocido y la otra conocida, la salida puede ser conocida ya que la And de 0 por cualquier valor es 0 y la Or de 1 más cualquier valor es 1. Esto se puede tener en cuenta para dibujar un 0 o un 1 en el cronograma en vez del valor *desconocido*, cuando esto sea posible.

En nuestro ejemplo sencillo sólo hay dos puertas. Para poder dibujar la salida de la And-2, que es la salida del circuito (w), tenemos que conocer primero la salida de la Not ($!x$) que es una de sus entradas. la única señal interna es la salida de la Not que denotamos como $!x$. El cronograma para la señal $!x(t)$ se muestra en la figura 3.47, considerando que $T_p(\text{Not}) = 10$ u.t. Se ha dibujado un valor *desconocido* de las entradas hacia el final del cronograma, cosa que podría no haberse hecho (como antes del inicio del cronograma donde no se han dibujado valores desconocidos). Además, se han dibujado las flechas indicando los cambios de la señal de salida y quién los provoca, así como los tiempos de retardo.

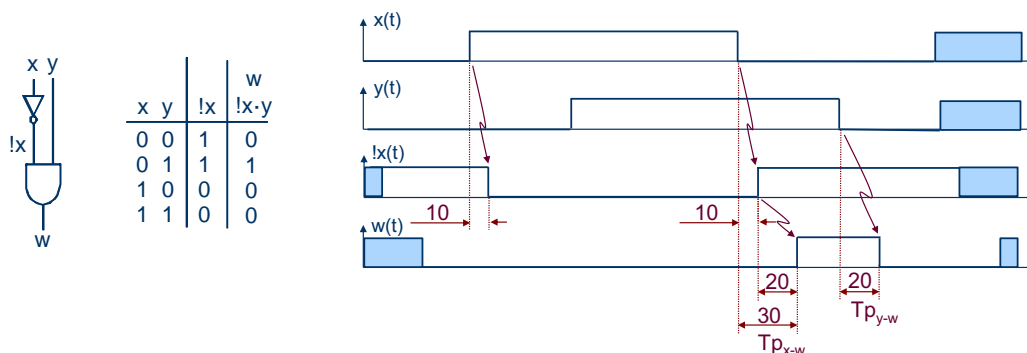


Fig. 3.47 Cronogram de un circuito que implementa la función $w = !x \cdot y$.

Una vez tenemos el comportamiento temporal de $!x$ ya podemos dibujar el de $w = !x \cdot y$ ya que también tenemos el cronograma de la señal de entrada y . Repetimos el proceso pero ahora aplicando los cambios lógicos de la And-2 con la señal de entrada $!x$ y la y . La señal w resultante se ve en la figura, suponiendo que $T_p(\text{And-2}) = 20$ u.t.

Para centrar la atención, en la figura 3.48 hemos dibujado solamente las señales de entrada y salida del circuito, extrayéndolas del cronograma anterior. Se observa que algunos cambios en las entradas no producen cambios en la salida: por ejemplo cuando las entradas (x, y) pasan de (0, 0) a (1, 0) o cuando pasan de (1, 0) a (1, 1). Pero cuando pasan de (1, 1) a (0, 1) la salida pasa de 0 a 1 con un retardo de 30 u.t. Este valor (30 y no 0) es el tiempo de propagación $T_{p_{x-w}}$, ya que es x la que ha provocado el cambio.

¿Por qué en este circuito $T_{p_{x-w}}$ es distinto de $T_{p_{y-w}}$? La respuesta está en la izquierda de la figura 3.48. Porque el camino de la entrada x a la salida es más largo (en tiempo) que el camino de la entrada y a la salida. Para que un cambio en la señal x llegue hasta la salida w ha de propagarse primero hasta la

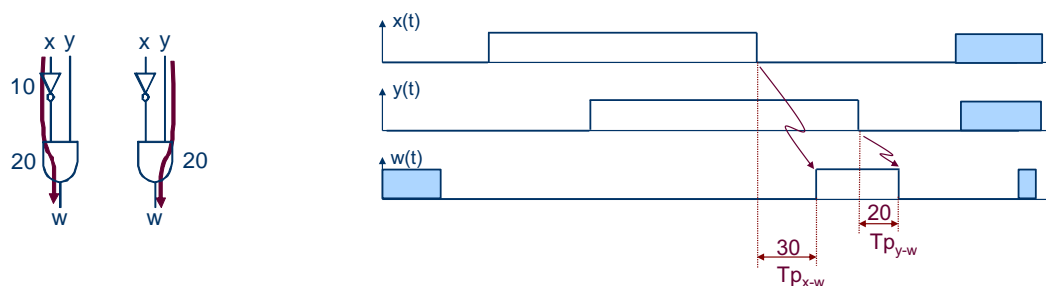


Fig. 3.48 Cronograma de las entradas y salida de un circuito que implementa la función $w=!x \cdot y$.

salida de la puerta Not, lo que requiere 10 u.t. y después ha de atravesar la puerta And-2, lo que supone otras 20 u.t más, sumando un total de 30 u.t. Sin embargo, un cambio en y sólo tiene que atravesar la puerta And-2 para llegar a la salida, lo que hace en 20 u.t. De esto podemos concluir lo siguiente.

En un circuito combinacional construido internamente por puertas lógicas conectadas entre sí (u otros dispositivos combinacionales), el tiempo de propagación de cada entrada a cada salida no tiene porqué ser el mismo para cada pareja entrada-salida.

Otra conclusión parcial de este ejemplo sencillo (ya que en la siguiente sección la matizamos) es que el tiempo de propagación de un circuito desde una entrada a una salida es, con lo que sabemos de momento, el tiempo del camino que va desde esa entrada a esa salida. El tiempo del camino es la suma de los tiempos de propagación de las puertas por las que pasa el camino. Así:

$$Tp_{x-w} = Tp(\text{Not}) + Tp(\text{And-2}) = 10 + 20 = 30$$

$$Tp_{y-w} = Tp(\text{And-2}) = 20$$

Pero, ¿qué pasa cuando desde una entrada a una salida de un circuito hay varios caminos posibles? Esto lo vemos a continuación.

Circuitos con más de un camino entre una entrada y una salida

Consideremos el circuito de la figura 3.49. Se han etiquetado (con A1 y A2) las dos puertas And-2 para diferenciarlas. Al lado de cada puerta se indica su tiempo de propagación, que son los de la Digit@Lib. La figura muestra, además, la tabla de verdad del circuito incluyendo las variables internas, que hemos denominado con su expresión lógica en función de las entradas.

Vamos a analizar (sin estudiar todos los casos) el comportamiento temporal de la salida al cambiar la entrada y para tres casos distintos. Para diferenciar cada uno de los tres casos se indica el valor de las variables de entrada (x, y, z) antes del cambio y seguido de una flecha, los valores después del cambio.

Caso 1: (1, 0, 0) -> (1, 1, 0)

En la figura 3.50 se muestra la tabla de verdad del circuito en la que se ha sombreado la fila anterior y posterior al cambio de la señal y que pasa de 0 a 1, mientras la x y la z se mantienen estables a los valores 1 y 0 respectivamente. Los cronogramas de las señales que son susceptibles de cambiar (no hemos dibujado x y z ya que son constantes y dan poca información) se muestran a la derecha. A la

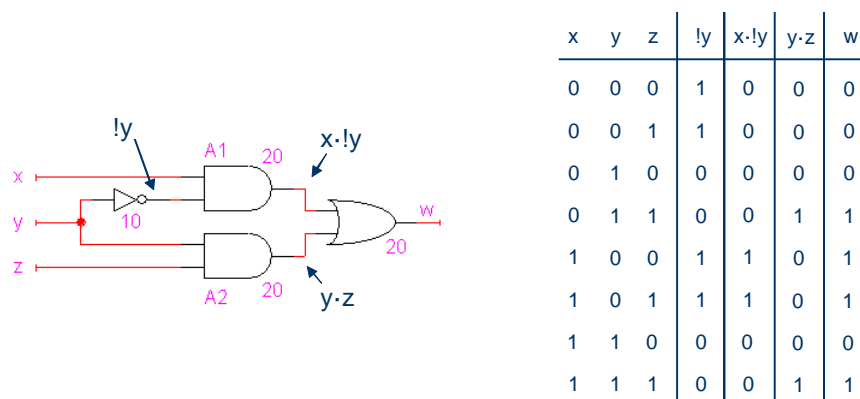


Fig. 3.49 Circuito con dos caminos de la entrada y a la salida y su tabla de verdad.

vista de la forma de la señal y a lo largo del tiempo, se puede construir el cronograma de !y. Después ya podemos dibujar la salida de la puerta A1, $x \cdot !y$, porque ya conocemos x y !y a lo largo del tiempo. La salida de A2, $y \cdot z$, vale 0 durante todo el tiempo ya que también lo vale la entrada z. Finalmente, dibujamos la salida w, la Or de $x \cdot !y$ e $y \cdot z$, que es igual a $x \cdot !y$ (ya que $y \cdot z$ vale 0 todo el rato) desplazada 20 u.t. a la derecha.

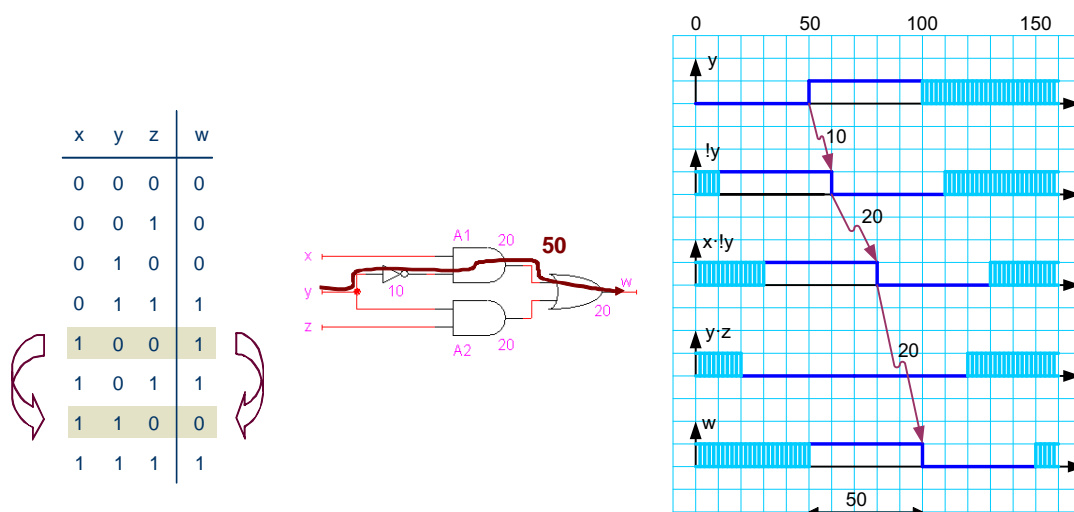


Fig. 3.50 Caso 1: (1,0,0) -> (1,1,0). Tabla de verdad, camino y cronograma.

En el centro de la figura 3.50 hemos dibujado sobre el circuito, a mano alzada, el camino desde la entrada y a la salida w que pasa por las puertas que han propagado el cambio de valor lógico de la entrada a la salida. La suma de los tiempos de propagación de las puertas por las que pasa este camino es el tiempo del camino, que es también el tiempo de propagación desde y a w para el caso de los valores de las entradas considerados. Este tiempo es de 50 u.t. como se ve también en el cronograma:

$$T_{\text{camino1}} = T_p(\text{Not}) + T_p(A1) + T_p(\text{Or-2}) = 10 + 20 + 20 = 50 \text{ u.t.}$$

Caso 2: (0, 1, 1) → (0, 0, 1)

La figura 3.51 es como la figura 3.50 pero para el caso que en el que y pasa de 1 a 0, x se mantiene a 0 y z a 1. Se observa que el tiempo de propagación de y a w es menor que para el caso anterior, ya que el camino de y a w que ha propagado el cambio es más corto en tiempo (no aparece la puerta Not).

$$T_{\text{camino2}} = T_p(A2) + T_p(\text{Or-2}) = 20 + 20 = 40 \text{ u.t.}$$

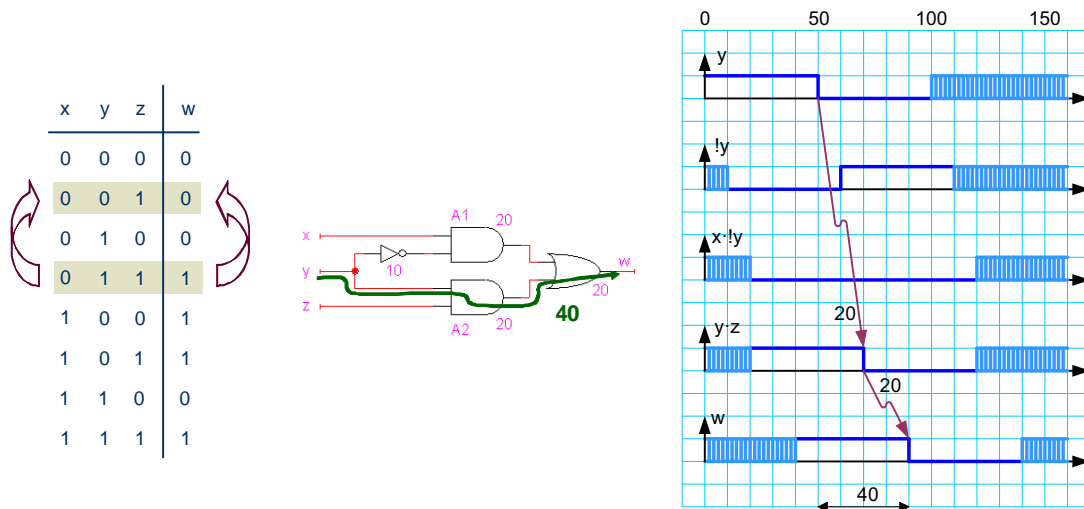


Fig. 3.51 Caso 2: (0,1,1) → (0,0,1). Tabla de verdad, camino y cronograma.

De estos dos casos estudiados ya podemos concluir lo siguiente.

El tiempo de propagar un cambio de una entrada a una salida de un circuito combinacional con varias puertas (u otros dispositivos), puede depender de qué valores lógicos concretos tienen las entradas del circuito que no cambian.

Cuando en un circuito el tiempo de propagar un cambio de una entrada concreta a una salida depende de los valores de las otras entradas, que permanecen constantes, es porque o bien

- hay varios caminos posibles de esa entrada a esa salida con distintos tiempos y unas veces el cambio se propaga por un camino y otras por otro, o bien
- solo hay un camino pero para ciertos valores la salida no debe cambiar, según la tabla de verdad, y no cambia (en este caso el tiempo desde el cambio de la entrada hasta que se ve el valor correcto en la salida es 0) mientras que para otros valores la salida sí que cambia (y tarda un tiempo distinto de 0 en propagarse).

¿Y qué pasa si el cambio de la entrada y se propaga a la vez por los dos caminos que hay de y a w? Veámoslo a continuación.

Caso 3: (1, 1, 1) -> (1, 0, 1)

Este caso es curioso ya que al variar y de 1 a 0, permaneciendo las otras dos entradas constantes al valor 1, la salida no debería cambiar, según la tabla de verdad de la figura 3.52, y sin embargo la salida cambia dos veces, como se ve en el cronograma.

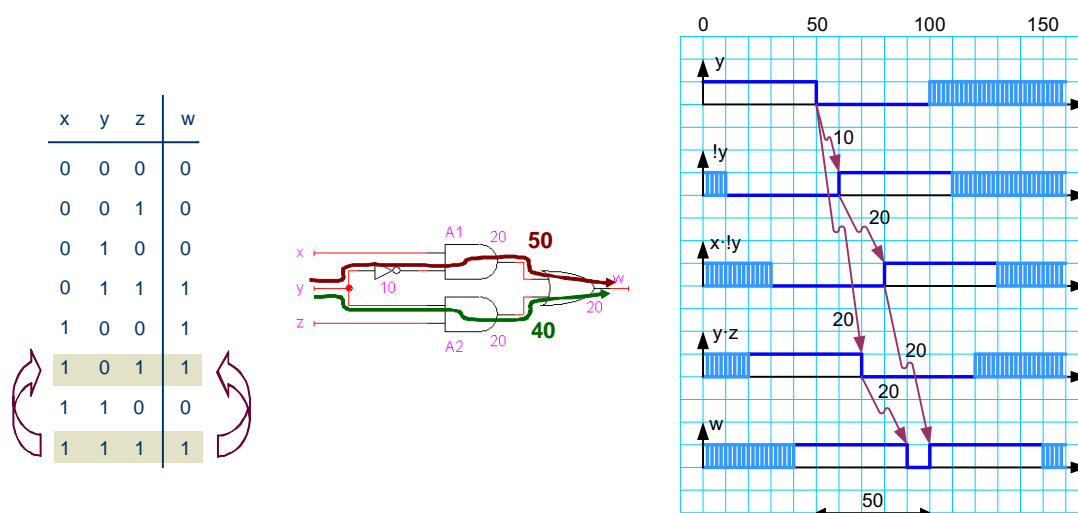


Fig. 3.52 Caso 3: (1,1,1) -> (1,0,1). Tabla de verdad, caminos y cronograma.

Ahora el cambio de y se propaga por los dos caminos que hay de y a w . Como el tiempo de atravesar el camino 2 es menor que el del camino 1, los cambios propagados por el camino 2 provocan un cambio temprano en la salida dejándola en un valor que no es el que indica la tabla de verdad para los nuevos valores de las entradas. Cuando llega el cambio propagado con más lentitud por el camino 1, la salida se estabiliza al valor correcto y permanece estable (hasta nuevas modificaciones de las entradas).

Considerando solamente el modelo lógico de las puertas (puertas sin retardo), en este caso la salida w no hubiera cambiado, se hubiera mantenido a 0. Este **pulso no deseado**, provocado por el tiempo de retardo de las puertas y por la existencia de más de un camino entre una entrada y una salida, se denomina **Glitch**. De esto podemos concluir lo siguiente.

En un circuito combinacional con varias puertas (u otros dispositivos), puede ocurrir que un cambio en una entrada, para ciertos valores concretos del resto de las entradas que no cambian, produzca durante un lapso de tiempo cambios no deseados en la salida, aunque la salida no deba cambiar según indica el comportamiento lógico del circuito (su tabla de verdad). No obstante, pasado este lapso de tiempo la salida siempre se estabiliza al valor lógico correcto para los nuevos valores de las entradas.

La aparición de pulsos no deseados puede ocurrir también cuando cambian a la vez varias entradas del circuito. Veamos esto con un ejemplo.

Ejemplo 12

La figura 3.53 muestra el cronograma en el que puede verse el pulso no deseado a la salida del circuito que calcula $\neg x \cdot y$ cuando las entradas (x , y) pasan de (0, 0) a (1, 1). Según indica la tabla de verdad del circuito la salida no debería cambiar ante este cambio de las entradas, sin embargo se produce un pulso no deseado de 10 u.t.: la diferencia entre el camino de x a w (30 u.t.) y el de y a w (20 u.t.). El tiempo de estabilizarse la salida al valor correcto es el del camino más largo (30 u.t.).

✍ 19

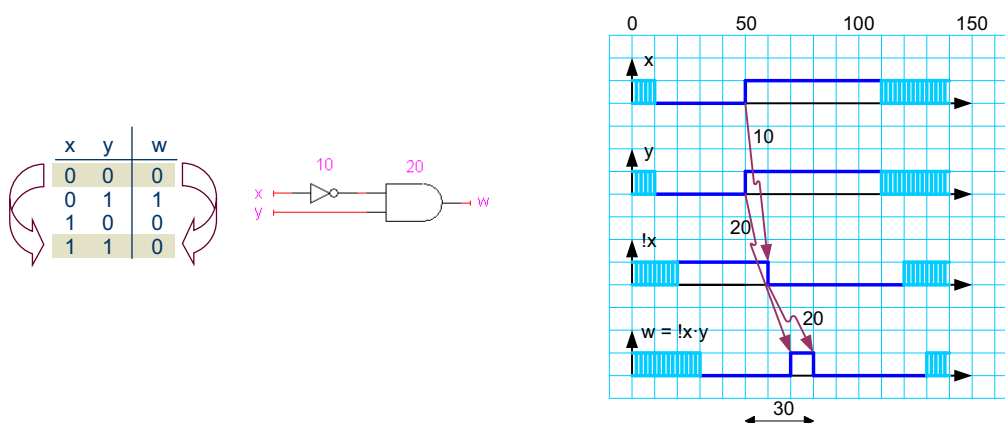


Fig. 3.53 Ejemplo de valor no deseado en la salida al cambiar a la vez dos entradas.

3.4.3 Tiempos de propagación de un circuito combinacional

Hemos visto el comportamiento temporal de circuitos combinacionales formados por la interconexión de puertas lógicas. Comportamientos similares encontramos en circuitos formados por dispositivos/bloques combinacionales conectados entre sí, ya que estos dispositivos están contruidos internamente con puertas lógicas. Pero, ¿por qué es tan importante el comportamiento temporal de los circuitos?

Importancia del tiempo de propagación de los dispositivos combinacionales en un computador

En un computador, los dispositivos combinacionales son los encargados de procesar la información, que se encuentra codificada de alguna forma en los bits de las señales de entrada del dispositivo. Por ejemplo, si el dispositivo es un incrementador de números naturales, con n señales de entrada ($x_{n-1}, x_{n-2}, \dots, x_1, x_0$) y $n+1$ señales de salida ($w_n, w_{n-1}, \dots, w_1, w_0$), el circuito incrementa el valor del número natural que se encuentra codificado en binario en los n bits de las entradas y lo muestra también codificado en binario en los $n+1$ bits de las salidas. El circuito calcula $W_u = X_u + 1$. Este circuito, que se encuentra formando parte de un computador, será usado mientras esté funcionando el computador para realizar el incremento de distintos números. Por ejemplo, en un momento se usa el incrementador para calcular el valor $325+1$, por lo que las señales de entrada codificarán el 325, pero un momento después el incrementador debe ser usado para calcular otro valor, por ejemplo el $367+1$. Las señales de entrada del circuito deben cambiar para que el circuito efectúe el nuevo cálculo, deben pasar de codificar el 325 a codificar el 367. Después de este cambio en la entrada, la salida del circuito, que

codificaba el valor 326, se va a modificar para pasar a codificar el valor 368, que es el resultado del nuevo cálculo.

No obstante, debido a la estructura concreta del circuito formado por puertas conectadas entre sí (y que veremos en el capítulo siguiente), al tiempo de propagación de las puertas y al hecho de que el tiempo de propagación de la señal desde una entrada a una salida es distinto para cada pareja entrada-salida, resulta que antes de que se “vea” la codificación del número 368 en los $n-1$ bits de la salida, se pueden “ver” durante unos instantes una secuencia de distintos números. Se pasa del 326 al 367, luego aparecen el 366, 364, 360, 352 y finalmente se estabilizan las señales de salida al valor 368, que es el resultado correcto del cálculo. Si “miramos” el valor de las salidas antes de que se estabilicen estamos viendo resultados incorrectos del cálculo para el que fue diseñado el circuito. Por esto es muy importante saber el tiempo de propagación de las señales en un circuito combinacional, para saber cuánto tiempo debemos esperar desde que cambian las señales de entrada a un nuevo valor hasta que se estabilizan las señales de salida al valor correcto (hasta que podamos “mirar” los nuevos valores correctos de las salidas).

Este “mirar” sólo en ciertos momentos a lo largo del tiempo es lo que se hace en un computador, que es un circuito secuencial síncrono. Los circuitos secuenciales síncronos los estudiamos en el capítulo 4. De momento decir que en los circuitos secuenciales síncronos hay una señal periódica, denominada señal de reloj, Clk, (*Clock*). La figura 3.54 muestra la señal de reloj de un computador. Cada flanco ascendente de la señal de reloj (cada vez que pasa de 0 a 1) indica el instante de tiempo en que se “miran” (y se almacenan) los valores de las señales de salida de algunos dispositivos combinacionales que forman el circuito (el computador). El periodo de la señal de reloj (el tiempo de cada ciclo de reloj, desde un flanco ascendente al siguiente) se denomina tiempo de ciclo (160 u.t. para el reloj de la figura).

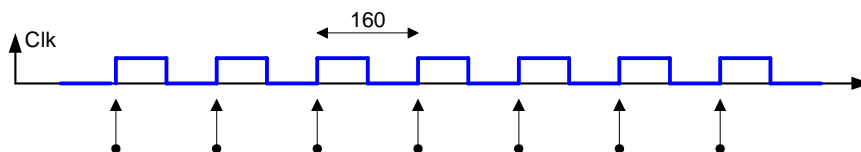


Fig. 3.54 Señal de reloj de un computador. Tiempo de ciclo de 160 u.t.

Para calcular el tiempo de ciclo de la señal de reloj del computador hay que conocer los tiempos de propagación de todos los dispositivos combinacionales que lo forman. Si el tiempo de ciclo es muy grande, si dejamos mucho tiempo entre flanco y flanco de reloj, el procesado de la información que realiza el computador progresará muy lentamente. Pero si el tiempo de ciclo es muy pequeño y resulta que estamos mirando y almacenando el valor de algunas señales antes de que se estabilicen al valor correcto, el computador irá muy rápido pero no hará los cálculos deseados correctamente. Así que es muy importante que el tiempo de ciclo de la señal de reloj sea lo menor posible, pero que sea suficiente para que las señales se estabilicen al valor correcto entre cada ciclo de reloj.

La figura 3.55 muestra el detalle de dos flancos ascendentes de reloj consecutivos y una señal de datos (por ejemplo el bit 8 de la señal de salida del incrementador del que hemos hablado). Se ve como en el primer flanco de reloj la señal w8 vale 0. En este momento empieza un nuevo ciclo y el incrementador comienza a calcular otro resultado. La señal w8 pasa por varios valores no deseados, no correctos,

hasta que se estabiliza al valor correcto, que para este cálculo es $w_8=1$. Cuando ya se ha estabilizado ya puede llegar el flanco ascendente de final de ciclo, para mirar y guardar el resultado de este cálculo. Ahora el incrementador puede iniciar un nuevo cálculo y se repite el proceso.

Usaremos lo que estamos aprendiendo sobre los tiempos de propagación de los circuitos combinacionales para calcular el tiempo de ciclo mínimo de los circuitos secuenciales que veremos en el capítulo 4. Vista la importancia que tienen los tiempos de propagación de los combinacionales, para terminar el capítulo vamos a formalizar un poco el tema.

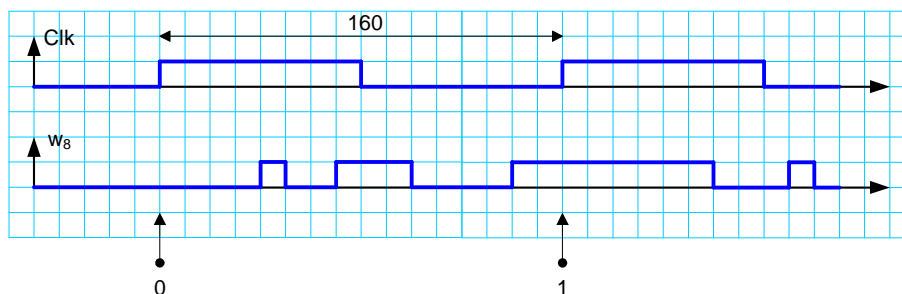


Fig. 3.55 Detalle de la señal de reloj y de una señal de datos sincronizada con ella

Tiempo de propagación y camino crítico de una entrada a una salida de un combinacional

Vamos a formalizar algunas definiciones a partir de lo que hemos visto en las secciones anteriores sobre el comportamiento temporal de los circuitos combinacionales.

Inicialmente definimos el **tiempo de propagación de un circuito combinacional desde una entrada e a una salida s**, $T_{p_{e-s}}$, como el tiempo máximo que tarda en estabilizarse la salida s al valor correcto (al que indica su tabla de verdad) ante un cambio en la entrada e, para los dos posibles cambios en e (de 0 a 1 y de 1 a 0) y para todas las posibles combinaciones de valores de las otras entradas que no cambian.

En el análisis temporal del circuito de la figura 3.49 hemos encontrado que en el caso 1 y el caso 3 el tiempo que tarda en estabilizarse la salida w ante un cambio en la entrada y es de 50 u.t., mientras que en el caso 2 es de 40 u.t. No hemos estudiado todos los casos posibles de cambios. Si lo hiciéramos, lo cual supone un total de 8 casos, veríamos que hay casos, como por ejemplo cuando y pasa de 0 a 1 mientras x y z permanecen a 0, caso $(0, 0, 0) \rightarrow (0, 1, 0)$, en el que la salida no tiene que cambiar según el comportamiento lógico, y no cambia, no aparecen glitches. Para este caso podemos decir que la salida se estabiliza inmediatamente, en 0 u.t. Si hacemos un análisis exhaustivo de los 8 casos veremos que el tiempo máximo es de 50 u.t. Por eso decimos que el tiempo $T_{p_{y-w}}$ es de 50 u.t.

De hecho, para calcular el tiempo de propagación de una entrada e a una salida s no haremos un análisis exhaustivo de los cambios de las entradas, ya que esto sería muy pesado. Primero encontraremos todos los posibles caminos que van de e a s, calcularemos el tiempo de propagación de cada camino y nos quedaremos con el máximo de esos tiempos.

3.13

Un **camino** desde la entrada **e** a la salida **s** de un circuito combinacional es una trayectoria o recorrido válido^a desde la entrada **e** a la salida **s**, pasando por dispositivos y las líneas que los conectan.

Puede haber varios caminos de una entrada **e** a una salida **s**. Para cada camino definimos el **tiempo de propagación del camino** como la suma de los tiempos de propagación de las puertas (o en general dispositivos combinacionales) que se encuentran en el camino.^b

Definimos también el **camino crítico** de la entrada **e** a la salida **s** de un circuito combinacional como el camino de **e** a **s** con mayor tiempo de propagación (el camino más largo en tiempo).

El **tiempo de propagación** desde una entrada **e** a una salida **s**, **T_{p_{e-s}}**, de un circuito combinacional es el tiempo del camino crítico de **e** a **s**.

- a. El recorrido es válido si cuando atraviesa un dispositivo sale de él por una de sus salidas (nunca debe salir por una entrada). Si se sigue esta regla y se trata de un circuito combinacional válido, como la trayectoria empieza en una entrada del circuito, siempre se entrará a un dispositivo por una de sus entradas.
- b. En nuestro modelo, ni las líneas ni las conexiones entre ellas introducen retardo.

Con esta última definición es más fácil calcular **T_{p_{e-s}}** que con la que dimos al principio. Es más fácil calcular todos los caminos de **e** a **s**, sus tiempos de propagación y elegir el mayor de todos ellos (el tiempo del camino crítico de **e** a **s**) que hacer los cronogramas para los casos en que **e** pase tanto de 0 a 1 como de 1 a 0, considerando todos los otros posibles valores que pueden tomar el resto de señales de entrada que no cambian. Por ejemplo, para un circuito con 3 entradas y una salida **s**, habría que considerar 8 casos para cada entrada **e** de la que queramos calcular **T_{p_{e-s}}**.

Para calcular el **T_{p_{y-w}}** del circuito de la figura 3.49 tenemos que encontrar los posibles caminos de **y** a **w**, que son 2, tal como se han dibujado sobre el circuito de la figura 3.52. Podemos denotar cada uno de estos caminos de la entrada **y** a la salida **w** como: (La puerta **Not** y la **Or** no tienen nombre específico, ya que sólo hay una de cada tipo):

- Camino 1: **y** → **Not** → **A1** → **Or** → **w**
- Camino 2: **y** → **A2** → **Or** → **w**

El tiempo de propagación del primer camino, **Tp1**, y del segundo, **Tp2**, como ya se ha visto es:

$$Tp1 = Tp(Not) + Tp(A1) + Tp(Or) = 10 + 20 + 20 = 50 \text{ u.t.}$$

$$Tp2 = Tp(A2) + Tp(Or) = 20 + 20 = 40 \text{ u.t.}$$

El camino crítico de **y** a **w** es el camino 1, ya que es el que tiene mayor tiempo. Así, **T_{p_{y-w}}** = 50 u.t.

Ejemplo 13 $T_{p_{e-s}}$ de un circuito vía el camino crítico de e a s

Consideremos el circuito combinacional con tres entradas y una salida de la figura 3.56 formado por puertas básicas y otros bloques combinacionales. En la parte inferior de la figura se indica el tiempo de propagación de la puerta y, mediante tablas, el tiempo de propagación de cada entrada (columna) a cada salida (fila) de los bloques. Para los bloques con varias entradas y varias salidas se especifica el tiempo de cada entrada a cada salida mediante una tabla de dos dimensiones. Por ejemplo, $T_{p_{b-d}}(\text{CLC1}) = 10$ u.t.

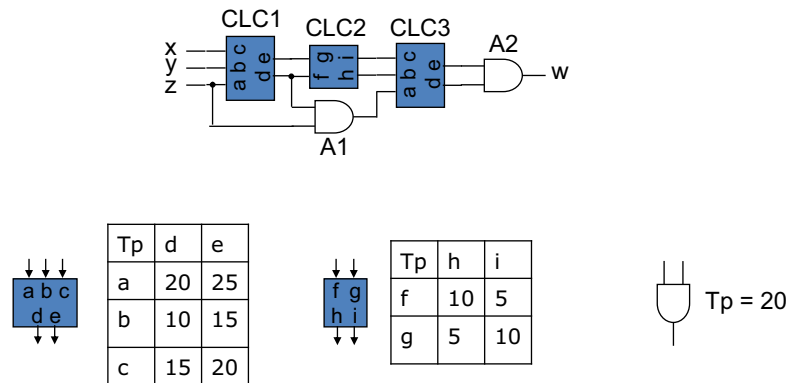
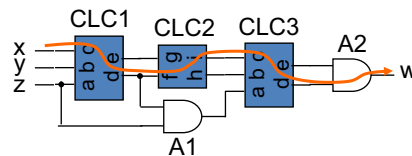


Fig. 3.56 Circuito combinacional formado por puertas y otros dispositivos combinacionales. Tiempos de propagación de los dispositivos involucrados

Vamos a calcular $T_{p_{x-w}}$. Para ello hay que calcular todos los posibles caminos válidos desde x a w y luego calcular los tiempos de propagación de cada camino y quedarnos con el camino de mayor tiempo. Este será el camino crítico de x a w y su tiempo el $T_{p_{x-w}}$ que buscamos.

Un ejemplo de camino y el cálculo de su tiempo se muestra en la figura 3.57. Pero, ¿Cómo encontramos todos los caminos válidos que hay de x a w?

Para encontrar todos los caminos podemos construir un árbol como el de la figura 3.58. Comenzamos por la raíz del árbol: la entrada x. La entrada x está conectada con la entrada c del bloque CLC1. Ahora dibujamos tantas ramas del árbol como salidas tiene el dispositivo CLC1.



Camino: $x \rightarrow c\text{-CLC1-d} \rightarrow f\text{-CLC2-i} \rightarrow c\text{-CLC3-d} \rightarrow A2 \rightarrow w$

$$T = T_{p_{c-d}}(\text{CLC1}) + T_{p_{f-i}}(\text{CLC2}) + T_{p_{c-d}}(\text{CLC3}) + T_p(\text{And-2}) = 15 + 5 + 15 + 20 = 55$$

Fig. 3.57 Un posible camino de x a w dibujado sobre el circuito. Notación del camino y cálculo del tiempo de propagación del camino.

Dibujamos 2 ramas, una para la salida e y otra para la d . Sobre cada una de estas dos ramas anotamos su tiempo de propagación, de c a e (20) y el de c a d (15). Esto nos ayudará a calcular el tiempo de cada camino. Ahora, como de la salida d (que es un nodo del árbol) se puede ir a la entrada f del CLC2 o a una de las entradas de la puerta $A1$, ponemos dos ramas que salen del nodo d . Vamos construyendo de esta forma el árbol y anotando los tiempos de propagación sobre las ramas del árbol. Finalmente todas las ramas terminan en un nodo diferente del árbol, pero todos ellos representan a la salida w , que es el final de cada camino. Vemos que hay 10 caminos diferentes.

Podemos calcular el tiempo de propagación de cada camino viajando de la raíz a cada hijo final del árbol y sumando los tiempos de los arcos. A la derecha de la figura se muestra el tiempo de cada camino. El camino crítico es el camino con tiempo mayor, en este caso 80 u.t. En este caso sólo hay un camino crítico, pero podría haber, por ejemplo, dos caminos distintos con tiempo 80 y los dos serían caminos críticos. En la figura se ha pintado el camino crítico sobre un esquema del circuito. Ahora podemos asegurar que el tiempo de propagación de x a w de este circuito es de 80 u.t.

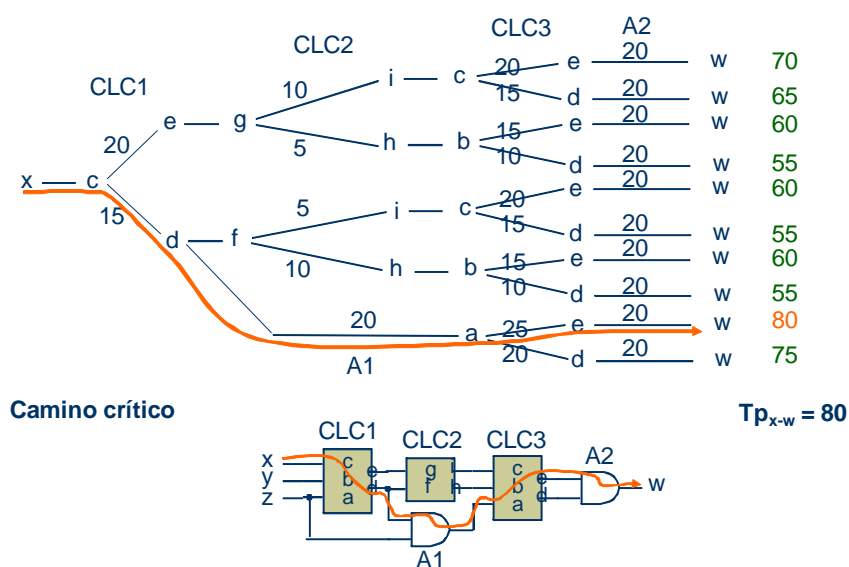


Fig. 3.58 Árbol para encontrar todos los caminos válidos de x a w y sus tiempos de propagación. Camino crítico dibujado sobre el esquema y su tiempo de propagación de 80 u.t.

Para calcular el $T_{p_{y-w}}$ y el $T_{p_{z-w}}$ habría que repetir un proceso equivalente para cada caso. No obstante, para circuitos más sencillos no hace falta ser tan exhaustivo: es cuestión de calcular el tiempo solamente de unos cuantos caminos: los que ya “a ojo” se intuyan como los más largos en tiempo.

✍ 20

Tiempo de propagación de un circuito

En general, en un circuito o dispositivo/bloque combinacional con varias entradas y varias salidas el tiempo de propagación de cada pareja entrada-salida concreta no tiene porqué ser el mismo. Definimos **tiempo de propagación “del circuito”** como el tiempo de propagación de la pareja entrada-salida con

mayor tiempo de propagación de esa entrada a esa salida. De la misma forma podemos hablar de **camino crítico de un circuito combinacional**, para referirnos al camino crítico de la entrada a la salida que tiene mayor tiempo de propagación.

En algunas ocasiones, para simplificar el cálculo del tiempo de propagación de una entrada a una salida de un circuito formado por bloques interconectados y no tener que recordar todos los tiempos de propagación de todas las parejas entrada-salida de cada bloque que forma el circuito, daremos solo el tiempo de propagación de cada bloque. En estos casos estamos suponiendo que los tiempos de propagación de cualquier entrada a cualquier salida de un bloque son el mismo, aunque realmente no sea así. Procediendo de esta forma calcularemos un tiempo de propagación de una entrada a una salida del circuito que será, en general, mayor que el que hubiéramos obtenido si hubiéramos usado los tiempos de propagación de cada entrada a cada salida de cada bloque.

APENDICE 1

(En construcción)

Ejercicios

1. Definiciones básicas (📖 3.1, nivel B1, sección 3.1.1)

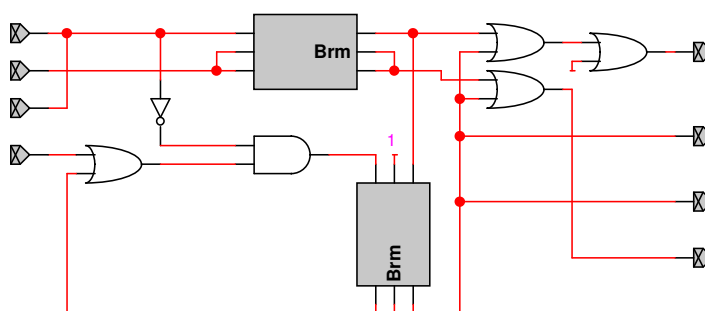
Definid con palabras propias los siguientes conceptos: a) circuito lógico combinacional, b) variable lógica binaria, c) función lógica y d) tabla de verdad.

2. Puertas lógicas (📖 3.3, nivel B1, sección 3.1.3)

Dibujad la puerta lógica adecuada para implementar cada una de las siguientes operaciones lógicas: a) $x = z \cdot k$; b) $p = q + r$; c) $v = !w$. Etiquetad, con el nombre correspondiente, cada entrada y salida de cada puerta. Junto a cada puerta especificad su tabla de verdad indicando correctamente los nombres de las variables en la cabecera de la tabla.

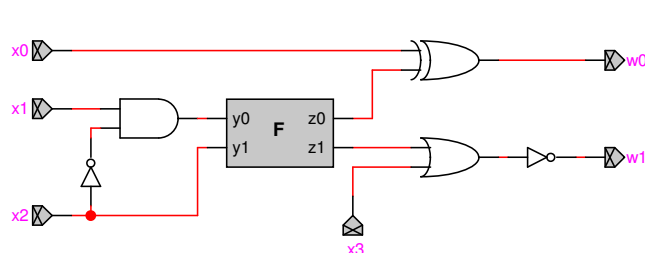
3. Reglas de interconexión para combinacionales (📖 3.4, nivel B2, sección 3.1.4)

Indicad si el siguiente esquema lógico es válido para un circuito combinacional. En caso de que no lo sea, marcad la parte o las partes del esquema que lo invalida e indicad en cada caso cuál de las tres reglas de interconexión para que el circuito sea válido no se cumple. El dispositivo combinacional Brm tiene 3 entradas y 3 salidas. Las 3 salidas están en el lado de la caja cercano a la etiqueta Brm. La forma de flecha de los conectores indica cuáles son las entradas del circuito (a la izquierda del esquema) y cuáles las salidas (a la derecha).



4. Análisis por filas (📖 3.5, nivel B2, sección 3.2.1)

A partir del esquema lógico del siguiente circuito combinacional escribid la fila 2 y la 7 de la tabla de verdad del circuito. La tabla de verdad del dispositivo (bloque) F se da en la figura. Esto es, ¿qué valor toman las salidas w1 y w0 para las dos combinaciones de valores de las entradas siguientes: a) $x_3 = 0$, $x_2 = 0$, $x_1 = 1$ y $x_0 = 0$, y b) $x_3 = 0$, $x_2 = 1$, $x_1 = 1$ y $x_0 = 1$.



TV bloque F

y1	y0	z1	z0
0	0	1	0
0	1	0	0
1	0	0	1
1	1	1	0

5. **Análisis por columnas** (H 3.5, nivel B2, sección 3.2.2)

Escribid la tabla de verdad completa del circuito combinacional del ejercicio 4.

6. **Expresión lógica en suma de minterms** (H 3.6.1, nivel B2, sección 3.3.4)

Escribid las expresiones lógicas en suma de minterms de las dos salidas del circuito del ejercicio 4, cuya tabla de verdad habéis obtenido al realizar el ejercicio 5.

7. **Síntesis en suma de minterms** (H 3.6.2, nivel B2, sección 3.3.4)

Dibujad el esquema lógico que resulta de la síntesis en suma de minterms del circuito del ejercicio 4, cuya expresión en suma de minterms habéis obtenido en el ejercicio 6.

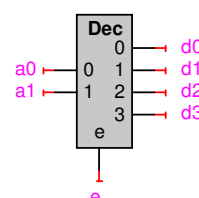
8. **Expresiones lógicas y síntesis en suma de minterms** (H 3.6, nivel B2, sección 3.3.4)

Escribid las expresiones en suma de minterms de las salidas del dispositivo F cuya tabla de verdad se define en el ejercicio 4. Dibujad el esquema lógico que implementa F en suma de minterms.

9. **Decodificador con entrada de enable** (H 3.2, H 3.7.2 y H 3.6.2, nivel B2, sección 3.1.2, sección 3.3.5)

Definimos el nuevo dispositivo combinacional denominado “Decodificador con entrada de enable” de n entradas (a_{n-1}, \dots, a_1, a_0) más la entrada de enable, e , y 2^n salidas ($d_{2^n-1}, \dots, d_1, d_0$), como sigue. Cuando $e = 0$ todas las salidas valen 0 mientras que cuando vale 1 la funcionalidad es como la del decodificador que no tiene entrada de enable. Así, la salida d_i es el producto lógico de e por la función minterm i de las n entradas

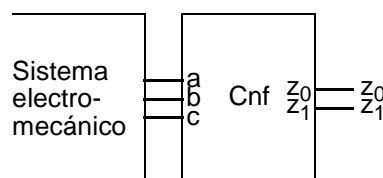
$$d_i = e \cdot m_i(a_{n-1}, \dots, a_1, a_0)$$



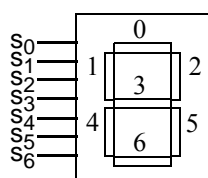
Escribid la tabla de verdad del decodificador con entrada de enable para $n = 2$ cuyo símbolo se muestra en la figura y su implementación en suma de minterms

10. **De la descripción funcional a la tabla de verdad** (H 3.2, nivel B2, sección 3.1.2)

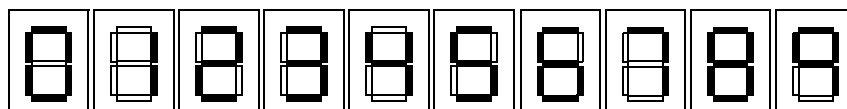
Escribir la tabla de verdad del circuito combinacional denominado **Cnf** (Codificador de nivel de fallo) que tiene 3 entradas (a, b, c) y 2 salidas (z_1, z_0). Cada entrada de Cnf viene de la salida de un sensor que detecta un tipo diferente de fallo en un sistema electro-mecánico (0 indica que el sensor no detecta fallo y 1 que sí). Los tres tipos de fallo tienen el mismo nivel de importancia. Las salidas del circuito $Z = (z_1, z_0)$ codifican en binario un número natural Z_u que indica el número de fallos que hay en el sistema.



11. **De la descripción funcional a la tabla de verdad** (H 3.2, nivel B2, sección 3.1.2)

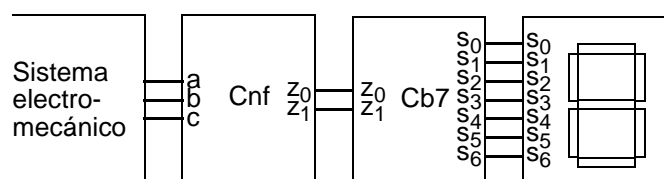


Se dispone de un visualizador de 7 segmentos que es un dispositivo con 7 señales binarias de entrada (s_6, s_5, \dots, s_0). y 7 segmentos luminosos dispuestos formando un 8 con trazos rectos, como muestra la figura. Cuando la señal s_k vale 1 el segmento k se ilumina y cuando vale 0 no. Por ello, para visualizar el número 3, como se muestra en la figura, las entradas deben valer $(s_6, s_5, \dots, s_0) = (1, 1, 0, 1, 1, 0, 1)$. Con este dispositivo se pueden visualizar los dígitos decimales, como se ve en la figura.



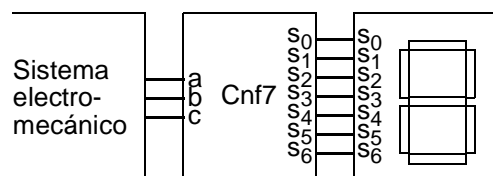
Escribid la tabla de verdad del circuito “Codificador de binario a 7 segmentos”, **Cb7**, que tiene dos entradas $Z=(z_1, z_0)$ y 7 salidas (s_6, s_5, \dots, s_0). El circuito obtiene el código 7 segmentos a partir del número natural Z_u codificado en binario en sus entradas

Conectando las salidas del circuito Cnf del ejercicio 10 con las entradas del Cb7 y las salidas del Cb7 con las entradas del visualizador 7 segmentos como se muestra en la siguiente figura, se puede visualizar el nivel de fallo del sistema electro-mecánico del ejercicio 10.



12. De la descripción funcional a la tabla de verdad (H 3.2, nivel B2, sección 3.1.2)

Escribid la tabla de verdad de un circuito codificador del nivel de fallos en 7 segmentos, **Cnf7**, que tiene tres entradas (a, b, c) y 7 salidas (s_6, s_5, \dots, s_0) que sustituya a los circuitos Cnf y Cb7 del circuito del ejercicio 11 manteniendo la misma funcionalidad.



13. De la descripción funcional a la tabla de verdad (H 3.2, nivel B2, sección 3.1.2)

Escribid la tabla de verdad de cada uno de los siguientes circuitos combinacionales con cuatro entradas (x_1, x_0, y_1, y_0) y dos salidas (w_1, w_0) cuya funcionalidad describimos mediante un pseudo-lenguaje de alto nivel. Para describir la funcionalidad consideramos que los vectores de bits $X = (x_1, x_0)$, $Y = (y_1, y_0)$ y $W = (w_1, w_0)$ representan en binario a los números naturales X_u , Y_u y W_u respectivamente (de esta forma podemos usar X_u , Y_u y W_u en expresiones aritméticas con operadores sobre números naturales). Las salidas de cada circuito se calcula como (en algún caso usamos la operación módulo: siendo $n \bmod k$ el resto de la división entera con resto positivo de n entre k . Por ejemplo: $3 \bmod 4 = 3$, $4 \bmod 4 = 0$, $6 \bmod 4 = 2$, $-2 \bmod 4 = 2$)

- a) $\text{if } (X_u > Y_u) \ W_u = 1 \ \text{else } W_u = 2;$
- b) $W_u = (X_u + Y_u) \% 4;$
- c) $\text{if } (X_u == 0) \ W_u = 0;$
 $\text{else if } (X_u == 1) \ W_u = Y_u;$
 $\text{else if } (X_u == 2) \ W_u = (Y_u + 1) \% 4;$
 $\text{else if } (X_u == 3) \ W_u = (Y_u - 1) \% 4;$

14. Xor, Decodificador, multiplexor (¶ 3.7, nivel B1, sección 3.1.2, sección 3.2.1, sección 3.3.4, sección 3.3.5)

Dibujad el símbolo, representad la tabla de verdad, dad una expresión lógica de cada una de las salidas, dibujad el esquema lógico interno usando puertas **Not**, **And** y **Or** y describid con palabras propias la funcionalidad de cada uno de los siguientes bloques combinacionales:

- puerta Xor de dos entradas (Xor-2),
- multiplexor 2-1 (Mx-2-1).
- decodificador 2-4 (Dec-2-4), y

15. Síntesis con decodificador y puertas Or (¶ 3.8, nivel B2, sección 3.3.5)

Dibujad el esquema lógico con un decodificador y puertas **Or** del dispositivo Cnf definido en el ejercicio 10, que es donde hemos obtenido su tabla de verdad.

16. ROM (¶ 3.9, nivel B1, sección 3.3.6)

Describid con palabras propias la funcionalidad de una memoria ROM. Dibujad el esquema lógico interno que modela de forma sencilla una memoria ROM de 2 entradas y 2 salidas mediante un decodificador, una matriz de interconexión y puertas **Or**.

17. Tamaño de la ROM (¶ 3.10, nivel B1, sección 3.3.6)

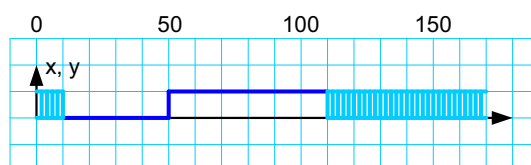
Especificad el tamaño mínimo de la ROM necesaria para implementar el dispositivo Cb7 definido en el ejercicio 11, que es donde hemos obtenido su tabla de verdad. Indicad el tamaño en número de palabras por número de bits por palabra.

18. Síntesis con una ROM (¶ 3.11, nivel B2, sección 3.3.6)

Dibujad el esquema lógico con una ROM del dispositivo Cnf7 definido en el ejercicio 12, que es donde hemos obtenido su tabla de verdad.

19. Cronograma (¶ 3.12, nivel B2, sección 3.4.2)

Dibujad, para el circuito de la figura 3.17, un cronograma con todas las señales internas (a, b,..., e) y la señal de salida w, considerando que las señales de entrada x e y tienen las dos el mismo comportamiento, que se muestra en el cronograma. Suponed $T_p(\text{Not}) = 10$, $T_p(\text{And-2}) = 20$, $T_p(\text{Or-2}) = 20$ u.t



20. Caminos críticos y tiempos de propagación (¶ 3.13, nivel B2, sección 3.4.3)

Indicad el camino crítico (o uno de ellos, si hay varios) y los tiempos de propagación desde cada entrada a cada salida del circuito combinacional del ejercicio 4. Considerad $T_p(\text{Not}) = 10$, $T_p(\text{And-2}) = 20$, $T_p(\text{Or-2}) = 20$, $T_p(\text{Xor-2}) = 50$ u.t. y los tiempos de propagación del bloque F desde cada entrada a cada salida de la tabla.

$T_p(F)_{y-z}$	z_0	z_1
y_0	30	70
y_1	70	80

