Capítulo 7

Sistemas Combinacionales

7.1. Componentes básicos.

Las componentes digitales electrónicas han evolucionado rápidamente. Se han logrado elevados niveles de integración; es decir gran número de compuertas y flip-flops en un solo dispositivo.

Para manejar el nivel de complejidad, de un proyecto que emplee elevado número de componentes, se han desarrollado herramientas de apoyo al diseño, las cuales también han evolucionado con las nuevas metodologías de diseño de software.

Debido a que los problemas computacionales que deben ser resueltos son de gran complejidad, y a menudo de costo exponencial, se han desarrollado heurísticas y nuevos conceptos para describir sistemas digitales.

Históricamente las compuertas y flip-flops se disponían en pastillas con integración en pequeña escala; ver Figura 1.15. Luego se dispuso de componentes integrados en mediana y gran escala; **MSI** y **LSI**, respectivamente (**Medium and Large Scale Integration**). En esta segunda etapa se introdujeron primitivas fijas, que implementan una función específica como: contadores, registros, multiplexores, decodificadores, codificadores, etc. El disponer de primitivas más poderosas que las compuertas lógicas básicas, hizo necesario métodos adecuados de diseño; comenzaron los lenguajes de descripción de hardware (HDL).

Posteriormente los procesos de fabricación permitieron obtener memorias más baratas, y comienzan a emplearse en el diseño digital memorias: PROM, EPROM, EEPROM, FLASH. Al mismo tiempo aparecen los primeros dispositivos programables: PAL, PLA, PLD, los cuales evolucionaron a los sistemas programables actuales: CPLD y FPGA.

Estudiaremos algunos bloques constructivos básicos: Veremos que ellos pueden implementarse en base a compuertas, o bien existen como dispositivos de mediana integración, o son partes o bloques que se pueden agregar a esquemáticos, o son elementos básicos de los dispositivos programables, o son módulos de software de los lenguajes descriptivos de hardware.

Estudiaremos los siguientes bloques básicos: Multiplexores, decodificadores, demultiplexores. A éstos pueden agregarse comparadores y bloques aritméticos.

7.2. Multiplexor. Mux.

Mediante señales de control se selecciona una de las 2^n entradas y se la dirige hacia una salida única. Se tienen n señales de control que, al ser decodificadas internamente, permiten establecer la única vía hacia el canal de salida. El dispositivo puede emplearse para convertir una entrada paralela en una salida en serie.

Un diagrama funcional se muestra en la Figura 7.1:

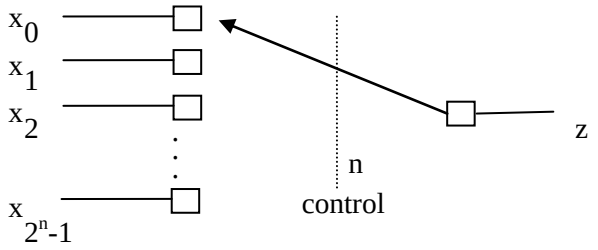


Figura 7.1 Esquema funcional multiplexor.

Suele existir una señal de habilitación (enable) que permite el control del multiplexor. Cuando, por razones que se verán más adelante, se autoriza el ingreso del control en un breve lapso de tiempo, la señal de habilitación toma la forma de un pulso angosto. Este pulso de control, se denomina **STROBE**, en inglés.

La Figura 7.2, muestra un símbolo lógico para un multiplexor de dos vías a una. El símbolo representa la ecuación: $f(c, x_0, x_1) = c' x_0 + c x_1$

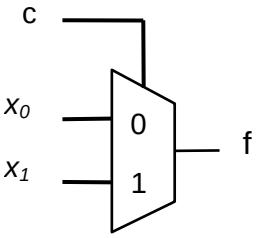


Figura 7.2 Multiplexor dos vías a una.

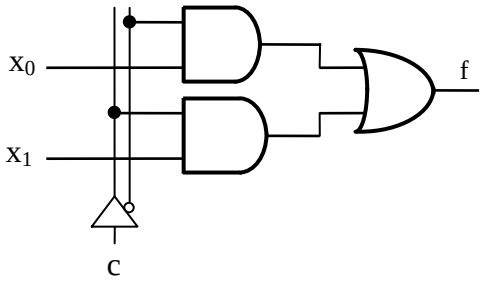


Figura 7.2.a. Multiplexor basado en compuertas.

La Figura 7.3, muestra la realización o síntesis de un mux de 4 vías a una, mediante tres muxs de 2 vías a una. La descomposición aumenta el número de niveles.

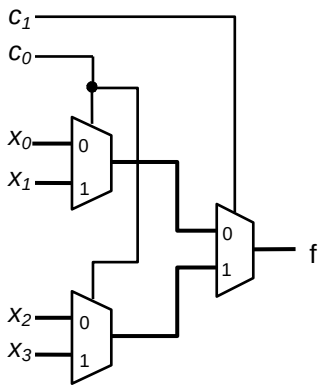


Figura 7.3 Multiplexor 4 a 1.

Para el mux de la Figura 7.3, se tiene la siguiente ecuación de salida:

$$f(c1, c0, x3, x2, x1, x0) = c1c0\ x3 + c1c0'\ x2 + c1'c0\ x1 + c1'c0'\ x0$$

Si se desea disminuir el número de niveles, la Figura 7.4, muestra un MUX (multiplexor) de 4 vías a una, implementado con dos niveles de compuertas. Los puntos donde existe conexión se han destacado con un punto grueso (soldadura).

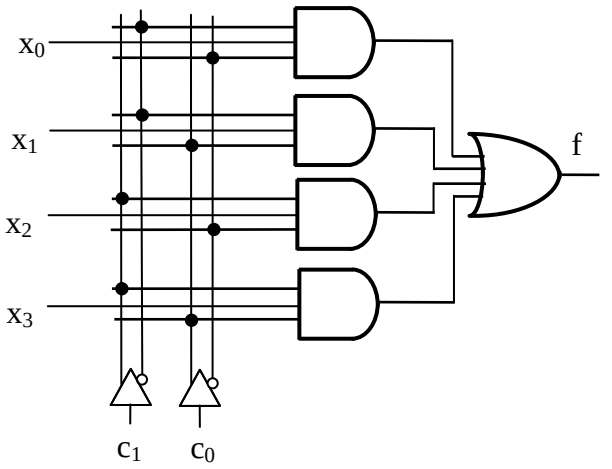


Figura 7.4 Diseño combinacional de multiplexor 4 a 1.

La Figura 7.5, muestra dos símbolos equivalentes para el mux 4 a 1.

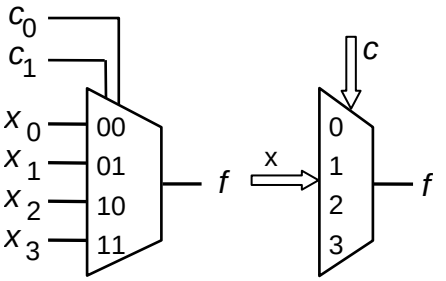


Figura 7.5 Símbolo mux 4 a 1.

La Figura 7.5 derecha, muestra conjuntos de señales con notación de buses o arreglos de alambres.

Mediante el teorema de expansión de Shannon pueden implementarse funciones lógicas mediante multiplexores.

Ejemplo 7.1.

Desarrollar mediante multiplexores la siguiente función:

$$f = x_1x_2 + x_1x_3 + \overline{x}_1\overline{x}_3$$

Aplicando expansión, respecto a x_1, mediante un mux de 2 a 1, se obtiene:

$$f(x_1, x_2, x_3) = x_1 f(1, x_2, x_3) + \overline{x}_1 f(0, x_2, x_3) = x_1(x_2 + x_3) + \overline{x}_1(\overline{x}_3)$$

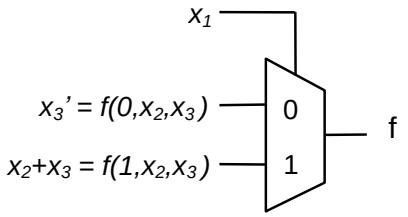


Figura 7.6 Implementación con mux 2 a 1.

Aplicando nuevamente la expansión a las funciones de dos variables, en x_2, x_3, se obtiene:

$$f(x_1, x_2, x_3) = x_1 f(1, x_2, x_3) + \overline{x}_1 f(0, x_2, x_3) =$$
$$= x_1(x_2 f(1,1, x_3) + \overline{x}_2 f(1,0, x_3)) + \overline{x}_1(x_2 f(0,1, x_3) + \overline{x}_2 f(0,0, x_3))$$
$$= x_1 x_2 f(1,1, x_3) + x_1 \overline{x}_2 f(1,0, x_3) + \overline{x}_1 x_2 f(0,1, x_3) + \overline{x}_1 \overline{x}_2 f(0,0, x_3)$$

La Figura 7.7, muestra el desarrollo mediante tres muxs 2 a 1.

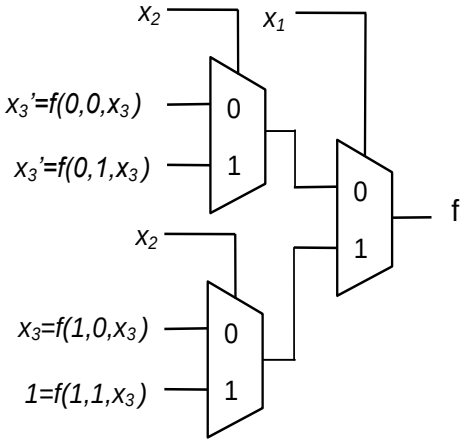


Figura 7.7 Implementación con muxs 2 a 1.

Expandiendo la función, del Ejemplo 7.1, se obtiene:

$$f = x_1 x_2 + x_1 x_3 + \overline{x}_1 \overline{x}_3 = x_1 x_2 + x_1 x_2 x_3 + x_1 \overline{x}_2 x_3 + \overline{x}_1 x_2 \overline{x}_3 + \overline{x}_1 \overline{x}_2 \overline{x}_3$$

Simplificando, se obtiene:

$$f = x_1 x_2 1 + x_1 \overline{x}_2 x_3 + \overline{x}_1 x_2 \overline{x}_3 + \overline{x}_1 \overline{x}_2 \overline{x}_3$$

Comparando coeficientes con la expansión realizada para tres variables, se obtienen las funciones de x_3, que se indican en la Figura 7.7.

Alternativamente, si se desea un diseño de menos niveles, puede emplearse un mux de 4 a 1, como se indica en la Figura 7.8.

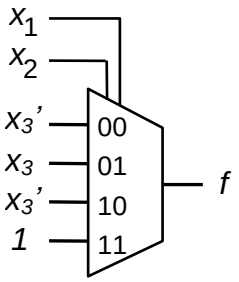


Figura 7.8 Desarrollo basado en mux 4 a 1.

Del diseño anterior puede verse que un mux de 4 a 1, permite implementar cualquier función de tres variables.
Un multiplexor de 8 vías a una, cuyo diagrama se muestra en la Figura 7.9, implementado en dos niveles, está disponible en la familia TTL, se tienen los dispositivos: 74151, 74152.

Un mux de 8 a 1, permite implementar cualquier función de cuatro variables.
Las señales de control: *c2*, *c1* y *c0*, representan a 3 de las 4 variables. La cuarta variable o su complemento se conecta en algunas de las entradas formando los mintérminos. Algunas de las entradas también pueden conectarse a 1 (Vcc) ó 0 (tierra), dependiendo de los mintérminos que deben cubrirse.

f=c2c1c0I7+c2c1c0'I6+c2c1'c0I5+c2c1'c0'I4+c2'c1c0I3+c2'c1c0'I2+c2'c1'c0I1+c2'c1'c0'I0

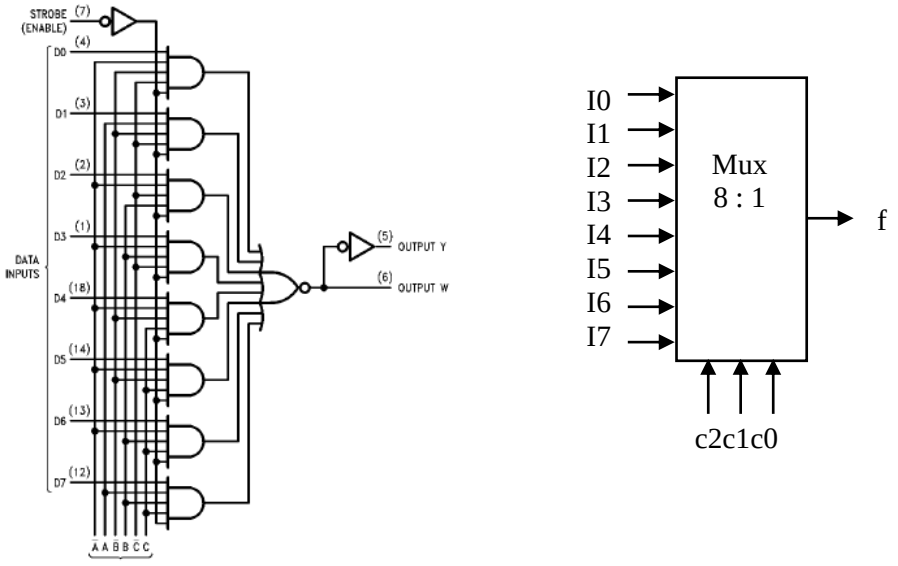


Figura 7.9. Multiplexor de 8 vías a una.

El 74150 implementa un mux de 16 vías a una; este mux permite implementar cualquier función de cinco variables.

Una alternativa electrónica es dotar a estos multiplexores de una salida de tercer estado. En este caso la compuerta de salida además de los valores lógicos 1 y 0, puede quedar en un estado de alta impedancia. En caso de TTL, se disponen conexiones para que ambos transistores del totem-pole de salida queden cortados. Este control adicional, permite conectar las salidas de dos multiplexores a un mismo alambre, ya que sólo uno de los multiplexores impone valores lógicos en la salida; el otro, está en tercer estado.
Esto permite componer un mux de *2n* vías a partir de dos muxs de *n* vías.

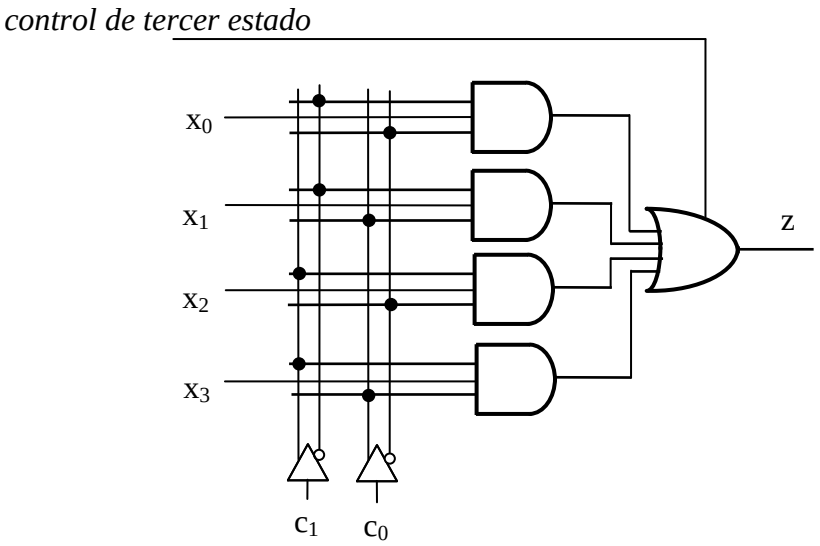


Figura 7.10. Multiplexor con salida de tercer estado.

Ejemplos de estas configuraciones de tercer estado son: 74251, 74253, 74257

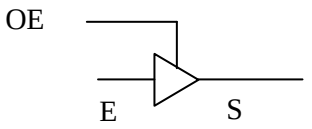
7.3. Tercer Estado. Alta Impedancia.

Además de los valores lógicos 0 y 1, hemos visto el valor superfluo (usualmente *X* o *d*), sin embargo estos valores no aparecen una vez que se ha realizado un diseño ya que éstos se han empleado para minimizar, y han sido reemplazados por unos o ceros según convenga.

Suele encontrarse un tercer estado en el cual puede estar una variable booleana, este valor es de alta impedancia, y se lo denomina corrientemente *Z*. Cuando una salida está en tercer estado, puede considerársela desconectada.

Las compuertas que pueden proveer esta salida tienen una entrada adicional, denominada habilitación de la salida (output enable). Cuando esta señal de control está desactivada la salida está en tercer estado; cuando está activa, la salida puede tomar valores lógicos, de acuerdo a las entradas.

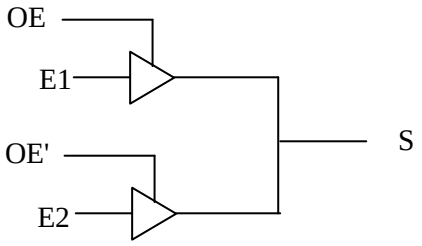
El siguiente diagrama ilustra un buffer de tercer estado (amortiguador entre etapas), junto a su tabla de verdad, en la cual aparecen los estados *X* (superfluo) y *Z* (de tercer estado):



E	OE	S
X	0	Z
0	1	0
1	1	1

Figura 7.11. Buffer de tercer estado.

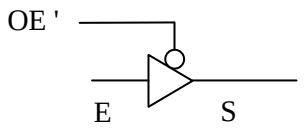
Empleando buffers de tercer estado puede diseñarse un multiplexor, según se ilustra en el diagrama de la Figura 7.12.



OE	S
1	E1
0	E2

Figura 7.12. Multiplexor mediante buffers de tercer estado.

En numerosos circuitos se proveen buffers de tercer estado que activan la salida cuando su nivel es bajo. En estos casos la señal de control, antes de ingresar al buffer, tiene un pequeño círculo, que se interpreta como un inversor. En estos casos de activación del control mediante nivel bajo, el nombre de la señal suele definirse como una señal negada, según se ilustra, en el siguiente diagrama:



E	OE '	S
X	1	Z
0	0	0
1	0	1

Figura 7.13. Buffer con salida activada por señal de lógica negativa.

7.4. Decodificadores.

Permiten seleccionar una de las ***N*** salidas de un circuito que tiene ***n*** entradas.

En decodificadores binarios se cumple la relación: $N = 2^n$.

Ejemplo 7.2

Para un decodificador de dos líneas de entradas, se tendrán cuatro salidas. Pueden denominarse decodificadores de 2 entradas a 4 salidas, anotando 2 : 4; o bien, una salida de las cuatro, lo que se anota: 1 de 4.

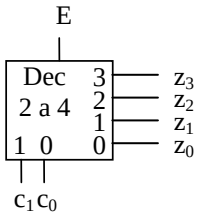
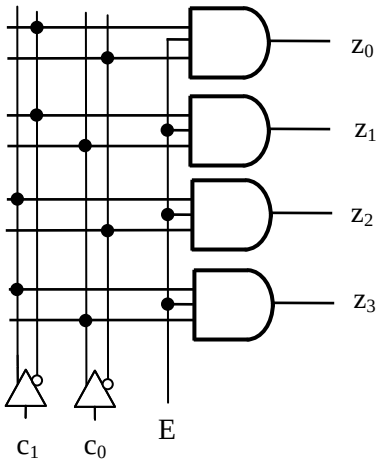


Figura 7.14. Decodificador binario.

Dependiendo de la combinación presente en la entrada, se tendrá una de las salidas en alto; el resto de las salidas serán bajas.
En caso de tener: $c_1 = 0$ y $c_0 = 1$, se tendrá que: $z_1 = 1$, $z_0 = z_2 = z_3 = 0$.

El circuito de la Figura 7.15, implementa un decodificador binario, 2:4, en base a compuertas, y en dos niveles. Se dispone de la señal *E*, que habilita al decodificador.



E	c1	c0	z3	z2	z1	z0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0
0	X	X	0	0	0	0

z0 = c1' c0' E
z1 = c1' c0 E
z2 = c1 c0' E
z3 = c1 c0 E

Figura 7.15. Diseño en base a compuertas

Existen decodificadores que no están basados en el sistema binario; por ejemplo, un decodificador muy usado es el **BCD** a decimal. En él, la entrada corresponde a un dígito **BCD,** en paralelo, es decir 4 entradas; y se tienen 10 salidas, una por cada dígito decimal. Un ejemplo de este decodificador es el 74145.

En toda memoria existe un decodificador del bus de direcciones, cuyas salidas activan sólo una de las palabras de la memoria.

Si en las entradas del decodificador se ingresan las variables, en las salidas se tendrán los mintérminos. Con un decodificador binario de *n* entradas se tendrán 2^n salidas. Con esta componente se pueden implementar todas las funciones de *n* variables. Para esto se conectan las señales asociadas a las variables en las entradas de control; la entrada (gate) se habilita en uno lógico, con esto se tendrán en las salidas los mintérminos de la función. Finalmente para implementar la función se efectúa un or de los mintérminos presentes en la función.

Pueden conectarse decodificadores en estructura de árbol, para formar decodificadores de mayores órdenes. En la Figura 7.15.a, se muestra un decodificador de 4 a 16, la configuración produce los 16 mintérminos asociados a las 4 variables.

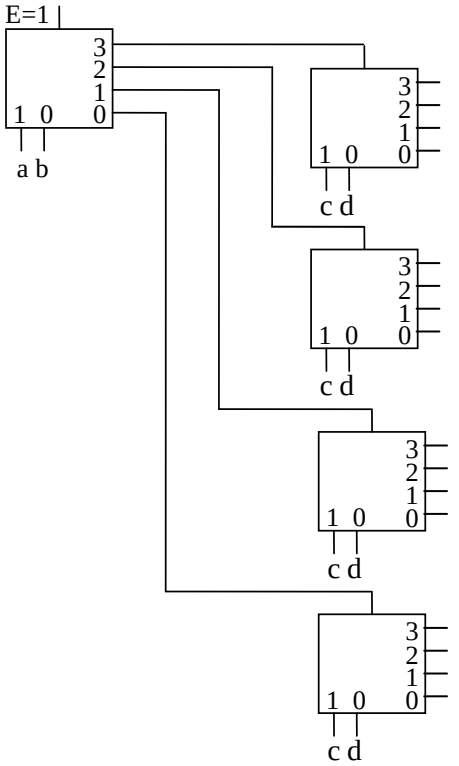


Figura 7.15.a. Decodificador binario 4 a 16

7.5. Demultiplexer, Distribuidor.

Una variante de los circuitos decodificadores es el demultiplexer, que esquemáticamente puede representarse por:

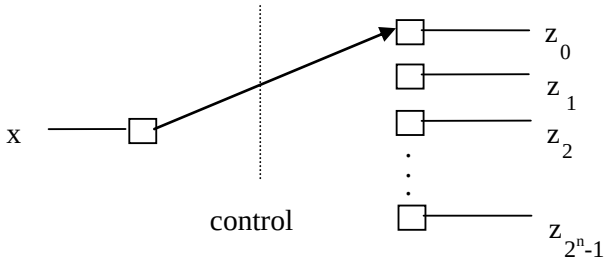


Figura 7.16. Esquema funcional de Demultiplexor

Permite dirigir la información que fluye desde *x*, por una (y sólo una) de las 2^n salidas, de acuerdo a una entrada de control codificada, en *n* líneas. La señal de control selecciona la vía por la que fluirá la única información de entrada.

La Figura 7.17 muestra el diseño de un demultiplexor en base a compuertas:

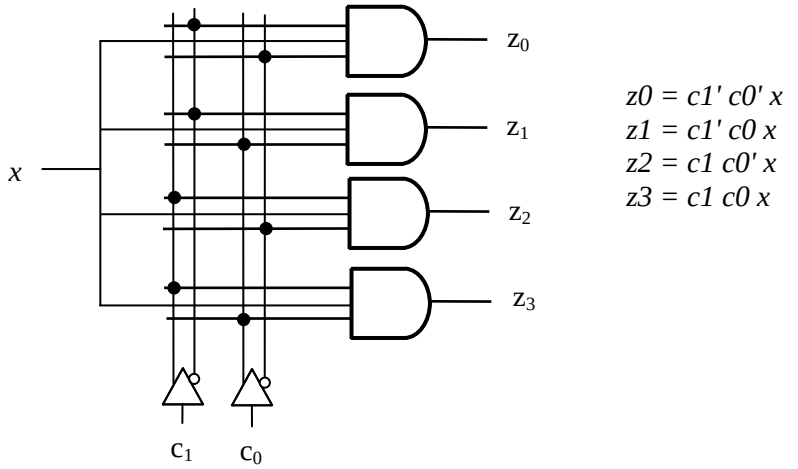


Figura 7.17. Diseño demultiplexor en base a compuertas.

La señal x fluye hacia la salida z3, si las señales de control toman valores: $c_1 = 1$, $c_0 = 1$. El resto de las salidas: z_0, z_1 y z_2 toman valor cero.

En los decodificadores, la señal *x* suele llamarse habilitadora (Gate o Enable).

En el caso de un demultiplexer *x* es la señal de entrada. En forma de bloques, suele emplearse la siguiente notación:

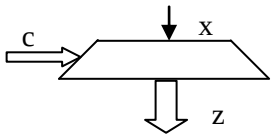


Figura 7.18. Símbolo demultiplexor.

Para un demultiplexor con una entrada, tres señales de control y 8 salidas, se tienen las ecuaciones:

z0 = c2' c1' c0' x
z1 = c2' c1' c0 x
z2 = c2' c1 c0' x
z3 = c2' c1 c0 x
z4 = c2 c1' c0' x
z5 = c2 c1' c0 x
z6 = c2 c1 c0' x
z7 = c2 c1 c0 x

El siguiente esquema ilustra la confección de un switch, en base a un multiplexor y un demultiplexor:

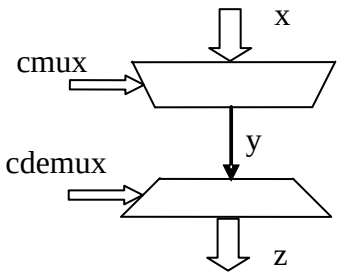


Figura 7.19. Diseño switch empleando multiplexor y demultiplexor.

Pueden usarse para resolver la interconexión entre múltiples fuentes y múltiples destinos.

Una aplicación importante es seleccionar los operandos de entrada a una unidad aritmética, y la posibilidad de llevar el resultado por diferentes rutas. La Figura 7.20 ilustra la posibilidad de escoger dos valores diferentes para cada operando de la unidad aritmética.

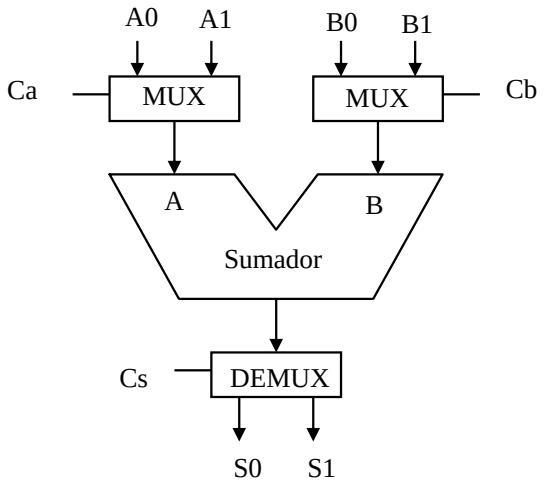


Figura 7.20. Operandos de entrada y salida de unidad aritmética.

En el diseño del camino de datos de un procesador suelen emplearse muxes para movilizar la información. En un caso práctico, *A0, A1, B0, B1, S0* y *S1* son buses formados por 16 ó 32 señales.

7.6. Primitivas programables

En circuitos con funciones cada vez más complejas, los diseños lógicos mediante componentes **SSI,** o primitivas fijas **MSI/LSI** ya no resultaron convenientes.
Comenzaron a emplearse componentes programables, por la disminución del tiempo de diseño y verificación; y además la posibilidad de reprogramación en caso de modificaciones al diseño. Esto estuvo ligado a la aparición de herramientas computacionales de apoyo al diseño.

Los primeros dispositivos programables fueron: **PROM**, memoria programable sólo de lectura (**P**rogramable **R**ead **O**nly **M**emory); **PLA**, arreglo**s** lógicos programables (**P**rogramable **L**ogic **A**rray**s**); **PAL** arreglo de and programable (**P**rogramable **A**rray **L**ogic).
Estos dispositivos poseen, como parte importante, una matriz de diodos.

7.6.1. Matriz de diodos

Básicamente el problema de las memorias es tener varias palabras almacenadas, y que el contenido de éstas esté disponible en los mismos cables de salida, de acuerdo a cuál palabra sea seleccionada o direccionada.

Supongamos que deseamos tener dos palabras de 3 bits cada una. Una de ellas debe ser 101 y la otra, 001. En la Figura 7.21 se tienen dos líneas horizontales o de palabras que pueden estar conectadas a voltajes altos o bajos. Se conectan diodos entre las líneas de palabra y las tres líneas verticales o de contenidos.

Si la línea ***p1*** se coloca en un voltaje alto, y la línea ***p0*** está en cero volts, se tendrá 101 en la salida. Es decir, *C2* y *C0* serán voltajes altos y *C1* será voltaje bajo.

Debe recordarse que un diodo abierto representa una alta impedancia; en el caso anterior, la línea de palabra ***p0*** queda físicamente desconectada de las líneas de salida.

Asumimos que los diodos que conducen dejan un voltaje igual al voltaje de la línea palabra menos 0,7 V, en las líneas de contenido.

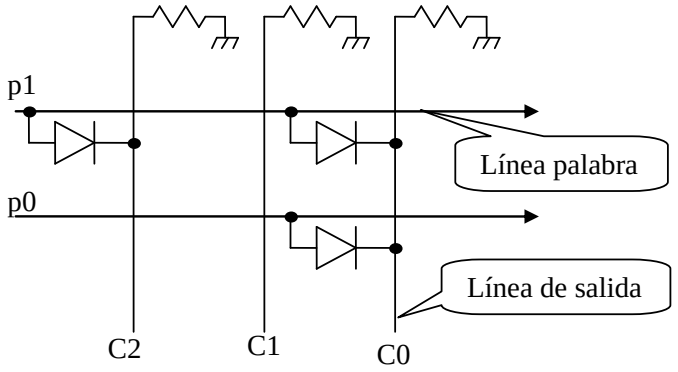


Figura 7.21. Matriz de diodos.

Para construir una memoria, sólo resta activar las líneas de palabras, de acuerdo al código binario de la dirección. Esto puede implementarse simplemente mediante un decodificador.

Se ilustra un esquema de una memoria de 2^m palabras con *n* bits cada una, con salidas de tercer estados controladas por la señal *leer*. En cada casilla de la matriz se tiene un diodo, que conecta la línea de palabra con la columna de salida, en caso de tener almacenado un uno; y no se tiene diodo, en caso de almacenar un cero.

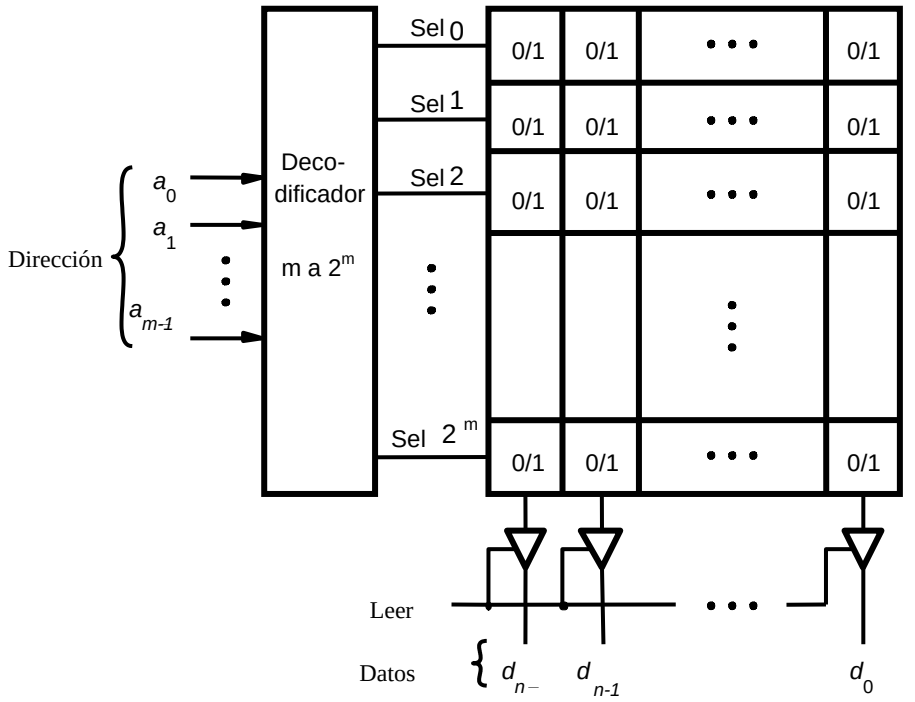


Figura 7.22. Memoria.

7.6.2. ROM

En una **ROM,** los datos quedan permanentemente almacenados. Esto se logra colocando o no diodos, mediante métodos fotográficos, al construir el circuito integrado. Otra posibilidad es construir el circuito con todos los diodos de la matriz; luego se aplica un voltaje adecuado de programación que rompe aquellos diodos asociados a los ceros de la palabra. El voltaje de programación se introduce por las líneas de salida.
La elección de una de estas técnicas de programación de **ROM**, dependen de la cantidad de dispositivos que se deseen programar. Estos dispositivos pueden tener diferentes usos, por ejemplo pueden usarse para: almacenar secuencias de control en un microprograma, generar caracteres, traductor de códigos usuales, etc.

En lo sucesivo, se empleará la siguiente notación para describir una matriz de diodos. La Figura 7.23 muestra una memoria de 4 palabras de 4 bits cada una, se ilustra el decodificador y luego la matriz de diodos.
No se dibujan las resistencias de terminación, y los diodos conectados se simbolizan por puntos gruesos:

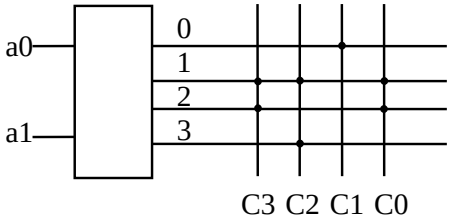


Figura 7.23. Representación simbólica de una matriz de diodos.

En la Figura 7.24 se muestran las direcciones y contenidos de la memoria que se representa simplificadamente en la Figura 7.23.

Dirección		Contenido			
a_1	a_0	c_3	c_2	c_1	c_0
0	0	0	0	1	0
0	1	1	1	0	1
1	0	1	0	0	1
1	1	0	1	0	0

Figura 7.24. Mapa de memoria de matriz de diodos de la figura 7.23.

7.6.3. PROM, EPROM.

El esquema en que se puede programar, pero sólo una vez, a través de la ruptura de algunos diodos del arreglo se clasificó como **ROM**. En las memorias **PROM,** puede colocarse información en la memoria, la P inicial recuerda que el dispositivo es programable.

Se emplean transistores **MOSFET** con la base aislada, en lugar de diodos. Mediante el efecto de avalancha se deposita carga en la base, quedando una baja impedancia o un '1' almacenado.

Para liberar la carga de la compuerta se aplica luz ultravioleta a través de una pequeña ventana que posee la **PROM**, dejándola con puros unos. Suele denominarse **EPROM** a las memorias de este tipo, donde la "**E**" simboliza "**Erasable**" (borrable).

Si se tienen ***n*** líneas para establecer la dirección, y si se tienen ***m*** líneas de contenido o salida, la capacidad total de la **PROM** queda dada por:

$$2^n \cdot m \text{ [bits]}$$

En la práctica, suele especificarse el número de direcciones, expresada en Kilos, multiplicado por el número de bits de salida o largo de palabra. Por ejemplo: 4 K * 8.

De tecnología más moderna son las EEPROM, cuya primera E indica que son electrónicamente borrables. Más recientemente se dispone de tecnología flash, la que tiene la ventaja que sus dispositivos pueden ser reprogramadas estando conectadas dentro de un circuito.

7.6.4. Usos de PROM en circuitos combinacionales.

a) Conversión de códigos.

La programación de la **PROM** está basada en la asignación de un contenido a cada dirección; es decir, se programa por palabras.

Un caso especial de traductor de código es una tabla de búsqueda. En éstas pueden almacenarse valores de funciones trigonométricas, logaritmos, etc. Además, todas las operaciones aritméticas que pueden ser planteadas con tablas, como la suma, resta y multiplicación, también pueden ser implementadas usando **PROM**.

Ejemplo 7.3. Conversión de BCD a exceso 3.

La tabla de conversión:

Dirección				Palabra	Contenido			
0	0	0	0	0	0	0	1	1
0	0	0	1	1	0	1	0	0
0	0	1	0	2	0	1	0	1
0	0	1	1	3	0	1	1	0
0	1	0	0	4	0	1	1	1
0	1	0	1	5	1	0	0	0
0	1	1	0	6	1	0	0	1
0	1	1	1	7	1	0	1	0
1	0	0	0	8	1	0	1	1
1	0	0	1	9	1	1	0	0

Figura 7.25. Mapa de memoria del cambiador de códigos.

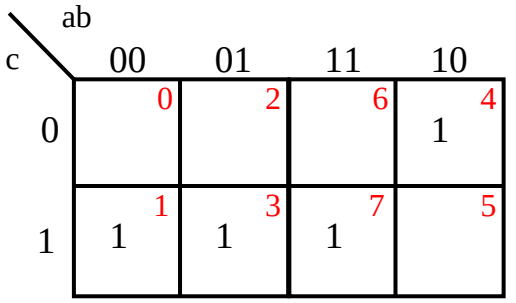
El resto de las combinaciones debe ser llenado con ceros. Nótese que se programa por palabras; es decir, se le asocia un significado al contenido de cada dirección. Se tiene una visión horizontal de la tabla.

b) Generadores de funciones.

En este caso se tiene una visión vertical de la tabla. Se asigna los valores de la tabla de verdad de la función, a una columna de contenido de la **PROM**.
Lo usual es generar varias funciones de las variables con la misma **PROM**.

Ejemplo 7.4.

La siguiente función, puede implementarse en una **PROM** de 8 por 1.



$f(a, b, c)=abc + ab'c' + a'b'c + a'bc$

Figura 7.26. Diseño de funciones mediante PROM.

Nótese que cada mintérmino de la función corresponde a un diodo conectado.

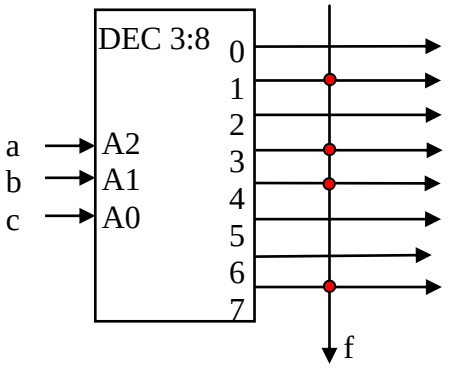


Figura 7.27. Esquema funcional del diseño de una función en base a PROM.

c) Extensión. Estructura de memorias.

Con una adecuada combinación de **EPROM,** es posible extender el largo de la palabra de salida y el número de direcciones.

Ejemplo 7.5

Obtener una **PROM** de 16 bytes por 6 a partir de dos **PROM** de 16 bytes por 3. Esto expande el largo de la palabra de salida.

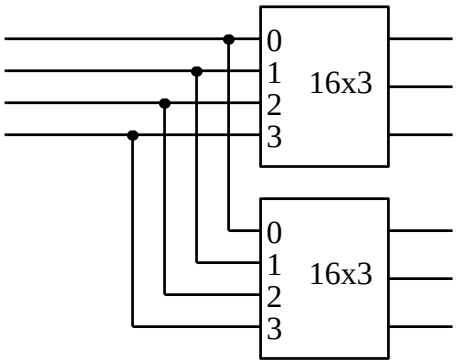


Figura 7.28 Extensión del largo de la palabra.

d) Descripción de archivos con código hexadecimal Intel.

Los archivos con extensión .hex se emplean para describir mapas binarios de contenidos asociados a direcciones.
Su principal uso es describir el contenido de una ROM, mediante un archivo de texto, formado por líneas. Normalmente es generado, en forma automática por una aplicación, por ejemplo un compilador.

El formato del archivo es el siguiente:
Cada línea debe cumplir el siguiente formato:
:10001000FE352FCD454BAEFFE43E5D55AAE435EEEF

:
El primer carácter de la línea es el símbolo dos puntos (colon en inglés).

: **10**
Los siguientes dos caracteres especifican, en hexadecimal, el número de bytes de datos presentes en la línea. (En el ejemplo que se ilustra, el número de bytes es 10H, es decir: 16 en decimal.)

:10 **0010**
Los siguientes cuatro caracteres, especifican la dirección del primer byte de datos de la línea (0010H equivale a la dirección 16 en decimal). Esto implica que el mayor tamaño de la memoria es FFFFH, es decir una memoria de 64 KB. Existen formatos de archivos para especificar memorias mayores a 64KB, entre ellos los registros S de Motorola y los archivos con formato Textronix.

:100010 **00**
Los siguientes dos caracteres indican el tipo de línea o registro. El tipo 00 indica un registro normal de datos; es decir, una línea que no es la última del archivo. El tipo de la última línea del archivo debe ser 01.

:10001000 **FE352FCD454BAEFFE43E5D55AAE435EE**

Los siguientes caracteres representan los datos, en forma de bytes, que se almacenaran en posiciones secuenciales a partir de la dirección inicial de la línea. Los 16 bytes de datos de la línea que se describe en el ejemplo son: FE, 35, 2F, CD, 45, 4B, AE, FF, E4, 3E, 5D, 55, AA, E4, 35, y EE.

:10001000FE352FCD454BAEFFE43E5D55AAE435EE **EF**
Los últimos dos caracteres son una suma de chequeo para la línea. Se denomina byte de paridad. Y se define como el valor que al ser agregado a la suma de los bytes anteriores (los bytes después del dos puntos, hasta el penúltimo) de la línea da como resultado cero. Puede decirse que es el negativo de la suma, descrita antes, expresada en complemento dos.
La suma se realiza en módulo 100H (módulo 256 en decimal).

Ejemplo 7.6

Las siguientes dos líneas son el contenido de un archivo en formato hexadecimal de Intel. El cual se emplea para grabar eprom. El archivo es de texto (puede editarse con notepad); es decir, está formado por caracteres imprimibles del código ASCII, organizado en líneas separadas por 0D, 0A (CR, LF carriage return y line feed)

```
:070100002402FDEC3400FCB9
:00000001FF
```

La última línea tiene tipo 01, que es la última línea del archivo.
Se cargan 7 bytes a partir de la dirección 0100H.

Si se visualiza el archivo de texto (hex) con un editor hexadecimal, se tendrá el siguiente contenido.

```
3A 30 37 30 31 30 30 30 30 32 34 30 32 46 44 45
43 33 34 30 30 46 43 42 39 0D 0A 3A 30 30 30 30
30 30 30 31 46 46
```

El código hexadecimal para el carácter dos puntos es 3A. Para el carácter 0 es 30 hexadecimal, y así sucesivamente. Existe una aplicación Windows llamada mapa de caracteres donde pueden visualizarse los símbolos disponibles en una fuente y su valor en hex.

Todos los cargadores de PROM aceptan directamente el formato Intel.
Existen programas que toman como entrada un archivo hexadecimal y lo transforman a una imagen binaria.

Si se transforma el archivo anterior, en formato Intel hex a binario, la información del archivo binario, vista con un editor hexadecimal se vería como sigue:

```
00000000 0000 0000 0000 0000 0000 0000 0000 0000 ................
00000010 0000 0000 0000 0000 0000 0000 0000 0000 ................
00000020 0000 0000 0000 0000 0000 0000 0000 0000 ................
00000030 0000 0000 0000 0000 0000 0000 0000 0000 ................
00000040 0000 0000 0000 0000 0000 0000 0000 0000 ................
00000050 0000 0000 0000 0000 0000 0000 0000 0000 ................
```

```
00000060 0000 0000 0000 0000 0000 0000 0000 0000 ................
00000070 0000 0000 0000 0000 0000 0000 0000 0000 ................
00000080 0000 0000 0000 0000 0000 0000 0000 0000 ................
00000090 0000 0000 0000 0000 0000 0000 0000 0000 ................
000000A0 0000 0000 0000 0000 0000 0000 0000 0000 ................
000000B0 0000 0000 0000 0000 0000 0000 0000 0000 ................
000000C0 0000 0000 0000 0000 0000 0000 0000 0000 ................
000000D0 0000 0000 0000 0000 0000 0000 0000 0000 ................
000000E0 0000 0000 0000 0000 0000 0000 0000 0000 ................
000000F0 0000 0000 0000 0000 0000 0000 0000 0000 ................
00000100 2402 FDEC 3400 FC                      $...4..
```

La primera columna indica la dirección inicial, en hexadecimal. Las siguientes columnas (exceptuando la última) contienen los datos asociados a las direcciones. Se tienen 16 bytes de datos por línea; por esta razón las direcciones de la primera columna terminan en 0. La última columna es la zona ASCII, donde se colocan los 16 símbolos del código ASCII asociados a los bytes.

Note que en la zona ASCII, los caracteres que no son gráficos o imprimibles (del 00 a 31 decimal, y los mayores que 7FH) se visualizan con puntos; y que también se rellenan con ceros las zonas donde no se han especificado datos; este es el caso desde la dirección 00000000 hasta 000000FF. Y sólo se cargan en este mapa de memoria los siete valores especificados, a partir de la dirección 00000100.
No puede verse un archivo binario, con un editor de texto.

Note que la fuente para ver alineadas las columnas debe ser de espacio no proporcional, como `courier` o similar.

Existe una aplicación, normalmente denominada dump, que pasa un archivo binario a un archivo de texto, con la información similar a la desplegada por un editor binario (o hexadecimal).

7.6.5. PLA. Arreglos Lógicos Programables.

En una **EPROM**, todos los mintérminos son generados por un decodificador fijo, luego los mintérminos requeridos para producir la función, son combinados mediante otro arreglo programable con el papel de un **OR**.

Sin embargo, para ciertos diseños lógicos, sólo se usa una pequeña fracción de los mintérminos generados por el decodificador.

La estructura arreglos lógicos programables (**PLA**) contiene un decodificador programable (arreglo lógico **AND**), y un arreglo lógico **OR** programable. De esta forma pueden implementarse funciones a partir de términos de mayor nivel que los mintérminos; es decir, a partir de los implicantes primos.

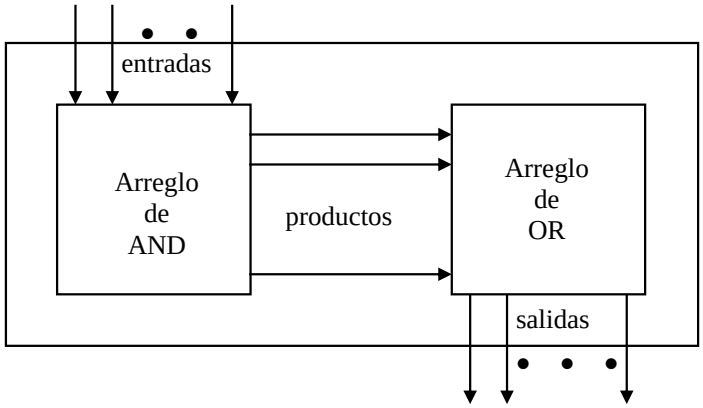


Figura 7.29. PLA Arreglos lógicos programables.

Ejemplo 7.7.

Implementar, mediante **PLA** , las siguientes funciones:

$$f_0 = \bar{A}BCD + \bar{D}E + D\bar{E}$$
$$f_1 = A\bar{B} + \bar{B}C\bar{D}E + D\bar{E}$$
$$f_2 = \bar{A}BCD + \bar{B}C\bar{D}E$$

Se reconocen 5 productos diferentes, que serán generados por el decodificador programable:

$$P_0 = \bar{A}BCD$$
$$P_1 = \bar{D}E$$
$$P_2 = D\bar{E}$$
$$P_3 = A\bar{B}$$
$$P_4 = \bar{B}C\bar{D}E$$

La Figura 7.30 ilustra los detalles.

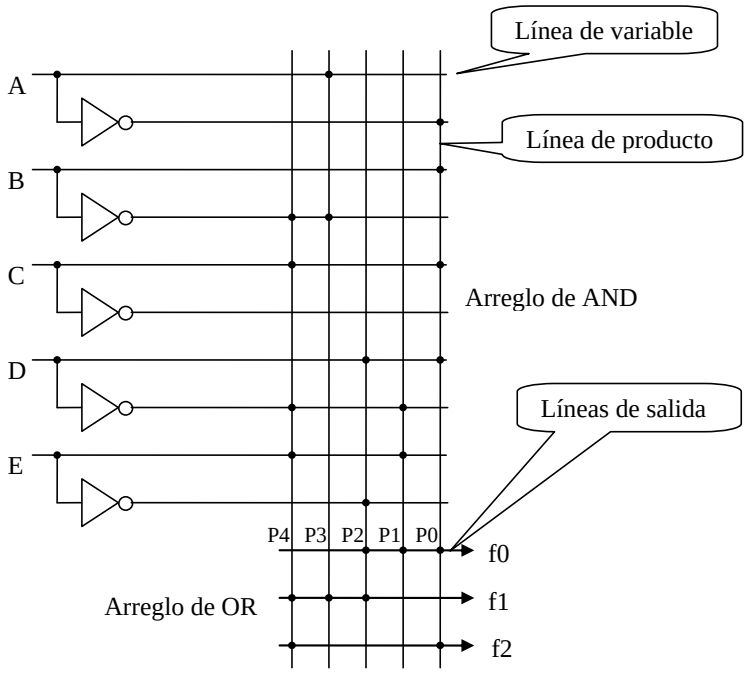


Figura 7.30. Esquema del diseño empleando PLA.

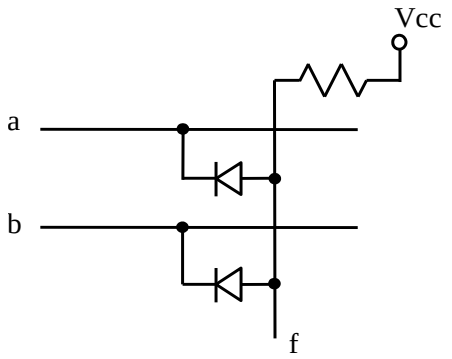
Las variables complementadas se generan internamente. El tamaño del arreglo de **AND** es proporcional al producto de las variables por el número de productos lógicos a generar.

Además, el tamaño del arreglo **OR** es proporcional al producto del número de funciones implementadas por el número de productos. Reduciendo los parámetros anteriores a través de los métodos de minimización, se logrará un diseño con una **PLA** de menor costo.

Como se verá más adelante, las **ROM** y **PLA** se emplean también en diseños de máquinas secuenciales.

Detalle arreglo de AND.

En la Figura 7.31, si ***a*** o ***b*** o ambas tienen voltaje cero, en ***f*** se tendrá el voltaje de conducción del (o de los) diodos. Este voltaje es cercano a los 0,7 [V] y corresponde a un cero lógico. Si ambos diodos están abiertos, con ***a*** y ***b*** altos, se tendrá un voltaje alto, proporcionando por la fuente y la resistencia, en la salida ***f***.



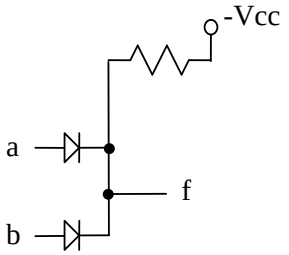
a	b	f
0	0	0
0	1	0
1	0	0
1	1	1

f = a b

Figura 7.31 Arreglo de AND.

La combinación fuente-resistencia se denomina **Pull-up**, y fija el voltaje alto en la salida.

Detalle arreglo OR.



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

f = a + b

Figura 7.32 Arreglo de OR.

En la Figura 7.32, basta que una (o ambas) de las entradas estén altas, para que en la salida ***f*** se tenga un voltaje alto, (V_{in} - 0,7), al cual se asocia un 1 lógico.

Si ambas entradas están bajas, la salida toma un voltaje de -0,7; al cual se asocia el 0 lógico.

Los circuitos que se ilustran en las Figuras 7.31 y 7.32, no son los únicos posibles. Las explicaciones simplificadas del funcionamiento de los arreglos, en base a diodos, son con fines ilustrativos de conceptos solamente.

La tecnología implementa los fusibles programables (diodos) empleando MOSFET.
Ver Apéndice 4.

Diagrama simplificado PLA.

El siguiente diagrama ilustra una PLA, en base a compuertas. En la práctica todas las compuertas son NAND (o NOR). Se muestran todas las conexiones antes de programar el dispositivo.

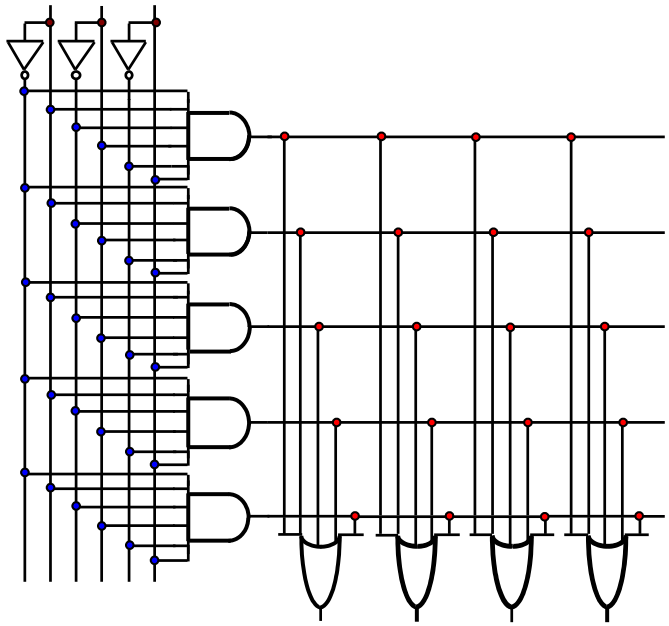


Figura 7.33 PLA en esquema de compuertas lógicas.

Luego las conexiones no deseadas se abren.

Existen dos modalidades, una denominada fusible (fuse) en la que las vías están normalmente conectadas, y se funden o abren las conexiones que no se desean.

Otra modalidad, llamada Antifusible (Anti-fuse), en la cual las conexiones están normalmente desconectadas, y la programación crea las conexiones.

Si se abren algunas conexiones, como se muestra en el diagrama de la Figura 7.34, pueden implementarse las funciones:

f1 = abc + a'bc' + b'c
f2 = a'bc' + a b + a
f3 = a + b'c
f4 = a'bc'

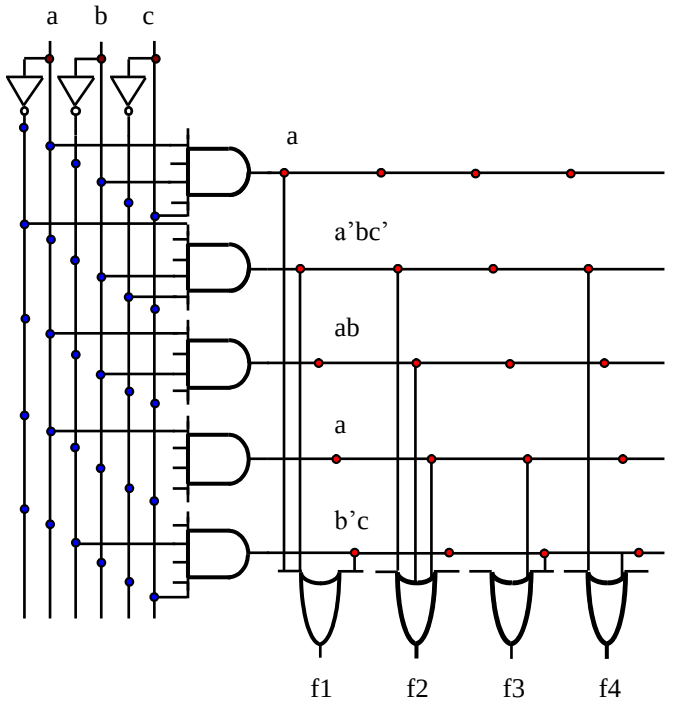


Figura 7.34 PLA programada. Se muestran abiertas las conexiones.

Cuando la densidad aumenta, se suele emplear la siguiente notación para las matrices de diodos: Se dibuja una sola línea de entrada a las compuertas, y se dibuja una marca cuando existe conexión entre las líneas perpendiculares. El siguiente diagrama ilustra una PLA de cuatro entradas y cuatro salidas, cuando todas las conexiones aún no han sido programadas.

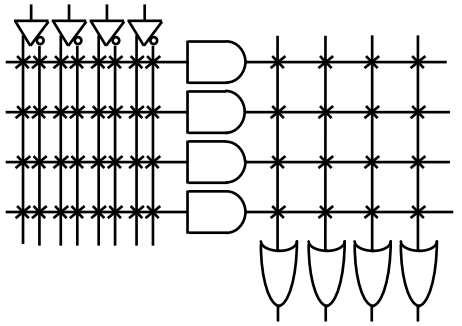


Figura 7.35 Esquema de compuertas simplificado de una PLA.

Usando este convenio, la programación de *f1= a'b + b'a* y *f2 = a'b'c'd' + abcd* puede representarse según:

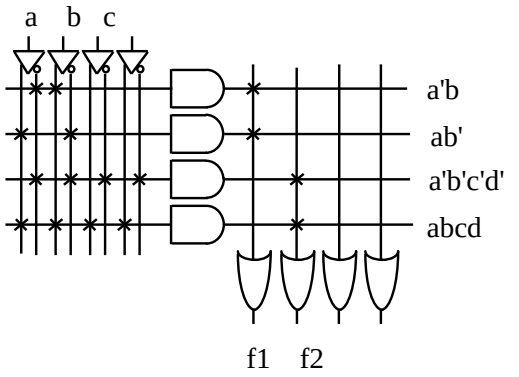


Figura 7.36 Esquema simplificado PLA, con conexiones abiertas.

Sin embargo cuando el número de fusibles es elevado, se han estandarizado formatos de archivos para describir las conexiones. Ver A4.9. Formato Jedec, en Apéndice 4.

7.6.6. PAL arreglo lógico (de and) programable (Programable Array Logic).

El arreglo de or es fijo. El número de productos lógicos que entran a un OR puede variar, usualmente de 2, 4, 8 y 16.

En la Figura 7.37, se ilustra un solo or con tres líneas de producto, y tres entradas. Note que en las entradas al arreglo de and se disponen de la señal y su complemento.

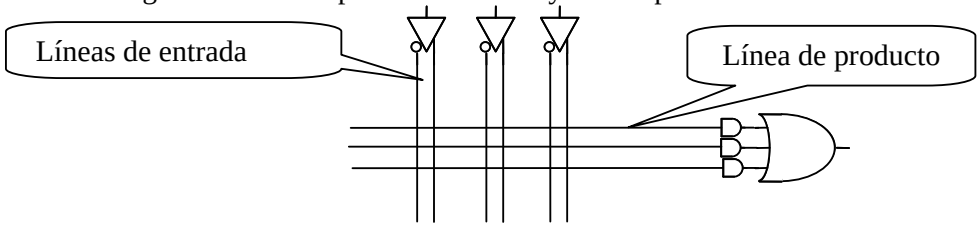


Figura 7.37. Esquema PAL

En cada línea de producto se puede programar un implicante de la función. En cada intersección de una línea de entrada con una línea de producto se tiene un fusible programable (diodo).

Existen PAL con lógica activa alta en la salida (Ejemplo P16H8) y con lógica activa baja en la salida (P16L8).

También suele proveerse una realimentación de la salida en las líneas de producto, esto permite programar realimentaciones, y puede emplearse para diseños asincrónicos.

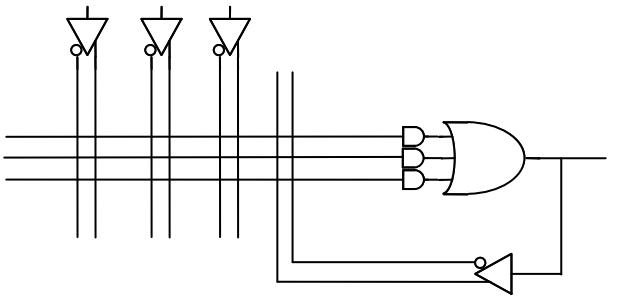


Figura 7.38. Esquema PAL con realimentaciones.

Si se está diseñando un conjunto de funciones, y existen productos que puedan ser compartidos las PLA son más eficientes que las PAL. Pero debido a que las conexiones basadas en fusibles tienen mayor resistencia que las conexiones físicas, las PLA son más lentas que las PAL.

7.6.7. PLD (Programable Logic Device).

El siguiente paso en la evolución de las PALs fue agregar un bloque reconfigurable (macrocelda) mediante fusibles en la salida del OR. La macrocelda contiene un flip-flop tipo D, que puede capturar la salida del OR; un buffer de tercer estado; y lógica para proveer realimentación a las líneas de producto de la salida del OR o del flip-flop. Entre los PLD, uno de los más conocidos es la GAL22V10 (**G**eneric Array **L**ogic). Estos dispositivos reemplazaron a las diferentes formas de las PAL.

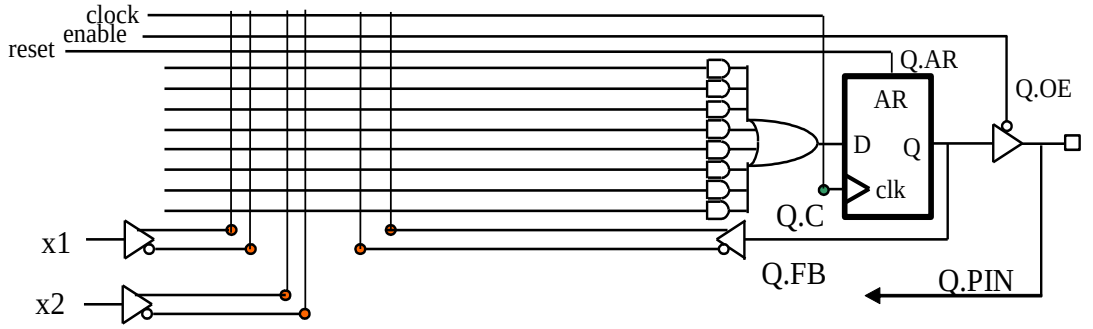


Figura 7.39. PLD con macrocelda.

En la Figura 7.39, la macrocelda muestra el flip-flop D, permanentemente conectado a la salida del or; sin embargo puede programarse que la macrocelda sea estrictamente combinacional, conectado la salida del or directamente a la salida. Se muestran dos entradas: x1 y x2.
Las líneas de productos de cada OR son fijas, pero se diseñan algunos OR con más líneas de productos que otros.

7.6.8. Comparaciones entre dispositivos programables.

Una ventaja de emplear ROM es que no se requiere minimizar las funciones de salida, y conviene emplearlas cuando se requieren muchas combinaciones de las entradas (por ejemplo, en cambiadores de código); las dificultades de su empleo es que por cada entrada adicional se duplica el tamaño de la ROM, y no puede emplearse minimización debida a condiciones superfluas. En una ROM puede implementarse cualquier función, y su costo es relativamente bajo, debido a los altos niveles de producción; pero su velocidad es baja relativa a las otras estructuras programables.

Los arreglos lógicos programables (PLA) comparten los productos entre varias funciones y su empleo es ventajoso si las funciones no tienen presentes mintérminos aislados. La mayor ventaja es que existen aplicaciones que permiten minimizar diseños de múltiples funciones. Es de mayor costo y menor velocidad que una PAL.

La principal desventaja del arreglo lógico programable (PAL) es que existen restricciones de fan-in en las entradas de los or fijos, cuestión que se ha superado en los diseños de las arquitecturas de los CPLD; su ventaja es la mayor rapidez relativa a PLA y ROM. En el diseño de funciones complejas, puede emplearse una salida como entrada, a la misma PAL, lo cual aumenta el número de niveles.

7.6.9. CPLD (Complex Programable Logic Device).

Se denomina CPLD al siguiente nivel de la tecnología. Tienen mayor nivel de integración (cerca de 6500 compuertas y 288 registros, lo cual es un elevado incremento comparado con los 10 registros de la GAL22V10). Se caracterizan por tener programable el número de líneas de producto que pueden sumarse en un or, y por la incorporación de un bloque de entrada-salida independiente de la macrocelda.

Suelen tener memoria flash, lo que permite reprogramarlos ya conectados al resto del circuito, y además conservan su programación al desconectarles la polarización.

El nivel de complejidad de las componentes actuales requiere de programas de apoyo al diseño. Además se han incorporado nuevos lenguajes para la descripción del hardware.

7.6.10. FPGA

Se denominan FPGA (Field Programmable Gate Array) a dispositivos basadas en arreglos bidimensionales de bloques lógicos y flip-flops, con programabilidad de las interconexiones y de las funciones lógicas y del control. Se las debe reconfigurar al aplicarles la polarización.

A medida que han ido aumentando el número de compuertas y flip-flops dentro del chip, también ha ido variando su estructura interna, actualmente algunas de ellas tienen memoria y procesadores incorporados.

Algunos fabricantes proporcionan el código HDL de microprocesadores de 16 y 32 bits, de unidades de multiplicación y acumulación para implementar filtros digitales, etc.

Algunos diseños comerciales de FPGA consideran como bloques lógicos: pares de transistores, compuertas nand de dos entradas, compuertas and con compuertas xor, multiplexores y tablas de búsqueda.

Si los bloques son muy simples, lo más probable es que se utilicen completamente, pero requieren un elevado número de alambres y switches programables; la programación de las interconexiones puede requerir mucho espacio en el área del chip o tener gran retardo.

Celdas basadas en multiplexores.

La celda de la Figura 7.40 tiene 8 entradas y una salida.

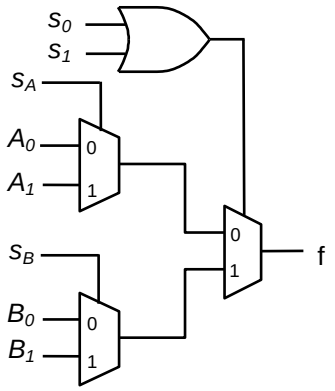


Figura 7.40. Celda basada en multiplexores.

La ecuación de la celda es:

$$f = \overline{(s_0 + s_1)}(\overline{s}_A A_0 + s_A A_1) + (s_0 + s_1)(\overline{s}_B B_0 + s_B B_1)$$

Esta celda puede implementar todas las funciones de dos variables, todas las de tres variables con al menos una entrada sin complementar, muchas de cuatro variables y hasta algunas de ocho variables.

Celdas basadas en tablas de búsqueda.

Pueden emplearse Tablas de Búsqueda para implementar funciones booleanas. Se denominan LUT (look-up table) en inglés. Están basadas en un registro cuya salida alimenta a un multiplexor.

La Figura 7.41 muestra una tabla de búsqueda de 4 variables de entrada, la cual permite implementar cualquier función de cuatro variables.

El principio de funcionamiento es el siguiente: primero se graba la tabla, para esto se direcciona un latch, el decodificador habilita para escritura sólo uno de los 16 latchs del registro, que tienen común la entrada D, y se almacena el 1 ó 0 que le corresponde en la tabla de verdad de la función; y así sucesivamente hasta grabar toda la tabla.

Una vez configurada la tabla de búsqueda, se procede a la lectura de ella; para ello basta direccionar el multiplexor con la combinación de las variables de entrada y en la salida se tendrá el bit correspondiente, que está almacenado en uno de los latchs del registro.

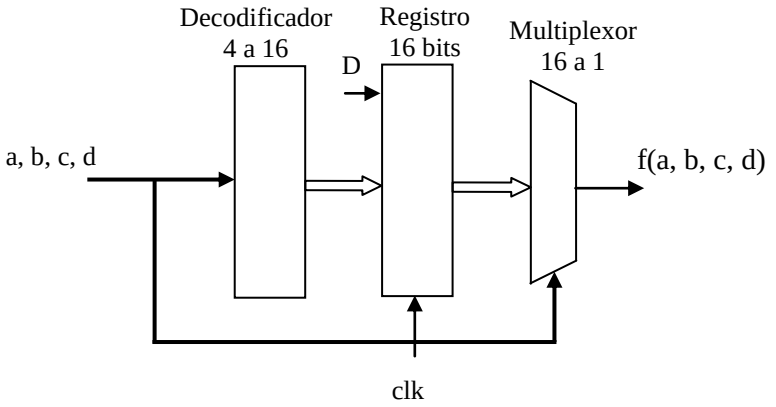


Figura 7.41. Celda basada en Tabla de búsqueda.

Un módulo básico conecta dos o tres tablas de búsqueda con multiplexores y flip-flops.

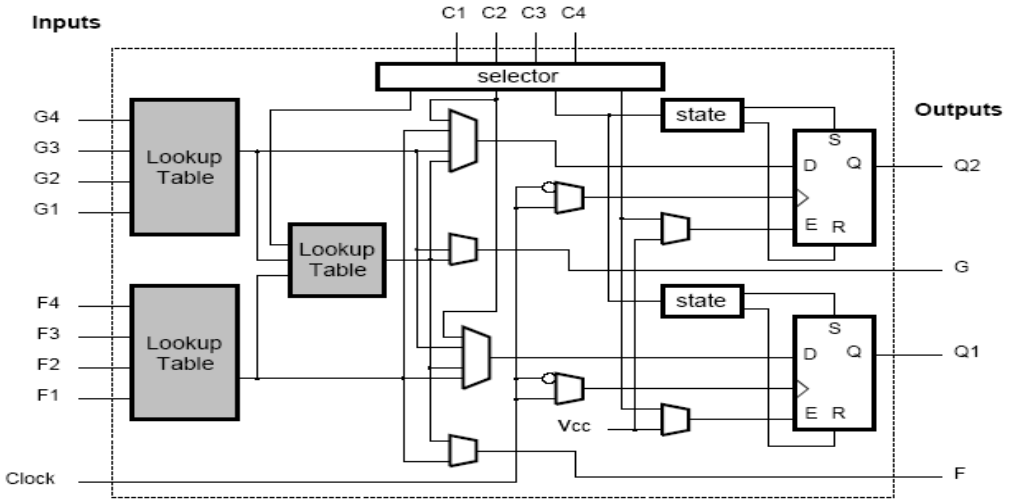


Figura 7.42. Bloque lógico configurable xilinx.

Ejemplo 7.8.

Diseñar la función de cuatro variables, empleando LUT:

$$f(x_1, x_2, x_3, x_4) = \overline{x}_2 x_3 + \overline{x}_1 x_2 \overline{x}_3 + x_2 \overline{x}_3 x_4 + x_1 \overline{x}_2 \overline{x}_4$$

Si se dispone de una tabla de búsqueda de 4 variables solo es necesario almacenar la tabla de verdad de la función.

Si el bloque constructivo es una LUT de 3 variables, puede expandirse la función en torno a la variable x_1, empleando el teorema de Shannon; de esta manera quedan dos funciones cofactores que dependen de tres variables, las cuales pueden ser directamente mapeadas en LUTs de 3.

$$f(x_1, x_2, x_3, x_4) = \overline{x}_1 f_{\overline{x}_1} + x_1 f_{x_1}$$
$$f_{\overline{x}_1} = f(0, x_2, x_3, x_4) = \overline{x}_2 x_3 + x_2 \overline{x}_3 + x_2 \overline{x}_3 x_4 = \overline{x}_2 x_3 + x_2 \overline{x}_3$$
$$f_{x_1} = f(1, x_2, x_3, x_4) = \overline{x}_2 x_3 + x_2 \overline{x}_3 x_4 + \overline{x}_2 \overline{x}_4$$

En la Figura 7.43 se muestra que, en este caso, la función cofactor $f_{\overline{x}_1}$, depende de dos variables.

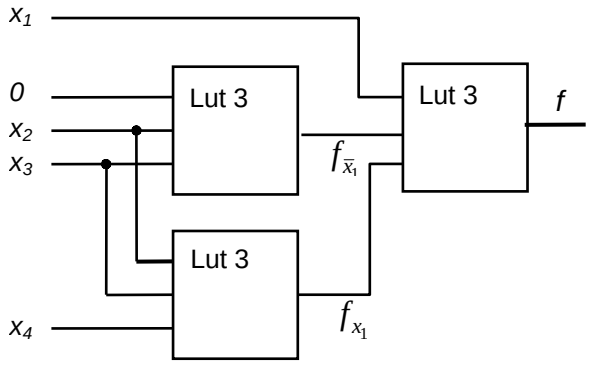


Figura 7.43. Implementación en LUTs de 3 variables.

Etapas o fases de diseño.

Trabajando con este tipo de componentes es preciso crear un proyecto e ingresar el diseño, ya sea usando diagramas esquemáticos o empleando un lenguaje de alto nivel como VHDL o Verilog. Esta etapa se denomina de **codificación** RTL, y consiste en la descripción estructural o de la conducta de un diseño.

Luego viene una fase de compilación y **simulación** funcional del diseño, en la que pueden corregirse errores. Puede verificarse el diseño lógico y el flujo de los datos.

A continuación se realiza la **síntesis**, que traslada el diseño a compuertas básicas, minimizando las ecuaciones.

Posteriormente se efectúa el **mapeo tecnológico** que implementa las ecuaciones con los elementos lógicos disponibles. En esta fase además de especificar las localizaciones que se emplearán también se efectúa los enrutamientos y conexiones entre los bloques lógicos (Place and route).

A continuación se efectúan **simulaciones temporales** considerando las compuertas y niveles del diseño.
Finalmente se genera un **archivo binario** de programación, el cual se graba en el dispositivo.

7.6.11. ASIC.

En algunas herramientas CAD, una vez verificado un diseño basado en FPGA y evaluado experimentalmente, puede migrarse el diseño a dispositivos ASIC; en los cuales se eliminan todos los circuitos de configuración y programación, de esta forma se puede diseñar un circuito integrado que opere a mayor velocidad y que consuma menos energía que el diseño basado en FPGA.

Por su menor costo de fabricación en elevados volúmenes y tener un comportamiento más rápido que los dispositivos programables, se suelen emplear dispositivos ASIC (Application Specific Integrated Circuit) cuando se tenga una producción en gran escala.

Problemas resueltos.

Problema 7.1. Expansión. Conexión cascada.

Un mux puede expandirse, si es combinado con otra componente del mismo tipo. Por ejemplo, puede formarse un MUX de 8 hacia 1, mediante dos MUX de 4 a 1 y uno de 2 vías a una.

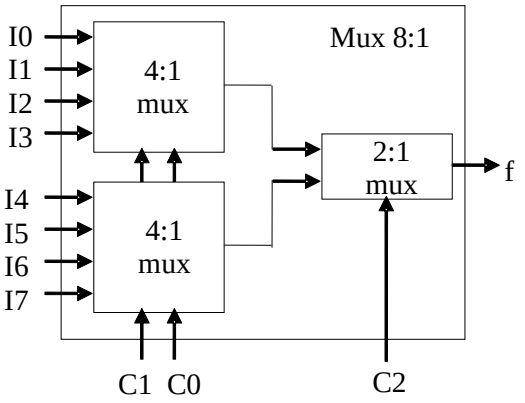


Figura P7.1. Conexión cascada de multiplexores.

En forma alternativa se muestra un mux de 8 vías a una, implementado con cuatro mux de 2 vías a una, y un mux de 4 vías a una. Ambos diseños difieren en el número de entradas.

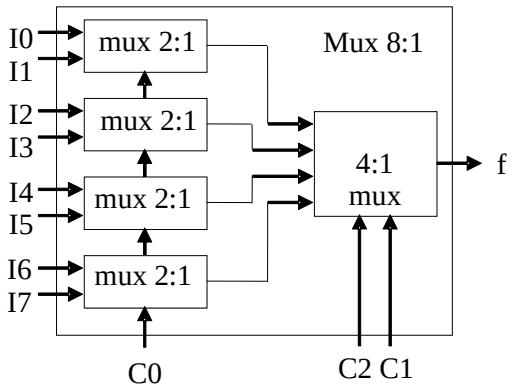


Figura P7.2. Conexión cascada de multiplexores.

Problema 7.2. Realización de funciones booleanas, mediante mux.

Mediante un multiplexor, pueden implementarse funciones lógicas. Consideremos el mux de 4 a 1, con **enable** =1, resulta:

$$f\,(c1, c0) = x3\ c1c0 + x2\ c1c0' + x1\ c1'c0 + x0\ c1'c0'$$

La expresión anterior muestra que, con un mux de 4 a 1, pueden implementarse todas las funciones booleanas de 2 variables. Para ello, basta hacer '1' ó '0' las entradas x_i, de acuerdo a los mintérminos de la función que se desee implementar.

Con un mux de 8 vías a 1, pueden obtenerse las 256 funciones booleanas de 3 variables, programando las entradas del mux con unos o ceros.

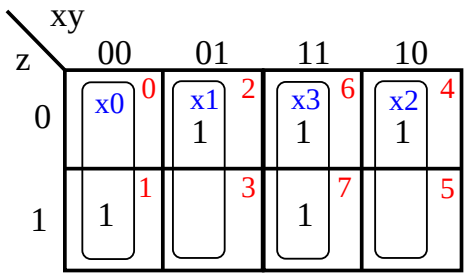
Si en las entradas de control del mux se aplican todas las variables de entrada menos una; y si se aplican en las entradas del mux: la variable de entrada restante o su complemento o unos y ceros, puede verse que con un mux de 4 vías a 1, pueden implementarse funciones booleanas de 3 variables, como se estudiará en el siguiente ejemplo.

Problema 7.3. Diseño de lógica combinacional empleando mux.

Programar un mux de 4 a 1 para implementar una función de tres variables:

$$f(x, y, z) = x'y'z + yz' + xz' + xy$$

Los mintérminos de f pueden dibujarse en un mapa:



f(x, y, z)=xy x3 + xy' x2 + x'y x1 + x'y' x0

Figura P7.3. Diseño empleando mux.

Asumiendo la señal enable en 1, y eligiendo **x** como la señal de control más significativa y la señal **y** como la línea de control menos significativa, la ecuación del mux muestra cómo es posible seleccionar los mintérminos de la función.

Por ejemplo, cuando *y* = 0 y *x* = 0:

- si la entrada *x0* se alimenta con el valor 1 lógico, se tendrán los mintérminos 0 y 1 presentes en la función;
- si *x0* se conecta a tierra, los mintérminos 0 y 1 no estarán presentes en la ecuación de la función;
- si la entrada *x0* se alimenta con la señal *z*, *f* tendrá sólo el mintérmino 1;
- si la entrada *x0* se alimenta con la señal *z'*, se tendrá, presente en *f*, el mintérmino 0;

Puede verse, en forma similar, que la señal *x3* controla la presencia de los mintérminos 6 y 7 en la función. Entonces se programa el multiplexor mediante: *x3* = 1; *x2* = z'; *x1* = z'; *x0* = z.

En el mapa se han marcado los grupos de mintérminos con los números de las señales de datos del mux.

El diseño anterior puede resumirse según:

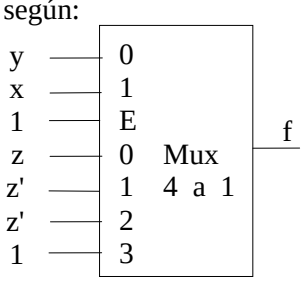


Figura P7.4 Esquema del diseño en base a mux.

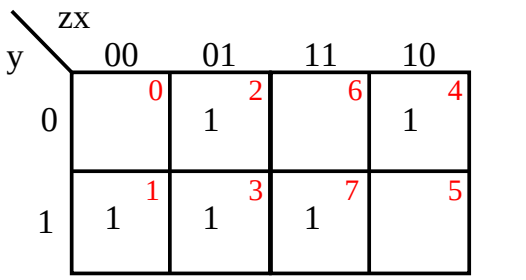
Nótese que la variable ***z*** se introduce en las entradas de datos del mux, y las variables *x*, e y como entradas de control.

Con un diseño tradicional SSI se necesitan 2 pastillas: Cuatro compuertas NAND de fan-in 2, y una compuerta OR de fan-in 4; el mux de 4 vías a una viene integrado en una sola pastilla.

Otra implementación para: $f = x'y'z + yz' + xz' + xy$ es considerar la señal enable en 1, y eligiendo ***z*** como la señal de control más significativa y la señal ***x*** como la línea de control menos significativa. La ecuación del mux resulta, en estas variables:

$$f(z, x, y) = zx\ x3 + zx'\ x2 + z'x\ x1 + z'x'\ x0.$$

Si se dibuja un mapa, y se marcan los mintérminos presentes de f, se logra:



$f(z, x, y) = zx\ x3 + zx'\ x2 + z'x\ x1 + z'x'\ x0$

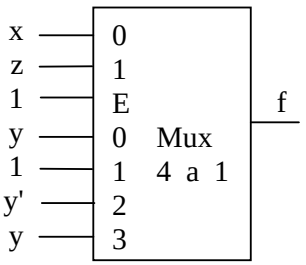


Figura P7.5. Otra implementación en base a mux.

Generalizando un mux de 8 vías a una, permite implementar todas las funciones de 4 variables. Basta elegir tres variables como entradas de control, y la cuarta se emplea en las entradas de datos.

Problema 7.4. Diseño empleando mux de 8:1

El mapa de 4 variables, de la Figura P7.6 define los unos de la función f.

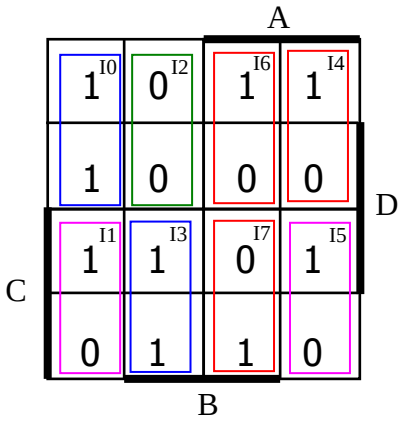


Figura P7.6. Grupos en en mapa de f.

La ecuación de un mux 8 a 1, es la siguiente:

f=c2c1c0I7+c2c1c0'I6+c2c1'c0 I5+c2c1'c0'I4+c2'c1c0I3+c2'c1c0' I2+c2'c1'c0 I1+c2'c1'c0' I0

Empleando A, B y *C* como entradas de control (*A* la más significativa), se obtiene:

f = ABC I7+ ABC' I6 + AB'C I5 + AB'C' I4 + A'BC I3 +A'BC' I2 + A'B'C I1 + A'B'C' I0

Luego se identifican las funciones de D, como entradas de datos. En el mapa se identifican pares de mintérminos asociados a un determinado valor de las entradas de control.

La Figura P7.7 resume el diseño:

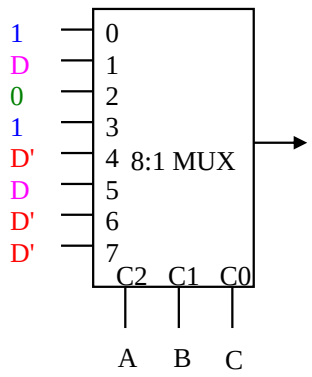


Figura P7.7. Diseño empleando multiplexor de 8 vías a una.

Problema 7.5. Mal uso de muxs

No siempre es conveniente diseñar con un mux. Por ejemplo, para la función de siete variables:

$$f = \bar{x}_1 + x_2 + x_3 + \bar{x}_4 + x_5 + \bar{x}_6 + x_7$$

Se requiere un mux de 128 vías a 1, o bien uno de 64 vías a una. También pueden emplearse 3 mux de 4 vías a 1, como se muestra en el siguiente ejemplo.

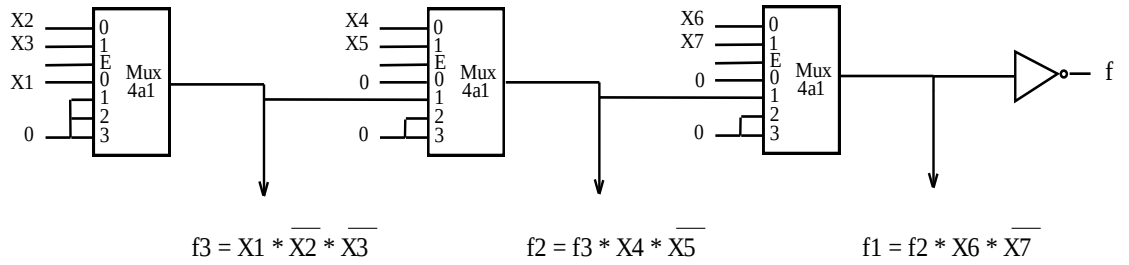


Figura P7.8. Diseño con exceso de muxs.

Con las señales enable (*E*) en uno.
La programación de los mux es por el alambrado (hardwired).

Nótese que en cada mux se utiliza solamente un mintérmino. Esto implica una mala utilización de las capacidades de cada mux. Un diseño **SSI,** mucho más simple, conveniente y directo es:

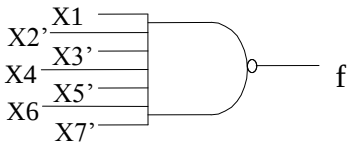


Figura P7.9. Diseño empleando compuertas en lugar de muxes.

En este caso se requiere sólo una pastilla, ya que se dispone comercialmente de un **NAND** de 8 entradas. En la implementación se conecta una de las entradas a Vcc, para disponer de un NAND de 7 entradas.

Problema 7.6. Diseño multifunción con decodificador.

Se desea diseñar las siguientes tres funciones de cuatro variables (diseño multifunción), empleando un decodificador 4: 16.
f1 = A′ B′ C D +A′ B C′ D + A B C D
f2 = A B C′ D' + A B C
f3 = (A′ + B′ + C′ + D′)

Se descomponen las funciones en suma de mintérminos, y luego se suman. El caso de *f3*, se resuelve por De Morgan según: f3′ = *ABCD*

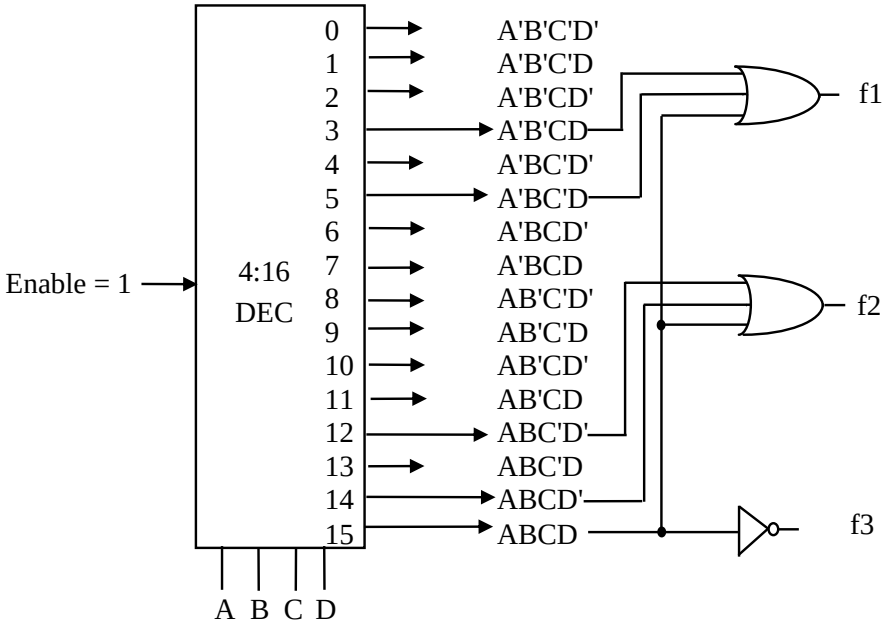


Figura P7.10. Diseño combinacional empleando decodificadores.

Problema 7.7. Diseño con decodificadores en base a mintérminos.

En la figura P7.11, se tiene un decodificador de 5:32, implementado en base a cuatro decodificadores 3: 8 y un decodificador de 2: 4. Con este decodificador es posible diseñar todos los mintérminos para funciones de cinco variables.

El primer decodificador genera, con *E* = 1, los grupos: *AB, AB', A'B, A'B'*.
Cada uno de los decodificadores 3:8, generan los ocho productos de la línea de entrada, con los mintérminos de las variables CDE.

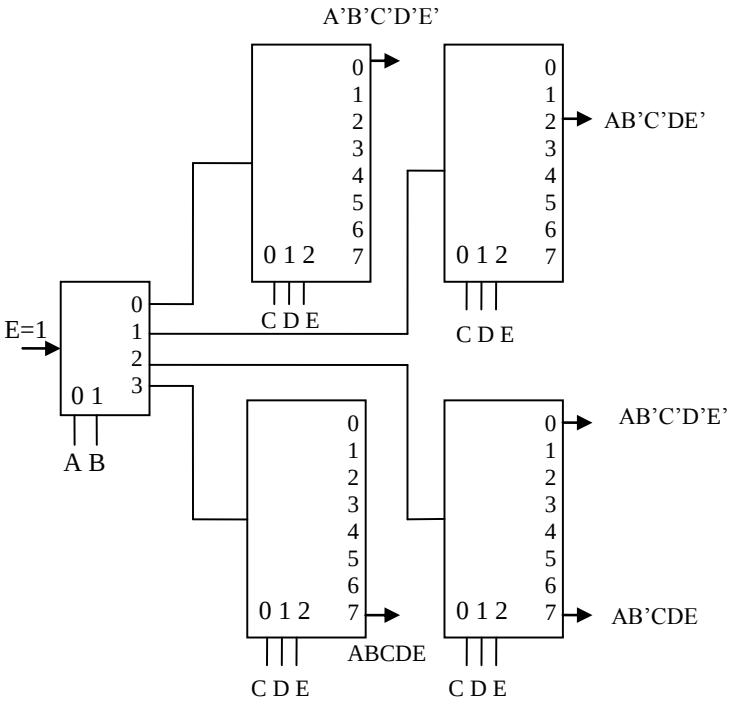


Figura P7.11. Diseño combinacional empleando decodificadores 3:8.

Problema 7.8. If then else anidados.

Un programador ha diseñado el siguiente diagrama de flujo, donde *C1*, *C2* y *C3* son condiciones y a, b, y c son grupos de acciones.

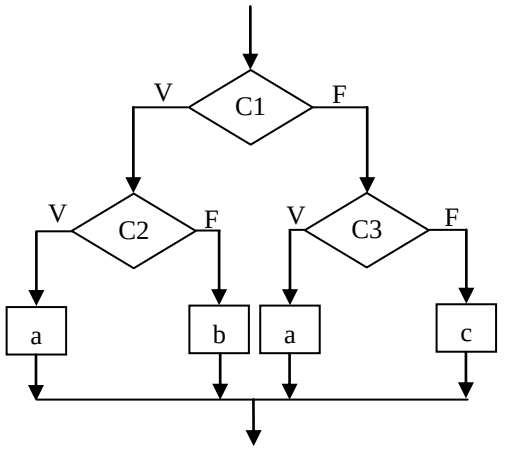


Figura P7.12. Diagrama de flujo Problema 7.8.

a) Determinar un nuevo diagrama de flujo, en el cual el grupo de acciones, denominado **a**, esté presente sólo una vez. Las condiciones del nuevo diagrama deben formularse en términos de *C1*, *C2* y *C3*. Indicar soluciones alternativas si existen varias.

b) Si analizando las variables, antes del ingresar al diagrama que se muestra, se conoce que la acción c nunca se realizará, determinar las condiciones para que se realicen las acciones a y b.

Solución.
a) Del diagrama se obtienen:
La acción a se realiza cuando la condición: C1C2 + C1' C3 es verdadera.
La acción b se realiza cuando la condición: *C1C2'* es verdadera.
La acción c se realiza cuando la condición: *C1'C3'* es verdadera.

Existen seis soluciones, en las cuales cada acción está presente una vez:

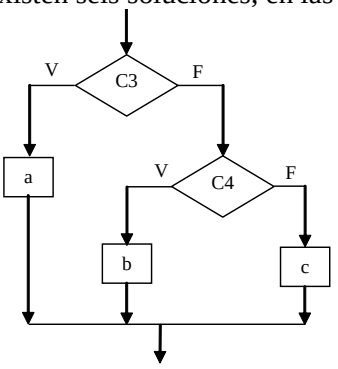


Diagrama 1.

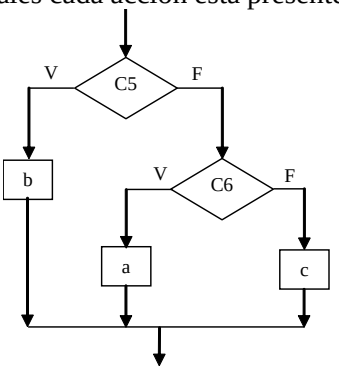


Diagrama 2.

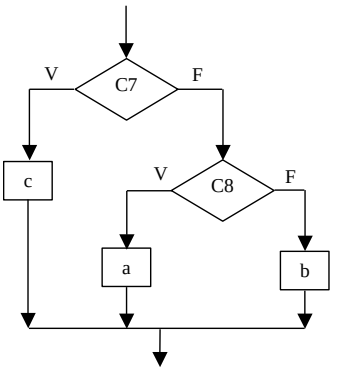


Diagrama 3.

Figura P7.13. Soluciones 1, 2 y 3.

C3 = C1C2 + C1' C3 *C4 = C1 C2'*
C5 = C1 C2' *C6 = C1C2 + C1' C3*
C7 = C1' C3' *C8 = C1C2 + C1' C3*

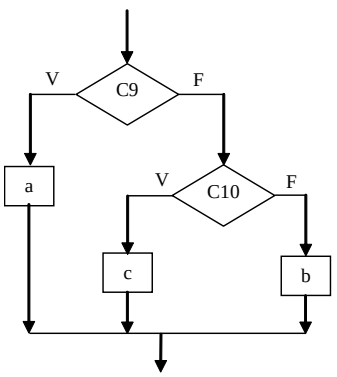


Diagrama 4.

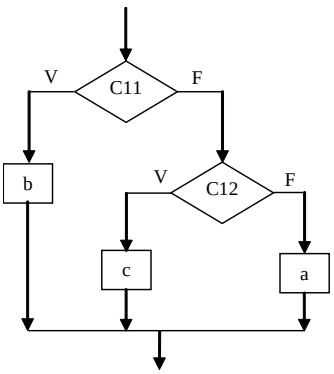


Diagrama 5.

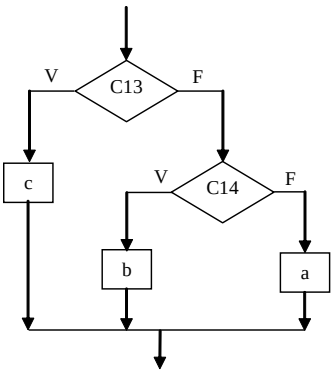


Diagrama 6.

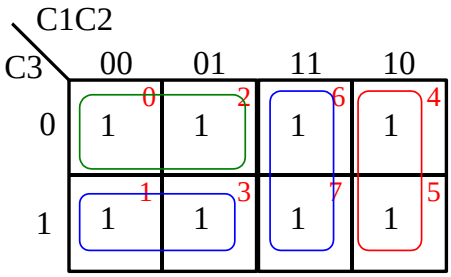
Figura P7.14. Soluciones 4, 5 y 6.

C9 = C1C2 + C1' C3 *C10 = C1' C3'*
C11 = C1 C2' *C12 = C1' C3'*
C13 = C1' C3' *C14 = C1 C2'*

Las condiciones más simples, de plantear, corresponden a los diagramas 5 y 6.
El código en C, para el diagrama 6 es:

```
if ( !C1 && !C3)  { c;}
else
   if(C1 && !C2)  {b;}  else {a;}
```

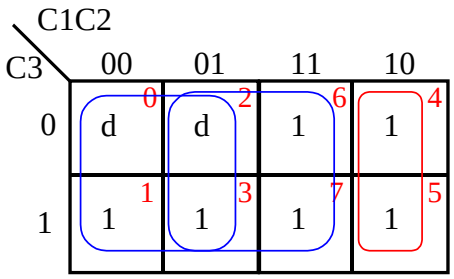
El siguiente mapa ilustra las condiciones para realizar las tres acciones: a, b y c, que son mutuamente excluyentes, ya que cubren todo el mapa.



a: C1C2 + C1' C3
b: C1C2'
c: C1'C3'

Figura P7.15. Condiciones para realización de a, b o c.

b) Si la acción *c* nunca se realiza, los mintérminos 0 y 2, se tratan como condiciones superfluas. En este caso la condición para la realización de la acción a, se simplifica a: *C1'+C2*. La condición para la realización de la acción b es: *C1C2'*, que resulta ser la negación de la anterior.



a: C1' + C2
b: C1C2'

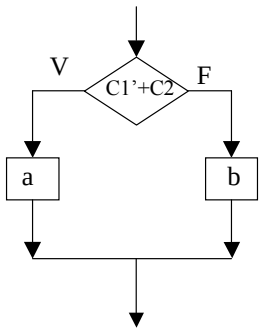


Figura P7.16. Condiciones para realización de a, b.

if(!C1 || C2) {a;} else {b;}

o bien:

if(C1 && !C2) {b;} else {a;}

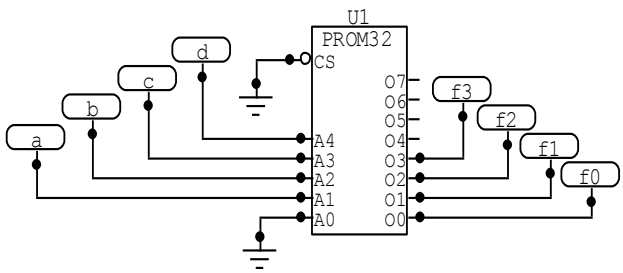
Si el grupo de acciones alternativas que deben realizarse, están condicionadas por un reducido número de mintérminos es conveniente emplear la sentencia switch, en lugar de sentencias anidadas if-then-else.

Problema 7.9. Diseño con PROM.

Se tiene una PROM de 32*8[bits], como se muestra en el esquema del circuito U1. Se entrega el mapa de memoria, tanto la dirección como el contenido se entregan en hexadecimal.

Dirección	Contenido
00	F3
01	15
02	D4
03	59
04	46
05	6B
06	01
07	C2
08	33
09	44
0A	57
0B	0F
0C	15
0D	9C
0E	23
0F	27

Dirección	Contenido
10	00
11	65
12	88
13	77
14	91
15	8D
16	F9
17	55
18	48
19	DD
1A	64
1B	13
1C	66
1D	88
1E	80
1F	22



Esquemático

Figura P7.17. Mapa de Memoria.

a) Dibujar el Mapa de Karnaugh para *f2*(*a*, *b*, *c*, *d*).
b) Determinar implicantes primos esenciales para *f2*.
c) Obtener la función mínima que permita una implementación de menor costo en base a compuertas para *f2*.

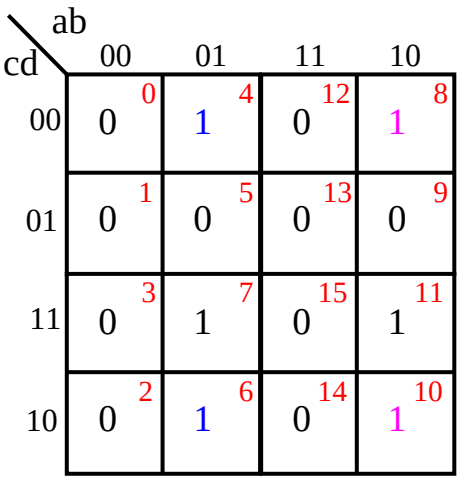
Solución.
Según el esquemático, *A0* y *O0* son los bits menos significativos de la dirección y el contenido respectivamente.

Se dibuja una tabla de verdad, con las direcciones pares de la memoria, identificando las posiciones de las variables *a, b, c* y *d.*

Se agrega en la tabla la columna H, que es la cifra hexadecimal menos significativa del contenido, y se expresa en binario en la cuarta columna, representando los bits de *f3 f2 f1 f0*. Entonces la tabla de verdad de *f2* corresponde al bit en la tercera posición, lo cual se agrega como última columna de la tabla.

Se dibuja un mapa para *f2*. Como el orden de las variables es (*a, b, c, d*) resultan los siguientes mintérminos para *f2*: 4, 6, 7, 8, 10, 11.

Dirección	dcba	H	binario	f2
0	0000	3	0011	0
2	0001	4	0100	1
4	0010	6	0110	1
6	0011	1	0001	0
8	0100	3	0011	0
A	0101	7	0111	1
C	0110	5	0101	1
E	0111	3	0011	0
10	1000	0	0000	0
12	1001	8	1000	0
14	1010	1	0001	0
16	1011	9	1001	0
18	1100	8	1000	0
1a	1101	4	0100	1
1c	1110	6	0110	1
1e	1111	0	0000	0



f2(a, b, c, d) = a'bd' + a'bc+ ab'd' + ab'c

Figura P7.18. Mapa de f2.

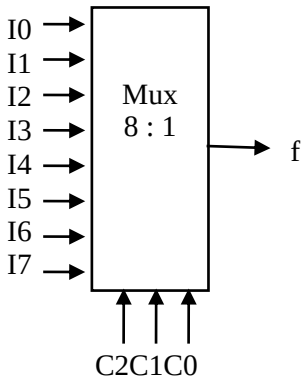
b) Según el mapa:
a'bd' es implicante primo esencial ya que es el único que contiene al mintérmino 4.
a'bc es implicante primo esencial ya que es el único que contiene al mintérmino 7.
ab'd' es implicante primo esencial ya que es el único que contiene al mintérmino 8.
ab'c es implicante primo esencial ya que es el único que contiene al mintérmino 11.

c) El diseño como suma de productos es: *f(a, b, c, d) = a'bd' + a'bc + ab'd' + ab'c*
Y tiene un costo de 16 entradas (12 literales).

El diseño como producto de sumas es, se obtiene agrupando los ceros de f2.
Se obtiene: *f*(*a, b, c, d*) = (*a* +*b*)(*a' + b'*)(*c + d'*), la cual tiene un costo de 9 entradas (6 literales), éste es el diseño mínimo.

Problema 7.10. Diseño con multiplexor.

Se ha diseñado la función *f*, empleando un multiplexor de 8 vías a una, cuyo esquema se muestra en la Figura P7.19 izquierda.
La siguiente tabla muestra las conexiones de las variables y constantes a las señales del multiplexor:



Señales	Variables
C2	a
C1	b
C0	c
I0	d
I1	d’
I2	ad
I3	b’d’
I4	0
I5	cd
I6	c’
I7	1

Figura P7.19. Mux Problema 7.10.

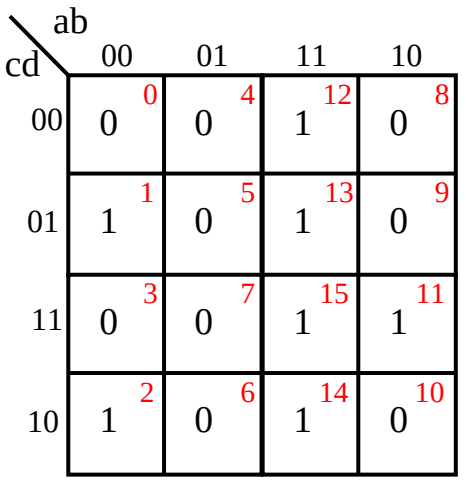
a) Determinar el mapa de Karnaugh para *f*.
b) Determinar implicantes primos esenciales de *f*(*a, b, c, d*).
c) Expresión lógica mínima para f como producto de sumas.

Solución:
a) Se tiene la siguiente ecuación para el multiplexor:
f = c2c1c0 I7+ c2c1c0' I6 + c2c1'c0 I5 + c2c1'c0' I4 + c2'c1c0 I3 + c2'c1c0' I2 + c2'c1'c0 I1 +c2'c1'c0' I0

Reemplazando las señales por las variables y constantes de la tabla, resulta:
f = abc1+ abc’ c’ + ab’c dc + ab’c’ 0 + a’bc b’d’ + a’bc’ ad + a’b’c d’ + a’b’c’ d

Simplificando, se logra:
f = abc + abc’ + ab’cd + a’b’c d’ + a’b’c’d

Puede dibujarse el mapa, y minimizando, se obtiene:
f(a, b, c, d) = ab + acd + a’b’c’d + a’b’cd



$f(a, b, c, d) = ab + acd + a'b'c'd + a'b'cd'$

Figura P7.20. Minimización Problema 7.10.

b) Se construye la tabla de implicantes:

	1	2	11	12	13	14	15
ab				x	x	x	x
acd			x				x
a'b'c'd	x						x
a'b'cd'		x					x

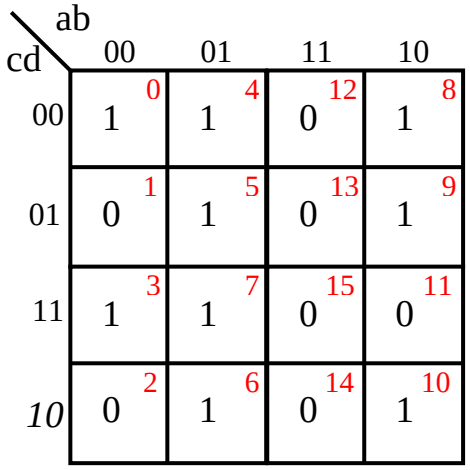
Figura P7.21. Tabla implicantes Problema 7.10.

Se tiene que el implicante *ab* es el mayor y único grupo que contiene a los mintérminos: 12, 13, y 14, por lo tanto es esencial.
El implicante *acd* es el único que contiene al mintérmino 11, por lo tanto es esencial.
Los mintérminos 1 y 2 no pueden agruparse y por lo tanto son implicantes primos esenciales.
La función mínima debe contener a los cuatros implicantes primos esenciales. Lo cual justifica el diseño de la parte a).

c) Para lograr la expresión lógica mínima para *f* como producto de sumas, debe agruparse los ceros de la función *f* (o los unos de la función *f '*, y luego complementar).

Para la tabla de implicantes de *f '* se tiene que:
a'b es implicante primo esencial(único grupo que contiene a los mintérminos 5 y 6);
a'cd es implicante primo esencial(único grupo que contiene al mintérmino 3);
ab'c' es implicante primo esencial(único grupo que contiene al mintérmino 9);
ab'd' es implicante primo esencial(único grupo que contiene al mintérmino 10);

Los implicantes primos *b'c'd'* y *a'c'd'* no son esenciales. Como en *f'*, deben estar presentes los implicantes primos esenciales, la tabla muestra que sólo resta cubrir al mintérmino 0; lo cual puede lograrse con el implicante primo *b'c'd'* o *a'c'd'*. Resultan las funciones f1' y *f2'* que se ilustran en el mapa.



	0	3	4	5	6	7	8	9	10
a'b			x	x	x	x			
a'cd		x				x			
ab'c'							x	x	
ab'd'							x		x
b'c'd'	x						x		
a'c'd'	x		x						

f1'(a, b, c, d) = a'b+a'cd+ab'c'+ab'd'+b'c'd'

f2'(a, b, c, d) = a'b+a'cd+ab'c'+ab'd'+a'c'd'

Figura P7.22. Mapa y Tabla implicantes producto de sumas.

Finalmente, complementando *f1'* y *f2'* (y aplicando De Morgan) se obtienen las dos soluciones mínimas como producto de sumas:

f1(a, b, c, d) = (a + b')(a + c' + d')(a' +b + c)(a' + b +d)(b + c + d)
f2(a, b, c, d) = (a + b')(a + c' + d')(a' +b + c)(a' + b +d)(a + c + d)

Ambas de 14 literales y de 19 entradas.

Problema 7.11. Multiplexor

Programar las entradas I7 a I0 del multiplexor para tener en la salida:

$$f(a, b, c, d) = \Sigma m(1, 3, 10, 11, 12, 15) + \Sigma d(5, 7, 13)$$

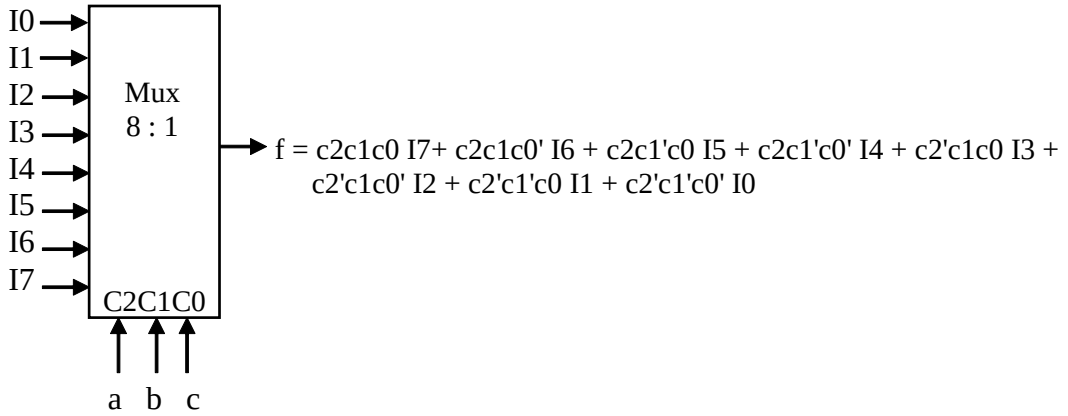


Figura P7.23. Mux Problema 7.11.

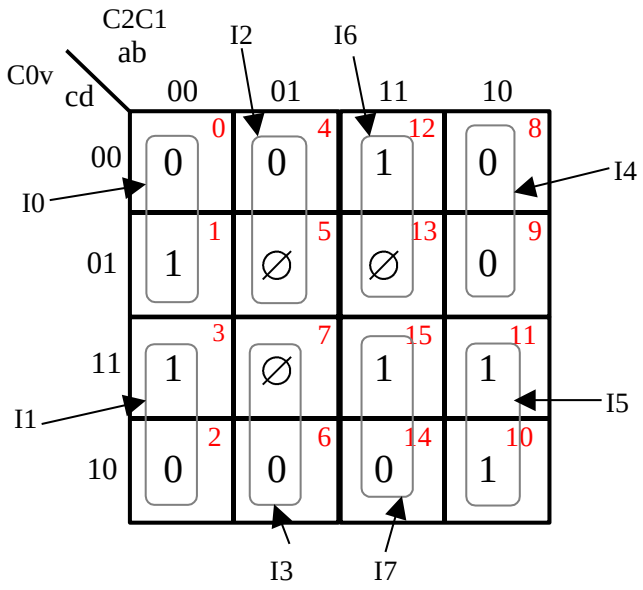
Indicar las diferentes soluciones, dependiendo de la elección del valor dado a las condiciones superfluas.

Solución:
En la ecuación del multiplexor, reemplazando *C2, C1* y C0 por *a, b* y *c* respectivamente, se tiene:

$$f = abc\ I7+abc'I6+ab'c\ I5+ab'c'\ I4+a'bcI3+a'bc'\ I2+a'b'c\ I1+a'b'c'\ I0$$

Para cada una de las entradas Ii podemos aplicar *d, d'*, 0 ó 1. Cada entrada permite incorporar distintas combinaciones de dos mintérminos (uno o el otro, o ambos o ninguno).

Ubicando los mintérminos y las condiciones superfluas en un mapa de cuatro variables, donde *a, b* y *c*, se han hecho equivalentes a *C2, C1* y *C0* respectivamente, se tiene:



f(a, b, c, d)= Σm(1, 3, 10, 11, 12, 15)+ Σd(5, 7, 13)

Figura P7.24. Mapa Problema 7.11.

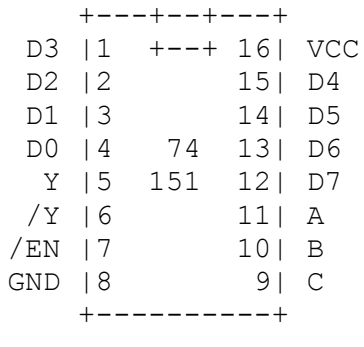
Si consideramos como ejemplo el término: T = *a'b'c'* I0
Con *I0=d* se tendrá *T = a'b'c'd* (m1). Con *I0=d'* se tendrá *T = a'b'c'd'* (m0).
Con *I0*=1, se tendrá *T=a'b'c'* (m0 +m1). Con *I0*=0, no estarán presentes m0 ni m1, o dicho de otra forma se aseguran los maxtérminos M0 y M1.

Deben escogerse: *I0 = d, I1 = d, I4* = 0, *I5* = 1, *I7* = d
Si *d5* se escoge 0, entonces *I2*=0. Si *d5* se escoge 1, entonces *I2=d*.
Si *d7* se escoge 0, entonces *I3*=0. Si *d7* se escoge 1, entonces *I3=d*.
Si *d13* se escoge 0, entonces *I6=d'*. Si *d13* se escoge 1, entonces *I6=1*.

Problema 7.12. Mux 74151

Se tiene los siguientes datos para un multiplexor 74151.
8-to-1 line data selector/multiplexer with complementary outputs.

C	B	A	EN	Y
X	X	X	H	L
L	L	L	L	D0
L	L	H	L	D1
L	H	L	L	D2
L	H	H	L	D3
H	L	L	L	D4
H	L	H	L	D5
H	H	L	L	D6
H	H	H	L	D7



a) Explicar brevemente el funcionamiento del multiplexor.
b) Diseñar la función *f(w,x,y,z)=x(y'+w) +w'(y'+x'z')* empleando un 74151.
Escoger *x = A, y = B, z = C*.
c) Se tiene un contador 74LS93 conectado al multiplexor 74LS151.
¿Qué relación existe entre los nombres de las señales de los pines del esquemático del 74LS151 y los de la hoja de datos?

Dibujar las formas de ondas en TP1 y TP2. Asumir para el dibujo que los circuitos no tienen retardos de propagación. Indicar función que realiza el sistema.

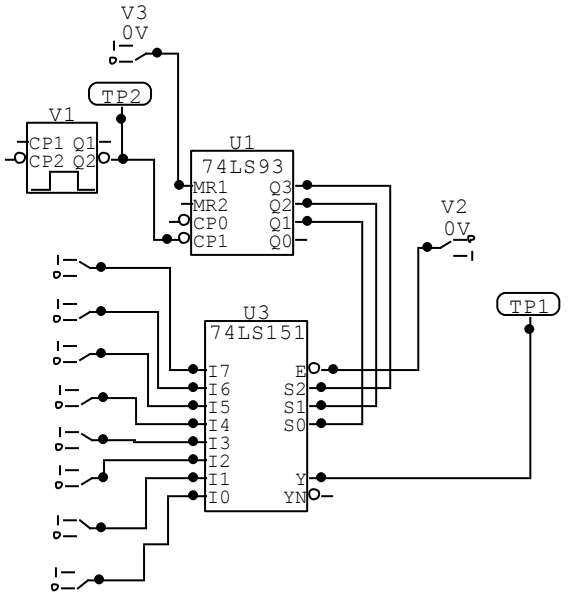


Figura P7.25. Conexiones Problema 7.12.

Solución.
a) Con *EN'* (enable) alto se tiene la salida *Y* en bajo. Es decir con *EN'* = 0 se habilitan la salida del mux.
En el manual TTL de TEXAS, la señal EN´se denomina *S* (strobe). Con *S* = 0 se habilita el mux.

Con $i = 2^2\ C + 2^1\ B + A$ se tiene $Y = Di$ ($i = 0 \ldots 7$) La entrada de control *C* es la más significativa.

Es decir:
Y =(D0 C'B'A' +D1 C'B'A+D2 C'BA'+D3 C'BA+D4 CB'A'+D5 CB'A+D6 CBA'+D7 CBA) S'

b) Se desea diseñar *f(w,A,B,C) = B'A + wA + w'B' + w'C'A'*

Si se dibuja un mapa de 4 variables con la función pedida, se tiene:

w/CBA	000	001	011	010	110	111	101	100
0	1	1	0	1	0	0	1	1
1	0	1	1	0	0	1	1	0
	D0 = w'	D1 = 1	D3= w	D2 = w'	D6 = 0	D7 = w	D5 = 1	D4 = w'

En el mapa se empleó código Gray, y se tiene que cada columna corresponde a una combinación de las señales de control del mux que activan una línea de datos de entrada. Para generar dicho mapa con el mux se deben efectuar las conexiones indicadas, es decir: en D0 se conecta la entrada w', en D1 se efectúa conexión a Vcc, etc.

Otra solución es expandir la función
$f(w,A,B,C) = (C'B'A+CB'A)+(wB'A+wBA)+(w'C'B'+w'CB')+(w'C'B'A'+w'C'BA')$ y luego volver a expandir

$f(w,A,B,C)=(C'B'A+CB'A)+(wC'B'A+wCB'A+wC'BA+wCBA)+(w'C'B'A'+ w'C'B'A+w'CB'A'+ w'CB'A)+(w'C'B'A'+w'C'BA')$

y ordenando, queda:
$f(w,A,B,C) = w'C'B'A'+ w'C'B'A'+ w'C'B'A+ wC'B'A+ C'B'A+w'C'BA'+ wC'BA+w'CB'A'+ w'CB'A+ wCB'A+CB'A+ wCBA$

Eliminando términos redundantes:
$f(w,A,B,C) = w'C'B'A' + C'B'A+w'C'BA'+ wC'BA+w'CB'A'+ CB'A+ wCBA$

Agregando algunos términos:
$f(w,A,B,C) = w'C'B'A' + 1C'B'A+w'C'BA'+ wC'BA+w'CB'A'+ 1CB'A+0CBA'+ wCBA$

Comparando con la ecuación del mux vista en 1, se logra identificar las funciones asociadas a los *Di*.

c) El contador genera la secuencia de valores 000, 001, 010, 011, 100, 101, 110, 111 que se aplican a las entradas de control del mux (es un contador módulo 8 en binario). Cuando se aplica en *S0*, *S1* y *S2* los valores 000, el mux saca en la salida Y el valor que esté conectado a *I0*, que en el ejemplo es 0; luego saca *I1* y así sucesivamente hasta sacar *I7*; y luego, la secuencia se repite.

Resumen: Genera la secuencia sincrónica de niveles: 01010001.

Se muestran las formas de ondas obtenidas con un simulador, la forma de onda TP1 muestra un retardo de propagación relativa al canto de bajada de TP2; este retardo contempla el del contador y del circuito combinacional del mux (se ilustra en el diagrama). Se ha destacado un ciclo completo de la salida.

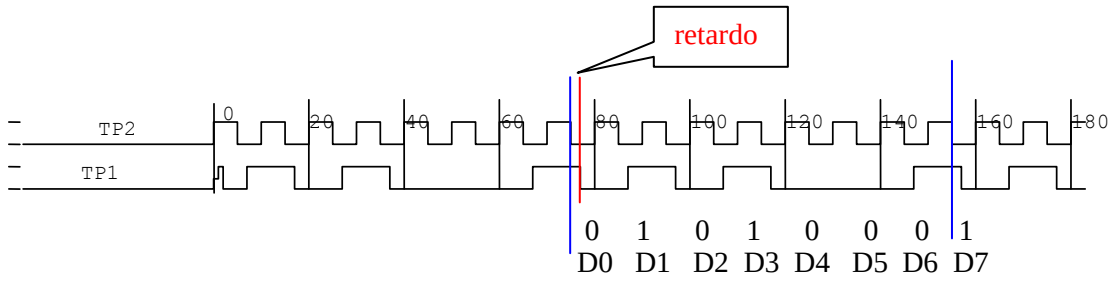
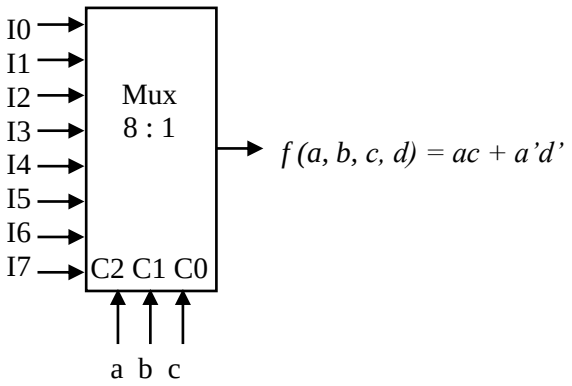


Figura P7.26. Formas de ondas. Problema 7.12.

Problema 7.13. Mux 8 a 1.

Determinar los valores con que deben programarse las entradas del mux para tener en la salida la función f.



I7	1
I6	0
I5	1
I4	0
I3	d'
I2	d'
I1	d'
I0	d'

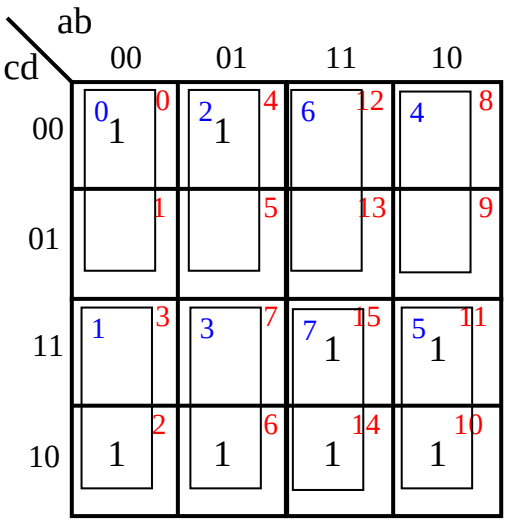
Solución.

Figura P7.27. Mux Problema 7.13.

Solución:
Se tiene:
f=c2c1c0 I7+c2c1c0' I6+c2c1'c0 I5+c2c1'c0' I4+c2'c1c0 I3+c2'c1c0' I2+c2'c1'c0 I1+c2'c1'c0' I0
Si se conectan las variables de entrada: a, b y c, según se indica en el diagrama, se tiene:
f = abc I7+ abc' I6+ ab'c I5+ ab'c' I4+ a'bcI3+ a'bc' I2+ a'b'c I1+ a'b'c' I0
Por otra parte pueden escribirse:
ac = abc + ab'c
a'd'= a'bd'+a'b'd'= a'bcd'+a'bc'd'+ a'b'cd'+a'b'c'd'
Es decir:
f (a, b, c, d) = ac + a'd'= abc + ab'c + a'bcd' + a'bc'd' + a'b'cd' + a'b'c'd'
Comparando coeficientes, resultan:

I7 = 1; I6 = 0; I5 = 1; I4 = 0; I3 = d'; I2 = d'; I1 = d'; I0= d'

Solución mediante mapas:



$f(a, b, c, d)$

$f\,(a, b, c, d) = \sum m(\,0, 2, 4, 6, 10, 11, 14, 15)$

Se identifican los pares de mintérminos controlados por las entradas al multiplexor.

Luego se obtienen los valores de las entradas al multiplexor, en función de la variable d.

Figura P7.28 Mapa Problema 7.13.

Problema 7.14. Programa en EPROM

Determinar el contenido de la EPROM, que implementa las funciones combinacionales: *f0*, *f1*, y *f2*, en función de las entradas *a*, *b*, *c*, *d*.

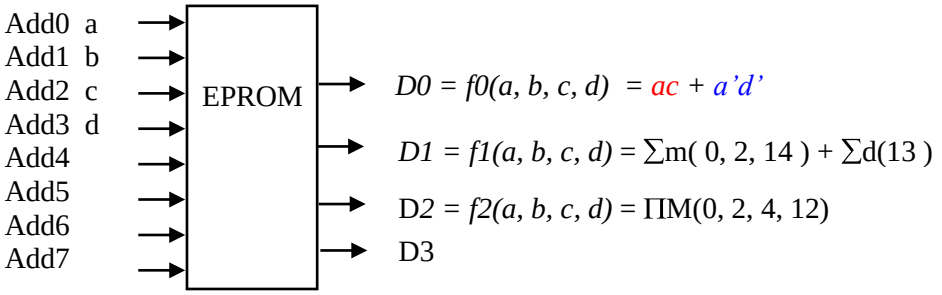


Figura P7.29 EPROM Problema 7.14.

Mostrar el contenido de la EEPROM según un mapa de direcciones versus contenidos.
Cómo deben conectarse las líneas de dirección que no se empleen. Valores que deben grabarse en *D3*.

Solución:
En una memoria sólo se pueden escribir unos o ceros.
Como no existe función asociada a la línea de datos *D3*, esta columna debe llenarse con cualquier secuencia de unos o ceros.

Para el caso del diseño de la función *f1*:

Puede escogerse que el mintérmino 13 esté presente en el conjunto on. Es decir se elige como 1 de la función. En este caso, la función resulta:
f1(a, b, c, d) = ∑m(0, 2, 14) + ∑d(13) = a'b'c'd' + a'b'cd' + abcd' +abc'd
En caso de escoger el término superfluo como cero, se tendrá:
f1(a, b, c, d) = a'b'c'd' + a'b'cd' + abcd'

La función *f2* establece los ceros de *D2*. También puede escribirse en función de los unos de *f2*, según: *f2*(*a*, *b*, *c*, *d*) = ∑m(1, 3, 5, 6, 7, 8, 9, 10, 11, 13, 14, 15)

Obtenidas las funciones deben escribirse las columnas de datos de la EPROM. Puede considerarse que cada dirección está asociada a un mintérmino; el número del mintérmino depende del ordenamiento dado a la función. En el caso propuesto, el orden es *a*, *b*, *c*, *d*; es decir, *a* es la variable más significativa para la numeración decimal de los mintérminos. Se ha colocado el número del mintérmino asociado a la tabla de contenidos versus direcciones.
Las líneas de dirección que no se empleen(add4, add5, add6 y add7) deben ser conectadas a tierra.

El siguiente es un mapa completo de la memoria.

#	Add7	Add6	Add5	Add4	Add3	Add2	Add1	Add0	D3	D2	D1	D0
					d	c	b	a		f2	f1	f0
0	0	0	0	0	0	0	0	0	1 ó 0	0	1	1
8	0	0	0	0	0	0	0	1	1 ó 0	1	0	0
4	0	0	0	0	0	0	1	0	1 ó 0	0	0	1
12	0	0	0	0	0	0	1	1	1 ó 0	0	0	0
2	0	0	0	0	0	1	0	0	1 ó 0	0	1	1
10	0	0	0	0	0	1	0	1	1 ó 0	1	0	1
6	0	0	0	0	0	1	1	0	1 ó 0	1	0	1
14	0	0	0	0	0	1	1	1	1 ó 0	1	1	1
1	0	0	0	0	1	0	0	0	1 ó 0	1	0	0
9	0	0	0	0	1	0	0	1	1 ó 0	1	0	0
5	0	0	0	0	1	0	1	0	1 ó 0	1	0	0
13	0	0	0	0	1	0	1	1	1 ó 0	1	1 ó 0	0
3	0	0	0	0	1	1	0	0	1 ó 0	1	0	0
11	0	0	0	0	1	1	0	1	1 ó 0	1	0	1
7	0	0	0	0	1	1	1	0	1 ó 0	1	0	0
15	0	0	0	0	1	1	1	1	1 ó 0	1	0	1

Figura P7.30 Mapa EPROM Problema 7.14.

Problema 7.15. Programar EPROM.

Se tiene una EPROM de 16 palabras de 4 bits cada una.
Se desea generar las siguientes funciones:
f0(a, b, c, d) = ∑m(2, 4 , 8, 15) + ∑d(3, 5)

$f1(d, c, a, b) = \sum m(2, 4, 8, 15) + \sum d(3, 5)$
$f2(a, b, c, d) = \prod M(2, 4, 8, 15) \prod D(3, 5)$
$f3 = f0 + f1$

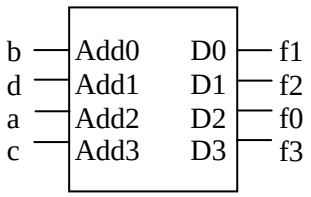
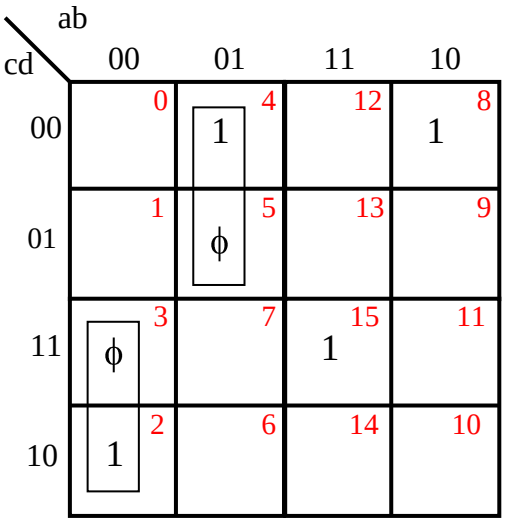


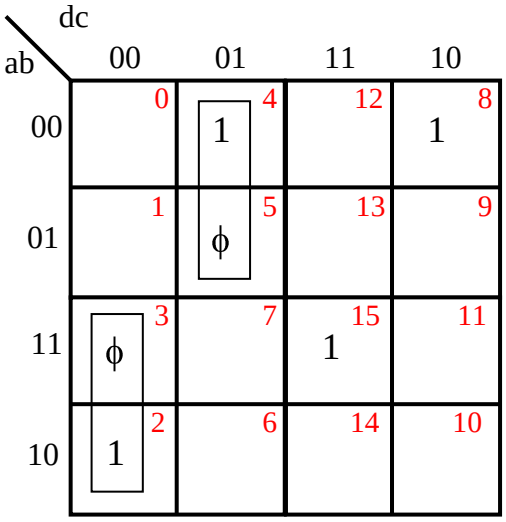
Figura P7.31 Variables y direcciones EPROM Problema 7.15.

a) Determinar las funciones en términos de las variables de entrada.
b) Determinar el mapa de la EPROM con las asociaciones entre las variables de entrada y las direcciones que se muestran en el diagrama; y con las funciones en las líneas de datos que se muestran en el diagrama. La tabla debe estar ordenada por direcciones ascendentes, y los contenidos el más significativo a la extrema derecha.

Solución:
Pueden dibujarse los mapas para *f0* y *f1*.



$f0(a,b,c,d) = a'bc' + a'b'c + ab'c'd' + abcd$



$f1(a,b,c,d) = ac'd' + a'cd' + a'b'c'd + abcd$

Figura P7.32 Mapas f0 y f1 Problema 7.15.

Para *f2*:
$f2 = f0' = (a + b' + c)(a + b + c')(a' + b + c + d)(a' + b' + c' + d')$
$f2 = a'b'c' + c'd' + abc' + a'bc + bcd' + ab'c$

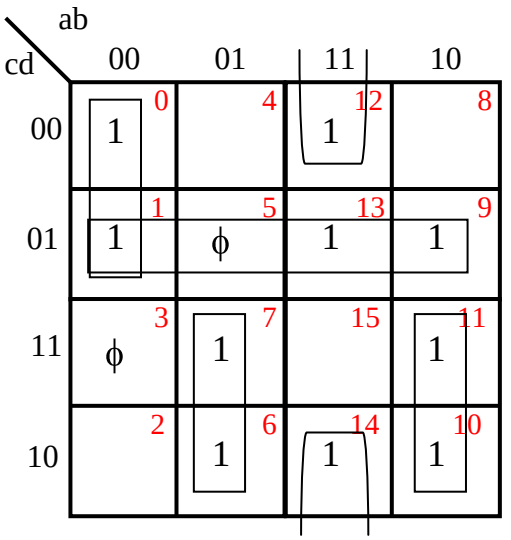


Figura P7.33 Mapas f2 Problema 7.15.

Finalmente f3, resulta:
f3 = f0 + f1 = a'b'cd' + a'bc'd' + ab'c'd' + a'b'c'd + abcd

	c	a	d	b	f3	f0	f2	f1
dirección	add3	add2	add1	add0	D3	D2	D1	D0
0	0	0	0	0	0	0	1	0
1	0	0	0	1	1	1	0	0
2	0	0	1	0	1	0	1	1
3	0	0	1	1	0	0	1	0
4	0	1	0	0	1	1	0	1
5	0	1	0	1	0	0	1	0
6	0	1	1	0	0	0	1	0
7	0	1	1	1	0	0	1	0
8	1	0	0	0	1	1	0	1
9	1	0	0	1	0	0	1	0
10	1	0	1	0	0	0	1	0
11	1	0	1	1	0	0	1	0
12	1	1	0	0	0	0	1	0
13	1	1	0	1	0	0	1	0
14	1	1	1	0	0	0	1	0
15	1	1	1	1	1	1	0	1

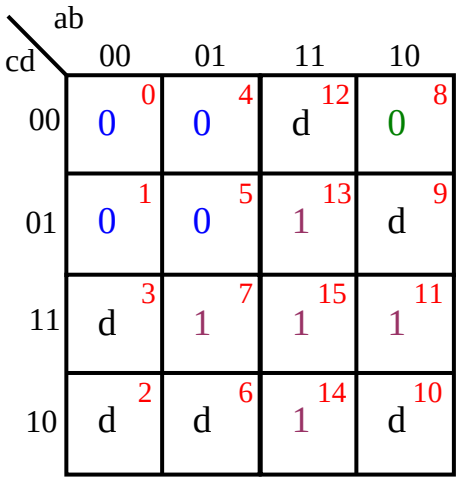
Figura P7.34 Mapas EPROM Problema 7.15.

Problema 7.16. Diseño con condiciones superfluas.

Se tiene que la función expresada como producto de sumas, considerando las condiciones superfluas con valores iguales a uno es: $f\,(a, b, c, d) = (a + c)(a' + b + c + d)$
Se tiene que la función mínima expresada como suma de productos, considerando las condiciones superfluas con valores iguales a cero es: $f\,(a, b, c, d) = abd + abc + bcd + acd$

a) Determinar los mintérminos superfluos para *f*.
b) Determinar el producto de sumas mínimo usando las condiciones superfluas.
c) Determinar la suma de productos mínima usando las condiciones superfluas.

Solución:
a) La primera proposición nos permite determinar los ceros de la función. La segunda determina los unos de la función. Dibujando en un mapa, pueden encontrarse las condiciones superfluas de *f*, como los mintérminos: 2, 3, 6, 9, 10, 12.



$f(a, b, c, d) = abd + abc + bcd + acd$ con Cond. Superfluas iguales a 0
$f(a, b, c, d) = (a + c)(a' + b + c + d)$ con Cond. Superfluas iguales a 1
Figura P7.35 Mapas Problema 7.16.

b) Agrupando los unos de la función complementada y considerando las condiciones superfluas, se obtienen los siguientes implicantes primos: *a'c', c'd', b'c', a'b', b'd'*.

	0	1	4	5	8
a'c'	x	x	x	x	
c'd'	x		x		x
b'c'	x	x			x
a'b'	x	x			
b'd'	x				x

Figura P7.36 Implicantes Problema 7.16.

Confeccionando una tabla de implicantes (donde no se anotan los superfluos), se obtiene que *a'c'* es esencial. Entonces sólo resta cubrir el mintérmino 8; esto es posible de tres formas, eligiendo: (*c'd'*), (*b'c'*) o (*b'd'*). Se tienen entonces:

$f\,1'(a, b, c, d) = a'c' + c'd'$
$f\,2'(a, b, c, d) = a'c' + b'c'$
$f\,3'(a, b, c, d) = a'c' + b'd'$

Complementado y aplicando De Morgan, se obtienen las tres soluciones, de igual costo (6 entradas, 4 literales):

$f\,1(a, b, c, d) = (a + c)(c + d)$
$f\,2(a, b, c, d) = (a + c)(b + c)$
$f\,3(a, b, c, d) = (a + c)(b + d)$

c) Agrupando los unos de la función y considerando las condiciones superfluas, se obtienen los siguientes implicantes primos: *c, ab, ad*. Confeccionando una tabla de implicantes (donde no se anotan los superfluos), se obtiene que *c* es implicante primo esencial. Sólo resta cubrir al mintérmino 13, lo cual puede lograrse de dos formas; eligiendo (*ab*) o (*ad*). Se tienen entonces, dos soluciones:

	7	11	13	14	15
c	x	x		x	x
ab			x	x	x
ad		x	x		x

Figura P7.37 Implicantes Problema 7.16, parte c.

$f\,1(a, b, c, d) = c + ab$
$f\,2(a, b, c, d) = c + ad$

Ambas de igual costo (4 entradas, 3 literales).

El método de minimización empleado, está basado en:

- obtener los implicantes primos (por inspección del mapa, o aplicando método de Quine McCluskey);
- luego se plantea la tabla de implicantes (se omiten las columnas asociadas a mintérminos superfluos, si existieren), y mediante ella se determinan los implicantes primos esenciales, que deben estar presentes en la función.
- Luego se reduce la tabla y se determinan todas las soluciones posibles. Se elige la de menor costo.

Problema 7.17. Diseño combinacional

Un sistema digital tiene cuatro entradas: *a, b, c, d* y una salida *z*, que debe colocarse alta cuando se cumplan las siguientes condiciones:

i) Si la entrada, en binario, es múltiplo de 3 ó 7.

Múltiplo está definido como el número que contiene a otro una o más veces exactamente. Considere que *a* es el bit más significativo y que *d* es el menos significativo.
ii) Si se activa *a*, no debe activarse *b*.
iii) Si no se activa *a*, entonces debe activarse *c* o *d* o ambos.

a) Expresar la condición i) como suma de mintérminos.
b) Expresar la condición ii) como producto de maxtérminos.
c) Expresar la condición iii) mediante un mapa de Karnaugh.
d) Minimizar la función z.

Solución:
a) El cero no se considera, con la definición dada, como múltiplo de 3 ó 7.
Condición i) = Σm(3, 6, 7, 9, 12, 14, 15)

#	a	b	c	d	i)	ii)	iii)	z
0	0	0	0	0	0	1	0	0
1	0	0	0	1	0	1	1	0
2	0	0	1	0	0	1	1	0
3	0	0	1	1	1	1	1	1
4	0	1	0	0	0	1	0	0
5	0	1	0	1	0	1	1	0
6	0	1	1	0	1	1	1	1
7	0	1	1	1	1	1	1	1
8	1	0	0	0	0	1	1	0
9	1	0	0	1	1	1	1	1
10	1	0	1	0	0	1	1	0
11	1	0	1	1	0	1	1	0
12	1	1	0	0	1	0	1	0
13	1	1	0	1	0	0	1	0
14	1	1	1	0	1	0	1	0
15	1	1	1	1	1	0	1	0

Figura P7.38 Tablas de verdad. Problema 7.17.

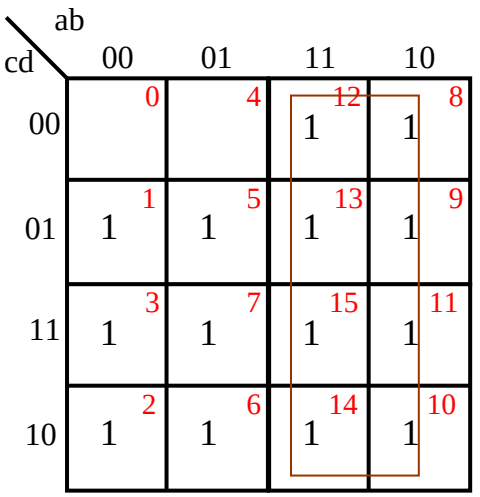
b) Condición ii)

$a \Rightarrow b' = a' + b'$
$= (a' + b')(c + c')$
$= (a' + b' + c')(a' + b' + c)(d + d')$
$= (a' + b' + c' + d)\,(a' + b' + c' + d')\,(a' + b' + c + d)\,(a' + b' + c + d')$
$= \quad M14 \qquad M15 \qquad M12 \qquad M13$
$= \Pi M(12,13,14,15).$

c) Condición iii)

$a' \Rightarrow (c + d) = a + c + d$

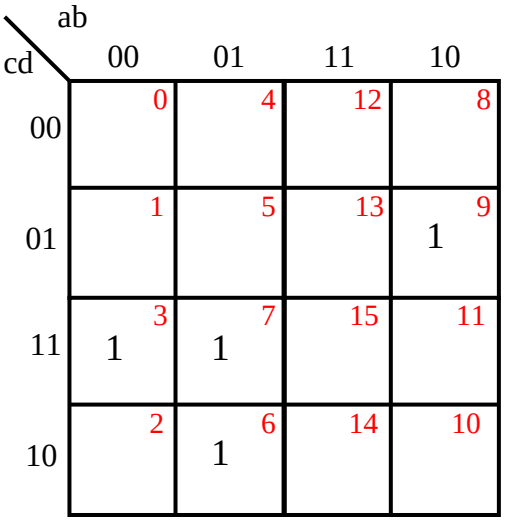
El mapa resulta:



Condición iii) = $a + c + d$

Figura P7.39 Mapa condición iii). Problema 7.17.

d) Cuando se cumplan las siguientes condiciones se interpreta como el and de las condiciones i), ii) y iii).



$z = \Sigma m(3, 6, 7, 9)$

Figura P7.40 Mapa con minimización de z. Problema 7.17.

Como suma de productos: $z = a'b'c'd + a'cd + a'bc$ 10 literales, 13 entradas.
Dos pastillas de baja integración: Un nand de tres entradas y un nand de 4 entradas.

Como producto de sumas: $z' = b'd' + a'c' + ac + ab$
$z = (b + d)(a + c)(a' + c')(a' + b')$ 8 literales, 12 entradas.

Dos pastillas de baja integración: Un nor de dos entradas y un nor de 4 entradas.
Entonces el diseño mínimo para *z* es mediante la forma producto de sumas.

Problema 7.18. Diseño de control de un display.

Se desea diseñar un circuito combinacional minimizado de 4 entradas (*A, B, C, D*) que produzca las señales que controlen un display de 4 segmentos. El display tiene 4 líneas de control (*C0, C1, C2, C3*) tales que cuando la línea de control está activada (es 1 lógico) se enciende el LED correspondiente; en caso contrario el segmento del LED permanece apagado.
Se sabe que las entradas no pueden estar todas en cero, y que tampoco pueden estar altas más de dos entradas simultáneamente. El diagrama ilustra el segmento que activa cada señal de control.

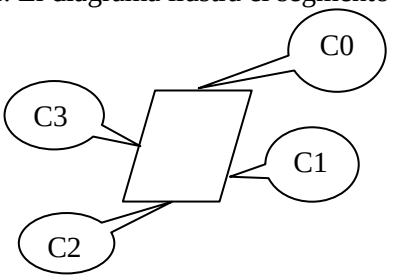


Figura P7.41 Esquema display. Problema 7.18.

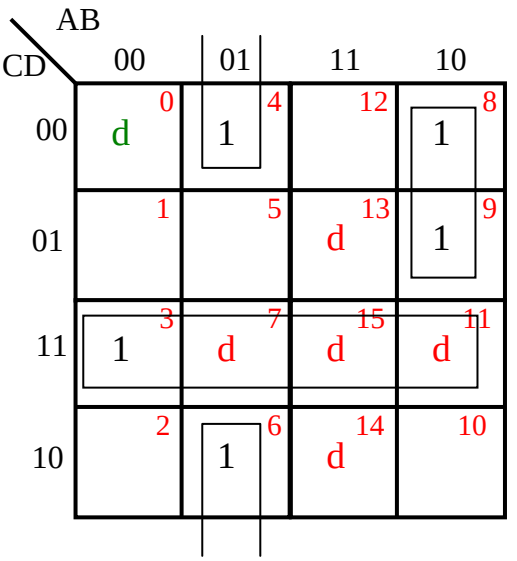
Se asume que las entradas *ABCD* representan un número binario, donde *A* es la cifra más significativa.
A continuación se ilustran los siguientes símbolos ordenados de izquierda a derecha. El ubicado más a la izquierda debe representar al número binario menor (dado por *ABCD*), y así sucesivamente hasta el número binario mayor que pueda representarse.

Solución.
Se puede confeccionar la siguiente tabla de verdad:

Las salidas del sistema combinacional (las señales que controlan los LEDs) se consideran condiciones superfluas para el primer renglón de la tabla, debido a que las entradas no pueden estar todas en cero. Y como no pueden existir más de dos entradas altas, también se consideran superfluos los mintérminos: 7, 11, 13, 14 y 15.

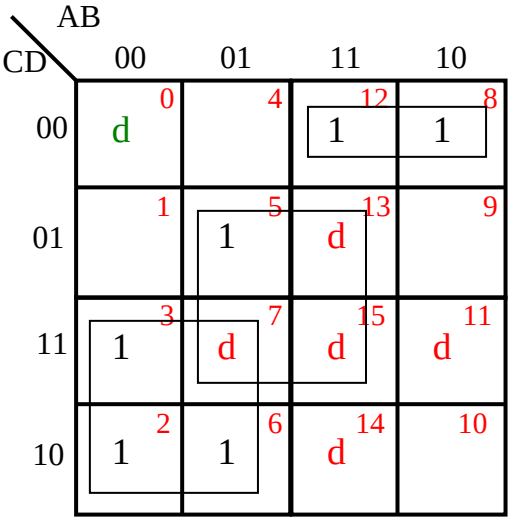
Luego se van llenando los valores de las señales *C0, C1, C2* y *C3*, renglón por renglón, considerando los segmentos que deben encenderse, de acuerdo al orden de los símbolos.

A	B	C	D	C0	C1	C2	C3
0	0	0	0	d	d	d	d
0	0	0	1			1	1
0	0	1	0		1	1	
0	0	1	1	1	1		
0	1	0	0	1			1
0	1	0	1		1	1	1
0	1	1	0	1	1	1	
0	1	1	1	d	d	d	d
1	0	0	0	1	1		1
1	0	0	1	1		1	1
1	0	1	0			1	
1	0	1	1	d	d	d	d
1	1	0	0		1		
1	1	0	1	d	d	d	d
1	1	1	0	d	d	d	d
1	1	1	1	d	d	d	d

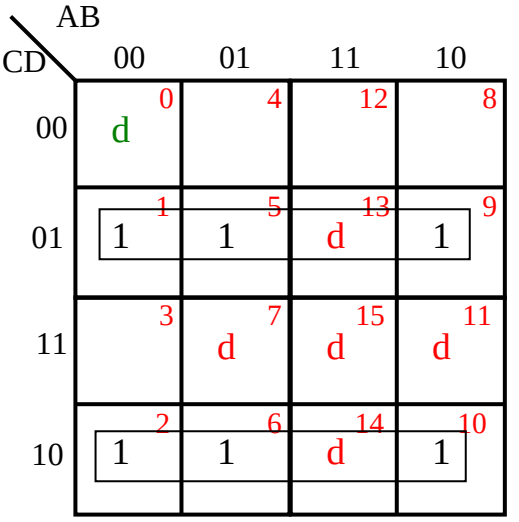


C0=A'BD'+AB'C'+CD

Figura P7.42 Tabla de verdad y Mapa de C0.

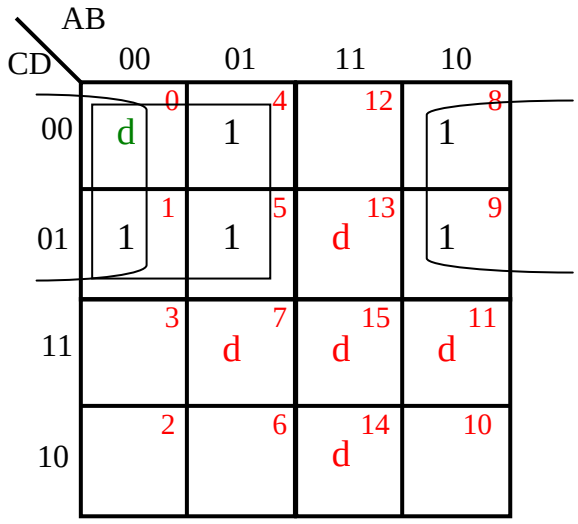


C1=AC'D'+BD+A'C



C2=C'D+CD'

Figura P7.43 Mapas de *C1* y *C2*.



C3=A'C'+ B'C'

Figura P7.44 Mapa de *C3*.

Observaciones respecto a las minimizaciones.
Los implicantes primos de *C0* son: *A'BD'* (4,6), *AB'C'* (8,9), *A'C'D'*(4,0), *BCD'*(6,14), *AC'D*(9,13), *A'BC*(6,7), *ABC*(14,15), *ABD*(13,15), *AB'D*(9,11), *B'C'D'*(8,0), *AD*(9,11,13,15), *BC*(6,7,14,15), *CD*(4,7,11,15).
Entonces *CD* debe estar presente ya que es el único que cubre al 3.
Se puede escoger *A'BD'* o *A'C'D'+BC* para cubrir el 4 y el 6. Obviamente es de menor costo *A'BD'*.
Se puede escoger *AB'C'* o *B'C'D'+AD* para cubrir el 8 y el 9. Obviamente es de menor costo *AB'C'*.

Los implicantes primos de *C1* son: *AC'D', A'B'D', B'C'D', AB, A'C, BD, CD, BC*.
BD debe estar presente ya que es esencial (el único que cubre al 3).
AC'D' cubre al 8 y 12. Igual cobertura se logra con mayor costo: *B'C'D'+AB*
A'C cubre al 2, 3 y 6. Mucho mayor costo para igual cobertura es: *CD+BC+A'B'D*

Los implicantes primos de *C2* son: *A'B'C', A'B'D', AD, BC, CD', C'D, BD*
La única minimización razonable que se considera es: *C'D+CD'*

Los implicantes primos de *C3* son: *A'C', B'C', AD, BD, C'D.*
La única minimización razonable que se considera es: *B'C'+A'C'*

No se requería emplear reducción de tablas de implicantes. Pero es un buen ejercicio practicar con las cuatro tablas anteriores, recordando que no deben considerarse los mintérminos superfluos en las tablas.

Ejercicios propuestos.

Ejercicio 7.1.

Diseñar con un multiplexor 8 a 1, la función: $f(a, b, c, d) = \sum m(0, 2, 4, 6)$

Ejercicio 7.2.

Diseñar con un decodificador 3 a 8, y compuertas or, la función: $f(a, b, c, d) = \sum m(0, 2, 4, 6)$

Ejercicio 7.3.

Desarrollar la función *f=ab+c'* en la celda de la Figura 7.40.

Ejercicio 7.4.

Desarrollar la función del ejemplo 7.8 empleando LUTs de 2 variables.

Índice general.

Índice de figuras