Tema 4: Sistemas Combinacionales

Universidad de Huelva

Escuela Técnica Superior de Ingeniería

Departamento de
Ingeniería Electrónica, Sistemas Informáticos y Automática

Tema 4: Sistemas Combinacionales

Introducción

En cualquier computador nos encontramos operaciones de codificación y decodificación de señales digitales, operaciones aritméticas, multiplexado y demultiplexado, transmisiones y control de datos usando buses, etc.

Los fabricantes de circuitos integrados disponen de bloques funcionales, cuyas primitivas fundamentales son las puertas lógicas estudiadas en los temas anteriores. Estos bloques se engloban dentro de los dispositivos MSI, y tienen la ventaja de poder usarse como una entidad de diseño de nivel superior a las puertas. De esta forma, no es necesario recordar su estructura interna de puertas, sino que es suficiente con conocer su comportamiento.

En este tema estudiaremos los más usuales y sus aplicaciones:

- ***Decodificadores***
- ***Codificadores***
- ***Multiplexores***
- ***Demultiplexores***
- ***Comparadores***
- ***Detectores/generadores de paridad***
- ***Aplicaciones de los circuitos enumerados***

Otra posibilidad a la hora de realizar sistemas digitales son los dispositivos programables. Estos elementos, de mayor complejidad estructural, permiten ser programados por el usuario con las funciones que necesitemos realizar. En este tema veremos los de tipo combinacional: estructuras PLA, PAL y memorias estáticas tipo ROM.

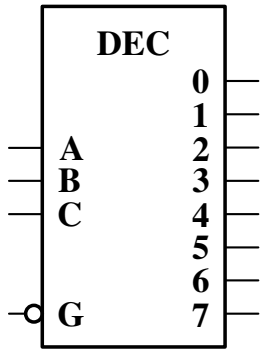
Durante el tema, introduciremos más conceptos sobre VHDL, y se verán ejemplos prácticos de descripciones de los sistemas estudiados.

Decodificadores (I)

Un decodificador es un circuito lógico con **n** entradas y **2^n** salidas como máximo, tal que para cada combinación de entradas se activa una salida que representa la combinación binaria aplicada a las entradas. Si existe una salida para cada combinación binaria de las variables de entrada, se denomina **decodificador completo**.

Por ejemplo, este es un circuito decodificador completo de 3 a 8 líneas. Dispone de una entrada de **HABILITACIÓN** (*enable*) que conecta o desconecta (coloca todas sus salidas al nivel no activo) el dispositivo. En este caso dicha entrada es activa a **nivel bajo**, ya que el dispositivo se activa cuando dicha entrada recibe un '0' lógico.

G	C	B	A	D0	D1	D2	D3	D4	D5	D6	D7
1	X	X	X	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0	0	0
0	0	0	1	0	1	0	0	0	0	0	0
0	0	1	0	0	0	1	0	0	0	0	0
0	0	1	1	0	0	0	1	0	0	0	0
0	1	0	0	0	0	0	0	1	0	0	0
0	1	0	1	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	0	0	0	1	0
0	1	1	1	0	0	0	0	0	0	0	1



Decodificadores (II)

Los decodificadores pueden dividirse en diferentes tipos:

- **EXCITADORES (DRIVERS)**, que controlan algún dispositivo, como por ejemplo visualizadores de 7 segmentos.
- **NO EXCITADORES**, los que no se usan para dicho fin.

Tanto las entradas como las salidas, principalmente estas últimas, pueden ser:

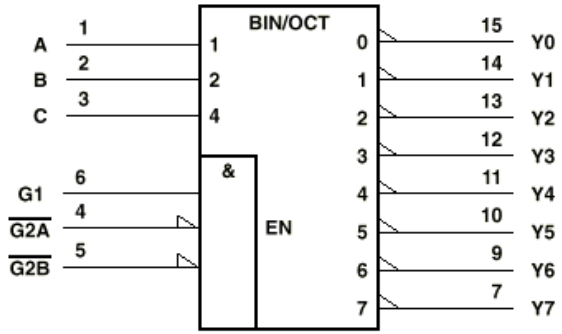
- ACTIVAS A **NIVEL ALTO**: la salida activa es **1** y la no activa **0**.
- ACTIVAS A **NIVEL BAJO**: la salida activa es **0** y la no activa **1**.

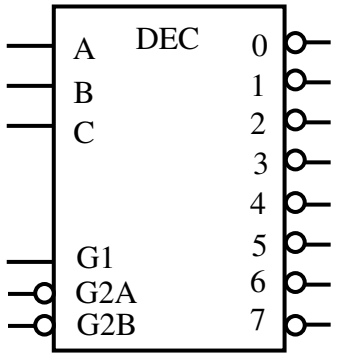
Además el número de entradas de habilitación puede ser de una o más, y pueden estar activas a nivel alto o bajo.
Podemos encontrar decodificadores de muy diversos "tamaños":

- **De 2 a 4 líneas**
- **De 3 a 8 líneas (bin a oct)**
- **De 4 a 16 líneas (bin a hex)**
- **Convertidores de códigos: BCD/decimal y BCD/7-seg**

Ejemplo de Decodificador completo de 3 a 8 líneas: CIRCUITO 74138

Símbolos Lógicos del Decodificador, según el Standard IEEE y tradicional





Decodificadores (III)

Tabla de funcionamiento del decodificador 74138 y encapsulado DIP y tiempos de respuesta según las diferentes subtecnologías.

FUNCTION TABLE

INPUTS						OUTPUTS							
ENABLE			SELECT										
G1	$\overline{G2A}$	$\overline{G2B}$	C	B	A	Y0	Y1	Y2	Y3	Y4	Y5	Y6	Y7
X	H	X	X	X	X	H	H	H	H	H	H	H	H
X	X	H	X	X	X	H	H	H	H	H	H	H	H
L	X	X	X	X	X	H	H	H	H	H	H	H	H
H	L	L	L	L	L	L	H	H	H	H	H	H	H
H	L	L	L	L	H	H	L	H	H	H	H	H	H
H	L	L	L	H	L	H	H	L	H	H	H	H	H
H	L	L	L	H	H	H	H	H	L	H	H	H	H
H	L	L	H	L	L	H	H	H	H	L	H	H	H
H	L	L	H	L	H	H	H	H	H	H	L	H	H
H	L	L	H	H	L	H	H	H	H	H	H	L	H
H	L	L	H	H	H	H	H	H	H	H	H	H	L

SWITCHING CHARACTERISTICS

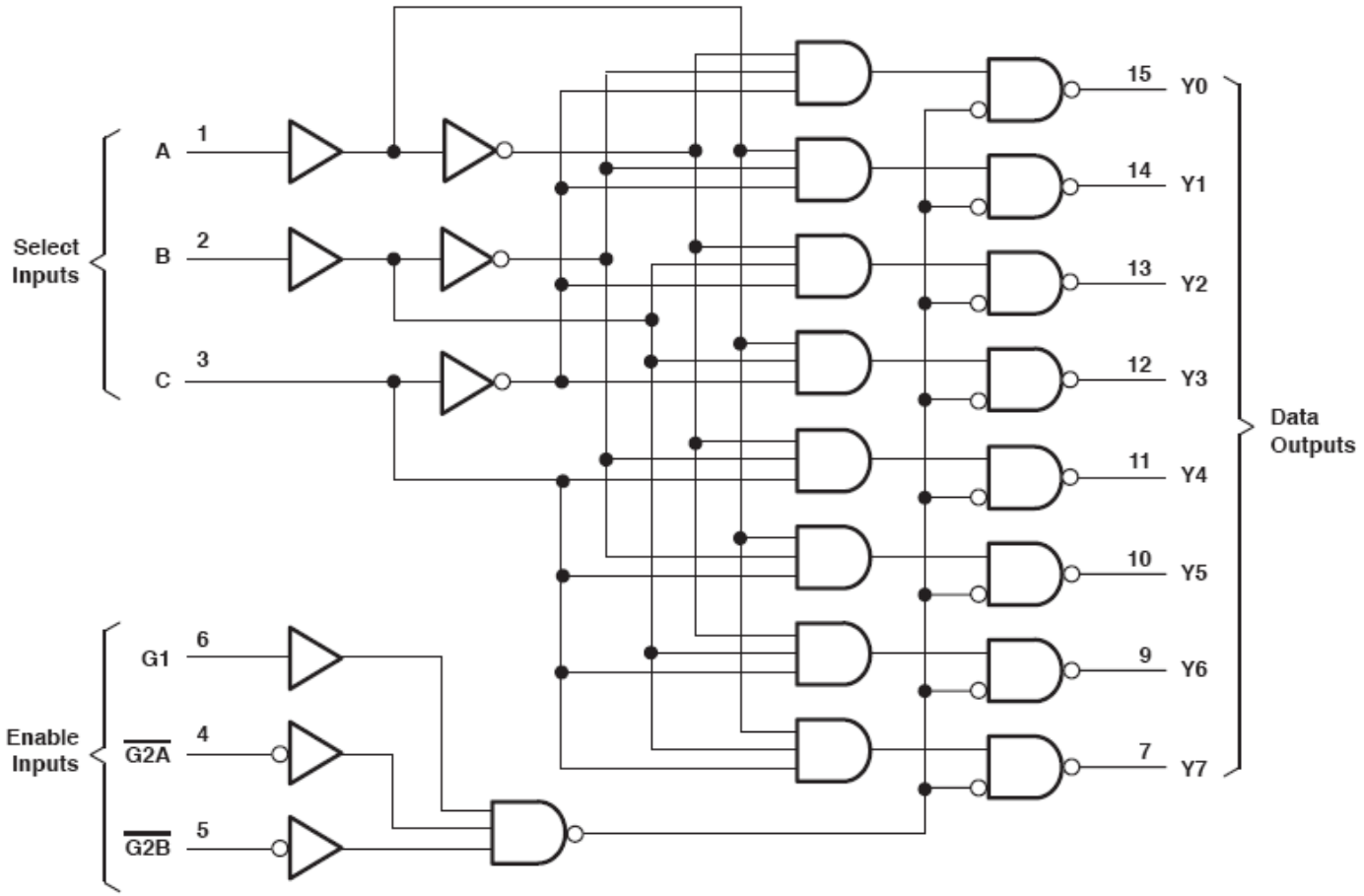
PARAMETER	INPUT	OUTPUT	MAX or MIN	LS	S	ALS	AS	F	SN74 HC	CD74 HC	SN74 HCT	CD74 HCT	AC 11
t_{PLH}	A, B, C	Y ($CD74:\overline{Y}$)	MAX	27	12	22	10	8.5	45	45	45	53	8.1
t_{PHL}			MAX	39	12	18	9.5	9	45	45	45	53	8.8
t_{PLH}	$\overline{G2}$	Y ($CD74:\overline{Y}$)	MAX	26	11	17	7.5	8	39	53	42	53	8.3
t_{PHL}			MAX	38	11	17	8.5	7.5	39	53	42	53	8.3
t_{PLH}	G1	Y ($CD74:\overline{Y}$)	MAX	26	11	17	10	9	39	53	42	53	7.5
t_{PHL}			MAX	38	11	17	10	8.5	39	53	42	53	7.7

PARAMETER	INPUT	OUTPUT	MAX or MIN	CD74 AC	ACT 11	CD74 ACT	AHC	AHCT	LV 3V	LV 5V	LVC 3V
t_{PLH}	A, B, C	Y ($CD74:\overline{Y}$)	MAX	11	9.8	12	11.5	13	18	11.5	6.7
t_{PHL}			MAX	11	9.7	12	11.5	13	18	11.5	6.7
t_{PLH}	$\overline{G2}$	Y ($CD74:\overline{Y}$)	MAX	10	8.9	10.5	11.5	12	18	11.5	6.5
t_{PHL}			MAX	10	8.9	10.5	11.5	12	18	11.5	6.5
t_{PLH}	G1	Y ($CD74:\overline{Y}$)	MAX	11	9.3	11	11.5	11.5	18.5	11.5	5.8
t_{PHL}			MAX	11	9.8	11	11.5	11.5	18.5	11.5	5.8

UNIT: ns

Decodificadores (IV)

Diagrama lógico del decodificador 74138.

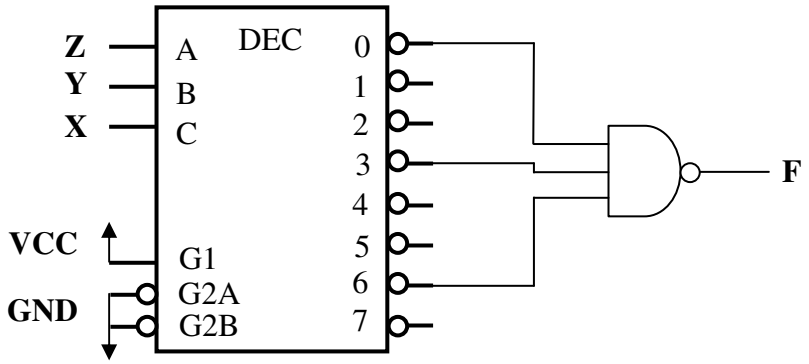


Realización de funciones con decodificadores

Un circuito decodificador completo genera todos los productos fundamentales (mintérminos) de las variables de entrada. Basta sumar (con puertas OR los términos que pertenecen a la función para obtener su implementación cuando las salidas del decodificador son activas a **nivel alto**.

Cuando las salidas del decodificador son activas a **nivel bajo**, para realizar la función en suma de productos basta con conectar las salidas correspondientes a los mintérminos de la función usando puertas NAND:

Por ejemplo: **$F(X,Y,Z) = \Sigma_3(0, 3, 6)$**



Asociación de decodificadores (I)

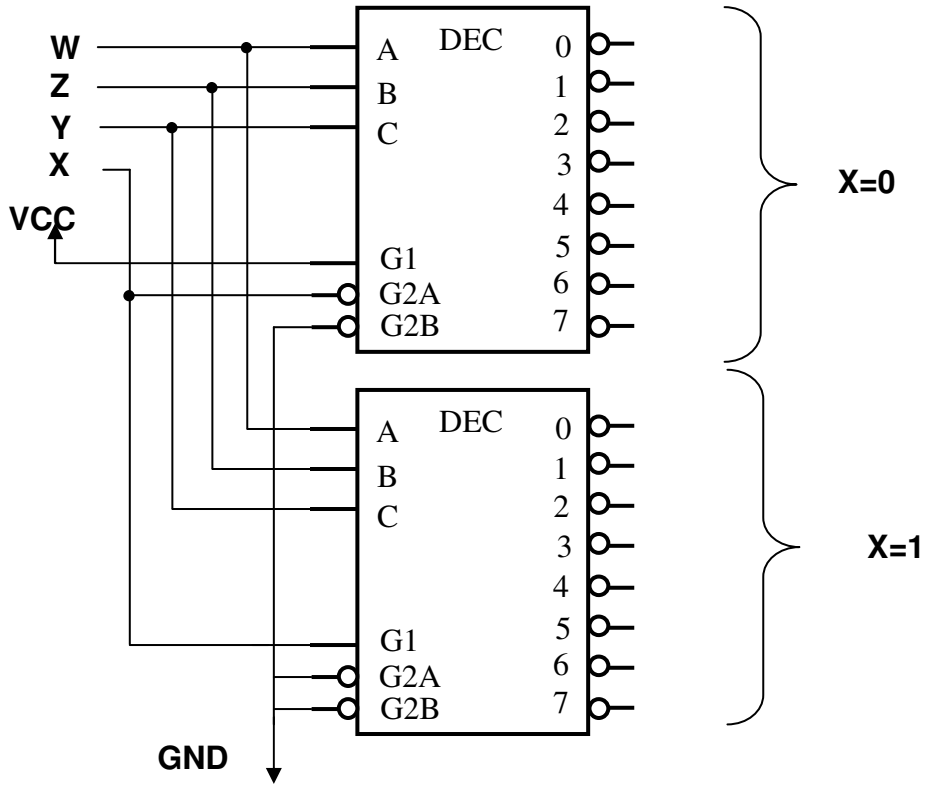
A veces puede ocurrir que necesitemos decodificar más líneas de las que nos permite nuestro circuito. Se debe entonces construir un decodificador de mayor tamaño usando los decodificadores disponibles. Para realizar esta asociación se divide el número de salidas necesarias entre las salidas de las que disponen los decodificadores que podemos usar. Esto nos proporciona el número de decodificadores que dan las salidas.

Después es necesario que sólo uno de estos decodificadores actúe en cada momento cuando el valor binario de la entrada de datos pertenezca al rango de salida que proporciona ese decodificador. Esto podemos conseguirlo con puertas lógicas o con otro/s decodificador/es.

Asociación de decodificadores (II)

Ejemplo:

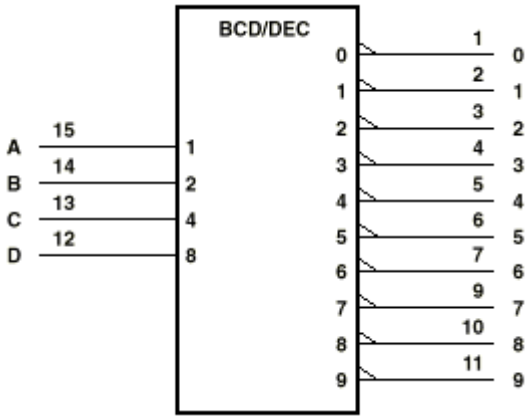
Realizar un decodificador de 4 a 16 líneas usando decodificadores 74138.



Convertidores de códigos (I)

Son circuitos Codificadores/Decodificadores que convierten los datos de un código a otro.

Ejemplo: El circuito 7442 es un convertidor de BCD a Decimal (de 4 a 10 líneas). Su símbolo lógico y tabla de funcionamiento se muestran a continuación.

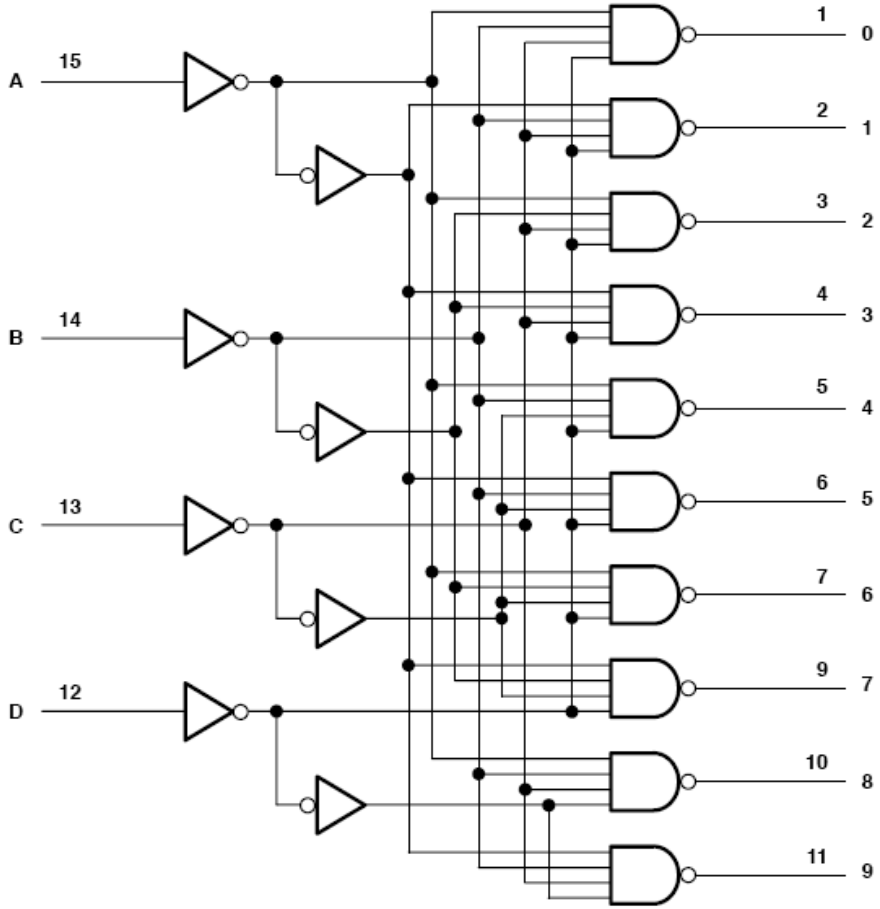


FUNCTION TABLE

NO.	INPUTS				OUTPUTS									
	D	C	B	A	0	1	2	3	4	5	6	7	8	9
0	L	L	L	L	L	H	H	H	H	H	H	H	H	H
1	L	L	L	H	H	L	H	H	H	H	H	H	H	H
2	L	L	H	L	H	H	L	H	H	H	H	H	H	H
3	L	L	H	H	H	H	H	L	H	H	H	H	H	H
4	L	H	L	L	H	H	H	H	L	H	H	H	H	H
5	L	H	L	H	H	H	H	H	H	L	H	H	H	H
6	L	H	H	L	H	H	H	H	H	H	L	H	H	H
7	L	H	H	H	H	H	H	H	H	H	H	L	H	H
8	H	L	L	L	H	H	H	H	H	H	H	H	L	H
9	H	L	L	H	H	H	H	H	H	H	H	H	H	L
Invalid	H	L	H	L	H	H	H	H	H	H	H	H	H	H
	H	L	H	H	H	H	H	H	H	H	H	H	H	H
	H	H	L	L	H	H	H	H	H	H	H	H	H	H
	H	H	L	H	H	H	H	H	H	H	H	H	H	H
	H	H	H	L	H	H	H	H	H	H	H	H	H	H
	H	H	H	H	H	H	H	H	H	H	H	H	H	H

Convertidores de códigos (II)

Diagrama lógico del decodificador de 4 a 10 líneas 7442

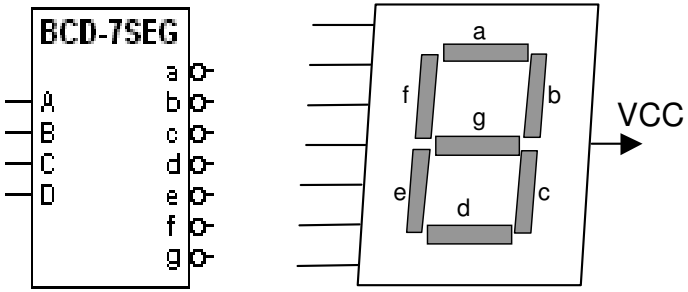


Decodificadores especiales: BCD/7 segmentos (I)

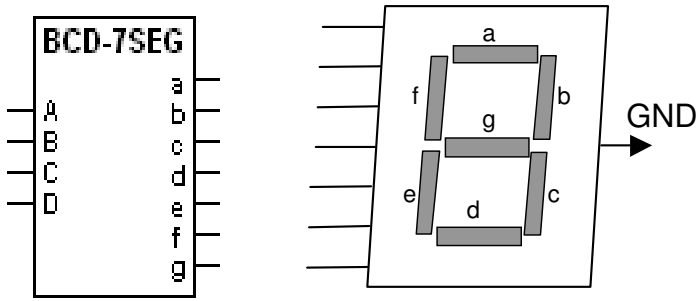
Estos decodificadores se usan para convertir un código BCD natural y representarlo en un visualizador de 7 segmentos. Son circuitos de tipo "driver".

Los visualizadores están formados por siete LEDs (Light Emitter Diode), que son dispositivos que emiten luz cuando la corriente que los atraviesa excede de un cierto valor (al igual que un diodo normal conduce cuando se supera una determinada tensión entre sus bornes), de ahí que para su excitación se necesiten dispositivos que proporcionen corriente suficiente (drivers). Existen dos tipos de visualizadores con LEDs principalmente: los de **ánodo común** y los de **cátodo común**. Los primeros se usan cuando el decodificador tiene salidas activas a nivel bajo, mientras que los segundos son para los decodificadores con salidas activas a nivel alto.

Otros visualizadores muy usados son los LCD (Liquid Crystal Display) que no están formados por diodos sino por una serie de plaquitas conductoras capaces de excitar un líquido que hay entre ellas.



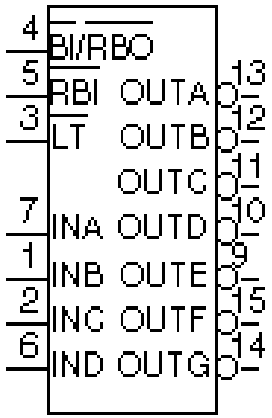
ÁNODO COMÚN



CÁTODO COMÚN

Decodificadores especiales: BCD/7 segmentos (II)

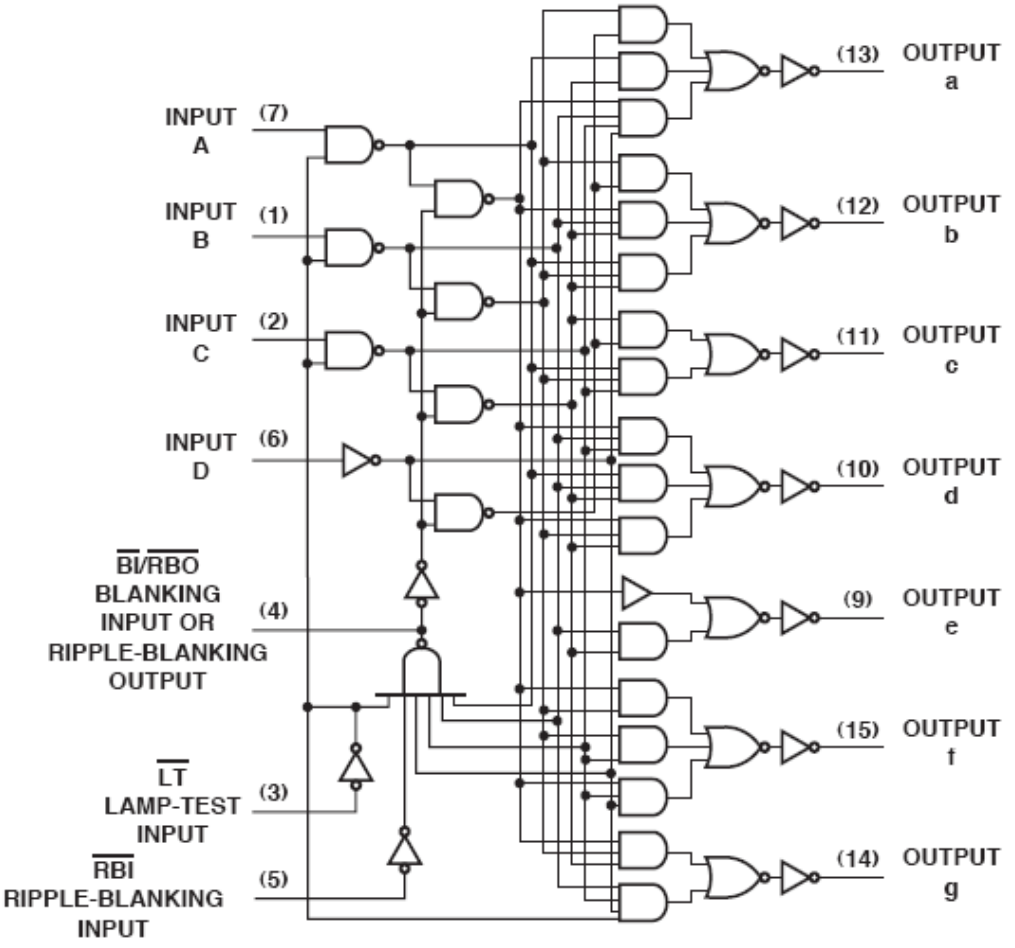
Ejemplo de decodificador BCD/7 segmentos: 7447



SALIDAS
a visualizador de 7 segmentos
ánodo común
(conexión mediante R=150Ω)

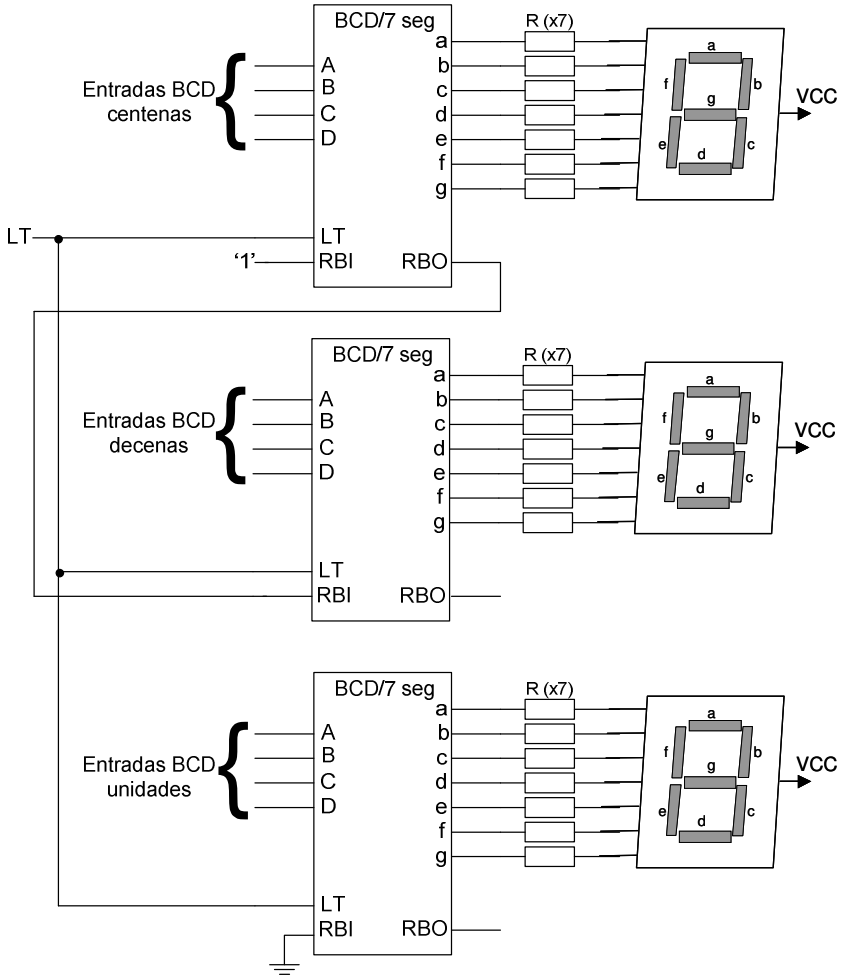
FUNCTION TABLE

No.	INPUTS						BI/RBO	OUTPUTS						
	LT	RBI	D	C	B	A		a	b	c	d	e	f	g
0	H	H	L	L	L	L	H	ON	ON	ON	ON	ON	ON	OFF
1	H	X	L	L	L	H	H	OFF	ON	ON	OFF	OFF	OFF	OFF
2	H	X	L	L	H	L	H	ON	ON	OFF	ON	ON	OFF	ON
3	H	X	L	L	H	H	H	ON	ON	ON	ON	OFF	OFF	ON
4	H	X	L	H	L	L	H	OFF	ON	ON	OFF	OFF	ON	ON
5	H	X	L	H	L	H	H	ON	OFF	ON	ON	OFF	ON	ON
6	H	X	L	H	H	L	H	OFF	OFF	ON	ON	ON	ON	ON
7	H	X	L	H	H	H	H	ON	ON	ON	OFF	OFF	OFF	OFF
8	H	X	H	L	L	L	H	ON	ON	ON	ON	ON	ON	ON
9	H	X	H	L	L	H	H	ON	ON	ON	OFF	OFF	ON	ON
10	H	X	H	L	H	L	H	OFF	OFF	OFF	ON	ON	OFF	ON
11	H	X	H	L	H	H	H	OFF	OFF	ON	ON	OFF	OFF	ON
12	H	X	H	H	L	L	H	OFF	ON	OFF	OFF	OFF	ON	ON
13	H	X	H	H	L	H	H	ON	OFF	OFF	ON	OFF	ON	ON
14	H	X	H	H	H	L	H	OFF	OFF	OFF	ON	ON	ON	ON
15	H	X	H	H	H	H	H	OFF	OFF	OFF	OFF	OFF	OFF	OFF
BI	X	X	X	X	X	X	L	OFF	OFF	OFF	OFF	OFF	OFF	OFF
RBI	H	L	L	L	L	L	L	OFF	OFF	OFF	OFF	OFF	OFF	OFF
LT	L	X	X	X	X	X	H	ON	ON	ON	ON	ON	ON	ON



Decodificadores especiales: BCD/7 segmentos (III)

Conexión de varios decodificadores BCD/7 segmentos con las líneas RBI y RB0

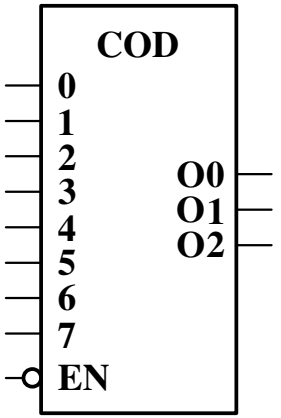


Codificadores (I)

Son los dispositivos MSI que realizan la operación inversa a la realizada por los decodificadores. Generalmente, poseen $\mathbf{2^n}$ entradas y **n** salidas. Cuando solo una de las entradas está activa para cada combinación de salida, se le denomina **codificador completo**.

Por ejemplo, el siguiente circuito proporciona a la salida la combinación binaria de la entrada que se encuentra activada. En este caso se trata de un codificador completo de **8** bits, o también llamado codificador de 8 a 3 líneas:

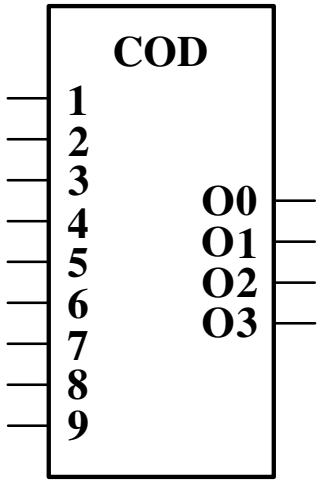
EN	I0	I1	I2	I3	I4	I5	I6	I7	O2	O1	O0
1	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**0**	**0**	**0**
0	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
0	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**
0	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**
0	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**1**	**1**
0	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**1**	**0**	**0**
0	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**1**	**0**	**1**
0	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**1**	**1**	**0**
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**1**	**1**



Codificadores (II)

En la siguiente figura se representa el diagrama lógico de un codificador completo de Decimal a BCD natural, junto a su tabla de funcionamiento.

I1	I2	I3	I4	I5	I6	I7	I8	I9	03	O2	01	O0
0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	1
0	1	0	0	0	0	0	0	0	0	0	1	0
0	0	1	0	0	0	0	0	0	0	0	1	1
0	0	0	1	0	0	0	0	0	0	1	0	0
0	0	0	0	1	0	0	0	0	0	1	0	1
0	0	0	0	0	1	0	0	0	0	1	1	0
0	0	0	0	0	0	1	0	0	0	1	1	1
0	0	0	0	0	0	0	1	0	1	0	0	0
0	0	0	0	0	0	0	0	1	1	0	0	1

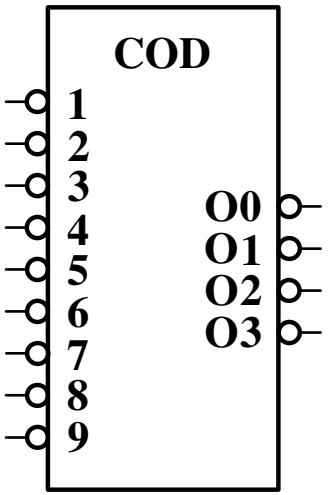


Las salidas codificadas generalmente se usan para controlar un conjunto de dispositivos, suponiendo claro está que sólo uno de ellos está activo en cualquier momento. Sin embargo cuando nos encontremos con que se deben controlar dispositivos que pueden estar activos al mismo tiempo, un problema que se suele encontrar en los sistemas microprocesadores, es preciso usar un dispositivo que nos proporcione a la salida el código del dispositivo que tenga más alta prioridad.

Codificadores (III)

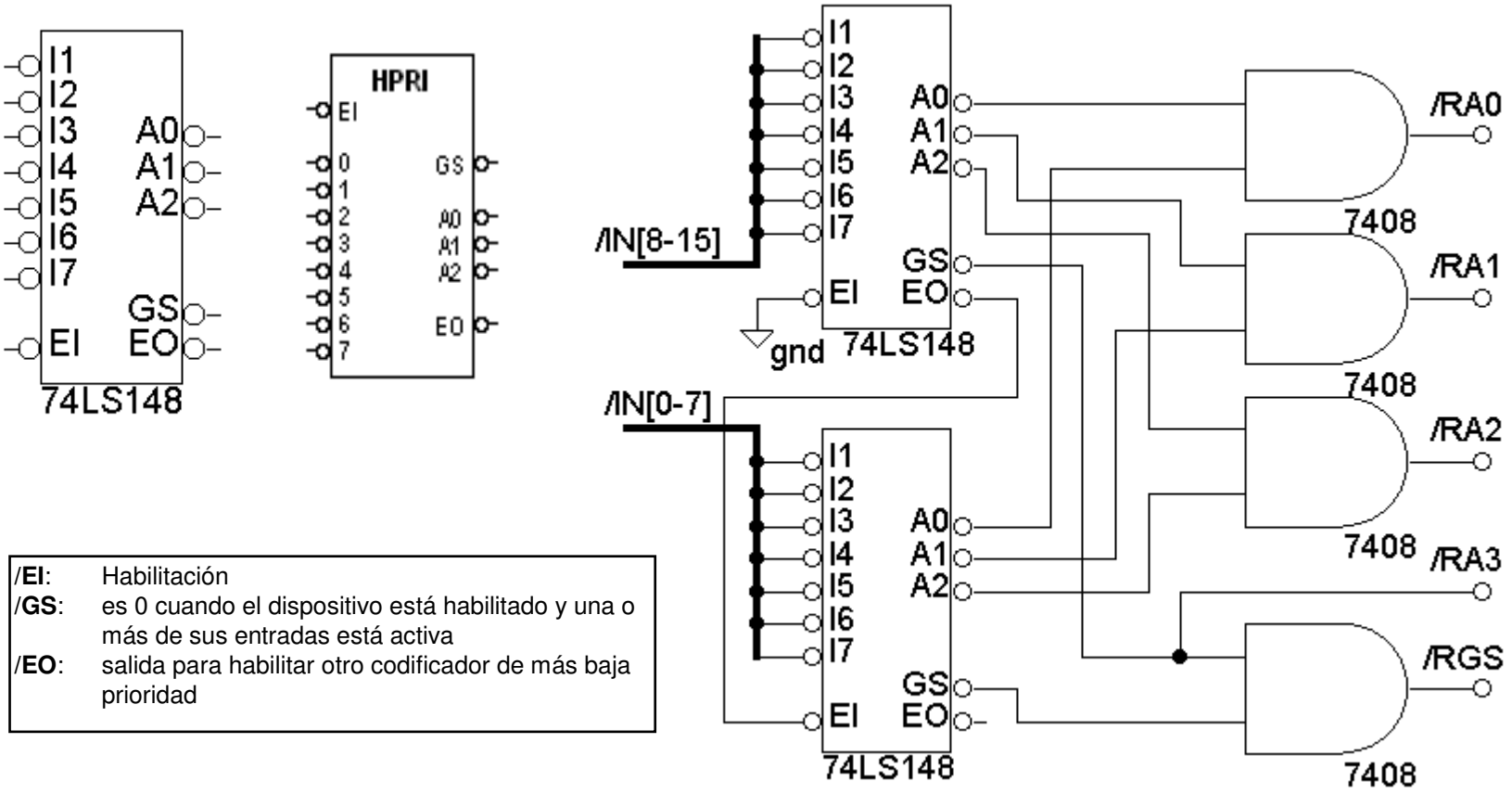
La figura siguiente representa el diagrama lógico del circuito 74147, que es un codificador de **prioridad** de Decimal a BCD natural; en la tabla de funcionamiento adjunta se puede notar la diferencia con el anterior ya que este codificador da prioridad a la entrada de más alto valor que esté activa en un instante dado.

I1	I2	I3	I4	I5	I6	I7	I8	I9	O3	O2	O1	O0
X	X	X	X	X	X	X	X	0	0	1	1	0
X	X	X	X	X	X	X	0	1	0	1	1	1
X	X	X	X	X	X	0	1	1	1	0	0	0
X	X	X	X	X	0	1	1	1	1	0	0	1
X	X	X	X	0	1	1	1	1	1	0	1	0
X	X	X	0	1	1	1	1	1	1	0	1	1
X	X	0	1	1	1	1	1	1	1	1	0	0
X	0	1	1	1	1	1	1	1	1	1	0	1
0	1	1	1	1	1	1	1	1	1	1	1	0
1	1	1	1	1	1	1	1	1	1	1	1	1



Codificadores (y IV)

Cuando se trata de establecer la prioridad con mayor número de bits, es preciso recurrir a la asociación de codificadores. El siguiente diagrama muestra un codificador de prioridad de 16 líneas a 4, usando codificadores de prioridad 74148, de 8 a 3 líneas.



/**EI**:	Habilitación
/**GS**:	es 0 cuando el dispositivo está habilitado y una o más de sus entradas está activa
/**EO**:	salida para habilitar otro codificador de más baja prioridad

Multiplexores (I)

El multiplexado es un proceso consistente en recibir mensajes de diferentes fuentes y enviarlas a un destino común. A la inversa, la técnica de demultiplexado permite enviar a varios puntos de destino diversos datos que proceden de una fuente común. A los multiplexores se les suele llamar también selectores de datos.

Un **MULTIPLEXOR (MUX)** es un circuito combinacional que selecciona una entrada y la transfiere a la salida. La selección de la entrada, o dato, se realiza mediante un conjunto de líneas de selección.

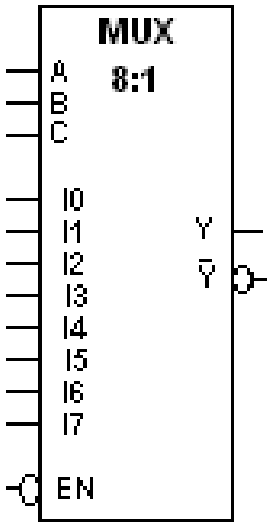
Poseen por tanto, **n** entradas de selección para $\mathbf{2^n}$ entradas de datos, proporcionando generalmente **dos salidas**: una para el dato directo y otra para el dato negado.

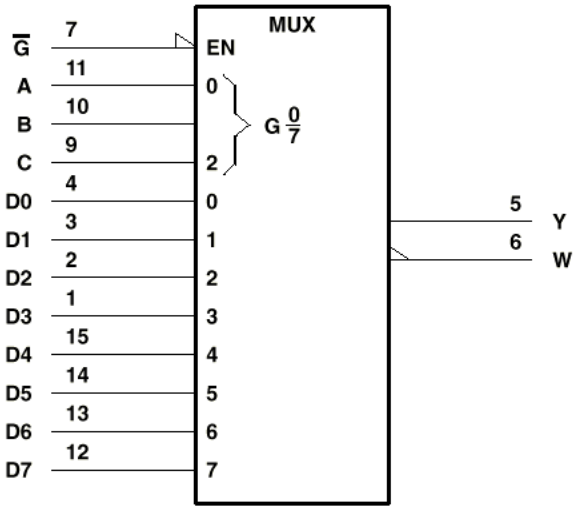
A continuación se presenta la tabla de funcionamiento y los símbolos clásicos e IEEE para un multiplexor de 8 a 1 líneas. Se trata del circuito 74151, con entrada de habilitación activa a nivel bajo.

FUNCTION TABLE

INPUTS				OUTPUTS	
SELECT			STROBE $\overline{G}$		
C	B	A		Y	W
X	X	X	H	L	H
L	L	L	L	D0	$\overline{D0}$
L	L	H	L	D1	$\overline{D1}$
L	H	L	L	D2	$\overline{D2}$
L	H	H	L	D3	$\overline{D3}$
H	L	L	L	D4	$\overline{D4}$
H	L	H	L	D5	$\overline{D5}$
H	H	L	L	D6	$\overline{D6}$
H	H	H	L	D7	$\overline{D7}$

H = high level, L = low level, X = irrelevant
D0, D1, . . . D7 = the level of the respective D input





Multiplexores (II)

Existen en el mercado multiplexores de diferentes tamaños en circuito integrado. Los tamaños más usuales en un mismo encapsulado son:

- **Un multiplexor de 8 canales (74151)**
- **Un multiplexor de 16 canales (Por ejemplo, el 74150)**
- **Dos multiplexores de 4 canales con líneas de selección comunes (74153)**
- **4 multiplexores de dos canales con línea de selección común (74157) (El 74158 es una versión del primero con las salidas activas a nivel bajo)**

Veremos a continuación estos circuitos, su tabla de funcionamiento y su diagrama lógico.

.

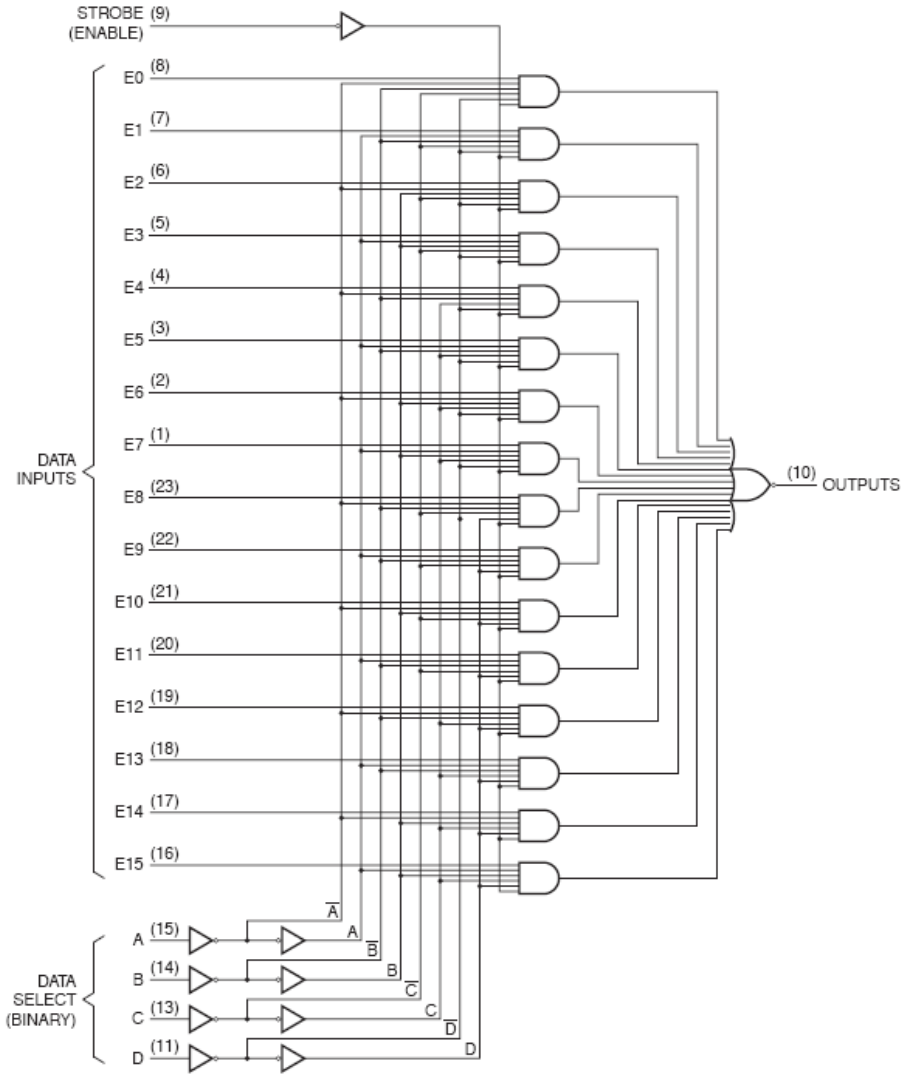
Multiplexores (III)

Multiplexor de 16 canales (74150)

FUNCTION TABLE

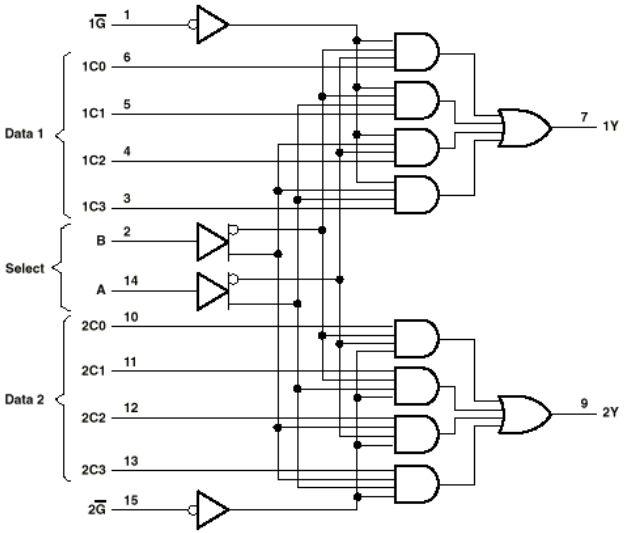
INPUTS					OUTPUT
SELECT				STROBE	W
D	C	B	A	$\overline{G}$	
X	X	X	X	H	H
L	L	L	L	L	$\overline{E0}$
L	L	L	H	L	$\overline{E1}$
L	L	H	L	L	$\overline{E2}$
L	L	H	H	L	$\overline{E3}$
L	H	L	L	L	$\overline{E4}$
L	H	L	H	L	$\overline{E5}$
L	H	H	L	L	$\overline{E6}$
L	H	H	H	L	$\overline{E7}$
H	L	L	L	L	$\overline{E8}$
H	L	L	H	L	$\overline{E9}$
H	L	H	L	L	$\overline{E10}$
H	L	H	H	L	$\overline{E11}$
H	H	L	L	L	$\overline{E12}$
H	H	L	H	L	$\overline{E13}$
H	H	H	L	L	$\overline{E14}$
H	H	H	H	L	$\overline{E15}$

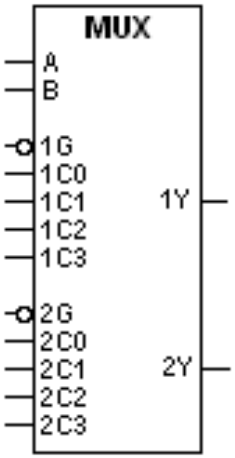
NOTES:
H = High Level, L = Low Level, X = irrelevant
$\overline{E0}$, $\overline{E1}$ ... $\overline{E15}$ = the complement of the level of the respective E input
D0, D1 ... D7 = the level of the D respective input

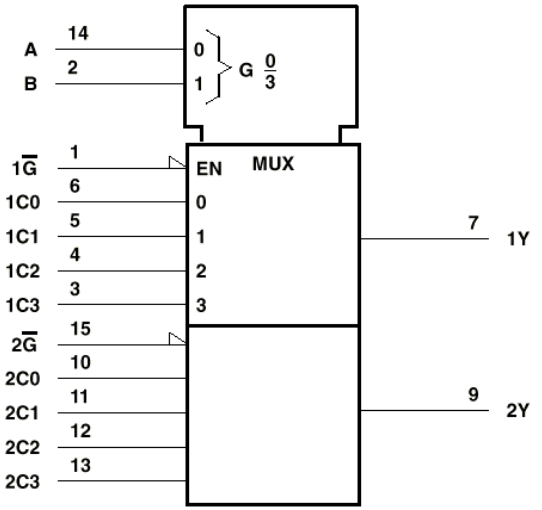


Multiplexores (IV)

Multiplexor doble de 4 canales (74153)







FUNCTION TABLE

INPUTS						STROBE $\overline{G}$	OUTPUT Y
SELECT		DATA					
B	A	C0	C1	C2	C3		
X	X	X	X	X	X	H	L
L	L	L	X	X	X	L	L
L	L	H	X	X	X	L	H
L	H	X	L	X	X	L	L
L	H	X	H	X	X	L	H
H	L	X	X	L	X	L	L
H	L	X	X	H	X	L	H
H	H	X	X	X	L	L	L
H	H	X	X	X	H	L	H

Select inputs A and B are common to both sections.

Multiplexores (V)

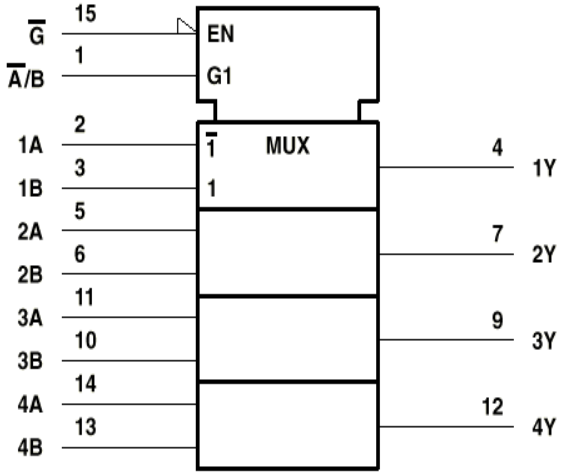
Multiplexor cuádruple de 2 canales (74157)

FUNCTION TABLE

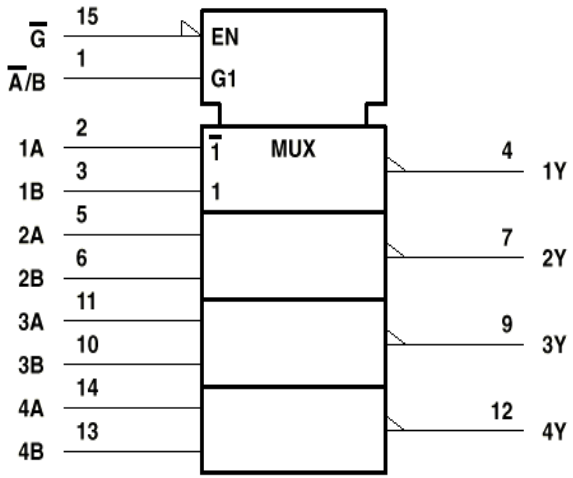
INPUTS				OUTPUT Y	
$\overline{G}$	$\overline{A}$/B	DATA A	DATA B	'ALS157A SN74AS157	'ALS158 SN74AS158
H	X	X	X	L	H
L	L	L	X	L	H
L	L	H	X	H	L
L	H	X	L	L	H
L	H	X	H	H	L

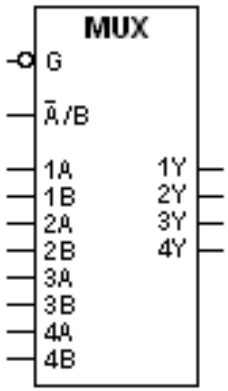
Este multiplexor funciona como selector de palabras: según sea el valor de selección G1, en las cuatro salidas aparece A[1-4] ó B[1-4], lo que lo hace muy útil cuando necesitemos seleccionar una entre dos combinaciones binarias.

'ALS157A, SN74AS157



'ALS158, SN74AS158



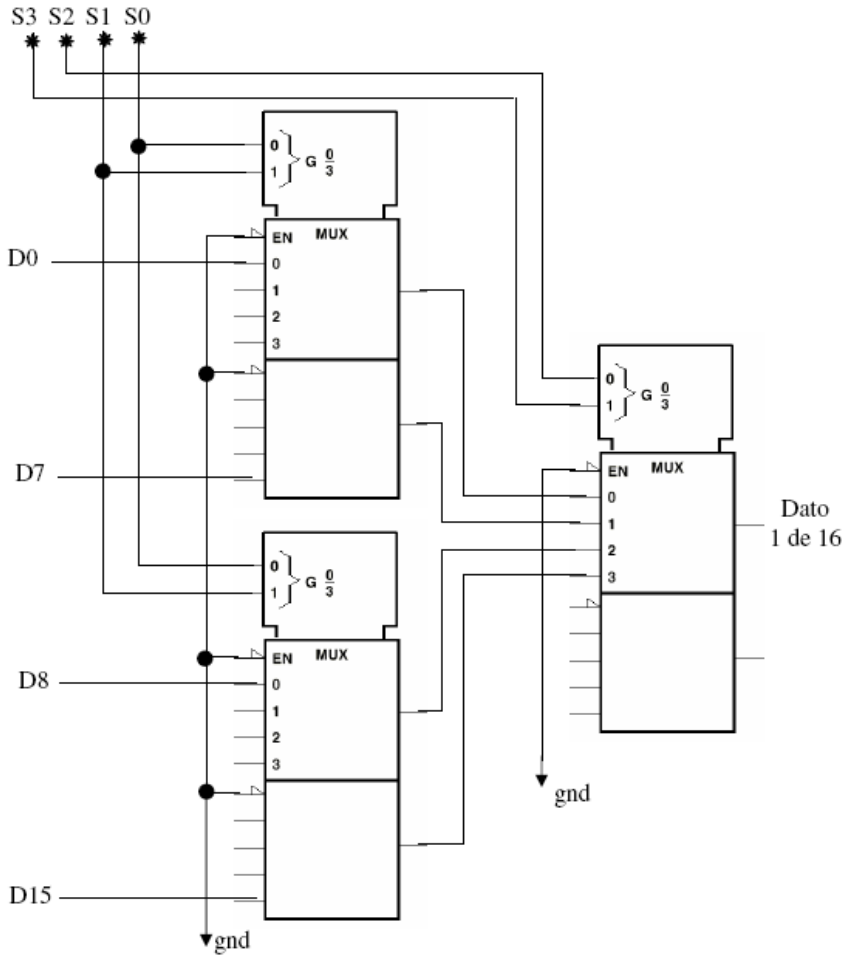


Asociación de multiplexores

Como siempre, cuando no se dispone del número necesario de entradas en un solo dispositivo, hay que recurrir a asociar dispositivos de menor número de entradas para obtener uno mayor.

El ejemplo muestra como construir un MUX de 16:1 usando MUX de 4:1.

La asociación es muy similar a la que se hace con los decodificadores, pero cambian la disposición de los MSB y LSB (bits más significativos y menos significativos).



Realización de funciones lógicas con multiplexores (I)

Mediante un multiplexor de 2^n entradas de datos (n entradas de selección) se puede realizar cualquier función lógica de n+1 variables

Se pueden usar dos métodos para implementar funciones con multiplexores:

- de forma algebraica;
- de forma tabular.

Nos centraremos en este último, y lo seguiremos mediante un ejemplo.

Ejemplo:

Realizar la función **$f(D,C,B,A)= \Sigma_4(0,2,3,7,8,13,15)$** mediante un multiplexor del tamaño más adecuado.

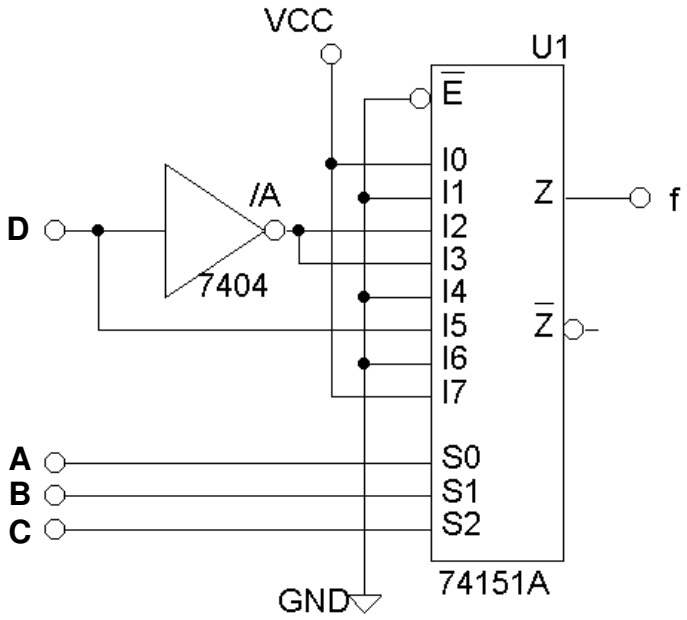
Al ser una función de 4 variables necesitamos un MUX de 8 a 1 líneas (es decir, con tres variables de control).

Se crea un mapa parecido a los de Karnaugh, pero con las siguientes diferencias:

- **Se colocan todas las variables menos una en la parte superior, y la variable que nos queda (que será la de mayor peso, MSB) a la izquierda de la tabla, señalando las filas.**
- **No es necesario emplear código reflejado. Es más fácil emplear código binario natural.**
- **Por seguir un orden, pondremos la variable de mayor peso (MSB) señalando las filas de la tabla.**
- **El resto de variables se ordenan convenientemente para que la numeración sea correlativa (0, 1, 2, 3, …)**
- **En la tabla, cada columna representa una entrada de datos del multiplexor. Así, las columnas vendrán determinadas por las variables de control del MUX. Las variables de control deben ser las de menor peso. Evaluando cada columna identificamos el valor que hay que colocar en cada entrada de datos del multiplexor.**
- **Dentro de la tabla, colocaremos aquellos términos que pertenecen a la función, señalándolos con un 1. Si la función tuviera condiciones no importa, también las colocaríamos, señalándolas con una 'X'.**
- **Examinaremos la tabla columna a columna, y señalaremos bajo cada una lo siguiente:**
  - **- Un '1' cuando las dos casillas de una columna contengan un '1', o bien un '1' y una 'X'.**
  - **- Un '0' cuando las dos casillas de una columna contengan un '0', o bien un '0' y una 'X'.**
  - **- La variable de mayor peso directa si encontramos un '1' en la casilla que señala a esa variable sin negar y un '0' en la otra.**
  - **- La variable de mayor peso negada si encontramos un '1' en la casilla que señala a esa variable negada y un '0' en la otra.**

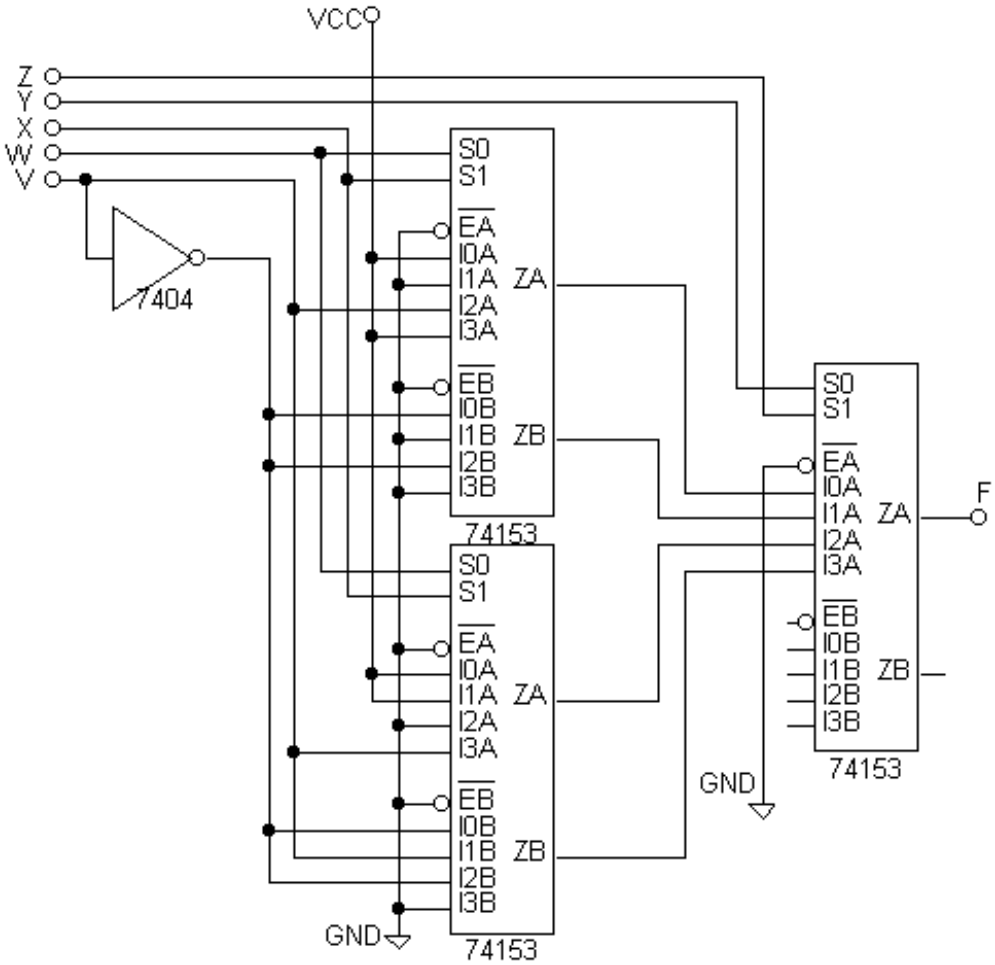
Realización de funciones lógicas con multiplexores (II)

CBA	I0 000	I1 001	I2 010	I3 011	I4 100	I5 101	I6 110	I7 111
D 0	1 (0)	0 (1)	1 (2)	1 (3)	0 (4)	0 (5)	0 (6)	1 (7)
1	1 (8)	0 (9)	0 (10)	0 (11)	0 (12)	1 (13)	0 (14)	1 (15)
	1	0	D'	D'	0	D	0	1



Realización de funciones lógicas con multiplexores (III)

Ejercicio: Analizar el circuito de la figura y averiguar cual es la función booleana que implementa.



Demultiplexores

En realidad no existen como tales, sino que vienen definidos como decodificadores/demultiplexores (decoder/demultiplexer).

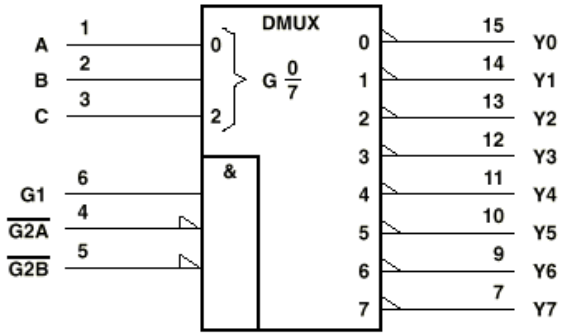
La función que debe realizar es la **inversa** de la que realiza el multiplexor, es decir, debemos seleccionar por cual de las salidas vamos a transmitir el dato de la entrada.

Por tanto, **el demultiplexor constará de *1* entrada de datos, *n* entradas de selección de salida, y 2^n salidas.**

El Decodificador/DEMUX 74138 que ya conocemos puede utilizar su entrada de habilitación **G1** para entrada de datos, en este caso las salidas se obtienen negadas. Si usamos **G2A'** o **G2B'** como entrada de datos, las salidas se obtienen de forma directa.

FUNCTION TABLE

INPUTS						OUTPUTS							
ENABLE			SELECT										
G1	$\overline{G2A}$	$\overline{G2B}$	C	B	A	Y0	Y1	Y2	Y3	Y4	Y5	Y6	Y7
X	H	X	X	X	X	H	H	H	H	H	H	H	H
X	X	H	X	X	X	H	H	H	H	H	H	H	H
L	X	X	X	X	X	H	H	H	H	H	H	H	H
H	L	L	L	L	L	L	H	H	H	H	H	H	H
H	L	L	L	L	H	H	L	H	H	H	H	H	H
H	L	L	L	H	L	H	H	L	H	H	H	H	H
H	L	L	L	H	H	H	H	H	L	H	H	H	H
H	L	L	H	L	L	H	H	H	H	L	H	H	H
H	L	L	H	L	H	H	H	H	H	H	L	H	H
H	L	L	H	H	L	H	H	H	H	H	H	L	H
H	L	L	H	H	H	H	H	H	H	H	H	H	L

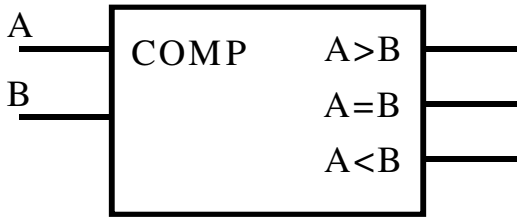


Comparadores de magnitud (I)

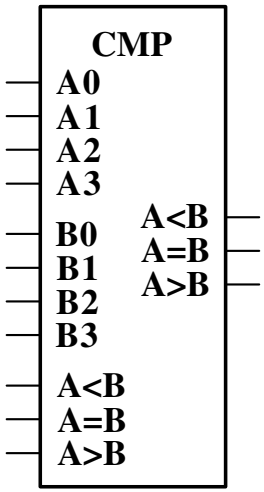
Son circuitos que comparan el valor binario de dos números, proporcionando información de cuál es mayor, menor, o si ambos son iguales.

Son sistemas muy usados en ingeniería. Su bloque y tabla de funcionamiento básico son los siguientes:

A	B	A>B	A=B	A<B
0	0	0	1	0
0	1	0	0	1
1	0	1	0	0
1	1	0	1	0



Existen comparadores de 4 bits y de 8 bits. Además de las correspondientes entradas de datos disponen de tres entradas más que pueden informar sobre una situación anterior, y que se usan para conectar en cascada distintos comparadores, de manera que pueda construirse uno de mayor capacidad.

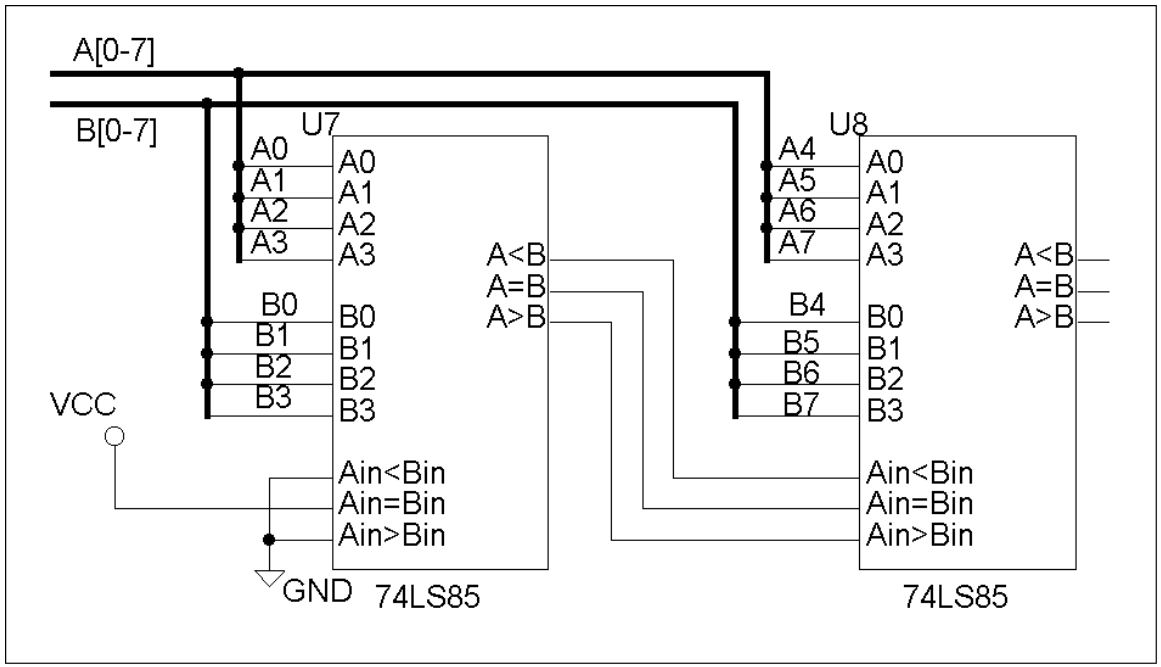


Comparadores de magnitud (II)

Las salidas del comparador se pueden deducir de las siguientes expresiones lógicas:

(A>B) = (Nº A > Nº B) or [(Nº A = Nº B) and (Ain>Bin)]
(A=B) = (Nº A = Nº B) and (Ain=Bin)
(A<B) = (Nº A < Nº B) or [(Nº A = Nº B) and (Ain<Bin)]

Haciendo uso de esas entradas de "comparación anteriores", podemos diseñar un comparador mayor, por ejemplo de 8 bits.



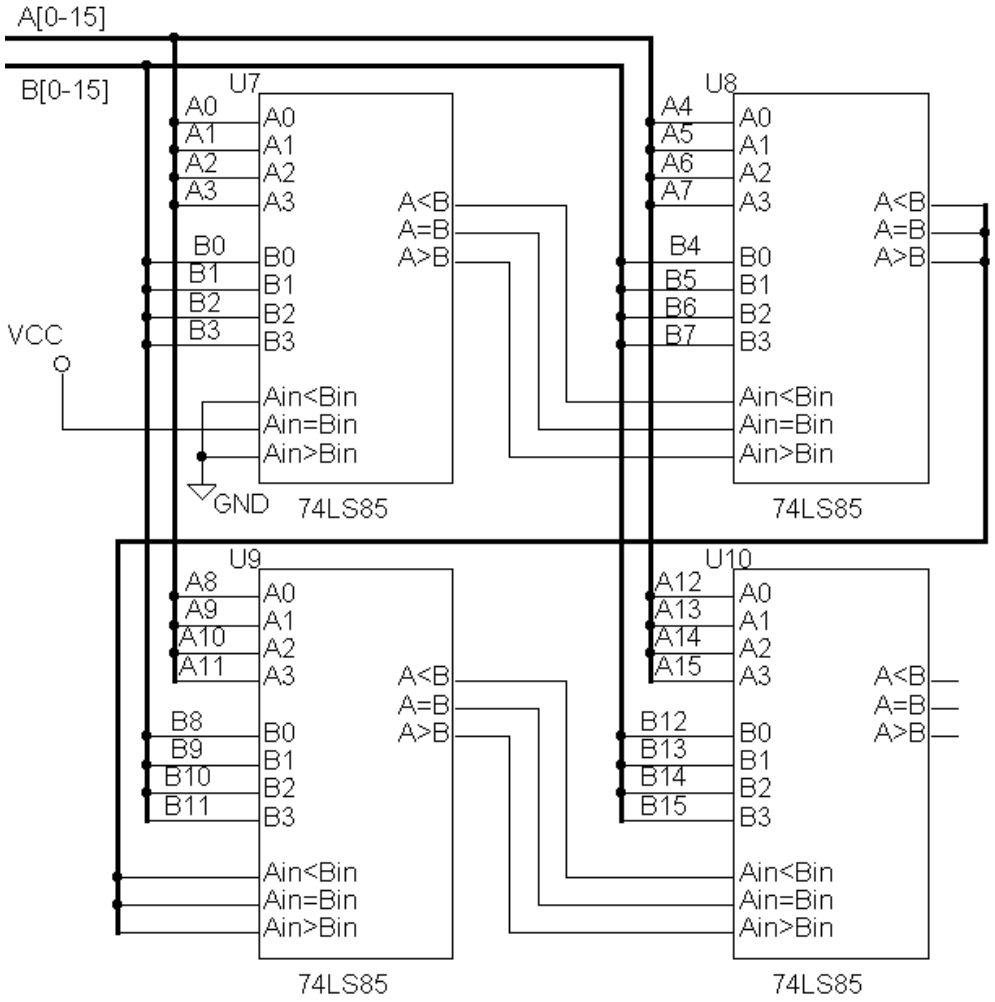
Así se comparan primero los bits menos significativos A[0-3] y B[0-3], para con la información obtenida comparar los más significativos; por ejemplo:

A: 00101100
B: 00100100

La comparación de los 4 bits menos significativos (1100 y 0100) activará la salida **A>B** del comparador de la izquierda y según las ecuaciones anteriores, al cumplirse que **(nº A=nº B)** (0010 y 0010), la salida que debe activarse en el segundo comparador es A>B. Es fácil hacer esta comprobación con todas las combinaciones posibles.

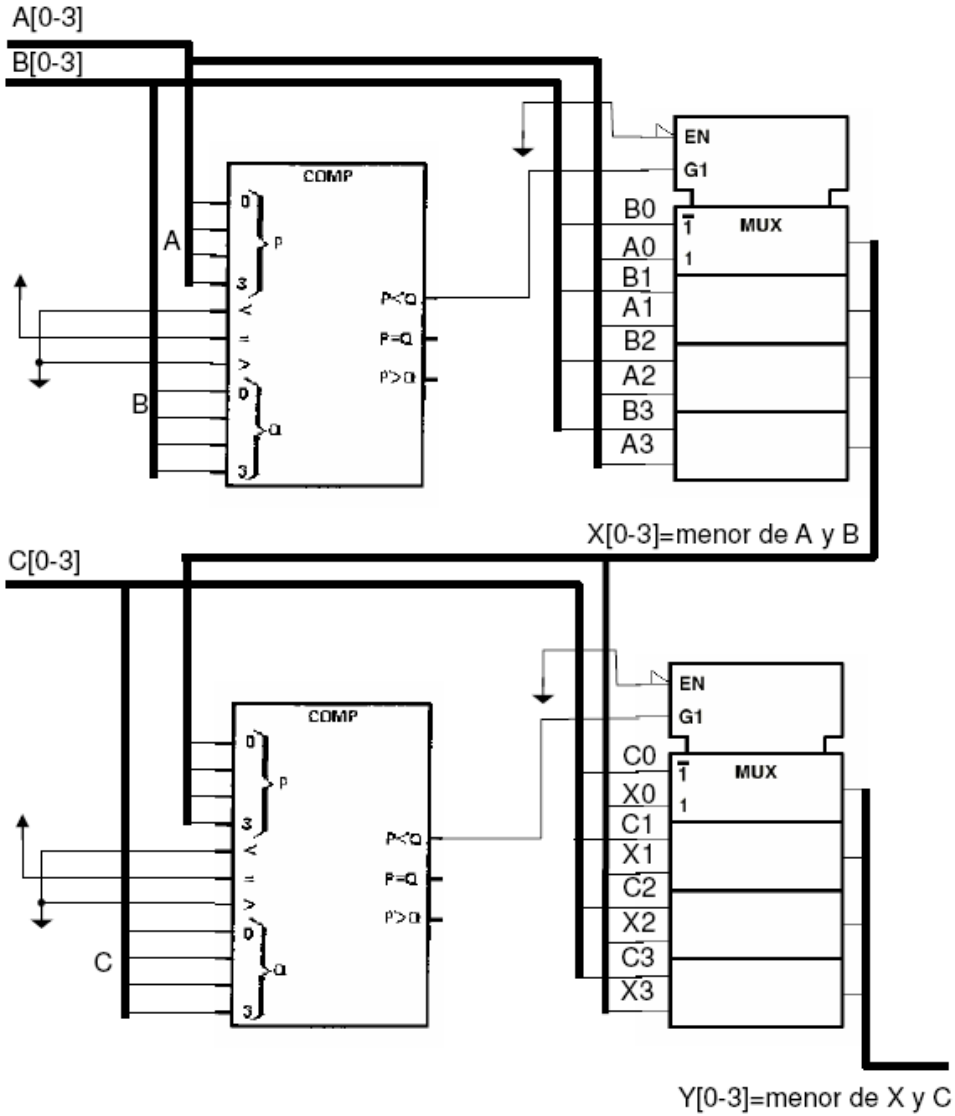
Comparadores de magnitud (III)

Ejemplo de construcción de un comparador de números de 16 bits bits a partir de comparadores de 4 bits.



Comparadores (y IV)

El circuito mostrado es una aplicación completa de comparadores, en la que se comparan tres números codificados en BCD (palabras de 4 bits) y se escoge el menor. Para ello se comparan dos números y se toma el menor y éste se compara con el tercero; se usa para "transmitir" el número seleccionado un MUX cuádruple de 2 a 1 líneas (se ha usado simbología según el estándar de IEEE).



Detectores/Generadores de Paridad

Son circuitos MSI que detectan si en la entrada hay un número par o impar de "unos", o sea, detectan la paridad de una palabra digital.

Se basan en la función XOR. Su aplicación principal se basa en la transmisión y detección de códigos en las comunicaciones digitales. Un tipo de código muy usado en las transmisiones digitales es aquel en que a una palabra digital se le añade un bit que indique la paridad de la palabra. Cuando nuestro circuito genere el bit de paridad, funcionará como transmisor, y cuando tenga que detectarlo, funcionará como receptor.

Supongamos que vamos a trasmitir la palabra de 7 bits [**1011110**] con paridad par, el bit que debemos añadir debe ser un 1, para que el total de unos sea par:

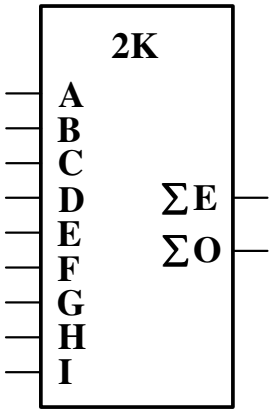
- *Como Transmisor:* ***1011110 1*** *nº de unos: 6*

En el receptor recibimos una palabra de 8 bits [**10111101**] detectamos su paridad y si es par (como ocurre en este caso), admitimos la palabra como correcta.

El circuito de la figura corresponde al 74280.

FUNCTION TABLE

NO. OF INPUTS A–I THAT ARE HIGH	OUTPUTS	
	Σ EVEN	Σ ODD
0, 2, 4, 6, 8	H	L
1, 3, 5, 7, 9	L	H



VHDL para descripción, simulación y diseño de circuitos digitales

En un principio, el diseño a partir de esquemáticos fue la forma habitual de diseño CAD (apoyado y almacenado en un computador). Pero, hoy en día, ha sido sustituida (casi por completo) por su descripción funcional en texto (programa que detalla el funcionamiento de las diversas partes del circuito y la conexión entre ellas), utilizando para ello un lenguaje de descripción de *hardware* (HDL). La forma textual presenta numerosas ventajas:

- **suele requerir menor tiempo y esfuerzo para comprender lo que el circuito hace (en el caso de sistemas complejos);**
- **es independiente de la implementación a bajo nivel (en puertas, bloques, biestables y registros);**
- **es directamente trasladable a los diversos dispositivos programables y a las diversas librerías de ASIC...**
- **y, sobre todo, resulta mucho más sencillo revisar e introducir modificaciones en el texto descriptor que en su esquema gráfico.**

Además, una misma descripción funcional puede configurarse circuitalmente de formas muy variadas, y tal configuración dependerá de la librería de celdas estándar disponibles (para el diseño de un ASIC) o, en su caso, del dispositivo programable en que se inserte; en principio no es necesario conocer su configuración a nivel booleano para utilizar eficientemente un circuito digital. Lo que sí es necesario es disponer de una adecuada y detallada descripción funcional.

Una vez descrito un circuito mediante VHDL, basta con "compilar" la descripción sobre una librería de recursos, librería que dependerá del circuito físico con el que implementar el diseño funcional expresado en VHDL. Iremos viendo a lo largo de la asignatura algunos de ellos, tales como los CPLD y las FPGA.

VHDL (que de forma más correcta correspondería a *Versatile Hardware Description Language*) es una excelente herramienta de documentación, simulación, validación y diseño de sistemas digitales, estandarizada y en permanente proceso de actualización bajo los auspicios del IEEE *(Institute of Electrical and Electronics Engineers)*. El único lenguaje alternativo que goza, también, de amplia aceptación es **Verilog**, pero su difusión en el contexto europeo es mucho menor (aunque, en buena medida, Verilog es un lenguaje más directo y más cercano al propio circuito digital).

VHDL básico para diseñar circuitos combinacionales (I)

Nociones básicas

Usaremos en esta asignatura el lenguaje VHDL para describir sistemas digitales, aunque el lenguaje permite la descripción de sistemas de cualquier tipo.

- **VHDL no distingue entre mayúsculas y minúsculas**
- **Los comentarios en los programas empiezan por - -**
- **Cada módulo descriptivo y cada asignación de valor a una señal termina con punto y coma (;)**
- **Los elementos básicos de la descripción digital son las señales (signal)**
- **VHDL es un lenguaje fuertemente tipado. Cada tipo de señal se define con anterioridad y no admite asignaciones a tipos incorrectos**
- **En las señales, sus valores se expresan siempre entre comillas. Suele emplearse el tipo std_logic (standard logic), que admite los siguientes nueve valores:**

'0','1'	**Valores booleanos típicos**
'X'	**Desconocido**
'Z'	**Alta impedancia**
'U'	**Sin inicializar (por ejemplo un biestable en su situación previa)**
'-'	**Condición No Importa (don't care)**
'L'	**'0' débil**
'H'	**'1' débil**
'W'	**desconocido débil**

VHDL básico para diseñar circuitos combinacionales (II)

- **Los valores '0', '1' y 'X' son de tipo fuerte, si se encuentran dos de ellos aplicados en un nodo, el resultado es 'X' (desconocido).**
- **Los valores débiles corresponden a determinadas situaciones circuitales, de manera que si confluyen con un valor fuerte, dan como resultado dicho valor fuerte.**
- **Si se encuentran dos valores débiles, el resultado es 'W' (desconocido débil).**
- **Un conjunto de señales constituye un vector. Este puede declararse en forma ascendente o descendente:**
  - ***Ascendente***: Ejemplo: **std_logic_vector (0 to 7)**
  - ***Descendente***: Ejemplo: **std_logic_vector (7 downto 0)**. Esta forma se utiliza mucho porque el bit más significativo se corresponde con el de mayor índice.
- **Los tipos *standard logic* se introdujeron en la normalización hecha por IEEE. Para poder usarlos, hay que incluir en los programas la declaración de la librería (y de los paquetes) que definen tales tipos y sus operaciones:**

```
library ieee;
use ieee.std_logic.all;
use ieee.std_logic_unsigned.all;
```

- **Con el paquete ieee.std_logic_unsigned las operaciones se realizan en binario natural. Si queremos trabajar con números negativos en complemento a 2, hay que usar el paquete ieee.std_logic_signed.**
- **El paquete ieee.std_logic_arith incluye dos funciones muy útiles:**
  - **CONV_INTEGER (a)** convierte el ***std_logic_vector*** *a* en **integer.**
  - **CONV_STD_LOGIC_VECTOR (b, n)** convierte el **integer** *b* en vector de longitud *n*

VHDL básico para diseñar circuitos combinacionales (III)

Las **operaciones básicas entre señales** se muestran en la siguiente tabla.

Asignación	***<=***
Operaciones booleanas	***and, or, not, xor, nand, nor***
Comparaciones	***=, /=, >, <, >=, <=***
Aritméticas	***+, -, ****
Concatenación	***&*** ***(la concatenación se refiere a colocar vectores juntos, formando un vector más largo)***

La **asignación de valores a las señales** puede hacerse directamente o por medio de operaciones entre señales.

Ejemplos:

```
signal a, b, c, F : std_logic_vector (3 downto 0);
signal m, n: integer range 0 to 15;
   a <= "1100";
   a <=(3 => '1', 0 => '0', others => '0');
   m <= 7;
   F <= ((not a) and b) or (a and (not b));    -- equivale a una operación xor
   F <= a + b; -- suma aritmética
```

VHDL básico para diseñar circuitos combinacionales (IV)

Estructura de una descripción en VHDL

En VHDL, las descripciones tienen tres partes bien diferenciadas:

- **Declaración de librerías y paquetes**: aquellas que vamos a usar en el diseño.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
```

- **Entidad (módulo de declaración de terminales)**: En esta parte se declaran los pines de entrada y salida del circuito que vamos a definir.

```
entity nombre_de_la_entidad is
  port( declaración de entradas y salidas
  );
end nombre_de_la_entidad;
```

- **Módulo de funciones** (descripción del funcionamiento del circuito):

```
arquitecture nombre_de_la_arquitectura of nombre_de_la_entidad is
  signal declaración de señales internas
  begin
        descripción del funcionamiento (asignaciones)
end nombre_de_la_arquitectura;
```

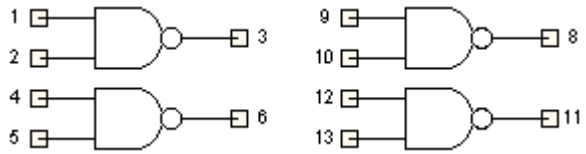
VHDL básico para diseñar circuitos combinacionales (V)

Ejemplo de una sencilla descripción en VHDL

Consideremos un circuito integrado sencillo como el 7400, que contiene cuatro puertas NAND. Se podría representar gráficamente la entidad con el chip y sus patillas, mientras que la arquitectura sería la función que es capaz de realizar el circuito:

```
entity cuatro_nand is
```


```
architecture puertas_nand is
```




```
library ieee;
use ieee.std_logic_1164.all;

entity cuatro_nand is
    port ( a1, b1, a2, b2, a3, b3, a4, b4: in std_logic;
           y1, y2, y3, y4: out std_logic);
end cuatro_nand;

architecture puertas_nand of cuatro_nand is
    begin
        y1 <= a1 nand b1;
        y2 <= a2 nand b2;
        y3 <= a3 nand b3;
        y4 <= a4 nand b4;
end puertas_nand;
```

```
library ieee;
use ieee.std_logic_1164.all;

entity cuatro_nand is
    port ( a, b: in std_logic_vector (1 to 4);
           y: out std_logic_vector (1 to 4) );
end cuatro_nand;

architecture puertas_nand of cuatro_nand is
    begin
        y <= a nand b;
end puertas_nand;
```

VHDL básico para diseñar circuitos combinacionales (VI)

Ejercicio

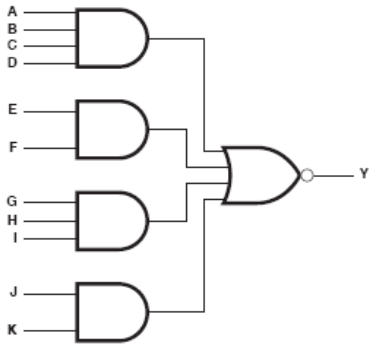
Modele en VHDL el circuito 7464, cuyo diagrama lógico y expresión algebraica se muestra a continuación.

64

4-2-3-2 INPUT AND-OR INVERT GATE

- $Y = \overline{ABCD + EF + GHI + JK}$

Logic Diagram




```
library ieee;
use ieee.std_logic_1164.all;

entity circuito_and_nor is
    port ( a, b, c, d, e, f, g, h, i, j, k: in std_logic;
           y: out std_logic);
end circuito_and_nor;

architecture descripcion_circuito of circuito_and_nor  is
    signal a0, a1, a2, a3: std_logic;
    begin
        a0 <= a and b and c and d;
        a1 <= e and f;
        a2 <= g and h and i;
        a3 <= j and k;
        y <= not (a0 or a1 or a2 or a3);
end descripcion_circuito;
```

Nota: En este ejemplo hemos empleado señales internas que nos facilitan la descripción del circuito: a0, a1, a2 y a3 son las salidas de cada una de las puertas AND

VHDL básico para diseñar circuitos combinacionales (VII)

Tipos de puertos

Los puertos (**port**) declarados en **entity** pueden ser:

- Entrada (**in**).
- Salida (**out**). No se pueden "leer" dentro del circuito.
- Bidireccionales (**inout**).
- Adaptados (**buffer**) Se utilizan menos que los anteriores.

Asignaciones concurrentes

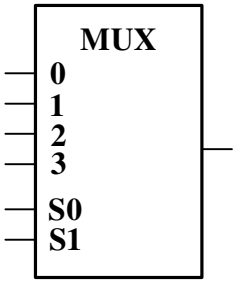
Son aquellas que se ejecutan siempre directamente sobre una señal. Una señal no puede recibir dos asignaciones concurrentes distintas (daría error al intentar asignar dos valores diferentes a la misma señal). Las asignaciones concurrentes pueden ser:

- Fijas: **señal <= valor_a_asignar;**
  señal <= operaciones_entre_señales, entre valores o entre ambos; *(los representamos por ------)*

- Condicionales:
```
señal <=    ------- when condicion_1 else
            ------- when condicion_2 else
            .
            .
            ------- when condicion_n else
            -------;
```

- Múltiples:
```
with -------- select
        señal <=    ------- when valor_1,
                    ------- when valor_2,
                    .
                    .
                    ------- when valor_n,
                    ------- when others;
```

VHDL básico para diseñar circuitos combinacionales (VIII)



Ejemplo de una descripción en VHDL usando asignaciones concurrentes

Describir un multiplexor de 4 canales usando:

- Asignaciones concurrentes condicionales
- Asignaciones concurrentes múltiples

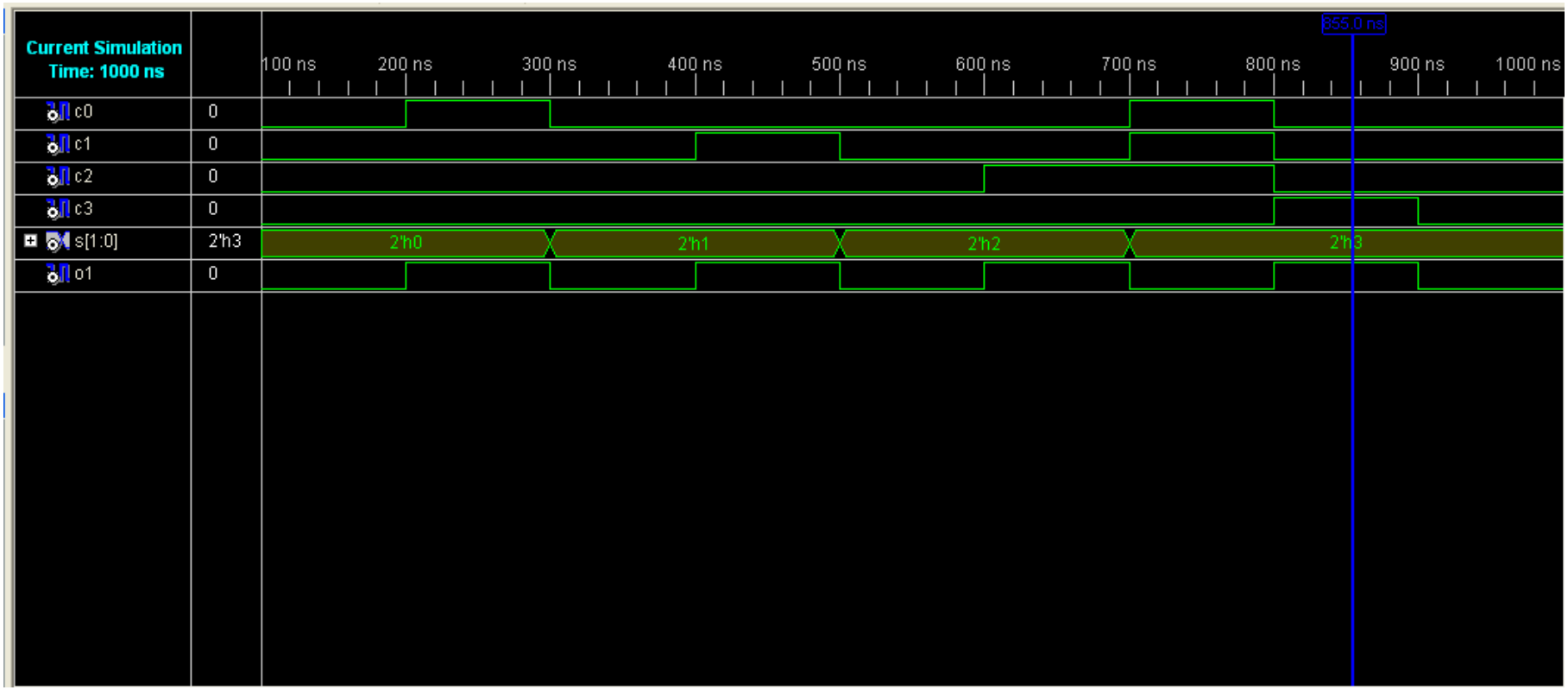
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity mux_4_a_1_T40_a is
    port ( c0, c1, c2, c3, s1, s0: in std_logic;
           s : out std_logic);
end mux_4_a_1_T40_a;

architecture mux_4_a_1 of mux_4_a_1_T40_a is
    signal control: std_logic_vector (1 downto 0);
    begin
        control <= s1 & s0;
        s <=    c0 when control = "00" else
                c1 when control = "01" else
                c2 when control = "10" else
                c3;
end mux_4_a_1;
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity mux_4_a_1_T40_b is
    port ( c0, c1, c2, c3: in std_logic;
           s: in std_logic_vector (1 downto 0);
           o1 : out std_logic);
end mux_4_a_1_T40_b;

architecture mux_4_a_1_b of mux_4_a_1_T40_b is
    signal control: integer range 0 to 3;
begin
    control <= conv_integer (s);
    with control select
        o1 <=  c0 when 0,
               c1 when 1,
               c2 when 2,
               c3 when others;
end mux_4_a_1_b;
```

VHDL básico para diseñar circuitos combinacionales (IX)

Simulación del multiplexor 4 a 1 a partir de la descripción VHDL

Esta simulación se ha realizado con el software *ISE Webpack* de Xilinx, que se usa también en las prácticas de la asignatura.



VHDL básico para diseñar circuitos combinacionales (X)

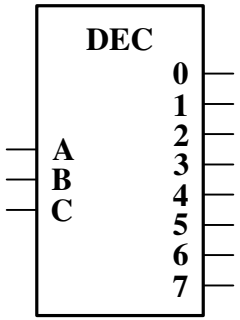
Ejemplo de descripción VHDL de un decodificador de 3 a 8 lineas

- Asignaciones concurrentes condicionales

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity dec3a8conc is
      port (a, b, c: in std_logic;
      salida: out std_logic_vector (0 to 7) );
end dec3a8conc;

architecture estruct_dec_1 of dec3a8conc is
      signal entrada: std_logic_vector (2 downto 0);

begin
      entrada <= c & b & a;
      salida <=      "10000000" when entrada = "000" else
                     "01000000" when entrada = "001" else
                     "00100000" when entrada = "010" else
                     "00010000" when entrada = "011" else
                     "00001000" when entrada = "100" else
                     "00000100" when entrada = "101" else
                     "00000010" when entrada = "110" else
                     "00000001";

end estruct_dec_1;
```

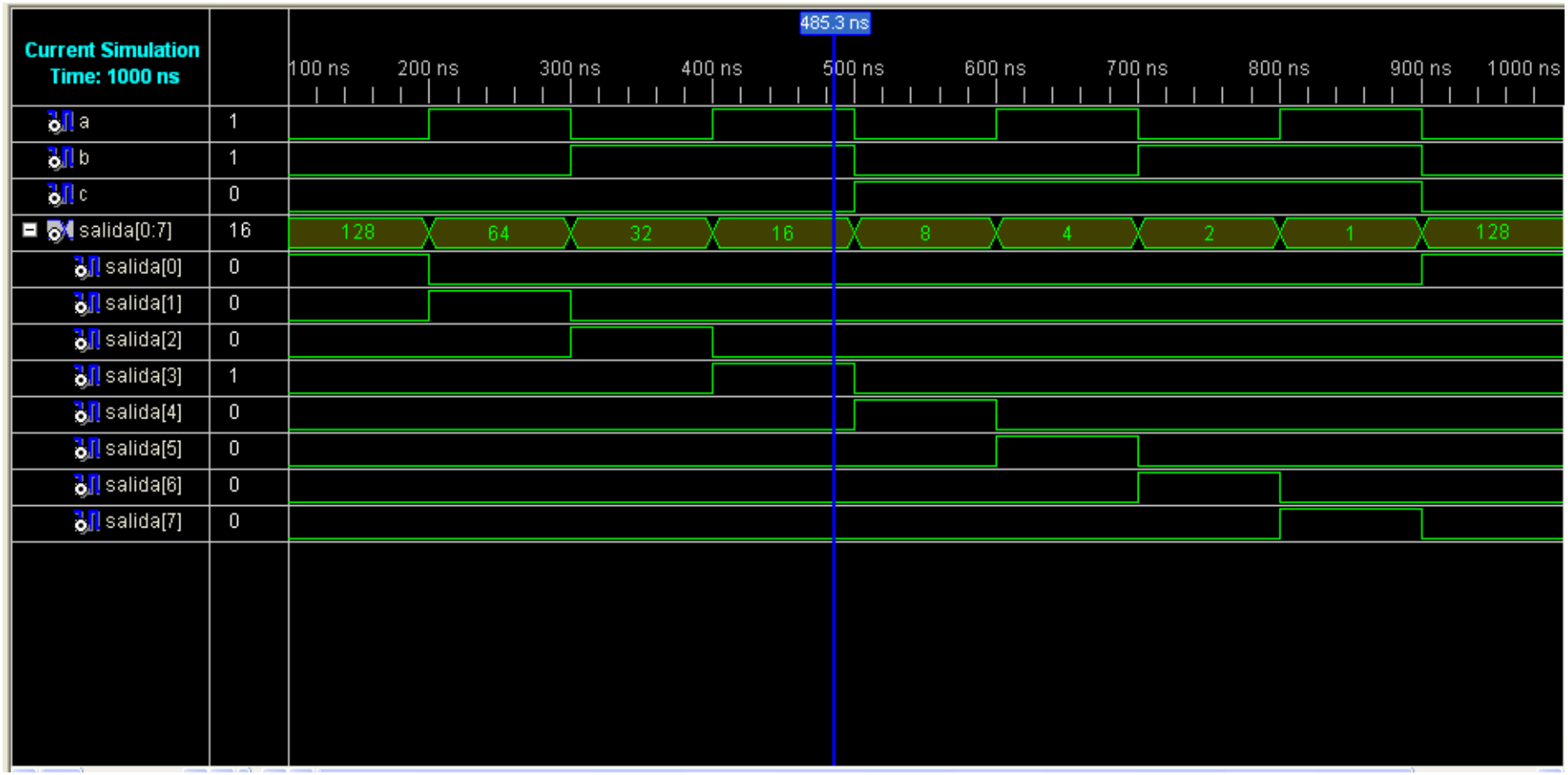
- Asignaciones concurrentes múltiples

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity dec3a8conc_b is
      port (a, b, c: in std_logic;
      salida: out std_logic_vector (0 to 7) );
end dec3a8conc_b;

architecture estruct_dec_2 of dec3a8conc_b is
      signal    entrada: std_logic_vector (2 downto 0);
      signal    entrada_int: integer range 0 to 7;
begin
      entrada <= c & b & a;
      entrada_int <= conv_integer (entrada);

      with entrada_int select
            salida <=      "10000000" when 0,
                           "01000000" when 1,
                           "00100000" when 2,
                           "00010000" when 3,
                           "00001000" when 4,
                           "00000100" when 5,
                           "00000010" when 6,
                           "00000001" when others;

end estruct_dec_2;
```



VHDL básico para diseñar circuitos combinacionales (XI)

Simulación del decodificador de 3 a 8 líneas

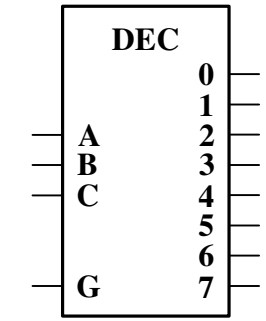


VHDL básico para diseñar circuitos combinacionales (XII)

Ejercicio (II)

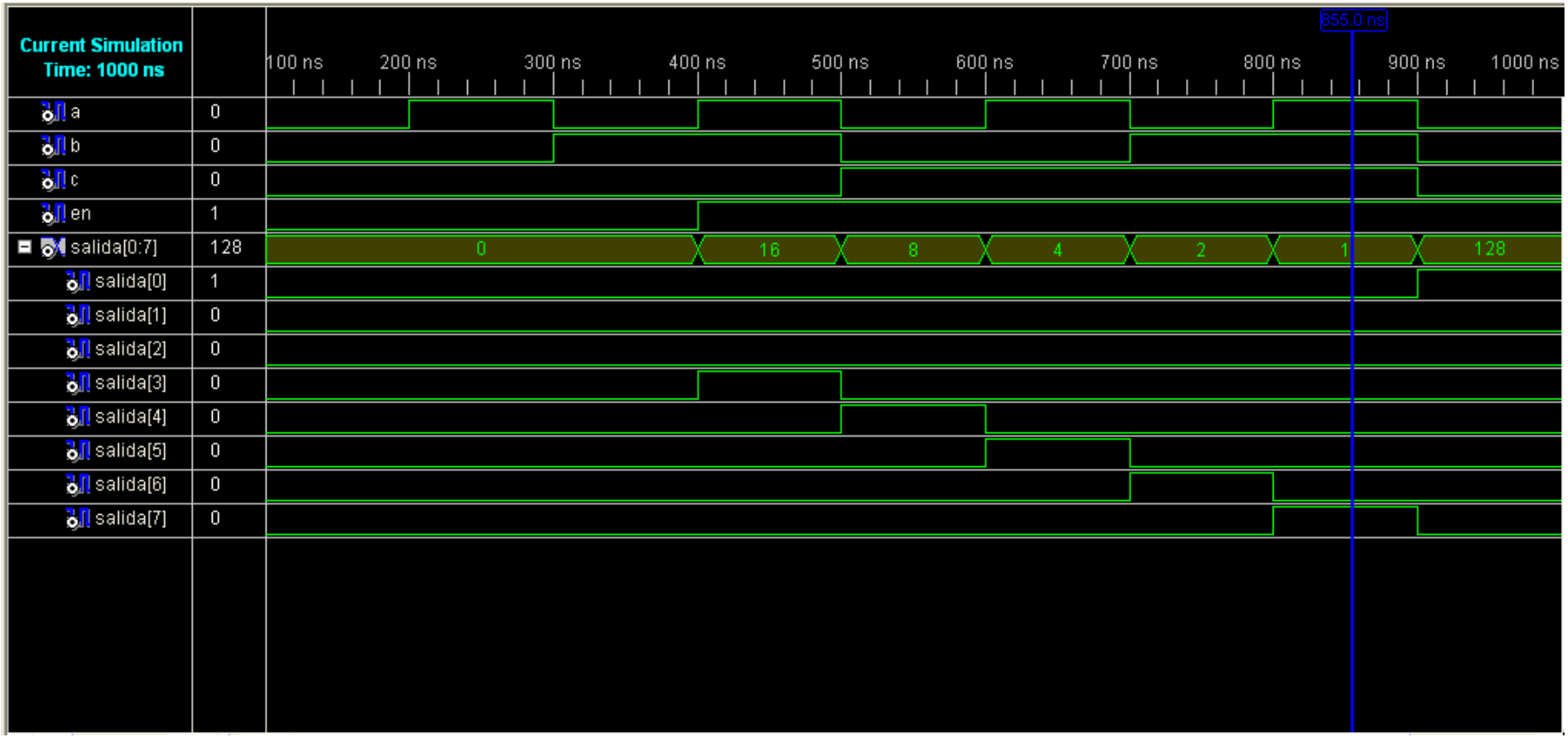
Modificar el ejemplo anterior para incluir en el decodificador una línea de enable (EN) activa a nivel alto.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity dec3a8_en is
   port (a, b, c, EN: in std_logic;
         salida: out std_logic_vector(0 to 7) );
end dec3a8_en;

architecture estruct_dec_en of dec3a8_en is
   signal   entrada:       std_logic_vector(2 downto 0);
   signal   entrada_int:   integer range 0 to 7;
   signal   salida_en:     std_logic_vector(0 to 7);
begin
   entrada <= c & b & a;
   entrada_int <= conv_integer(entrada);
   with entrada_int select
      salida_en <=   "10000000" when 0,
                     "01000000" when 1,
                     "00100000" when 2,
                     "00010000" when 3,
                     "00001000" when 4,
                     "00000100" when 5,
                     "00000010" when 6,
                     "00000001" when 7,
                     "00000000" when others;
      -- Salidas a nivel no activo si EN = 0 y funcionara cuando EN = 1
      salida <= salida_en when EN ='1' else "00000000";
end estruct_dec_en;
```



VHDL básico para diseñar circuitos combinacionales (XIII)

Simulación del decodificador de 3 a 8 líneas con entrada de habilitación (EN)



VHDL básico para diseñar circuitos combinacionales (XIV)

Asignaciones secuenciales (I)

Estas asignaciones se encuentran dentro de un módulo denominado **proceso** (process) y no se ejecutan hasta que no se ha terminado de leer todo el módulo. Si hay en un módulo más de una asignación a una señal, es válida únicamente la última de ellas.

En el caso de asignaciones condicionales (lo más usual), es válida la última asignación cuyas condiciones se cumplen. Los procesos se declaran de la siguiente forma:

```
nombre_del_proceso (opcional):   process (lista de sensibilidades)
                                 begin
                                        asignaciones
                                 end process
```

- La **lista de sensibilidades** se refiere a las señales que afectan al proceso, normalmente se colocan las entradas al proceso.
- Cada proceso en sí puede considerarse una asignación concurrente, por tanto no pueden realizarse asignaciones a una misma señal en dos procesos diferentes.
- Las **asignaciones secuenciales** pueden ser **fijas**, **condicionales** o **múltiples**. Las fijas ya las conocemos, veremos a continuación las condicionales y las múltiples.

Condicionales

```
if…… then          señal <= ----------; señal <= ----------;
    elsif …….. then  señal <= ----------; señal <= ----------;
    else             señal <= ----------; señal <= ----------;
end if;
```

VHDL básico para diseñar circuitos combinacionales (XV)

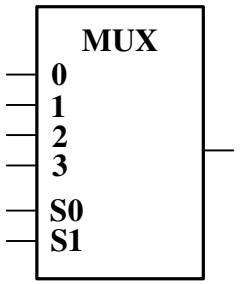
Asignaciones secuenciales (II)

Múltiples

```
case…… is
    when “”””””     =>     señal <= ----------; señal <= ----------;
    when “”””””     =>     señal <= ----------; señal <= ----------;
    when “”””””     =>     señal <= ----------; señal <= ----------;
    when others    =>     señal <= ----------; señal <= ----------;
end case;
```

IMPORTANTE: Dentro de un proceso no pueden utilizarse asignaciones con **when…else** o con **with…select**, y de igual forma, las estructuras **if** y **case** no pueden utilizarse fuera de los procesos.

VHDL básico para diseñar circuitos combinacionales (XVI)



Ejemplo de una descripción en VHDL usando asignaciones secuenciales

Describir un multiplexor de 4 canales usando:

- Asignaciones secuenciales condicionales
- Asignaciones secuenciales múltiples

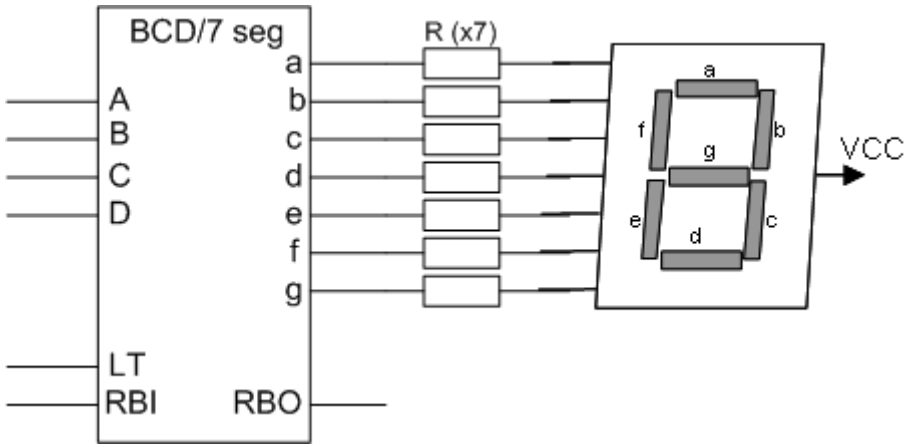
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity mux4a1_sec_cond is
      port ( c0, c1, c2, c3: in std_logic;
            s: in std_logic_vector (1 downto 0);
            salida : out std_logic);
end mux4a1_sec_cond;

architecture estruct_mux4a1sec_c of mux4a1_sec_cond is
      signal control:  integer range 0 to 3;
      begin
            control <= conv_integer (s);
            process (control, c0, c1, c2, c3)
                  begin
                        if control = 0 then salida <= c0; end if;
                        if control = 1 then salida <= c1; end if;
                        if control = 2 then salida <= c2; end if;
                        if control = 3 then salida <= c3; end if;
            end process;

end estruct_mux4a1sec_c;
```

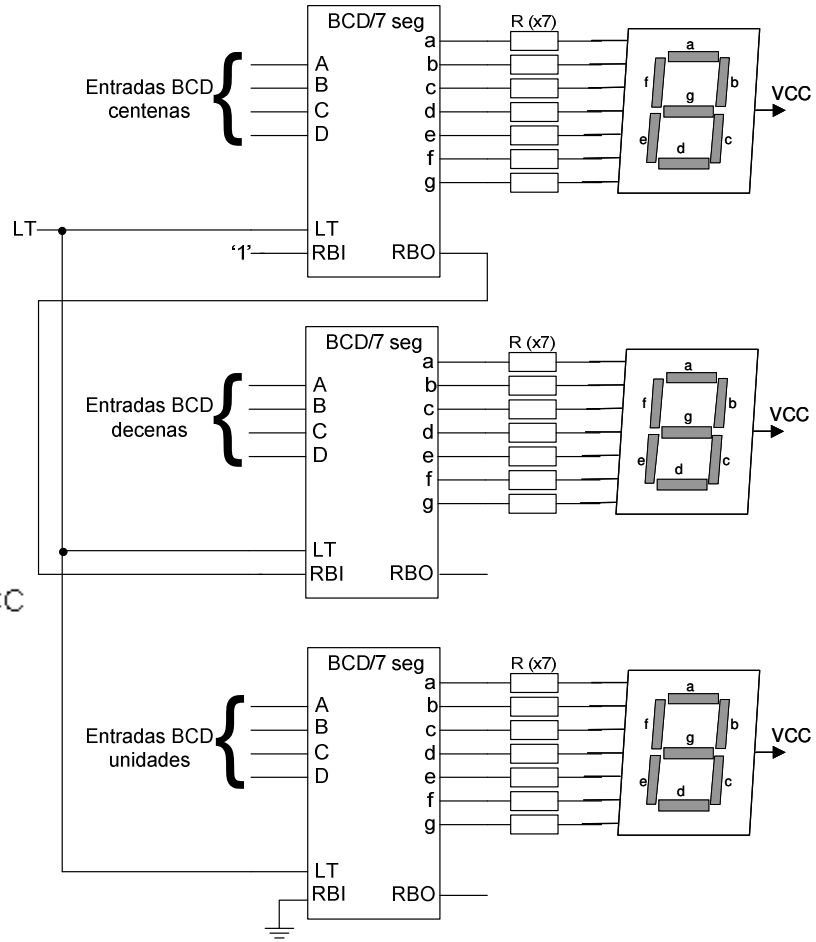
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity mux4a1sec_mul is
          port (c0, c1, c2, c3: in std_logic;
                s: in std_logic_vector (1 downto 0);
     salida: out std_logic);
end mux4a1sec_mul;

architecture estruct_mux4a1secmul of mux4a1sec_mul is

    begin
          process (s, c0, c1, c2, c3)
              begin
               case s is
                    when "00" => salida <= c0;
                    when "01" => salida <= c1;
                    when "10" => salida <= c2;
                    when others => salida <= c3;
               end case;
          end process;

end estruct_mux4a1secmul;
```

VHDL básico para diseñar circuitos combinacionales (XVII)

Ejercicio: Modelar en VHDL un decodificador BCD/7 segmentos para visualizadores de ánodo común (salidas del decodificador activas a nivel bajo) que disponga además de las siguientes características:

- Una entrada **LT** (*lamp test*) que cuando vale '1' sirve para encender todos los segmentos del visualizador para comprobar su funcionamiento.
- Una salida **RBO** que debe valer 1 cuando la entrada de datos en BCD sea '0000' y además la entrada **RBI** valga '1'. Estas líneas se emplean para el apagado de ceros no significativos.
- Una entrada RBI que cuando recibe un '1' y la entrada BCD es '0000' debe apagar el display.





VHDL básico para diseñar circuitos combinacionales (XVIII)

Algunas observaciones importantes sobre el lenguaje VHDL

- Las asignaciones condicionales concurrentes deben ser completas, es decir, es necesario especificar que debe ocurrir si no se cumplen las condiciones. Por tanto, deben llevar **else** en el caso de **when** y deben llevar **when others** en el caso de **with**.
- La asignación múltiple **case** también debe ser completa, si no se recorren todos los casos posibles, hay que incluir al final **when others =>**.
- La asignación **if** también debe ser completa, llevando al final **else** o asignando todos los valores posibles en las condiciones. Una excepción es cuando empleamos la memoria implícita de los procesos, que veremos en el estudio de los sistemas secuenciales.
- Dentro de **if** o **case**, se pueden hacer asignaciones a varias señales: señal1 <= -----; señal2 <= -----; señal3 <= ------ ;
- En una asignación múltiple (**with** o **case**) se puede hacer referencia a varios casos mediante el símbolo '|' para separarlos:

```
case entrada is
        when 1 | 3 | 6 => f <= '1'; …


with entrada select
        f <= '1' when 1 | 3 | 6, …
```

Otros recursos de VHDL (I)

- Se pueden declarar y dar valores a constantes mediante **:=**. Deben declararse después de la declaración **architecture** y antes del **begin** de la misma. Estas constantes pueden emplearse para asignación de valores a señales o parámetros:

```
constant siete: std_logic_vector (3 downto 0) := "0111";
g <= siete;
constant k     : integer:=8;
signal m       : std_logic_vector (k-1 downto 0);
```

VHDL básico para diseñar circuitos combinacionales (XIX)

Otros recursos de VHDL (II)

- Se pueden también definir parámetros o valores constantes en la declaración de entidad, antes de los puertos, con la declaración generic (genéricos):

```
generic (m:  integer :=8);
port (…
```

- Los vectores pueden usarse de forma parcial o por componentes. También se pueden asignar valores a los vectores a través de sus componentes:

```
signal a:  std_logic_vector (15 downto 0);
signal b:  std_logic_vector (7 downto 0);
signal c:  std_logic;
signal d, e: std_logic_vector (3 downto 0);
b <= a(15 downto 8);
c <=a(6);
d <= (3 => '1', 2 =>'1', others => '0');
e <= (others =>'0'); -- equivale a e <= "0000";
```

- Se puede definir una matriz, que es un conjunto ordenado y numerado de vectores:

```
type coleccion is array (1 to 128) of std_logic_vector (7 downto 0); -- 128 vectores de 8 bits cada uno
signal a    :coleccion;
begin
   a(34) <= '01110100';
   a(67)(7) <= '1';
```

- Se pueden definir números hexadecimales con la notación 16#número#:

```
VIA <= '1' when DIR = 16#FFC0# else '0';
```

VHDL básico para diseñar circuitos combinacionales (XX)

Otros recursos de VHDL (III)

- En los procesos y en las funciones y procedimientos que se verán más adelante en el temario, pueden emplearse **VARIABLES** internas. Mientras que las señales no cambian de valor dentro del proceso, sino al final del mismo, las variables cambian de valor en el propio proceso, en cuanto reciben una asignación aplicada a ellas. Las asignaciones a variables se hace con el símbolo "**:=**" en vez de "**<=**".

Veamos la diferencia entre variable y señal con dos ejemplos:

Ejemplo de **p** como señal

```
architecture ejemplo1 of entidad_declarada_antes is
        signal a, b, y, w:  std_logic_vector (3 downto 0);
        signal p:  std_logic_vector (3 downto 0);
        begin
                p1: process (a, b, p)
                        begin
                                p <= a or b;
                                y <= p;
                                p <= a and b;
                                w <= p;
                end process;
…
```

Ejemplo de **p** como variable

```
architecture ejemplo1 of entidad_declarada_antes is
        signal a, b, y, w:  std_logic_vector (3 downto 0);
        variable p: std_logic_vector (3 downto 0);
        begin
                p1: process (a, b, p)
                        begin
                                p := a or b;
                                y <= p;
                                p := a and b;
                                w <= p;
                end process;
…
```

En el ejemplo de la izquierda, el valor de "y" y "w" será **a and b**, ya que como señal, "p" recibe asignación al final del proceso. En el ejemplo de la derecha, la asignación como variable es inmediata, por tanto "y" tomará el valor **a or b** y "w" tomará el valor **a and b**.

VHDL básico para diseñar circuitos combinacionales (XXI)

Otros recursos de VHDL (IV)

- **Alta impedancia**: Para configurar señales triestado se emplea el valor 'Z':

```
salidas  :out std_logic_vector (3 downto 0);
salidas <= "ZZZZ" when enable = '0' else f;
-- otra opción
salidas <= (others => 'Z') when enable = '0' else f;
```

- **Bucles 'FOR'**: Un bucle va siempre dentro de un proceso. Pueden llevar opcionalmente un identificador.

```
identificador: for i in … to … loop
  asignaciones
end loop;
```

Ejemplo: Decodificador de 6 a 64 líneas:

```
signal entrada:  std_logic_vector (5 downto 0);
signal salida:   std_logic_vector (0 to 63);
signal p:        integer range 0 to 63;
begin
  process (entrada)
      p <= conv_integer (entrada);
      for i in salida'range loop                    – otra opcion: for i in 0 to 63 loop
          if p = i then salida(i) <= '1';
          else salida(i) <='0';
          end if;
      end loop;
end process;
```

VHDL básico para diseñar circuitos combinacionales (XXII)

Otros recursos de VHDL (V)

- **Bucles 'WHILE'**: También pueden usarse bucles del tipo 'WHILE'. En estos, el bucle sigue ejecutándose mientras se cumpla una condición especificada. También se usan sólo dentro de un proceso.

```
identificador: while condición loop
  asignaciones
end loop;
```

Ejemplo: Inicializa a '0' los últimos 8 bits del vector entrada:

```
process (lista de sensibilidades)
variable  i:   integer := 0;
begin
  ciclo: while i < 8 loop
       entrada(i) <= (others => '0');
       i := i + 1;
  end loop;
end process;
```

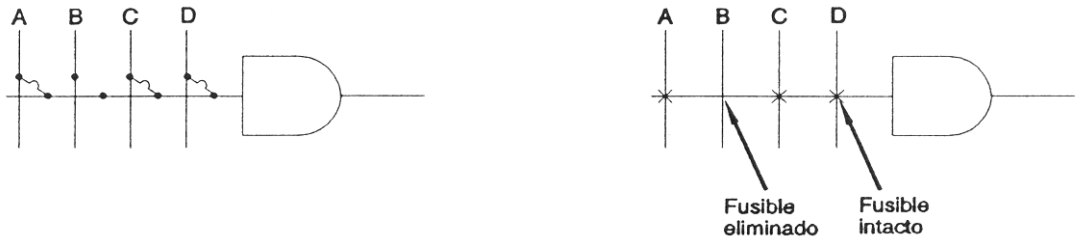
- **Sentencias NEXT y EXIT:** Dentro de un bucle podemos usar también las sentencias NEXT y EXIT. La primera permite detener la ejecución de la iteración actual y pasar a la siguiente. La segunda detiene la ejecución en ese instante y se sale del bucle. En el caso de tener varios bucles anidados, la salida se realiza del bucle donde se encuentre la instrucción.

Dispositivos Programables Combinacionales (I)

Un sistema combinacional programable puede definirse como aquel cuya tabla de verdad puede ser cambiada, sin modificar el cableado entre los elementos que lo constituyen.

Los dispositivos lógicos programables incorporan una matriz lógica genérica, que puede programarse de forma que el dispositivo (circuito integrado) realice las funciones que se desee. Normalmente, la estructura programable principal consiste en una estructura combinacional, formada por una matriz de puertas AND, a cuyas entradas se conectan las entradas del dispositivo tanto de forma directa como negada. Esta primera matriz estará seguida de una segunda matriz formada en el caso más general por puertas OR, de manera que pueda realizarse fácilmente una suma de productos.

Algunas matrices programables están formadas por ***fusibles***, que el usuario puede eliminar o dejar intactos para generar la lógica deseada. Estos fusibles pueden ser de diferente tecnología, pudiendo también ser regenerados para volver a programar el dispositivo y que éste realice otras funciones lógicas. Para simplificar la representación de estas estructuras, las diferentes entradas de una puerta AND se representan con una sola línea denominada línea producto. En la figura que se muestra a continuación se representan cómo se sitúan los fusibles y la representación gráfica, donde se observa que un fusible intacto se representa con una 'X' y un fusible eliminado sin ningún símbolo especial en la unión correspondiente.

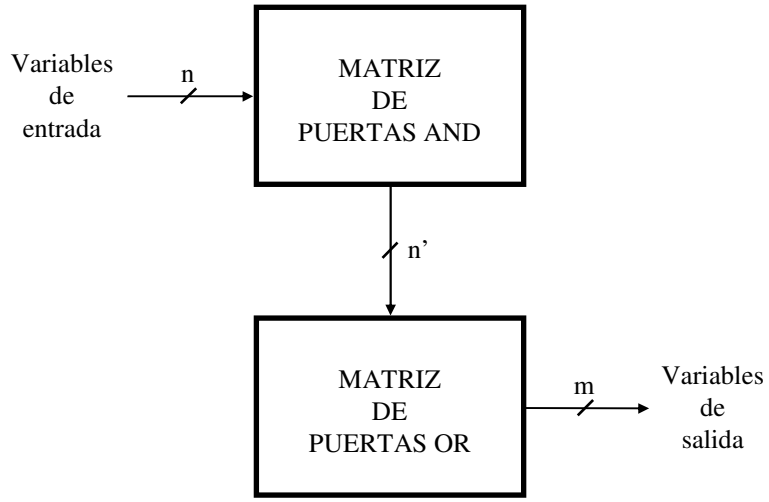


Dispositivos Programables Combinacionales (II)

La estructura genérica de estos dispositivos se muestra en la siguiente figura. Hoy en día, los dispositivos programables suelen ser de tipo secuencial, estando algunos de ellos basados en esta arquitectura AND-OR y añadiendo elementos de tipo secuencial, tales como biestables. Estos dispositivos pueden realizar tanto sistemas secuenciales como sistemas combinacionales.

Los sistemas combinacionales programables universales son aquellos dispositivos programables que permiten la implementación de cualquier función lógica. Existen dos tipos de sistemas combinacionales programables universales:

- ***S. C. P. incompletos***: ***Arquitecturas PLA y PAL***. Estos dispositivos no pueden implementar todos los mintérminos posibles (2^n) de n entradas. Las funciones se simplifican normalmente antes de implementarlas en estos circuitos.
- ***S. C. P. completos***: ***Memorias pasivas***. Estos dispositivos pueden realizar todos los mintérminos posibles (**2^n**) de **n** entradas.



Según la forma en que se realiza la conexión entre las puertas AND y las OR, pueden considerarse dos tipos de arquitecturas de sistemas combinacionales programables incompletos:

- ***Arquitectura "AND Programable - OR Programable".*** Ambas matrices son programables. Se conoce como **PLA** (Programmable Logic Array: 'Matriz lógica programable').
- ***Arquitectura "AND Programable - OR Fija".*** Sólo la matriz AND es programable. Cada cierto número de puertas AND están conectadas de forma fija a una puerta OR. Se conoce como **PAL** (Programmable AND Array Logic: 'Matriz lógica "Y" programable').

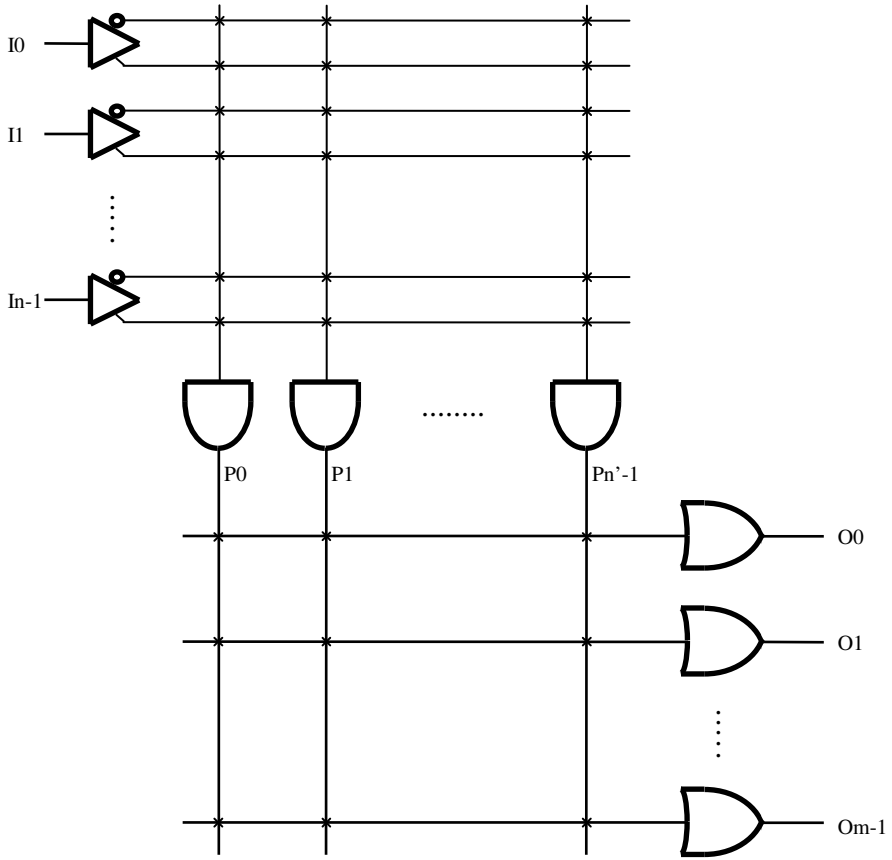
Dispositivos Programables Combinacionales (III)

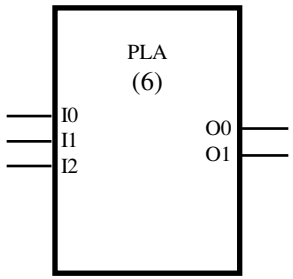
Matrices lógicas programables (PLA)

Una PLA está constituida por una matriz de *n*' (*n*' < 2^n, donde *n* es el número de entradas) puertas AND y una matriz de puertas OR, ambas programables. Las puertas AND poseen 2*n* entradas que se unen a cada variable de entrada a través de conexiones que pueden ser eliminadas.

Las puertas OR poseen tantas entradas como puertas AND existen en el circuito (*n*'), y se conectan a las salidas de éstas mediante conexiones igualmente suprimibles.

La estructura general de una arquitectura PLA se muestra a continuación, así como el símbolo lógico de una PLA con tres entradas, seis términos producto y dos salidas.





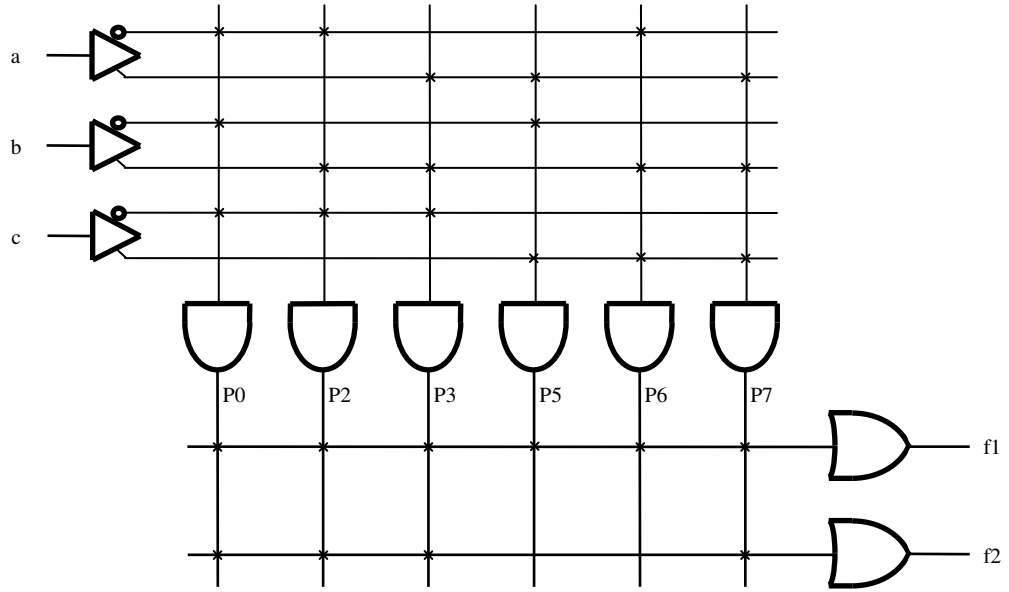
Dispositivos Programables Combinacionales (IV)

Ejemplo de diseño con una PLA (I):

Implementar mediante una PLA el sistema combinacional formado por las siguientes funciones:

$$f_1 = \sum_3 (0,2,3,5,6,7)$$
$$f_2 = \sum_3 (0,2,3,7)$$

La PLA mínima necesaria para la implementación de esta multifunción, deberá tener tres entradas (número de entradas del sistema), seis términos producto (número de términos producto distintos del sistema) y dos salidas (número de salidas del sistema). Una vez programada, su estructura interna quedaría como sigue:



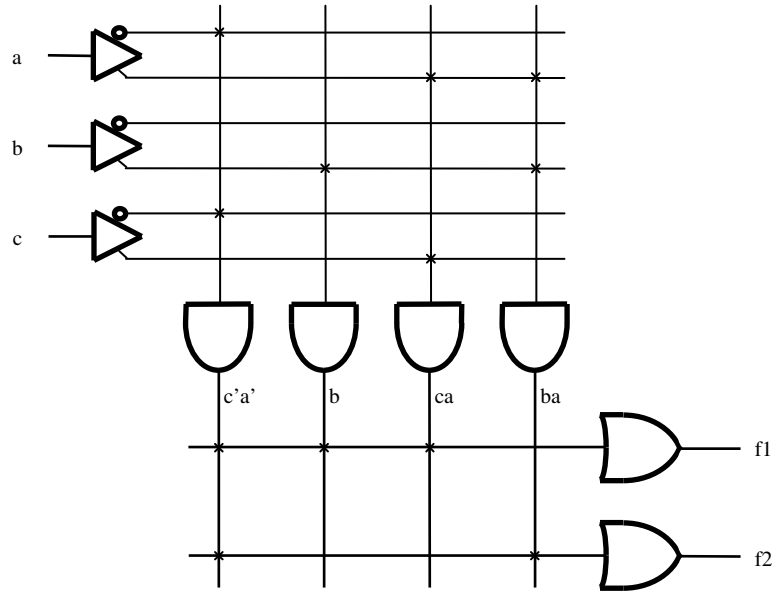
Dispositivos Programables Combinacionales (V)

Ejemplo de diseño con una PLA (II):

A veces, la simplificación de las expresiones en forma de suma de productos de las funciones a implementar, permite la reducción del tamaño de la PLA necesaria para la implementación del sistema. Así, si simplificamos por Karnaugh las funciones anteriores obtendremos:

$$f_1 = \bar{c}\,\bar{a} + b + c\,a$$
$$f_2 = \bar{c}\,\bar{a} + b\,a$$

El tamaño mínimo de la PLA necesaria para la implementación del sistema, utilizando las expresiones simplificadas, será ahora 3 x 4 x 2 (3 entradas, 4 términos producto distintos entre las dos expresiones y 2 salidas). El circuito, una vez programado, quedará como se muestra en la figura.



AND	I2	I1	I0	O1	O0
1	L	-	L	A	A
2	-	H	-	•	A
3	H	-	H	•	A
4	-	H	H	A	•
VAR	c	b	a	f2	f1

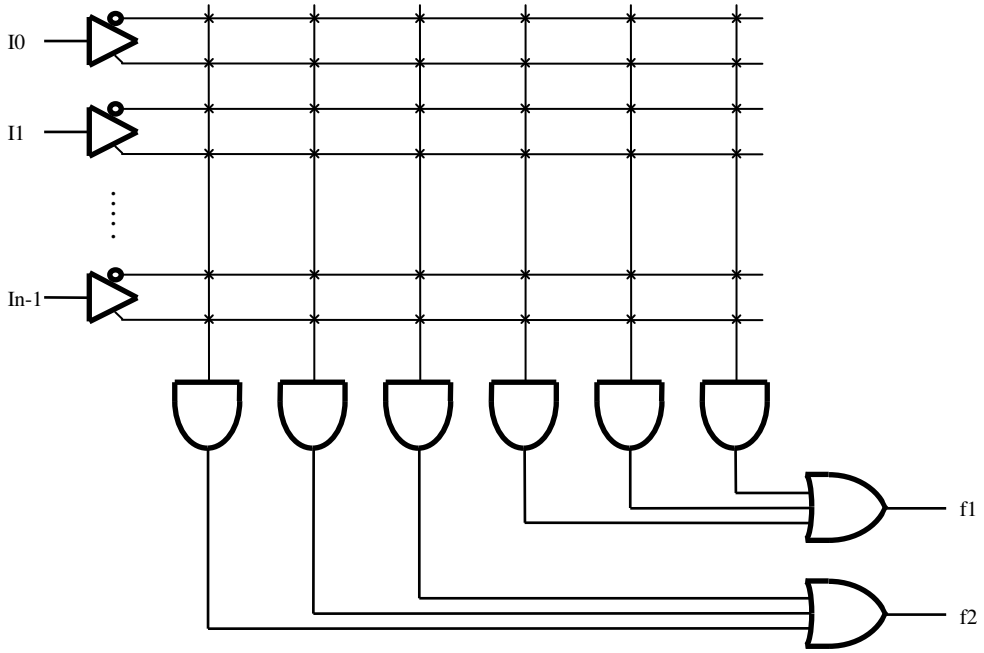
Dispositivos Programables Combinacionales (VI)

Matrices lógicas Y – Programables (PAL)

Una PAL es un sistema combinacional programable universal incompleto, de estructura similar a la de una PLA, de la cual se diferencia en que cada una de las puertas OR están conectadas rígidamente a un conjunto de puertas AND cada una. En general, si el número de términos producto es *n*' y el de salidas *m*, cada una de estas últimas irá conectada a *n*'/*m* puertas AND.

Las PAL son dispositivos menos flexibles que las PLA y necesitan más puertas AND que éstas, ya que si un producto forma parte de varias funciones, deberá ser programado para cada una de ellas. En compensación, su tiempo de programación es más pequeño, disipan menos potencia, ocupan memos superficie de silicio y su programación es más sencilla.

La estructura interna general de una PAL, con 6 términos producto y dos salidas, se representa en la siguiente figura.

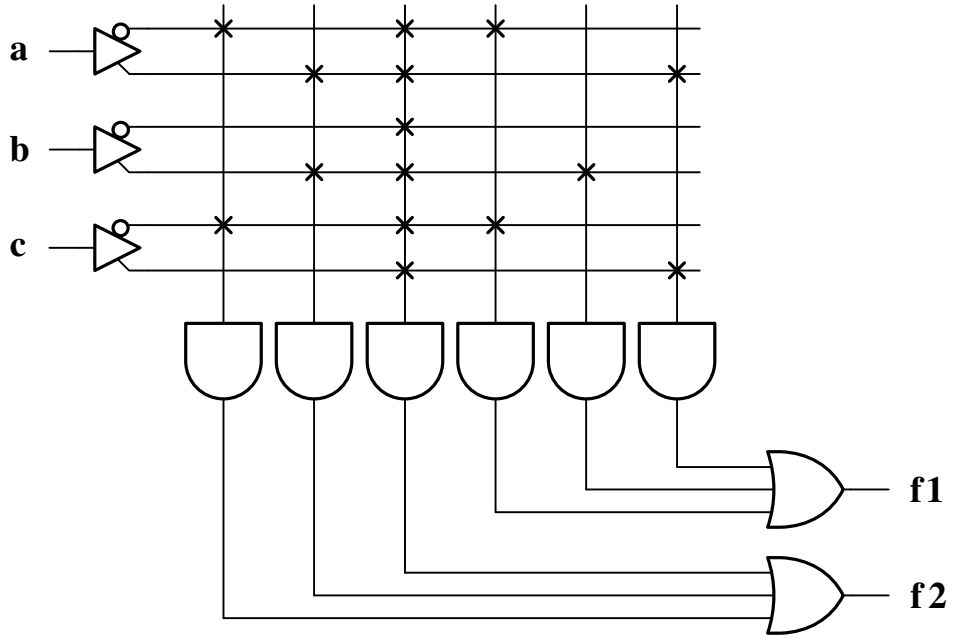


Dispositivos Programables Combinacionales (VII)

Ejemplo de diseño con una PAL (I):

Vamos a realizar las dos funciones anteriores mediante una PAL. Se muestra una representación de la PAL mínima necesaria, así como una tabla donde se detalla su programación.

$$f_1 = \bar{c}\,\bar{a} + b + c\,a$$
$$f_2 = \bar{c}\,\bar{a} + b\,a$$



AND	I2	I1	I0	O1	O0
1	L	-	L	-	A
2	-	H	-	-	A
3	H	-	H	-	A
4	L	-	L	A	-
5	-	H	H	A	-
6				•	-
VAR	c	b	a	f2	f1

Dispositivos Programables Combinacionales (VIII)

Procedimiento genérico de diseño con Dispositivos Lógicos Programables

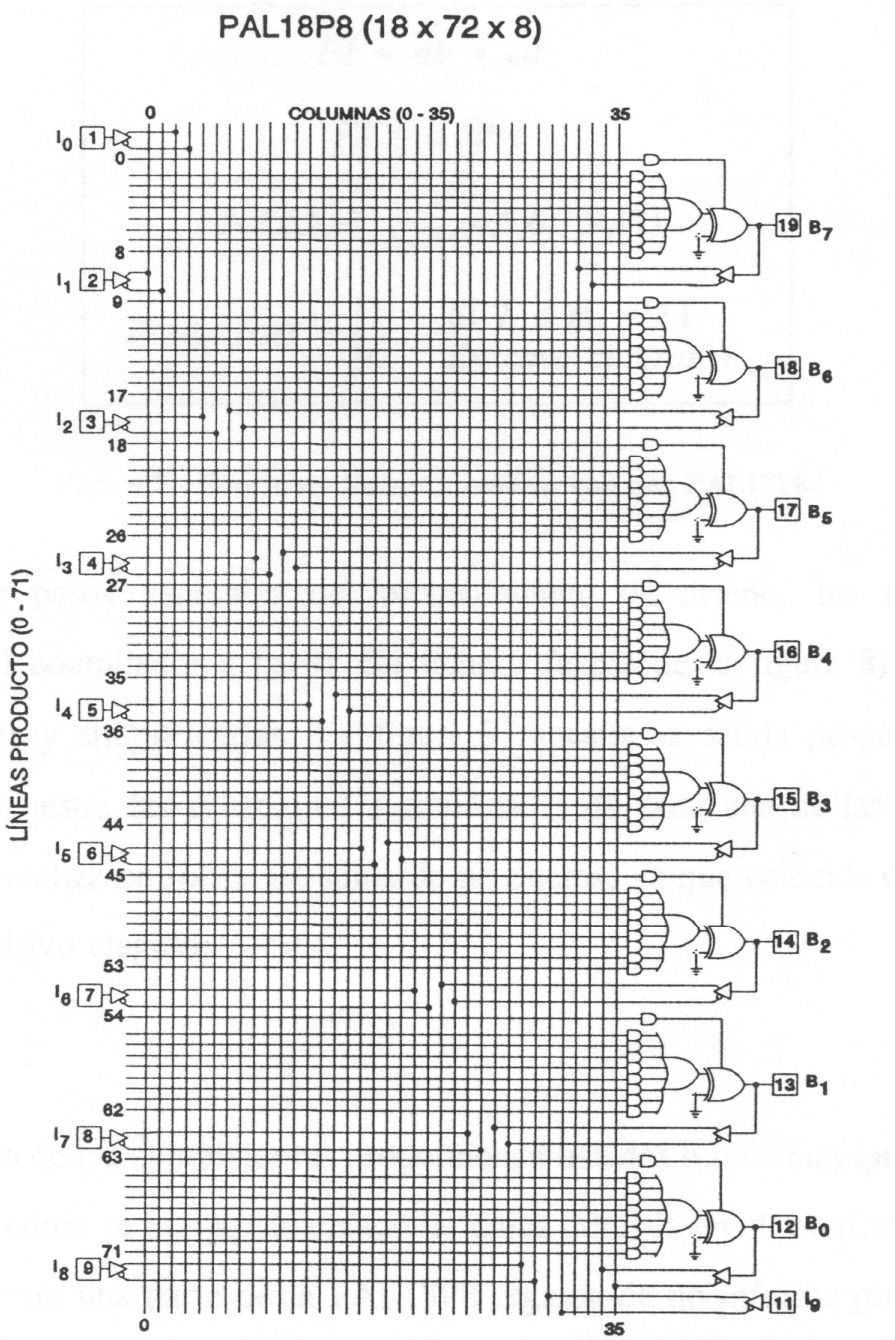
El procedimiento de diseño basado en estos dispositivos pasa por los siguientes puntos:

1. ***Especificación de la función que el circuito debe realizar***. Esta especificación puede consistir en ecuaciones booleanas, tablas de verdad, diagramas de estados, diagramas lógicos, VHDL y otros formatos.
2. ***Generación de ecuaciones***. A partir de las especificaciones iniciales, se generan las ecuaciones que las representan.
3. ***Simplificación de las ecuaciones lógicas***. Este paso es necesario para asegurar que la lógica a implementar puede realizarse en un dispositivo concreto, ya que éstos tienen un número limitado de recursos.
4. ***Generación del mapa de programación del dispositivo***. Consiste en obtener un fichero que proporcionado al programador de dispositivos permite generar la lógica deseada.
5. ***Simulación lógica***. Este paso es opcional, aunque muy importante en el diseño de circuitos. Consiste en determinar el funcionamiento exacto que tendrá el dispositivo una vez grabado, pero sin necesidad de programarlo. La simulación se realiza en el ordenador.
6. ***Programación del dispositivo seleccionado***. Una vez comprobado el diseño, puede ya grabarse el dispositivo físicamente, labor que realiza el programador de dispositivos. Este instrumento recibe el mapa de programación y de él extrae la información necesaria para grabar el dispositivo aplicando ciertos impulsos eléctricos. Algunos dispositivos pueden programarse sin necesidad de extraerlos del circuito impreso donde se encuentren. Cuando tienen esta capacidad se denominan ***ISP*** (***In System Programmable***). Los estudiaremos en el tema 6.
7. ***Chequeo o test del dispositivo ya programado***. Este paso (opcional) consiste en comprobar que un dispositivo ha sido grabado correctamente. Esta comprobación se realiza con la ayuda del mismo programador de dispositivos.

Dispositivos Programables Combinacionales (IX)

Ejemplo de dispositivo PAL real: PAL18P8

Como ejemplo de arquitectura PAL se muestra en la figura 6 la estructura interna del dispositivo PAL18P8. En este dispositivo sólo puede programarse la matriz AND, ya que la matriz OR tiene conexiones fijas. Este dispositivo puede configurarse para cambiar el número de entradas y de salidas. Además, las salidas pueden negarse o no, según se necesite. Como ejercicio, describa en qué forma una salida puede convertirse en una entrada en este dispositivo.



Dispositivos Programables Combinacionales (X)

Unidades usadas en almacenamiento de datos

- **Byte**: Es la unidad de información formada por ocho bits.
- **Kilobyte** (KB): 1024 bytes. (2^{10} bytes)
- **Megabyte** (MB): 1024 kilobytes. (1.048.576 bytes)
- **Gigabyte** (GB): 1024 Megabytes (1.073.741.824 bytes)
- **Terabyte** (TB): 1024 Gigabytes

…..

- **Yottabyte** (YB): 1024 Zettabytes

File Storage Capacity by Bits and Bytes					
	Bit	byte	Kilobyte	Megabyte	Gigabyte
bit	1				
byte	8	1			
Kilobyte	8,192	1,024	1		
Megabyte	8,388,608	1,048,576	1,024	1	
Gigabyte	8,589,934,592	1,073,741,824	1,048,576	1,024	1
Terabyte	8,796,093,022,208	1,099,511,627,776	1,073,741,824	1,048,576	1,024
Petabyte	9,007,199,254,740,990	1,125,899,906,842,620	1,099,511,627,776	1,073,741,824	1,048,576
Exabyte	9,223,372,036,854,780,000	1,152,921,504,606,850,000	1,125,899,906,842,620	1,099,511,627,776	1,073,741,824
Zettabyte	9,444,732,965,739,290,000,000	1,180,591,620,717,410,000,000	1,152,921,504,606,850,000	1,125,899,906,842,620	1,099,511,627,776

Dispositivos Programables Combinacionales (XI)

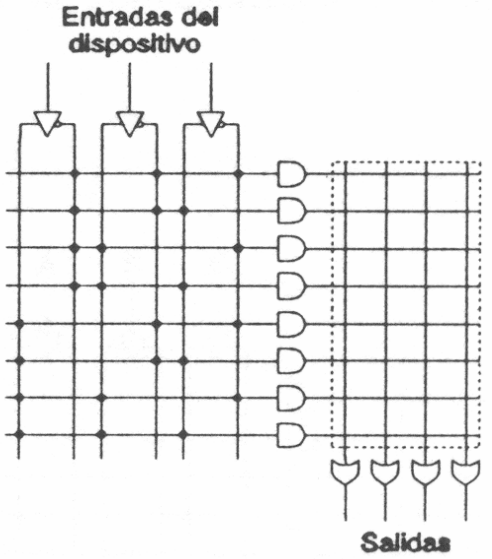
Memorias tipo ROM (Read Only Memory) (I)

Estos sistemas proporcionan todos los términos posibles que podemos obtener con las variables de entrada. Su estructura es de tipo ***"AND Fija - OR Programable"***. Sólo la matriz OR es programable. Esta es una estructura **ROM (Read Only Memory: 'Memoria programable de sólo lectura')**. La figura muestra un ejemplo simplificado de esta estructura.

Una memoria está organizada en **2^n palabras de m bits cada una**. Se dice que tiene **n entradas de dirección**, **m salidas de datos,** y en general disponen de entradas de control para habilitar salidas, habilitar el chip, etc. (**p** entradas de control).

Cada celda de memoria puede almacenar un bit. Si la memoria puede modificarse escribiendo en ella, se dice que es de **lectura y escritura**. Si por el contrario, la información de la memoria no puede modificarse durante la operación normal de la memoria, se dice de **sólo lectura**.

A cada palabra de memoria le corresponde una dirección única, estando formada esa dirección por una combinación específica de las *n* entradas de dirección. Para leer una palabra, además de poner las entradas de control a los niveles adecuados, en las entradas de dirección se aplica la dirección correspondiente a esa palabra y en las salidas de datos aparece la información escrita en esa palabra de memoria.



AND FIJA - OR PROGRAMABLE

PROM (Programmable Read Only Memory)
Memoria programable de sólo lectura

Dispositivos Programables Combinacionales (XII)

Memorias tipo ROM (Read Only Memory) (II)

Existen diferentes tipos de memorias ROM:

- **Memorias ROM programables por máscara**: Son memorias cuya programación se fija durante el proceso de fabricación. No pueden volver a programarse y sólo tienen sentido para producciones extensas.
- **Memorias PROM (Programmable ROM)**: También denominadas *Field Programmable* (programable 'in situ'). Permite a un usuario programar la memoria con el contenido que desee, pero una vez programada, no puede volver a borrarse o reprogramarse.
- **Memorias tipo EPROM (Erasable Programmable ROM)**: Estas memorias pueden grabarse por parte del usuario,y permiten borrarse para volver a ser programadas. Normalmente, mediante una luz ultravioleta se vuelve la memoria a su estado inicial, para luego volver a programarla.
- **Memorias tipo EEPROM (electrically Erasable Programmable** ROM): Estas memorias también pueden volver a reprogramarse, pero con la ventaja de que puede hacerse en el mismo lugar donde están insertadas, aplicando ciertos valores de tensión indicados por el fabricante.
- **Memorias tipo FLASH**: En estas memorias, no se pueden borrar posiciones individuales, sino que se borran simultáneamente todas las celdas del chip.

Las aplicaciones de las memorias ROM son muy variadas: grabación de tablas de consulta frecuente, generación de secuencias específicas de caracteres, grabación permanente de rutinas muy usadas en un sistema digital (como por ejemplo, la BIOS de un ordenador tipo PC), etc.

Estas memorias PROM grabables por el usuario se programan normalmente en un **programador de dispositivos**, que puede ser tanto de tipo genérico (puede grabar dispositivos de distintos fabricantes), como específico para los dispositivos de un único fabricante.

Dispositivos Programables Combinacionales (XIII)

Ejemplo de dispositivo real FLASH: AT29C010A

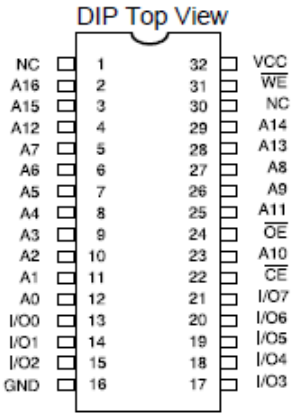
Este dispositivo es una memoria tipo FLASH de 1 Megabit (**128K x 8**). Está por tanto organizada como 131072 palabras de 8 bits cada una. Una de sus ventajas es que no es necesario un voltaje distinto al de operación para ser reprogramada (5 voltios). Puede ser reescrita más de 10000 veces manteniendo su funcionamiento. Leer datos en este dispositivo es similar a hacerlo en una EPROM. No vamos a comentar los pasos necesarios para reprogramarla. En muchos casos, si se hace en un programador de dispositivos, el software asociado y el hardware del programador se encargarán de programarlo de forma casi "transparente" al usuario, que sólo se preocupa del contenido que quiere grabar en la memoria.

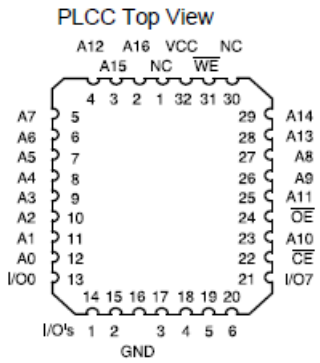
Pin Configurations

Pin Name	Function
A0 - A16	Addresses
$\overline{CE}$	Chip Enable
$\overline{OE}$	Output Enable
$\overline{WE}$	Write Enable
I/O0 - I/O7	Data Inputs/Outputs
NC	No Connect

AC Read Characteristics

		AT29C010A-70		AT29C010A-90		AT29C010A-12		AT29C010A-15		
Symbol	Parameter	Min	Max	Min	Max	Min	Max	Min	Max	Units
t_{ACC}	Address to Output Delay		70		90		120		150	ns
t_{CE} (1)	$\overline{CE}$ to Output Delay		70		90		120		150	ns
t_{OE} (2)	$\overline{OE}$ to Output Delay	0	35	0	40	0	50	0	70	ns
t_{DF} (3, 4)	$\overline{CE}$ or $\overline{OE}$ to Output Float	0	25	0	25	0	30	0	40	ns
t_{OH}	Output Hold from $\overline{OE}$, $\overline{CE}$ or Address, whichever occurred first	0		0		0		0		ns





Waveforms (1, 2, 3, 4)