Fundamentos de Computadores

1º curso del Grado en Ingeniería Informática

Tema 4

Bloques funcionales combinacionales

Universidad de Huelva

ETSI ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA

Departamento de Ingeniería Electrónica, de Sistemas Informáticos y Automática

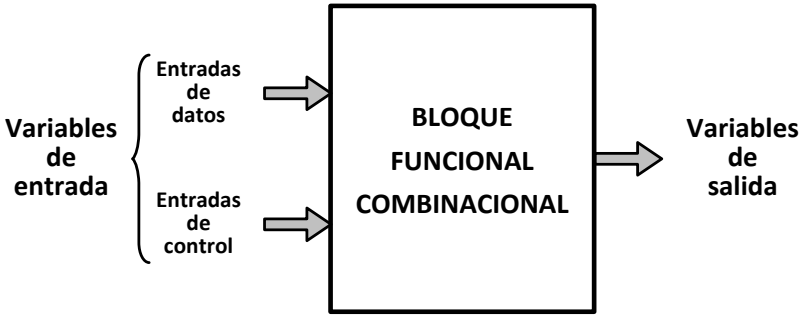
Tema 4. Bloques funcionales combinacionales
Introducción a los bloques funcionales combinacionales

En cualquier computador es preciso realizar 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 han desarrollado diversos bloques funcionales, cuyas primitivas fundamentales son las puertas lógicas estudiadas en los temas anteriores.

Estos bloques se engloban dentro de los dispositivos de media escala de integración o MSI (*Medium Scale Integration*) y poseen la ventaja de poder usarse como entidades de diseño de nivel superior a las puertas. De esta forma, no es necesario recordar la estructura interna de puertas de dichos bloques, sino que es suficiente con conocer sus símbolos y sus comportamientos.

Simbología general de los bloques funcionales combinacionales



En este tema estudiaremos los bloques funcionales combinacionales más frecuentemente utilizados y sus aplicaciones:

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

Un **decodificador** es un circuito lógico combinacional que posee **n** entradas de selección y **2^n** salidas como máximo.

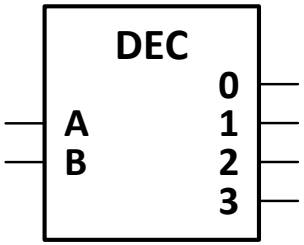
Su funcionamiento es tal que para cada combinación de las variables de entrada se activa una única salida, que es la que corresponde a la combinación binaria aplicada a las entradas.

Si el decodificador posee una salida para cada combinación de las variables de entrada (**2^n** salidas) se denomina **decodificador completo**, y si no es así (menos de **2^n** salidas) se denomina **decodificador incompleto**.

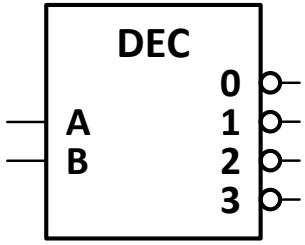
La principal aplicación de los decodificadores es la selección de un único dispositivo en cada momento (módulo de memoria, buffer triestado, etc.) entre todos los conectados a sus salidas, en función de la combinación aplicada a sus líneas de selección. No obstante, los decodificadores también pueden emplearse para implementar funciones lógicas sin necesidad de simplificarlas.

Los decodificadores pueden tener salidas activas a nivel alto (salida inactiva igual a 0 y salida activa igual a 1) o salidas activas a nivel bajo (salida inactiva igual a 1 y salida activa igual a 0).

Decodificadores de 2 a 4 líneas con salidas directas y negadas



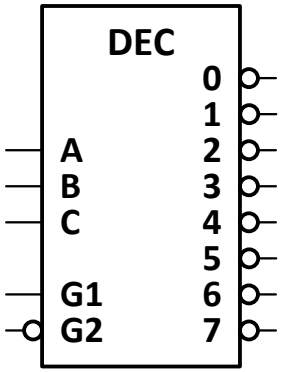
B	A	Y_0	Y_1	Y_2	Y_3
0	0	1	0	0	0
0	1	0	1	0	0
0	0	0	0	1	0
0	1	0	0	0	1



B	A	Y_0	Y_1	Y_2	Y_3
0	0	0	1	1	1
0	1	1	0	1	1
0	0	1	1	0	1
0	1	1	1	1	0

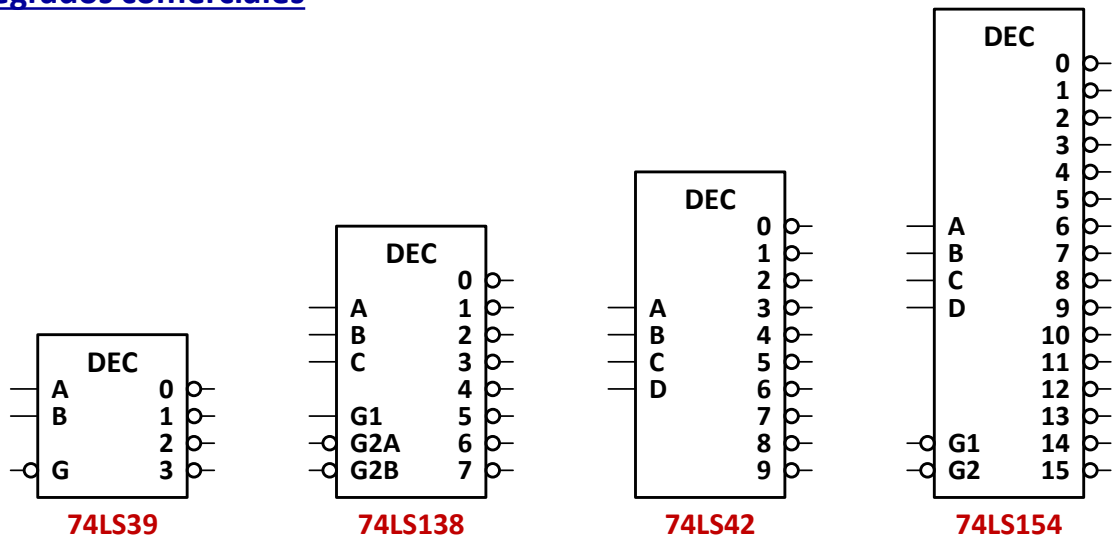
Entradas de habilitación

Algunos decodificadores poseen una o varias entradas de habilitación (**G**), las cuales deben encontrarse en un determinado estado lógico para que se produzca la decodificación. En caso contrario, el circuito no decodifica, es decir, no se selecciona ninguna salida.



G_2	G_1	C	B	A	Y_0	Y_1	Y_2	Y_3	Y_4	Y_5	Y_6	Y_7
1	X	X	X	X	1	1	1	1	1	1	1	1
X	0	X	X	X	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	1	1	1
0	1	0	0	1	1	0	1	1	1	1	1	1
0	1	0	1	0	1	1	0	1	1	1	1	1
0	1	0	1	1	1	1	1	0	1	1	1	1
0	1	1	0	0	1	1	1	1	0	1	1	1
0	1	1	0	1	1	1	1	1	1	0	1	1
0	1	1	1	0	1	1	1	1	1	1	0	1
0	1	1	1	1	1	1	1	1	1	1	1	0

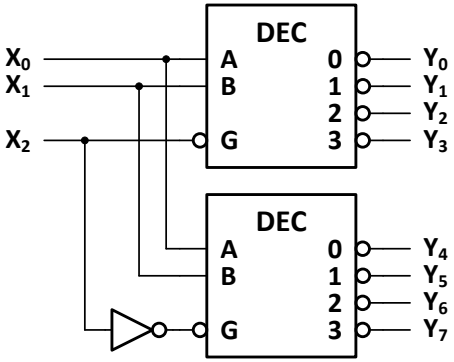
Decodificadores integrados comerciales

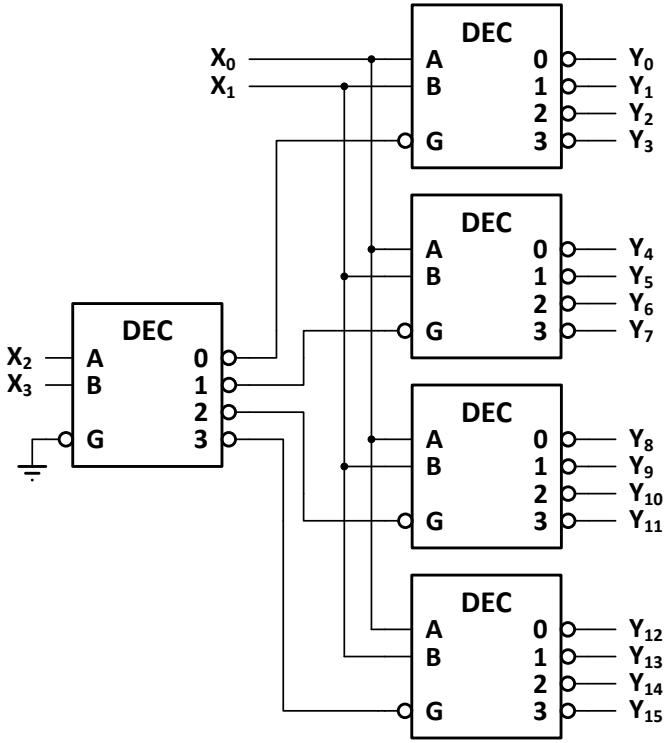


Si se necesita obtener un decodificador con un tamaño diferente al de los disponibles comercialmente, se debe realizar una asociación de decodificadores. Para ello, se divide el número de salidas necesarias entre el número de salidas de los decodificadores de mayor tamaño disponibles, obteniendo así el número de decodificadores necesarios en la etapa de salida. A estos decodificadores se les aplicará en paralelo las variables de **menor peso** de la combinación de entrada.

Por último, es necesario que sólo uno de los decodificadores anteriores actúe en cada momento, cuando el valor binario de la combinación de entrada pertenezca al rango de salidas que proporciona ese decodificador. Esto se consigue añadiendo un nuevo decodificador (o puertas lógicas) para seleccionar uno entre los decodificadores de salida. A este último decodificador se le aplicarán las variables de **mayor peso** de la combinación de entrada.

Ejemplos: Obtener un decodificador de 3 a 8 líneas y otro de 4 a 16 líneas usando decodificadores de 2 a 4 líneas.





Un decodificador completo genera todos los minitérminos del conjunto de variables de entrada.

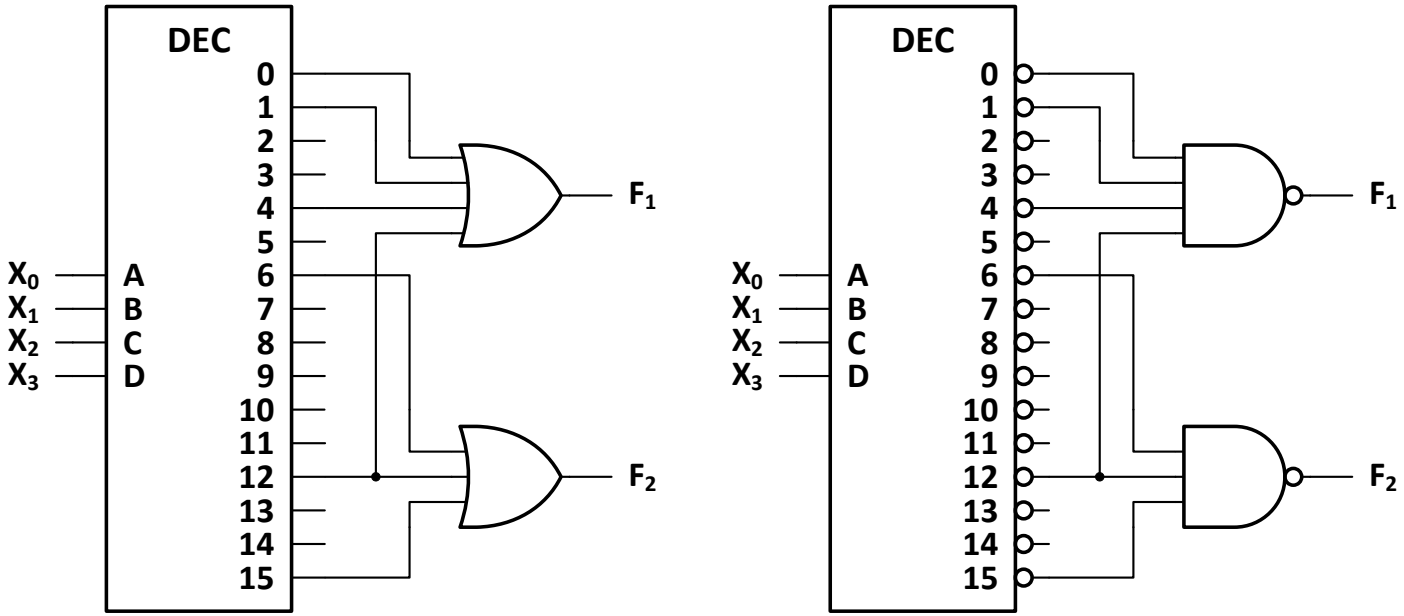
Así pues, para realizar funciones basándose en el empleo de un decodificador, basta con sumar (mediante el empleo de puertas **OR**) los términos pertenecientes a la expresión canónica disyuntiva numérica, en el caso de que las salidas del decodificador sean activas a **nivel alto**.

Si las salidas del decodificador son activas a **nivel bajo**, se procederá del mismo modo, excepto que se emplearán puertas **NAND** en lugar de puertas OR.

Ejemplo: Implementar mediante un decodificador las siguientes funciones:

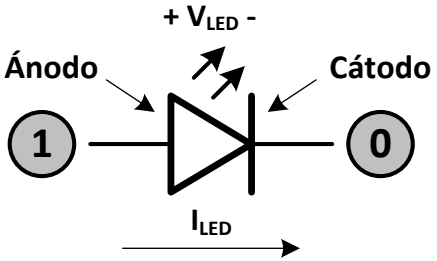
$$F_1 = \sum_4 (0, 1, 4, 12)$$

$$F_2 = \sum_4 (6, 12, 15)$$

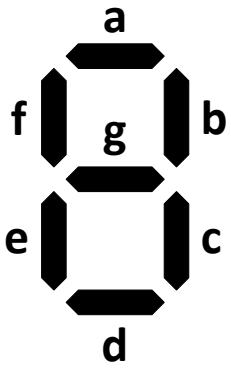


Un conversor de BCD a 7 segmentos es un tipo especial de decodificador que recibe a su entrada una combinación expresada en código BCD natural y genera las señales necesarias para representar el dígito decimal correspondiente en un visualizador (display) de 7 segmentos. Posee salidas de tipo **driver** o **excitador**, capaces de proporcionar corrientes superiores a las de los circuitos estándar.

Un visualizador de siete segmentos es una estructura integrada por siete LEDs (*Light Emitter Diode*). Los LEDs son dispositivos que emiten luz cuando la corriente que los atraviesa excede de un cierto valor, de ahí que para su excitación se necesiten dispositivos que proporcionen corriente suficiente (drivers). Al igual que los diodos normales, los LEDs conducen cuando se polarizan de forma directa con una tensión que supere un determinado valor.

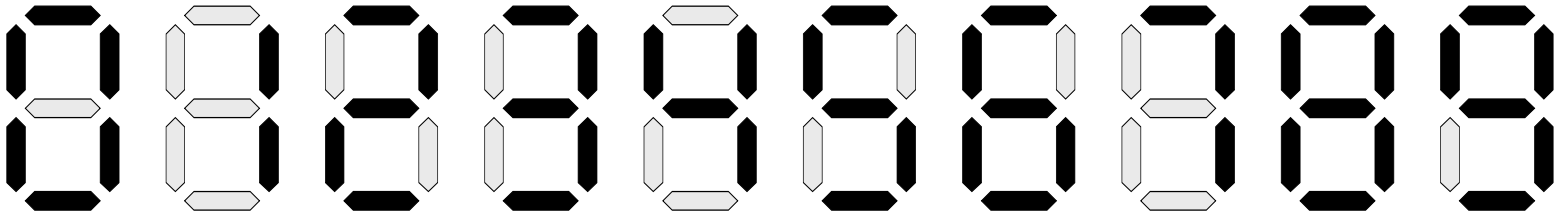


Modo de polarización de un LED



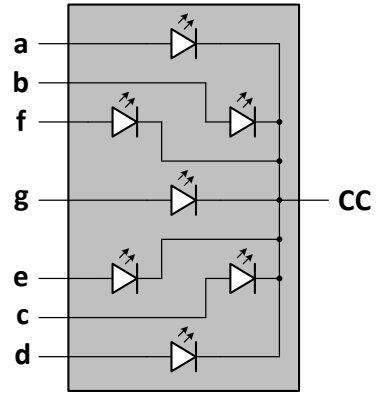
Disposición de los LEDs en un display de 7 segmentos

Representación de los dígitos del 0 al 9 en un display de 7 segmentos

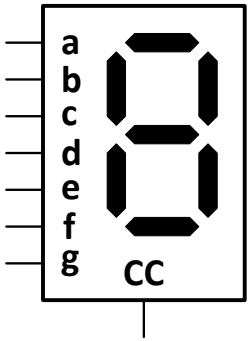


Existen dos tipos de visualizadores de 7 segmentos: de ánodo común y de cátodo común.

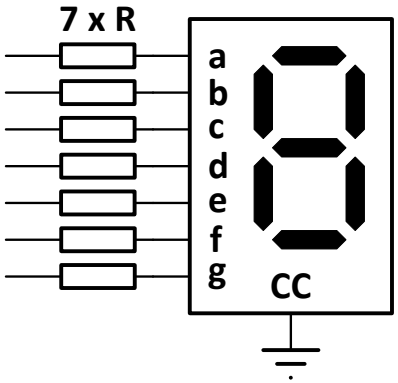
Display de siete segmentos de cátodo común



Conexión interna de los LEDs

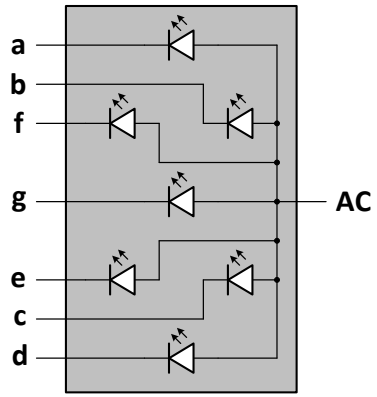


Símbolo

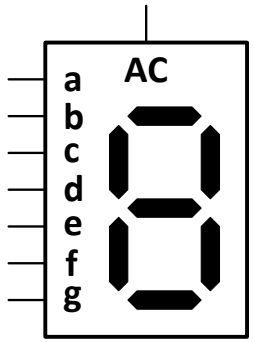


Modo de conexión

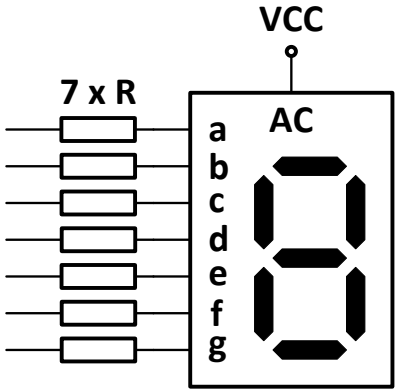
Display de siete segmentos de ánodo común



Conexión interna de los LEDs



Símbolo

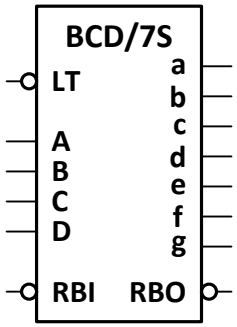


Modo de conexión

Tipos de conversores de BCD a 7 segmentos

Dado que existen dos configuraciones diferentes de displays de siete segmentos (cátodo común y ánodo común), también existen dos tipos de conversores de BCD a 7 segmentos, para controlar a ambos tipos de visualizadores.

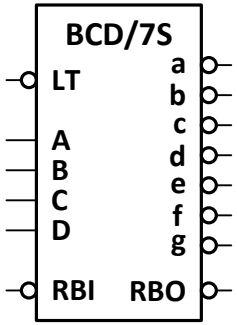
Conversores BCD-7 segmentos para displays de cátodo común



Salidas activas a nivel alto

LT	RBI	D	C	B	A	a	b	c	d	e	f	g	RBO
0	X	X	X	X	X	1	1	1	1	1	1	1	1
1	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	0	0	0	0	1	1	1	1	1	1	0	1
1	X	0	0	0	1	0	1	1	0	0	0	0	1
1	X	0	0	1	0	1	1	0	1	1	0	1	1
1	X	0	0	1	1	1	1	1	1	0	0	1	1
1	X	0	1	0	0	0	1	1	0	0	1	1	1
1	X	0	1	0	1	1	0	1	1	0	1	1	1
1	X	0	1	1	0	1	0	1	1	1	1	1	1
1	X	0	1	1	1	1	1	1	0	0	0	0	1
1	X	1	0	0	0	1	1	1	1	1	1	1	1
1	X	1	0	0	1	1	1	1	1	0	1	1	1

Conversores BCD-7 segmentos para displays de ánodo común

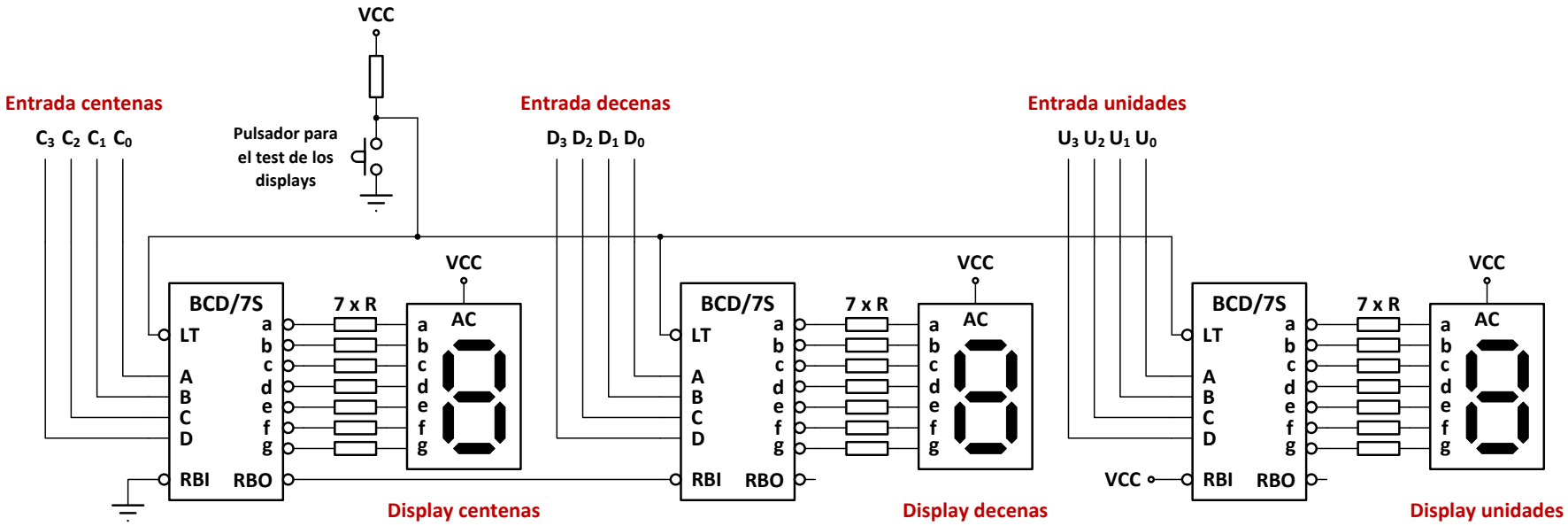


Salidas activas a nivel bajo

LT	RBI	D	C	B	A	a	b	c	d	e	f	g	RBO
0	X	X	X	X	X	0	0	0	0	0	0	0	1
1	0	0	0	0	0	1	1	1	1	1	1	1	0
1	1	0	0	0	0	0	0	0	0	0	0	1	1
1	X	0	0	0	1	1	0	0	1	1	1	1	1
1	X	0	0	1	0	0	0	1	0	0	1	0	1
1	X	0	0	1	1	0	0	0	0	1	1	0	1
1	X	0	1	0	0	1	0	0	1	1	0	0	1
1	X	0	1	0	1	0	1	0	0	1	0	0	1
1	X	0	1	1	0	0	1	0	0	0	0	0	1
1	X	0	1	1	1	0	0	0	1	1	1	1	1
1	X	1	0	0	0	0	0	0	0	0	0	0	1
1	X	1	0	0	1	0	0	0	0	1	0	0	1

Conexión de varios conversores de BCD/7 segmentos usando las líneas RBI y RB0

En este circuito no aparecerán los ceros a la izquierda correspondientes a las centenas y las decenas. Los ceros en las unidades siempre se visualizarán.

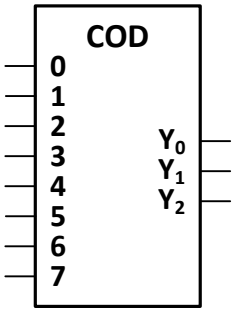


Los codificadores son dispositivos MSI que realizan la operación inversa a la realizada por los decodificadores.

Generalmente, poseen 2^n (0 - 2^n-1) entradas y n salidas.

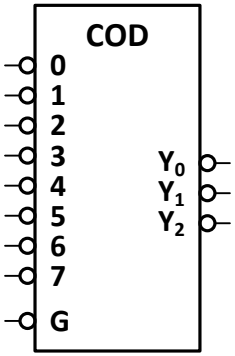
Su funcionamiento es tal, que cuando una de las entradas adopta su estado lógico activo, aparece a la salida la combinación binaria correspondiente al valor decimal asignado a dicha entrada.

Símbolo y tabla de verdad de un codificador de 8 a 3 líneas con entradas y salidas directas



X_7	X_6	X_5	X_4	X_3	X_2	X_1	X_0	Y_2	Y_1	Y_0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

Símbolo y tabla de verdad de un codificador de 8 a 3 líneas con entradas y salidas negadas y entrada de habilitación

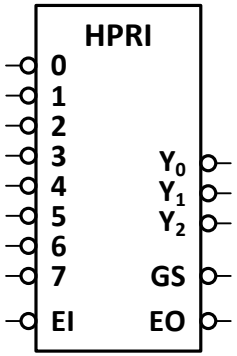


G	X_7	X_6	X_5	X_4	X_3	X_2	X_1	X_0	Y_2	Y_1	Y_0
1	X	X	X	X	X	X	X	X	1	1	1
0	1	1	1	1	1	1	1	0	1	1	1
0	1	1	1	1	1	1	0	1	1	1	0
0	1	1	1	1	1	0	1	1	1	0	1
0	1	1	1	1	0	1	1	1	1	0	0
0	1	1	1	0	1	1	1	1	0	1	1
0	1	1	0	1	1	1	1	1	0	1	0
0	1	0	1	1	1	1	1	1	0	0	1
0	0	1	1	1	1	1	1	1	0	0	0

Codificadores con prioridad

En los codificadores sin prioridad, como los vistos anteriormente, en cada instante debe estar activa una (y solo una) de las entradas. En caso de activarse varias entradas al mismo tiempo, la salida será igual a la suma lógica de las combinaciones correspondientes a cada una de las entradas activas, y por tanto será errónea.

Para evitar esta limitación de funcionamiento se han desarrollado los **codificadores con prioridad**, en los cuales pueden estar activas varias entradas simultáneamente. Cuando esto ocurre, la combinación de salida será la correspondiente a la entrada activa de mayor valor decimal, si se trata de un **codificador con prioridad alta** (HPRI), o a la de menor valor decimal, en el caso de un **codificador con prioridad baja** (LPRI).



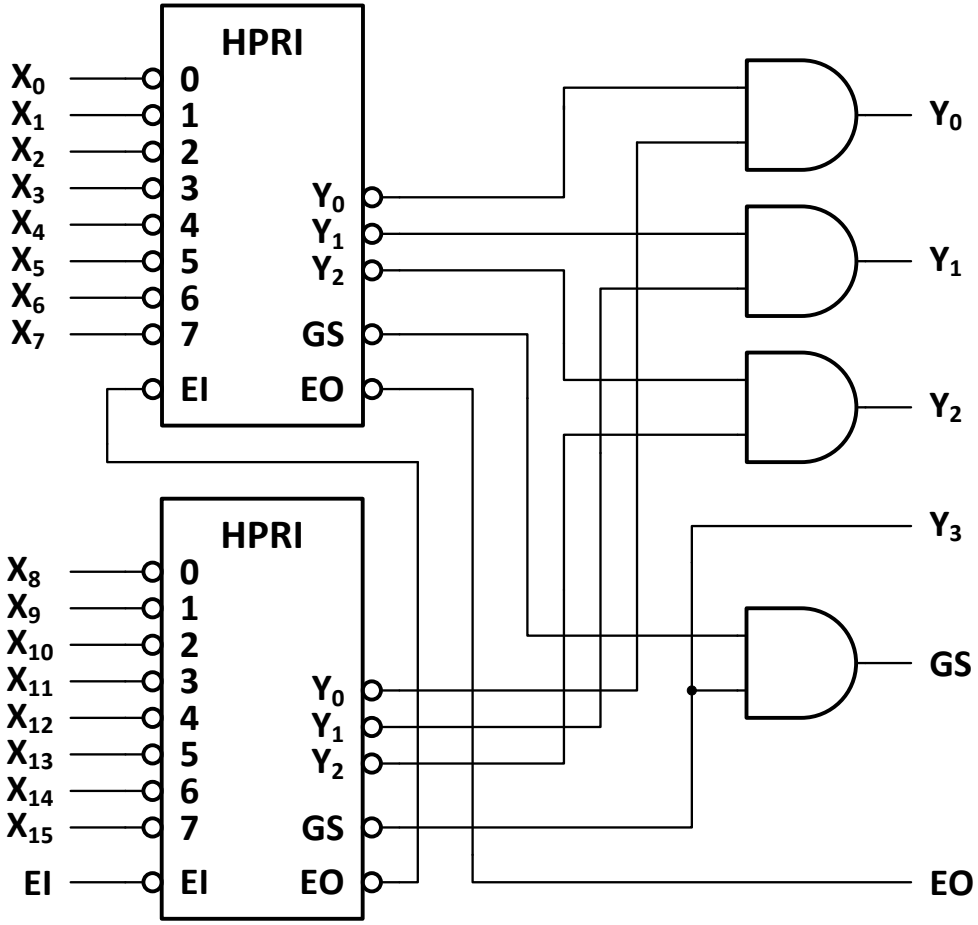
EI	X_7	X_6	X_5	X_4	X_3	X_2	X_1	X_0	Y_2	Y_1	Y_0	GS	EO
1	X	X	X	X	X	X	X	X	1	1	1	1	1
0	1	1	1	1	1	1	1	1	1	1	1	1	0
0	1	1	1	1	1	1	1	0	1	1	1	0	1
0	1	1	1	1	1	1	0	X	1	1	0	0	1
0	1	1	1	1	1	0	X	X	1	0	1	0	1
0	1	1	1	1	0	X	X	X	1	0	0	0	1
0	1	1	1	0	X	X	X	X	0	1	1	0	1
0	1	1	0	X	X	X	X	X	0	1	0	0	1
0	1	0	X	X	X	X	X	X	0	0	1	0	1
0	0	X	X	X	X	X	X	X	0	0	0	0	1

Funcionamiento del circuito

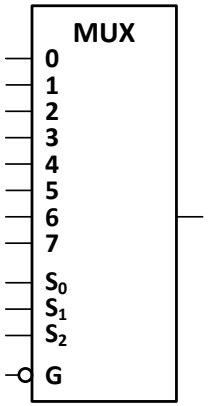
- El circuito sólo codifica cuando se aplica a su entrada de habilitación **EI** (*Enable Input*) un nivel bajo.
- Si el circuito está habilitado y no tiene ninguna entrada activa se activa la salida de habilitación **EO** *(Enable Output)*.
- Si el circuito está habilitado y alguna de las entradas está activa se activa la salida **GS** *(Group Signal)* y la combinación de salida $Y_2Y_1Y_0$ indica cual es la entrada activa con mayor valor decimal (en binario y de forma complementada).

Cuando se necesita establecer la prioridad entre un número de líneas superior al número de entradas de un circuito, es preciso realizar una asociación de codificadores.

El siguiente diagrama muestra el modo de obtener un codificador con prioridad alta de 16 a 4 líneas usando codificadores de prioridad alta de 8 a 3 líneas.



Un **multiplexor** es un circuito combinacional que posee **m** canales de entrada (numerados de 0 a m-1), **n** líneas de selección (tal que $m = 2^n$) y una salida, cuyo funcionamiento es tal que la combinación binaria aplicada a las líneas de selección determina cual es el canal de entrada cuya información aparece en la salida.



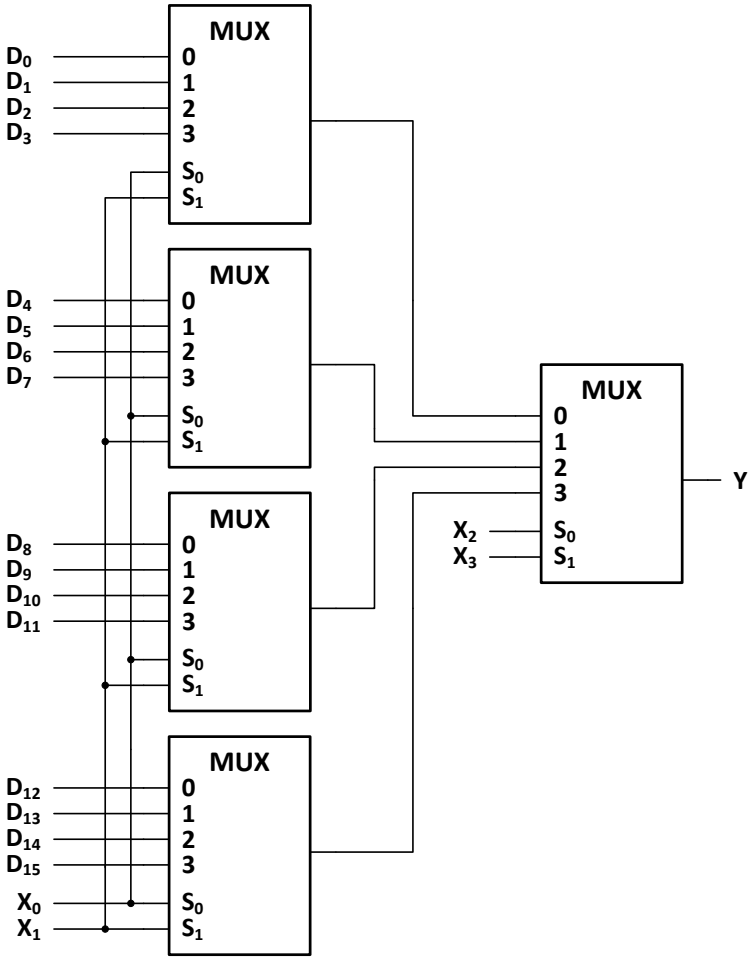
G	S_2	S_1	S_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Y_2
1	X	X	X	X	X	X	X	X	X	X	X	0
0	0	0	0	X	X	X	X	X	X	X	A	A
0	0	0	1	X	X	X	X	X	X	B	X	B
0	0	1	0	X	X	X	X	X	C	X	X	C
0	0	1	1	X	X	X	X	D	X	X	X	D
0	1	0	0	X	X	X	E	X	X	X	X	E
0	1	0	1	X	X	F	X	X	X	X	X	F
0	1	1	0	X	H	X	X	X	X	X	X	H
0	1	1	1	I	X	X	X	X	X	X	X	I

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

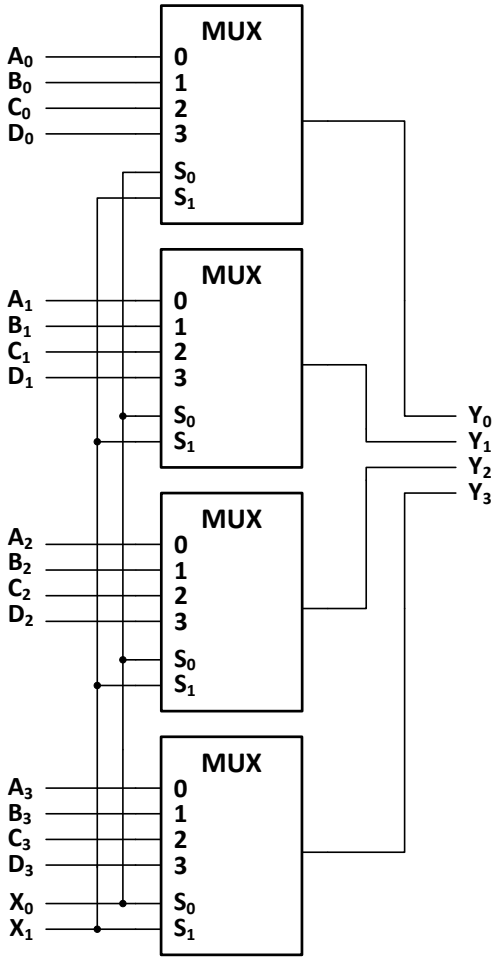
- **Multiplexor de 8 canales (74151).**
- **Multiplexor de 16 canales con salida negada (74150).**
- **Doble multiplexor de 4 canales con líneas de selección comunes (74153).**
- **Cuádruple multiplexor de dos canales con líneas de habilitación y de selección comunes (74157).**

La principal aplicación de los multiplexores es el envío a un único destino de la información procedente de varias fuentes. Los datos a seleccionar pueden ser de un solo bit o combinaciones de varios bits (2, 4, 8, etc.).

Existen dos razones por las que puede ser necesario asociar multiplexores: aumentar el número de canales o generar buses de información.



Aumento del número de canales



Generación de un bus de información

Otra aplicación de los multiplexores es la generación de funciones lógicas. Para implementar una función de **n** variables se necesita un multiplexor con sólo **n-1** líneas de selección.

Para generar funciones con multiplexores se pueden emplear dos métodos:

- **Método algebraico.**
- **Método tabular.**

Seguidamente se estudia el modo de aplicar el segundo de ellos.

Método tabular para la generación de funciones mediante multiplexores

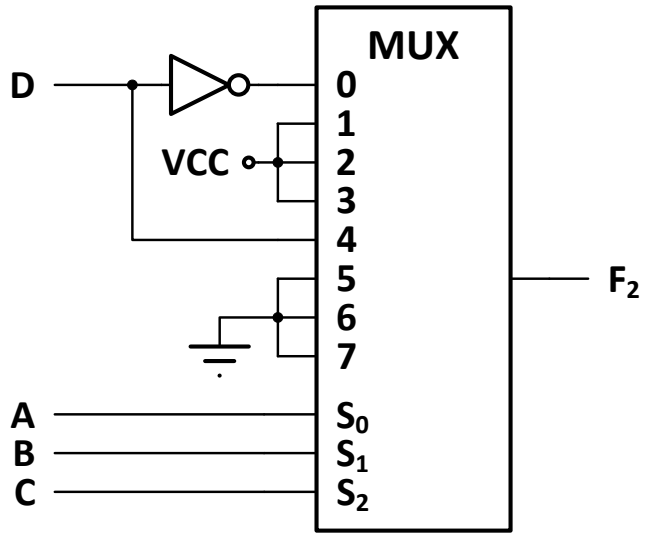
- **Se parte de una de las expresiones canónicas numéricas de la función a generar.**
- **Se representa un mapa con dos filas, asignando a las columnas las diferentes combinaciones de las variables de menos peso de la función, y a las filas los dos valores (0 y 1) de la variable de mayor peso.**
- **Las variables de las columnas se ordenan de mayor a menor peso y se codifican empleando el código binario natural.**
- **Cada columna representa una entrada de datos del multiplexor. Así pues, las variables asignadas a las columnas serán las que se conecten a las líneas de selección del multiplexor.**
- **Se representa dentro de cada casilla de la tabla el valor que posee la función para la combinación asociada a la misma.**
- **Se evalúa el contenido de cada columna, para determinar qué valor hay que aplicar al canal correspondiente del multiplexor, representándolo en la parte inferior de la misma:**
  - **Si una columna posee dos ”0”, o un ”0” y una ”X” se colocará un ”0”.**
  - **Si una columna posee dos ”1”, o un ”1” y una ”X” se colocará un ”1”.**
  - **Si una columna posee un ”0”, en la primera fila y un ”1” en la segunda se colocará la variable de mayor peso directa.**
  - **Si una columna posee un ”1”, en la primera fila y un ”0” en la segunda se colocará la variable de mayor peso negada.**
  - **Si una columna posee dos ”X”, se puede colocar ”0” o ”1” indistintamente.**
- **Finalmente se representa el símbolo del multiplexor, conectando a las líneas de selección las variables de menor peso de la función (teniendo en cuenta los pesos) y a los diferentes canales los valores indicados en la tabla debajo de cada columna.**

Ejemplo: Realizar la función **F (D, C, B, A) = $\sum_4$ (0, 1, 2, 3, 9, 11, 12) + $\sum_\phi$ (6, 7, 10, 15)** mediante un multiplexor del tamaño más adecuado.

Al tratarse de una función de 4 variables se necesitará un multiplexor de 8 canales (es decir, con tres líneas de selección).

	D	C	B	A	F
0	0	0	0	0	1
1	0	0	0	1	1
2	0	0	1	0	1
3	0	0	1	1	1
4	0	1	0	0	0
5	0	1	0	1	0
6	0	1	1	0	X
7	0	1	1	1	X
8	1	0	0	0	0
9	1	0	0	1	1
10	1	0	1	0	X
11	1	0	1	1	1
12	1	1	0	0	1
13	1	1	0	1	0
14	1	1	1	0	0
15	1	1	1	1	X

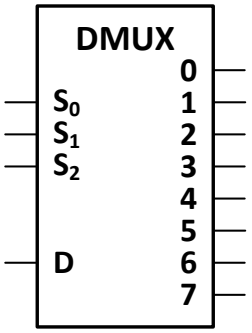
D \ CBA	000	001	010	011	100	101	110	111
0	1 (0)	1 (1)	1 (2)	1 (3)	0 (4)	0 (5)	X (6)	X (7)
1	0 (8)	1 (9)	X (10)	1 (11)	1 (12)	0 (13)	0 (14)	X (15)
	$\overline{D}$	1	1	1	D	0	0	X



Un **demultiplexor** es un sistema combinacional con una entrada de información, **n** líneas de selección y **m** salidas ($m = 2^n$) numeradas de 0 a m-1.

Su funcionamiento es tal que cuando se aplica una combinación a las líneas de selección, la información presente en la entrada de datos aparece en la salida cuyo valor decimal corresponde a dicha combinación. El resto de las salidas mantienen su valor por defecto, que puede ser "0" o "1".

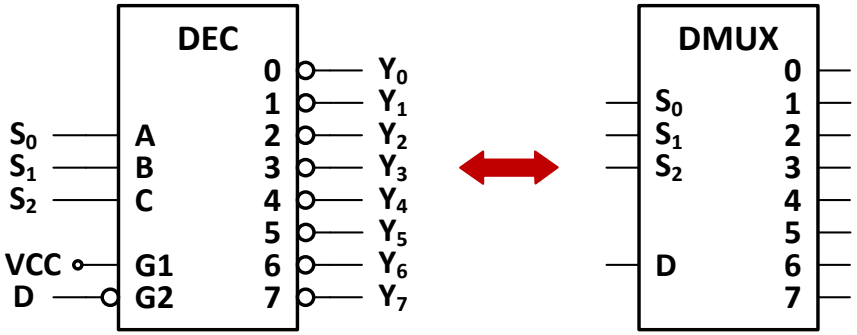
Símbolo y tabla de verdad de un demultiplexor de 1 a 8 líneas



S_2	S_1	S_0	D	Y_0	Y_1	Y_2	Y_3	Y_4	Y_5	Y_6	Y_7
0	0	0	E	E	0	0	0	0	0	0	0
0	0	1	E	0	E	0	0	0	0	0	0
0	1	0	E	0	0	E	0	0	0	0	0
0	1	1	E	0	0	0	E	0	0	0	0
1	0	0	E	0	0	0	0	E	0	0	0
1	0	1	E	0	0	0	0	0	E	0	0
1	1	0	E	0	0	0	0	0	0	E	0
1	1	1	E	0	0	0	0	0	0	0	E

Utilización de un decodificador como demultiplexor

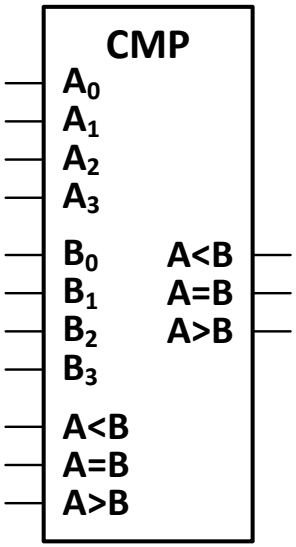
En realidad no existen demultiplexores integrados como tal, sino que se utilizan decodificadores como demultiplexores.



Un **comparador** es un circuito que indica si dos combinaciones expresadas en binario natural son iguales, y si no es así cual de ellas es mayor.

Existen comparadores para combinaciones de 4 bits y de 8 bits.

Comparador de 4 bits



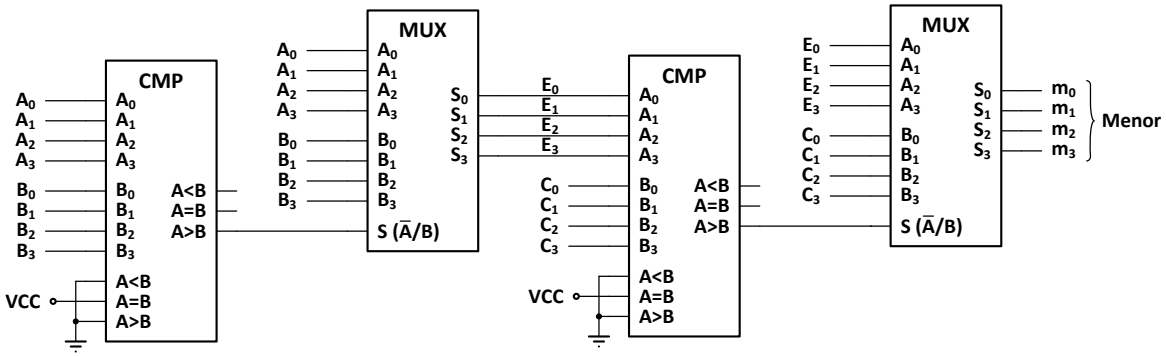
ENTRADAS				SALIDAS		
A - B	A < B	A = B	A > B	A < B	A = B	A > B
A < B	X	X	X	1	0	0
A > B	X	X	X	0	0	1
A = B	1	0	0	1	0	0
A = B	0	1	0	0	1	0
A = B	0	0	1	0	0	1

- **Si las combinaciones de entrada A y B son diferentes, se activarán las salidas A<B o A>B, según corresponda.**
- **Si las combinaciones de entrada son iguales, las salidas A<B, A=B y A>B tomarán los mismos valores que las entradas del mismo nombre**

Una de las aplicaciones de los comparadores es la de seleccionar combinaciones de forma automática en función de su valor en combinación con multiplexores.

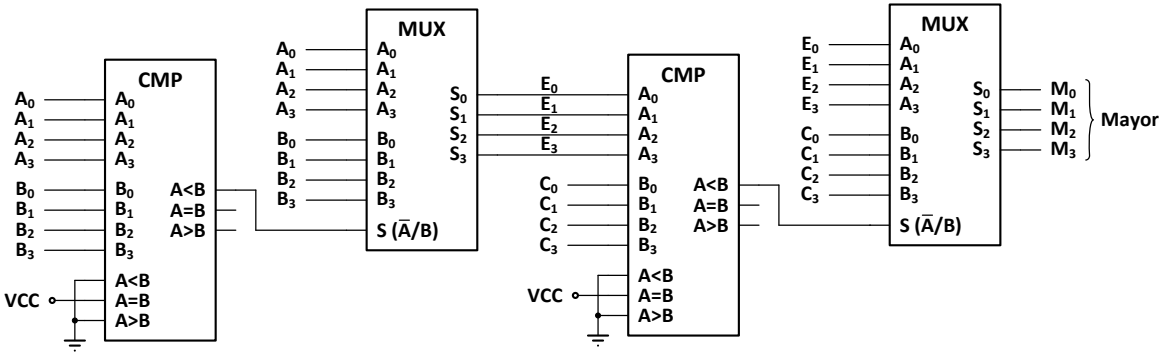
Selección de la combinación de menor valor

El circuito de la figura proporciona a su salida la combinación de menor valor entre las combinaciones de entrada **A**, **B** y **C**. Para ello se emplean dos comparadores de cuatro bits y dos multiplexores cuádruples de dos canales.



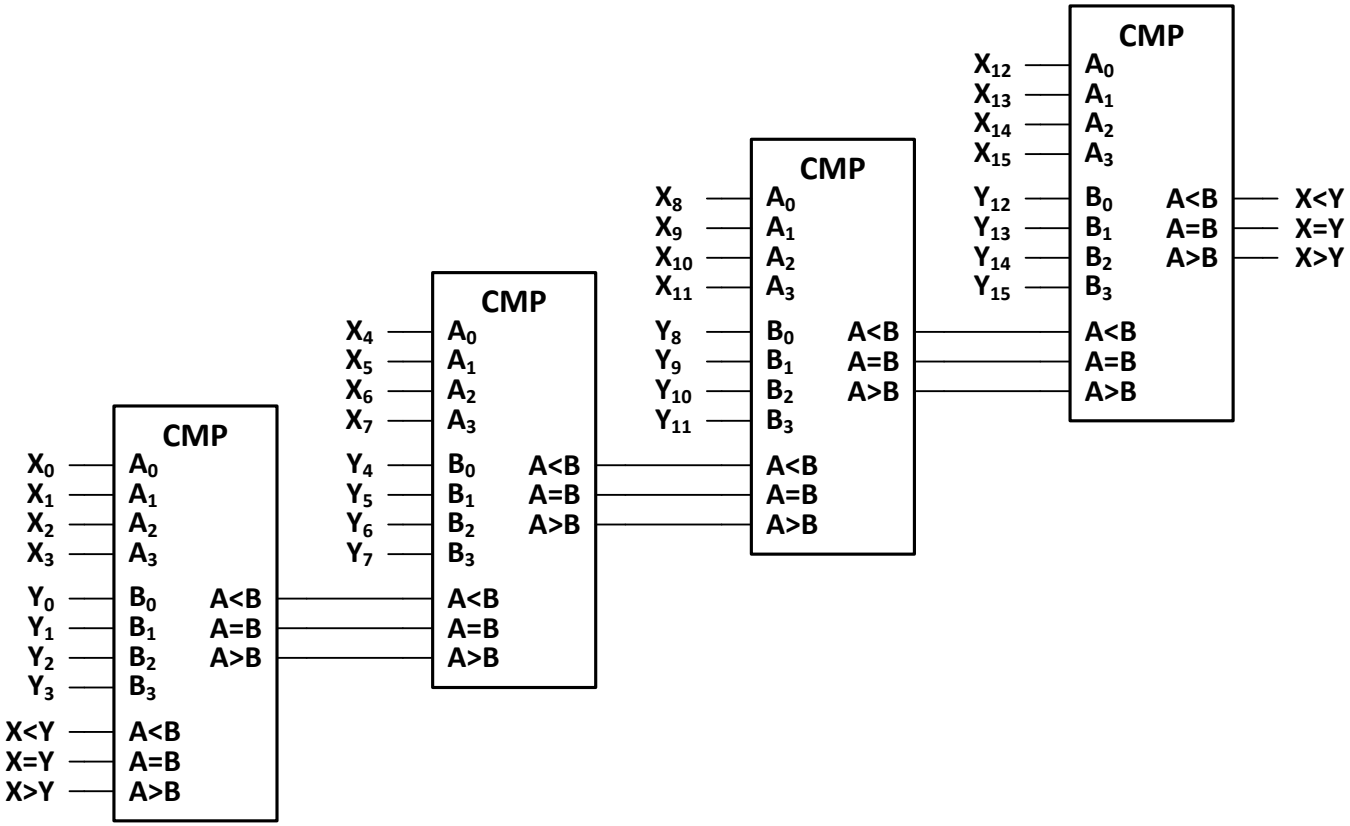
Selección de la combinación de mayor valor

El circuito de la figura proporciona a su salida la combinación de mayor valor entre las combinaciones de entrada **A**, **B** y **C**. Para ello se emplean dos comparadores de cuatro bits y dos multiplexores cuádruples de dos canales.



Los comparadores disponibles en el mercado suelen tener un tamaño de 4 bits, como el estudiado anteriormente. No obstante, si se necesita comparar combinaciones de una longitud superior, se puede obtener el comparador necesario mediante la conexión en cascada de varios de estos circuitos.

Para ello, las salidas de cada comparador se conectan a las entradas del mismo nombre del comparador que recibe los cuatro bits de peso inmediatamente superior de las combinaciones de entrada. Las entradas A<B, A=B y A>B del comparador resultante serán las correspondientes al comparador al que se aplican los cuatro bits de menos peso de ambas combinaciones, y las salidas del mismo serán las de aquel al que se aplican los 4 bits de más peso.



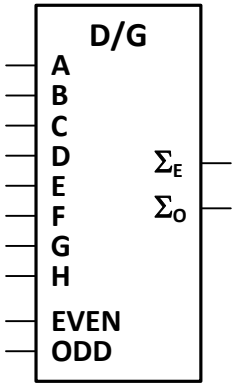
Detectores / generadores de paridad

En los sistemas digitales, sobre todo en aquellos capaces de detectar o corregir errores, puede ser necesario detectar la paridad de combinaciones binarias o generar bits de paridad a partir de las mismas.

La paridad de una combinación binaria viene determinada por el número de unos que ésta posee. Una combinación tiene paridad impar (**ODD**) cuando contiene un número impar de unos, y paridad par (**EVEN**) en caso contrario.

- **Detectar** la paridad de una combinación es averiguar si el número de unos que posee es par o impar. Cuando se **detecta paridad impar** sobre una combinación, el resultado será 1 si el número de unos de la misma es impar y 0 en caso contrario. Por el contrario, cuando se **detecta paridad par**, el resultado será 1 si el número de unos de la misma es par (incluyendo el 0) y 0 en caso contrario. Por tanto, el resultado de detectar paridad impar el es contrario de detectar paridad par, y viceversa.
- **Generar** paridad es obtener un nuevo bit a partir de una combinación dada de n bits, cuyo valor sea tal que al ser añadido a la misma, haga que la combinación de n+1 bits resultante posea una determinada paridad (par o impar). Cuando se **genera paridad impar** el número de unos de la combinación resultante debe ser impar, por lo que si la combinación de partida tiene paridad par el valor del bit generado debe ser 1 y si tiene paridad impar dicho valor debe ser 0. Por el contrario, cuando se **genera paridad par** el número de unos de la combinación resultante debe ser par, por lo que si la combinación de partida tiene paridad par el valor del bit generado debe ser 0 y si tiene paridad impar dicho valor debe ser 1. Por tanto, el resultado de generar paridad impar el es contrario de generar paridad par, y viceversa.

Para detectar y generar paridad en los sistemas digitales se suele hacer uso de unos circuitos combinacionales denominados **detectores/generadores de paridad**. Estos circuitos permiten, según se configuren, la detección o generación tanto de paridad par como impar.

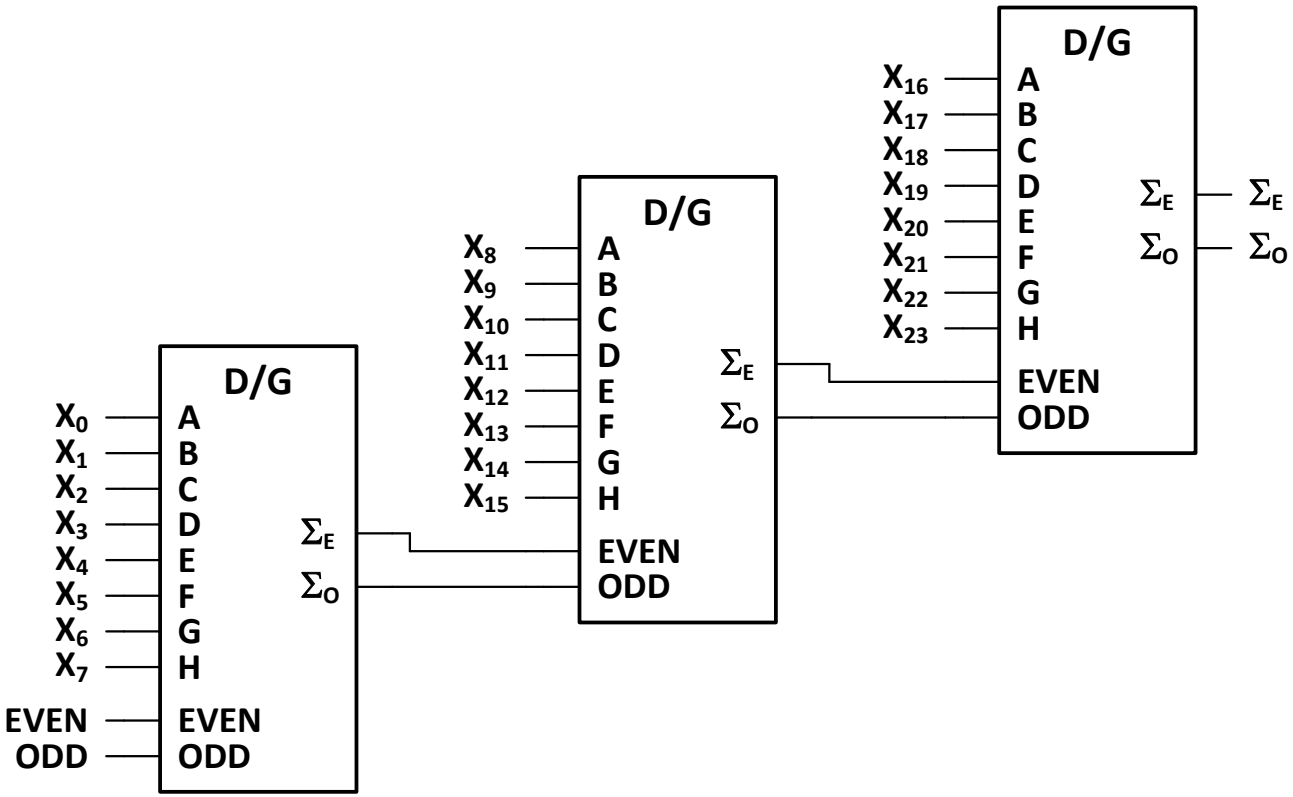


PARIDAD COMBINACIÓN ENTRADA	EVEN	ODD	Σ_E	Σ_O	MODO
PAR	1	0	1	0	DETECTOR
IMPAR	1	0	0	1	
PAR	0	1	0	1	GENERADOR
IMPAR	0	1	1	0	
X	0	0	1	1	NO FUNCIONA
X	1	1	0	0	

Además de servir para la configuración del circuito como detector o como generador de paridad, las entradas **EVEN** y **ODD** también permiten obtener un circuito con más entradas de datos mediante la asociación de varios bloques.

Para ello, las salidas par (Σ_E) e impar (Σ_O) de cada circuito deben conectarse, respectivamente, a las entradas par (**EVEN**) e impar (**ODD**) del siguiente.

Ejemplo: Obtener un detector / generador de paridad para combinaciones de 24 bits.



Usaremos en esta asignatura el lenguaje VHDL para describir sistemas digitales, aunque el lenguaje permite la descripción de sistemas de cualquier tipo. Las siguientes son características básicas del lenguaje:

- **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, es decir, 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* (abreviatura de *standard logic*, lógica estándar), 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.**

Los 5 primeros tipos de valores son los más usados, y en general con ellos puede describirse cualquiera de los circuitos que diseñaremos en la asignatura.

Un conjunto de señales (bits) 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.

➢ **L**os 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 (***library***) y de los paquetes dentro de la librería que se usarán (*use*) y que definen tales tipos y sus operaciones. Esto se sitúa al principio del código:

```
library ieee;
use ieee.std_logic_1164.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_unsigned** 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.*

Las operaciones básicas entre señales se muestran a continuación:

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
```

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)**: Es la parte del módulo VHDL donde se declaran los terminales del circuito a diseñar, es decir, las entradas, salidas y, en algunos casos, las líneas bidireccionales de E/S. Los puertos (**port**) declarados en la entidad pueden ser de entrada (**in**), de salida (**out**), bidireccionales (**inout**) y adaptados (**buffer**), estos últimos menos usados que los anteriores.

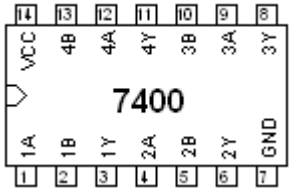
```
entity nombre_de_la_entidad is
  port( declaración de entradas y salidas
  );
end nombre_de_la_entidad;
```

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

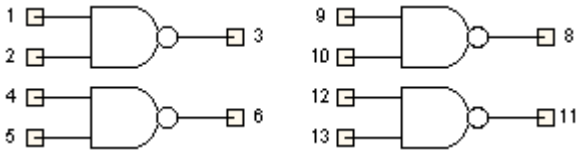
```
arquitecture nombre_de_la_arquitectura of nombre_de_la_entidad is
  signal declaración de señales internas
  begin
        descripción del funcionamiento (asignaciones)
end nombre_de_la_arquitectura;
```

Consideremos un circuito integrado sencillo como el **7400**, que contiene cuatro puertas **NAND** de dos entradas. 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;
```

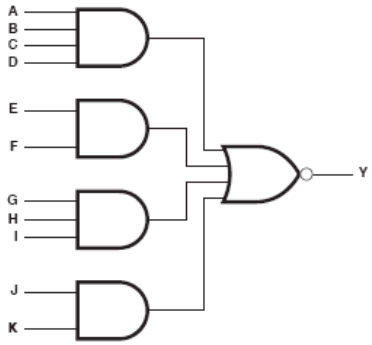
Modele en VHDL el circuito **7464**, cuyo diagrama lógico y expresión algebraica se muestra a continuación.

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;
```

Se pueden emplear señales internas (que se definen en la parte de la arquitectura, antes del *begin*) para definir los puntos de entrada a la puerta NOR. De esta forma es más fácil definir la función completa y queda más fácil de leer.

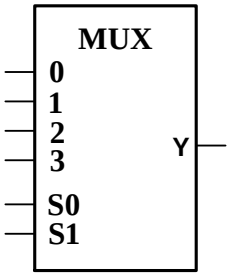
Nota: En este ejemplo hemos empleado señales internas que nos facilitan la descripción del circuito: a0, a1, a2 y a3 son las salidas de cada una de las puertas AND.

Las asignaciones concurrentes en VHDL 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;



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_4a1 is
    port ( c0, c1, c2, c3, s1, s0: in std_logic;
           y : out std_logic);
end mux_4a1;

architecture interior of mux_4a1 is
    signal control: std_logic_vector (1 downto 0);
    begin
        control <= s1 & s0;
        y <=    c0 when control = "00" else
                c1 when control = "01" else
                c2 when control = "10" else
                c3;
end interior;
```

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

entity mux_4a1 is
    port ( c0, c1, c2, c3: in std_logic;
           s: in std_logic_vector (1 downto 0);
           y : out std_logic);
end mux_4a1;

architecture interior of mux_4a1 is
    signal control: integer range 0 to 3;
begin
    control <= conv_integer (s);
    with control select
        y <=    c0 when 0,
                c1 when 1,
                c2 when 2,
                c3 when others;
end interior;
```

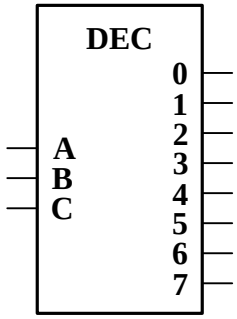
Descripción VHDL de un decodificador de 3 a 8 líneas

❑ 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;
```

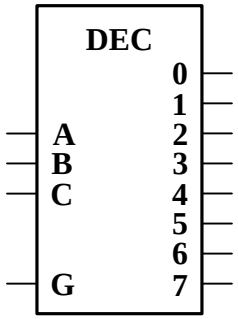
❑ Asignaciones concurrentes múltiples

```
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;
```



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;
```



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.

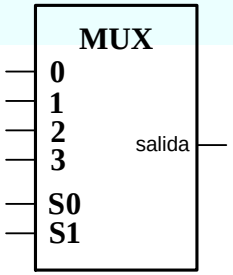
Condicionales

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

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.



Descripción en VHDL usando asignaciones secuenciales de un multiplexor

- Asignaciones secuenciales condicionales

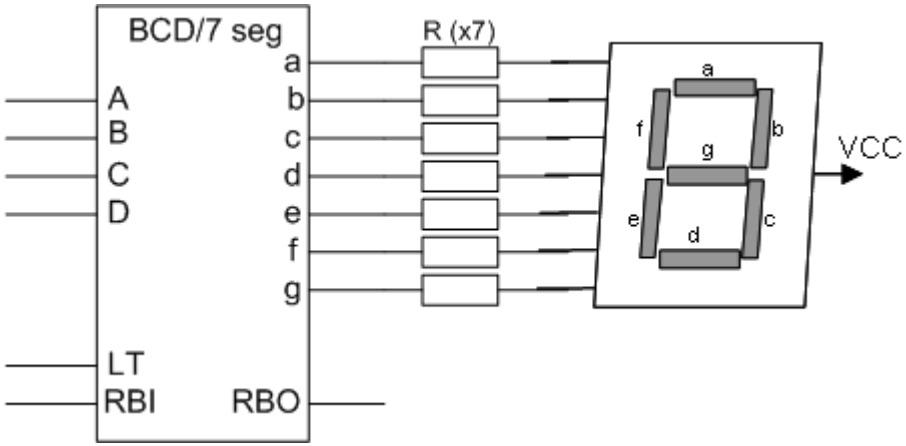
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

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;
                        elsif control = 1 then salida <= c1;
                        elsif control = 2 then salida <= c2;
                        else salida <= c3; end if;
            end process;

end estruct_mux4a1sec_c;
```

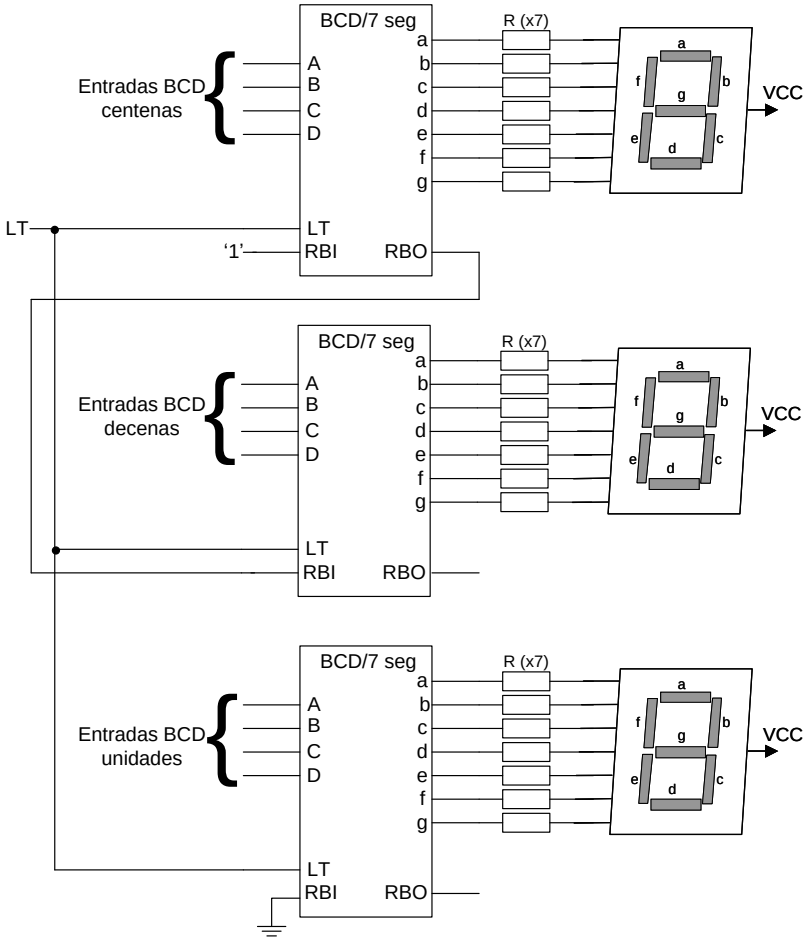
- Asignaciones secuenciales múltiples

```
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;
```

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.





- 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:

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

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

- 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);
```

- 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';
```

- 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);
        begin
                p1: process (a, b, p)
                variable p: std_logic_vector (3 downto 0);
                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 la señal "p" sólo recibe la 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**.

- **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 i = p then salida(i) <= '1';
          else salida(i) <='0';
          end if;
      end loop;
end process;
```

- **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) <= '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.