Escuela de Ingenierías Industriales

Titulación: **Grado en Ingeniería Electrónica, Robótica y Mecatrónica**

Asignatura: **Electrónica Digital**

Tema 6: **Bloques Funcionales Combinacionales**

ANDALUCÍA TECH
Campus de Excelencia Internacional
International Campus of Excellence

TEMA 6: BLOQUES FUNCIONALES COMBINACIONALES

6.1. Bloques para el procesamiento y enrutado de datos.

6.1.1 Decodificadores. Codificadores. Conversores de código.

6.1.2 Multiplexores y demultiplexores.

6.1.3 Comparadores

6.2. Bloques aritméticos.

6.2.1 Sumadores binarios: semisumador y sumador completo. Sumador de n bits con acarreo en serie. Sumador de n bits con acarreo anticipado. Sumador/restador.

6.2.2 Multiplicadores binarios.

6.2.3 ALU: Unidad aritmético lógica

6.3. Bloques Reconfigurables. Lógica programable.

6.3.1 Generadores Universales de Funciones Booleanas: ROM, PLA PAL

6.4. Bloques funcionales en HDL

Bloques para el procesamiento de datos

Decodificadores

Un decodificador es un circuito combinacional con **n** entradas y **m** salidas.

Cada una de las combinaciones de entrada es una palabra de un código.

Cada una de las salidas tiene asignado un nº de orden en el intervalo 0 a m-1.

Para cada palabra de código sólo se activa una de las líneas de salida, la correspondiente al nº de orden de la palabra de código en la entrada.

Según sea el código de entrada se habla de diferentes tipos de decodificadores.

Ejemplos:

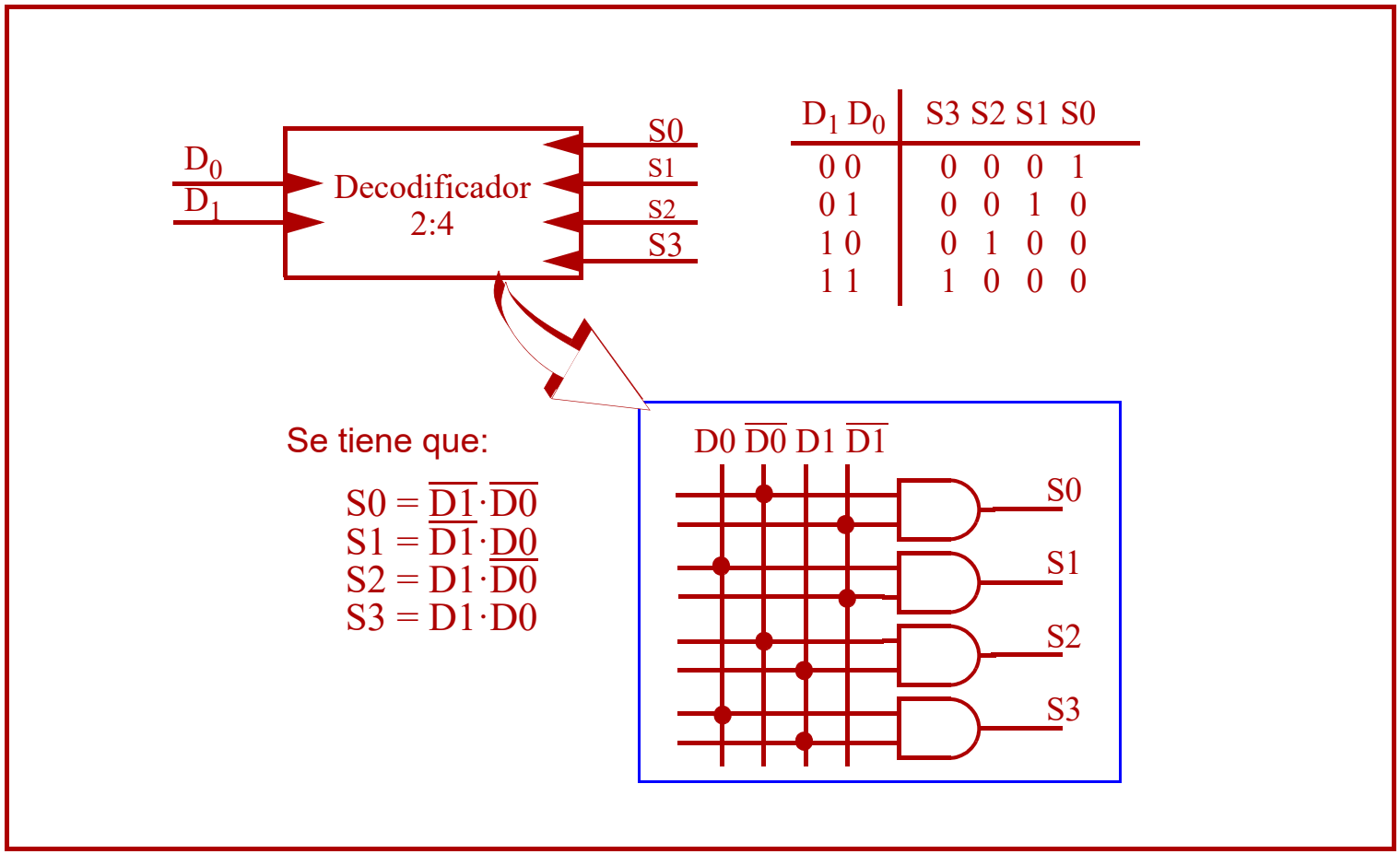
- Si el código de entrada es binario, se habla de **decodificador binario**. Se cumple que $\mathbf{m = 2^n}$.
- Si el código de entrada es BCD, **n** ha de ser múltiplo de cuatro, y se cumple que $\mathbf{m = 2^{4N}}$, con **N** igual al nº de digitos BCD del código.

Las principales aplicaciones de los decodificadores son:

- Decodificación de códigos.
- Direccionamiento de posiciones de memoria.
- Implementación de demultiplexores.
- Implementación de funciones booleanas.

Bloques para el procesamiento de datos. Decodificadores

Ejemplo: Decodificador binario 2:4



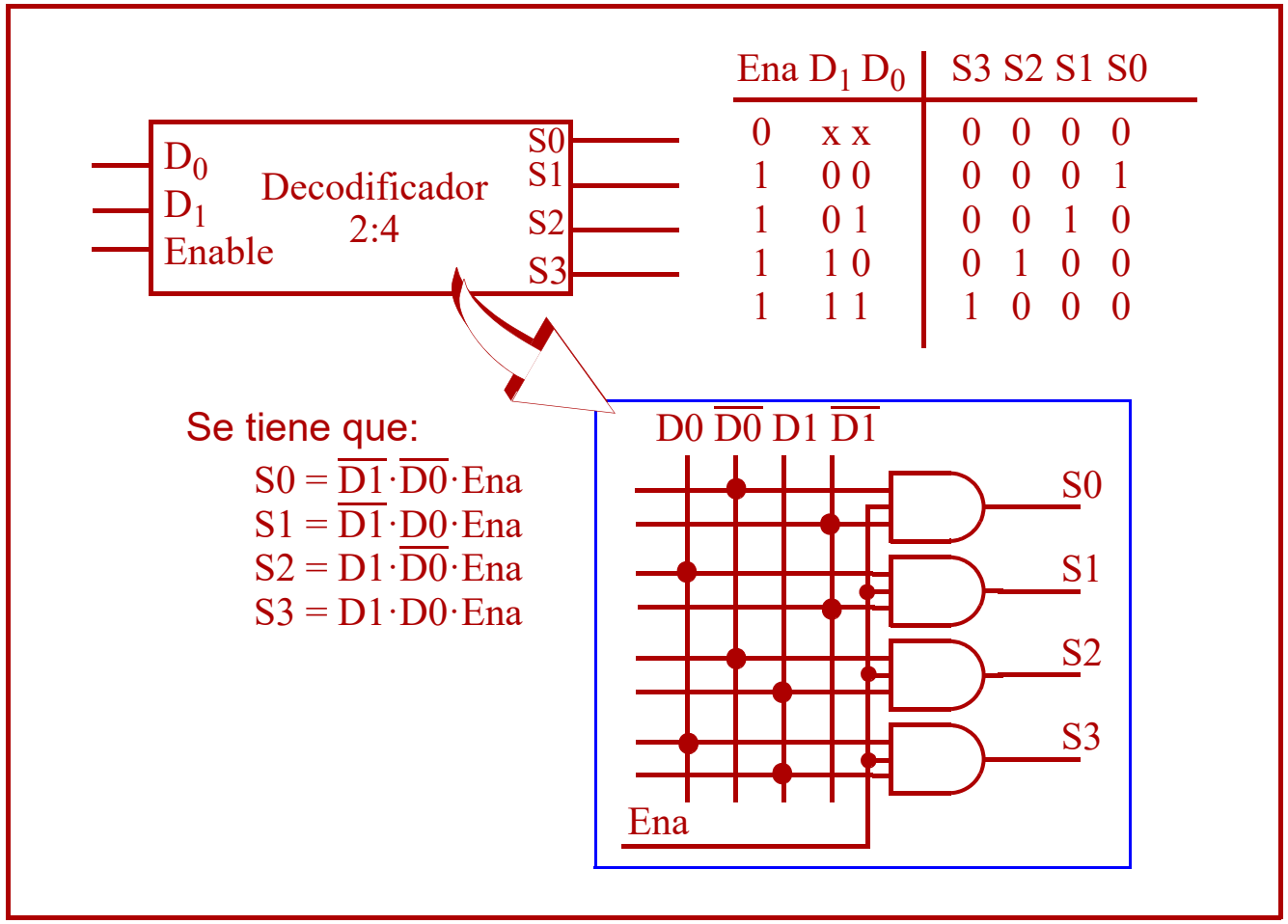
Cada salida es un minitérmino de dos variables

Bloques para el procesamiento de datos. Decodificadores

- Los decodificadores pueden incorporar una línea de habilitación o “**enable**”.

  Esta entrada, en su estado “ no activo ” deshabilita el funcionamiento como decodificador y fuerza a la salida un valor determinado, independientemente del código de entrada.
  Así se facilita la expansión modular.

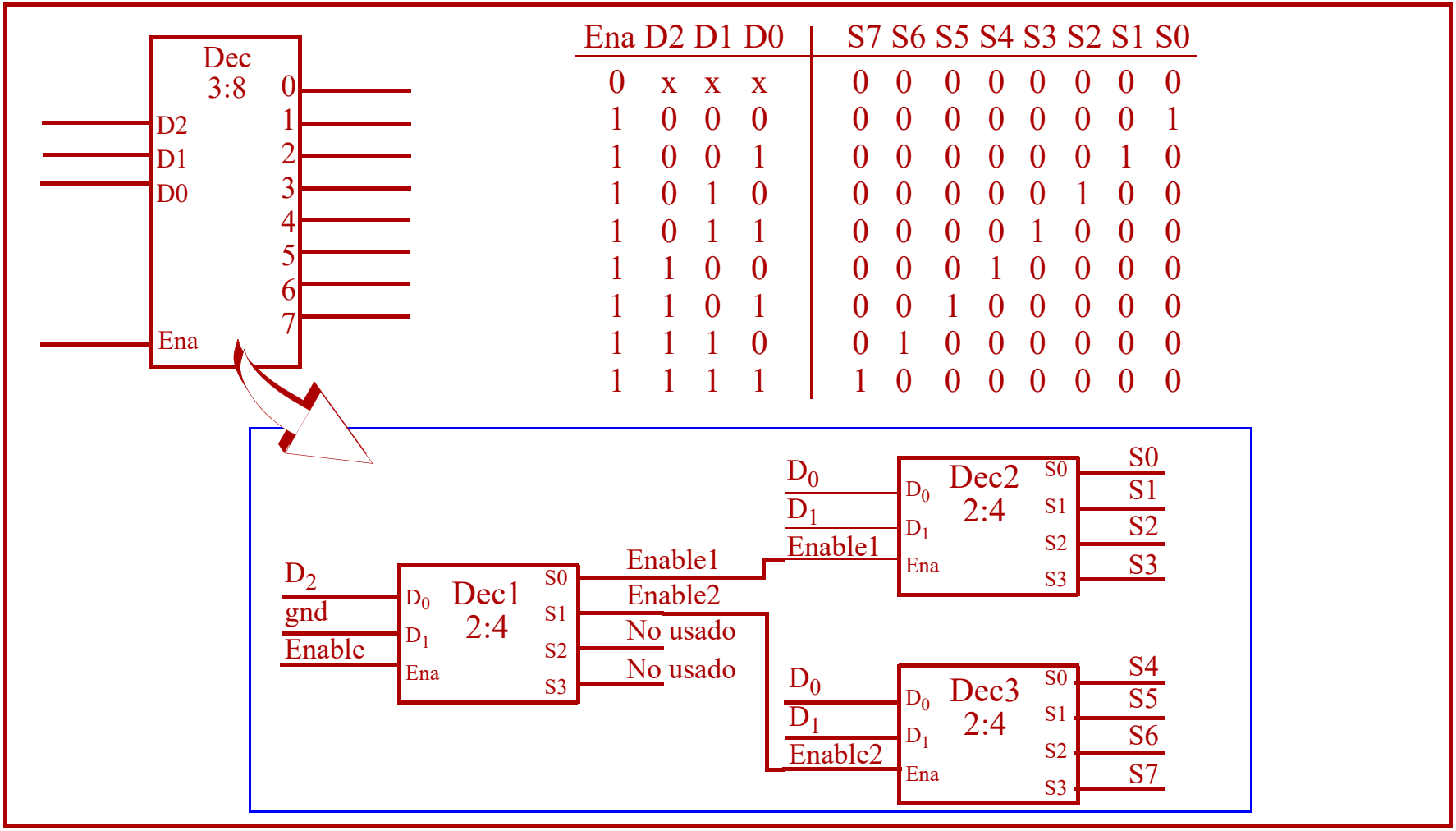
  Ejemplo: Decodificador binario 2:4 con entrada de habilitación



Bloques para el procesamiento de datos. Decodificadores

- La entrada de habilitación es útil para construir decodificadores de palabras de mayor nº de bits. a partir de decodificadores más sencillos.

  En el ejemplo se construye un decodificador 3:8 con entrada de habilitación (**enable**) a partir de decodificadores 2:4 con entrada de habilitación



Bloques para el procesamiento de datos. Decodificadores

Implementación de funciones booleanas

Un decodificador de **n** variables implementa todos los minterms de **n** variables, por tanto cualquier función booleana de **n** variables puede ser implementada mediante un decodificador n: 2^n y puertas OR. Además, cada miniterm puede ser usado para realizar distintas funciones booleanas, por lo que la afirmación anterior puede extenderse a multifunciones booleanas.

Ejemplo: Generar simultáneamente las funciones NAND, NOR y XOR de tres variables.

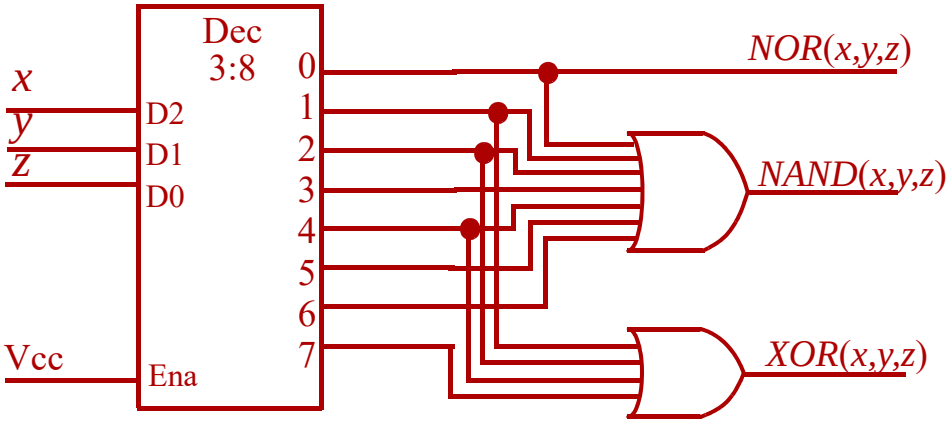
Esto es posible utilizando un decodificador 3:8 y las puertas OR adecuadas.

Dado que: $NAND(x,y,z) = \sum_3(0,1,2,3,4,5,6)$ $NOR(x,y,z) = \sum_3(0)$ y $XOR(x,y,z) = \sum_3(1,2,4,7)$

y que el decodificador 3:8 se comporta según la tabla de verdad:

Ena	D2	D1	D0	S7	S6	S5	S4	S3	S2	S1	S0
0	x	x	x	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	1
1	0	0	1	0	0	0	0	0	0	1	0
1	0	1	0	0	0	0	0	0	1	0	0
1	0	1	1	0	0	0	0	1	0	0	0
1	1	0	0	0	0	0	1	0	0	0	0
1	1	0	1	0	0	1	0	0	0	0	0
1	1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0	0	0

Es fácil verificar el correcto funcionamiento de este diseño



Bloques para el procesamiento de datos. Codificadores

- Un codificador es un circuito combinacional con **m** entradas y **n** salidas.
  Cada una de las combinaciones de salida corresponde a una palabra de un código.
  Cada una de las entradas tiene asignado un nº de orden en el intervalo 0 a m-1.
  Cada palabra de código se asocia a una de las líneas de entrada.
  Un codificador realiza la función inversa de un decodificador.

Según sea el código de salida
se habla de diferentes tipos de codificadores.

Ejemplos:

- Si el código de salida es binario,
  se habla de **codificador binario**.
  En este caso se cumple que $\mathbf{m = 2^n}$.

- Si el código de salida es BCD, **n** ha de ser
  múltiplo de cuatro y se cumple que $\mathbf{m = 2^{4N}}$,
  con **N** igual al nº de dígitos BCD del código.

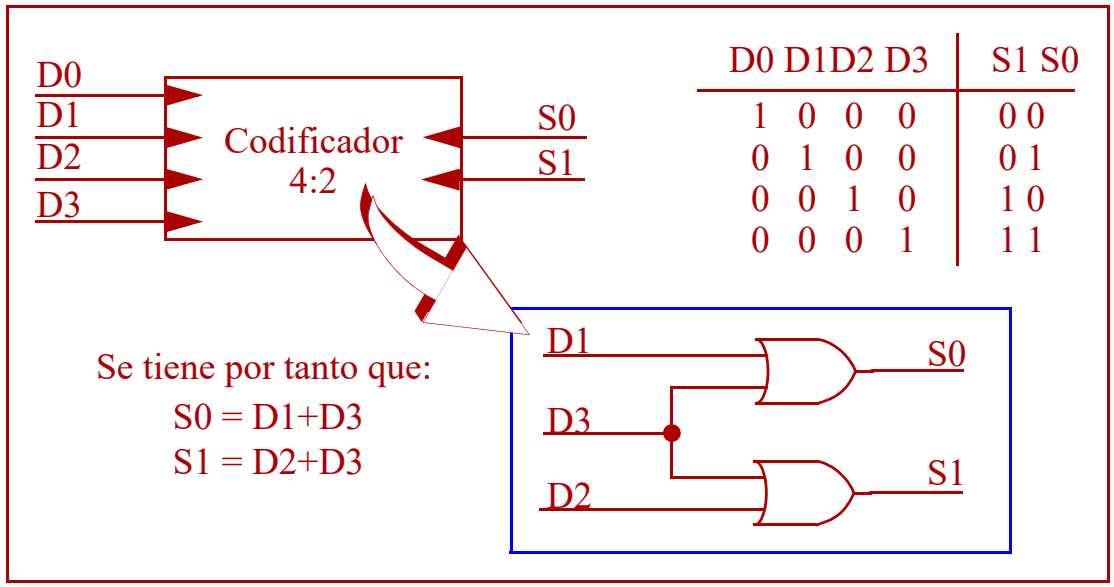
Las principales aplicaciones
de los codificadores son:

- Conversores analógico-digital.
- Conversores de código.
- Asignación de prioridad a distintos elementos que piden un recurso.
- Codificación de información en general.

Bloques para el procesamiento de datos. Codificadores

Codificador sin prioridad

Ej: Codificador binario 4:2



Es un circuito simple pero presenta dos características que a veces pueden ser inconvenientes:

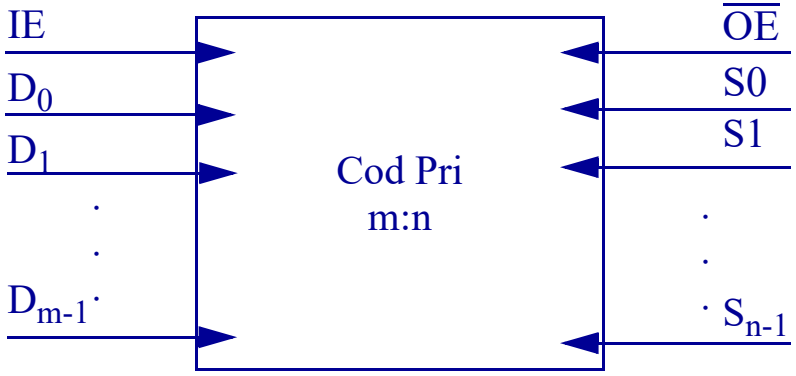
- La salida no distingue entre entrada D0 activa y ninguna entrada activa.

- Funciona correctamente si se garantiza que sólo se activa una entrada cada vez.

Codificador sin prioridad

Bloques para el procesamiento de datos. Codificadores

Codificador de prioridad, entrada de habilitación y validación salida.



Permite que varias de sus entradas puedan estar activas a la vez en cualquier momento, mostrando a la salida el código de la entrada a la que se ha asignado mayor prioridad.

Los codificadores de prioridad se clasifican en:

- **Codificadores con prioridad creciente**, aquellos para los que la entrada m es menos prioritaria que la m+1.
- **Codificadores con prioridad decreciente**, aquellos para los que la entrada m es más prioritaria que la m+1.

Un bloque codificador estándar incorpora dos líneas adicionales.

Una entrada de habilitación IE (normalmente activa a nivel alto)

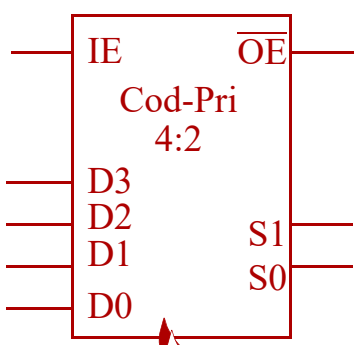
Una validación de código de salida $\overline{OE}$ (habitualmente activa a nivel bajo)

Estas señales resuelven el problema de ambiguedad entre D0 activa y ninguna entrada activa, además de facilitar la expansión modular.

Bloques para el procesamiento de datos. Codificadores

Codificador de prioridad, entrada de habilitación y validación salida.

Ejemplo: Codificador 4:2 con prioridad ascendente, entrada (**IE**), y salida (**$\overline{OE}$**).

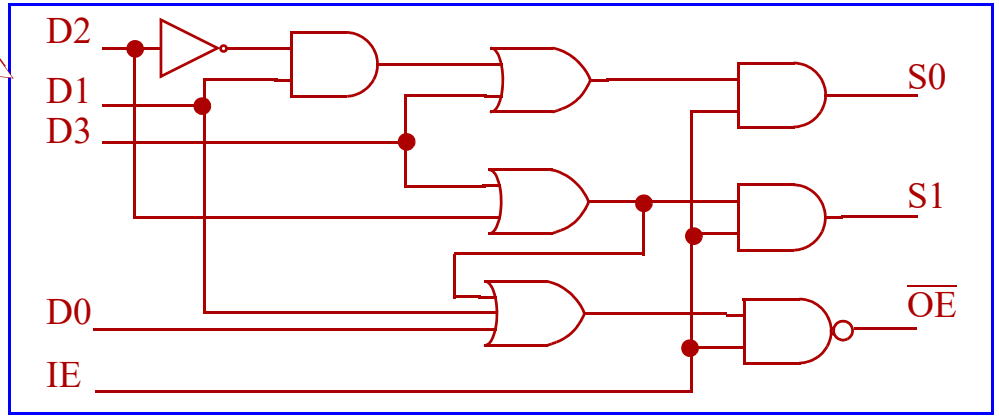


IE	D0	D1	D2	D3	$\overline{OE}$	S1	S0
0	x	x	x	x	1	0	0
1	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0
1	x	1	0	0	0	0	1
1	x	x	1	0	0	1	0
1	x	x	x	1	0	1	1

Se tiene que:

$$\overline{OE} = \overline{(D0 + D1 + D2 + D3)\cdot EI}$$

$$S1 = (D2 + D3)\cdot IE \qquad S0 = (D1\cdot \overline{D2} + D3)\cdot IE$$



IE (Input Enable): Si está activa (en este caso nivel 1) permite que el circuito codifique.

Si está inactiva la señal **$\overline{OE}$** está inactiva, (en este caso valor 1) y S1S0 deben ser ignoradas.

$\overline{OE}$ (Output Enable): Si esta activa (en este caso nivel 0) indica que en S1S0 hay un palabra de código válida.

Si está inactiva, (en este caso valor 1) el estado de las salidas no es válido.

Bloques para el procesamiento de datos. Codificadores

Expansión modular

Ejemplo: Construir un codificador 8:3 con prioridad ascendente a partir de codificadores 4:2 como el anterior

Símbolo del bloque

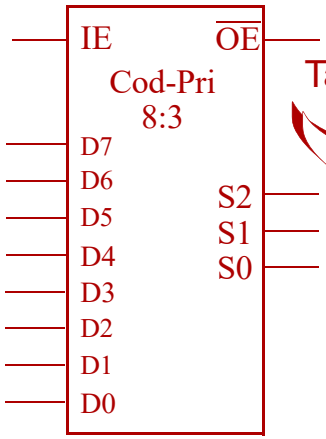


Tabla de verdad

IE	D0	D1	D2	D3	D4	D5	D6	D7	$\overline{OE}$	S2	S1	S0
0	x	x	x	x	x	x	x	x	1	0	0	0
1	0	0	0	0	0	0	0	0	1	0	0	0
1	**1**	**0**	**0**	**0**	0	0	0	0	**0**	**0**	**0**	**0**
1	**x**	**1**	**0**	**0**	0	0	0	0	**0**	**0**	**0**	**1**
1	**x**	**x**	**1**	**0**	0	0	0	0	**0**	**0**	**1**	**0**
1	**x**	**x**	**x**	**1**	0	0	0	0	**0**	**0**	**1**	**1**
1	x	x	x	x	**1**	**0**	**0**	**0**	**0**	**1**	**0**	**0**
1	x	x	x	x	**x**	**1**	**0**	**0**	**0**	**1**	**0**	**1**
1	x	x	x	x	**x**	**x**	**1**	**0**	**0**	**1**	**1**	**0**
1	x	x	x	x	**x**	**x**	**x**	**1**	**0**	**1**	**1**	**1**

Es fácil ver que:

$IE1 = IE \qquad D[3:0] = D[7:4]$

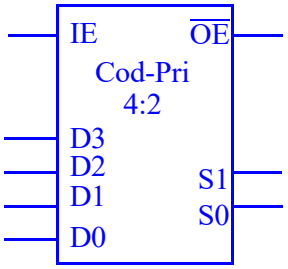
$IE0 = IE \qquad D[3:0] = D[3:0]$

$\overline{OE} = \overline{OE1} \cdot \overline{OE0}$

$S2 = \overline{\overline{OE1}}$

$S1 = \overline{OE1} \cdot S1 + \overline{\overline{OE1}} \cdot S1$

$S0 = \overline{OE1} \cdot S0 + \overline{\overline{OE1}} \cdot S0$



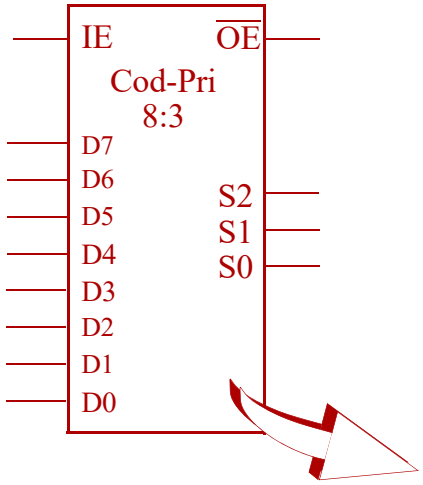
IE	D0	D1	D2	D3	$\overline{OE}$	S1	S0
0	x	x	x	x	1	0	0
1	0	0	0	0	1	0	0
1	**1**	**0**	**0**	**0**	0	0	0
1	**x**	**1**	**0**	**0**	0	0	1
1	**x**	**x**	**1**	**0**	0	1	0
1	**x**	**x**	**x**	**1**	0	1	1

IE	D0	D1	D2	D3	D4	D5	D6	D7	$\overline{OE0}$	$\overline{OE1}$	S1	S0	S1	S0	$\overline{OE}$	S2	S1	S0
0	x	x	x	x	x	x	x	x	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1	0	0	0
1	**1**	**0**	**0**	**0**	0	0	0	0	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
1	**x**	**1**	**0**	**0**	0	0	0	0	**0**	**1**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**1**
1	**x**	**x**	**1**	**0**	0	0	0	0	**0**	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**1**	**0**
1	**x**	**x**	**x**	**1**	0	0	0	0	**0**	**1**	**1**	**1**	**0**	**0**	**0**	**0**	**1**	**1**
1	x	x	x	x	**1**	**0**	**0**	**0**	**x**	**0**	**x**	**x**	**0**	**0**	**0**	**1**	**0**	**0**
1	x	x	x	x	**x**	**1**	**0**	**0**	**x**	**0**	**x**	**x**	**0**	**1**	**0**	**1**	**0**	**1**
1	x	x	x	x	**x**	**x**	**1**	**0**	**x**	**0**	**x**	**x**	**1**	**0**	**0**	**1**	**1**	**0**
1	x	x	x	x	**x**	**x**	**x**	**1**	**x**	**0**	**x**	**x**	**1**	**1**	**0**	**1**	**1**	**1**

Bloques para el procesamiento de datos. Codificadores

Expansión modular

Ejemplo: Construir un codificador 8:3 con prioridad ascendente a partir de codificadores 4:2 como el anterior



$IE1 = IE$ $\quad D[3:0] = D[7:4]$

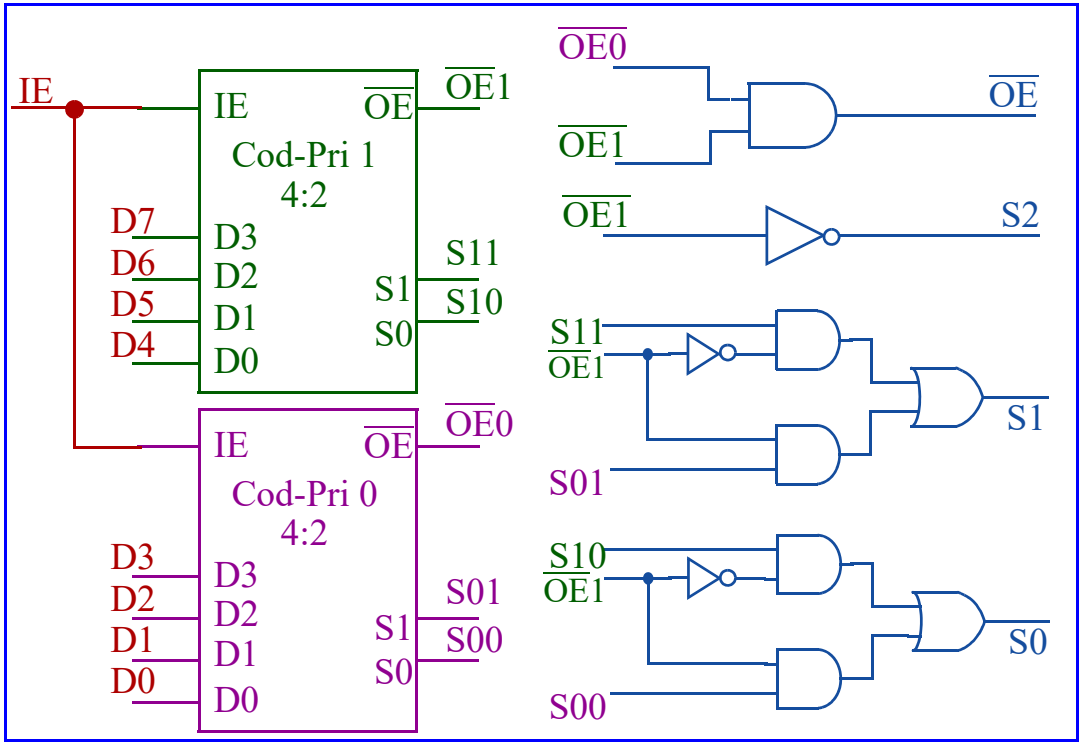
$IE0 = IE$ $\quad D[3:0] = D[3:0]$

$\overline{OE} = \overline{OE1} \cdot \overline{OE0}$

$S2 = \overline{\overline{OE1}}$

$S1 = \overline{OE1} \cdot S1 + \overline{\overline{OE1}} \cdot S1$

$S0 = \overline{OE1} \cdot S0 + \overline{\overline{OE1}} \cdot S0$



Bloques para el procesamiento de datos. Convertidores de código

- Son sistemas utilizados para traducir información codificada en un código a otro distinto.

Algunos de los más usados son convertidores:

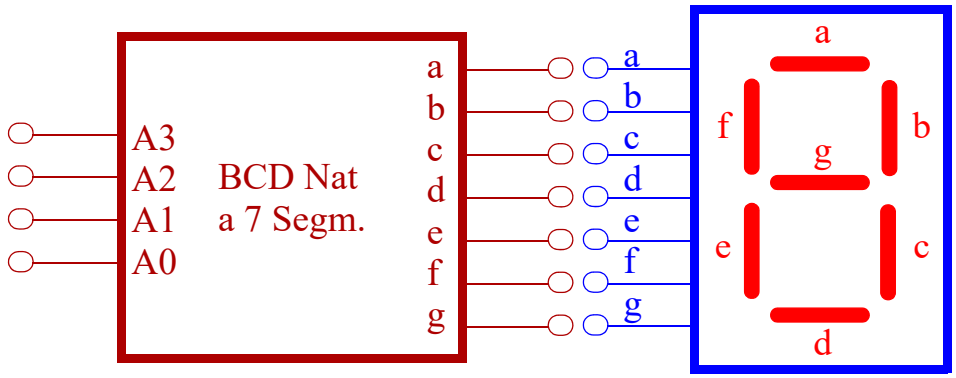
Binario natural - BCD natural
Binario natural - BCD exceso3
Signo magnitud - Binario complento a dos.
BCD natural - Display 7 segmentos

Para implementar un convertidor de código se tienen dos opciones:

a) Conectar en cascada un decod. del código de entrada y un cod. del código de salida.

b) Implementar la correspondiente función booleana.

Ejemplo: Conversor BCD natural - Display 7 segmentos



Digito BCD	a b c d e f g
0	1 1 1 1 1 1 0
1	0 1 1 0 0 0 0
2	1 1 0 1 1 0 1
3	1 1 1 1 0 0 1
4	0 1 1 0 0 1 1
5	1 0 1 1 0 1 1
6	1 0 1 1 1 1 1
7	1 1 1 0 0 0 0
8	1 1 1 1 1 1 1
9	1 1 1 0 0 1 1

Bloques para el procesamiento de datos. Multiplexores

- Un multiplexor es un circuito combinacional que posee ***n*** entradas de dato, una salida **S** y ***c*** entradas de control, tal que $n = 2^c$; de modo que cada combinación de las entradas de control permiten seleccionar y presentar a la salida una de las entradas de dato.
  Se nombran Multiplexor (n x 1), o bien MUX*c*.

- Los bloques multiplexores pueden incorporar también una entrada adicional de habilitación, **Enable**. Y algunos de ellos proporcionan además de la salida S, la salida $\overline{S}$.

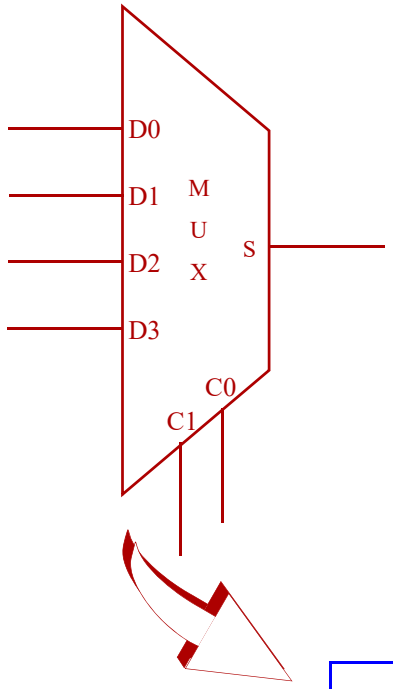
- Los bloque multiplexores pueden asociarse para realizar expansiones bien en serie, aumentando el nº de señales de control, y por tanto el nº de líneas de entrada de dato; bien en paralelo, conservando el nº de señales de control y aumentando el tamaño de la palabra en cada línea de entrada de dato.

- Las aplicaciones básicas son:

  - Selección de datos,
  - Interconexión o enrutado de datos,
  - Implementación de funciones booleanas.

Bloques para el procesamiento de datos. Multiplexores

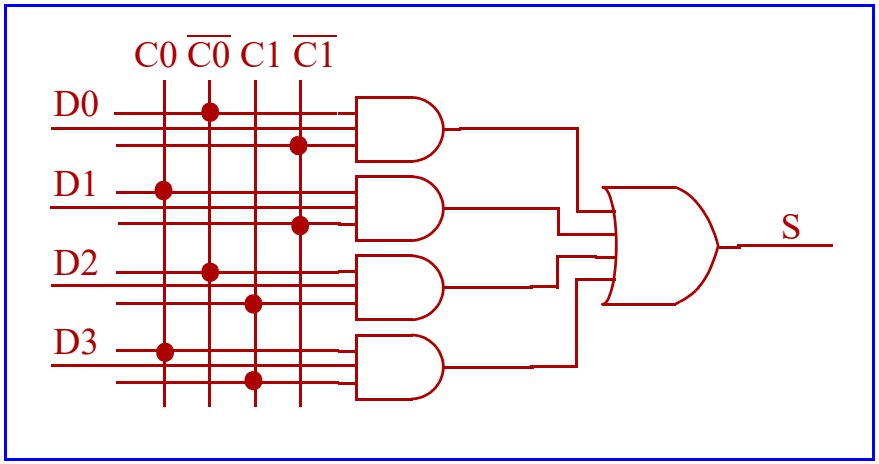
- Ejemplo: Multiplexor 4x1, o MUX2



C_1 C_0	S
0 0	D0
0 1	D1
1 0	D2
1 1	D3

Por tanto:

$$S = \overline{C1}\cdot\overline{C0}\cdot D0 + \overline{C1}\cdot C0\cdot D1 + C1\cdot\overline{C0}\cdot D2 + C1\cdot C0\cdot D3$$

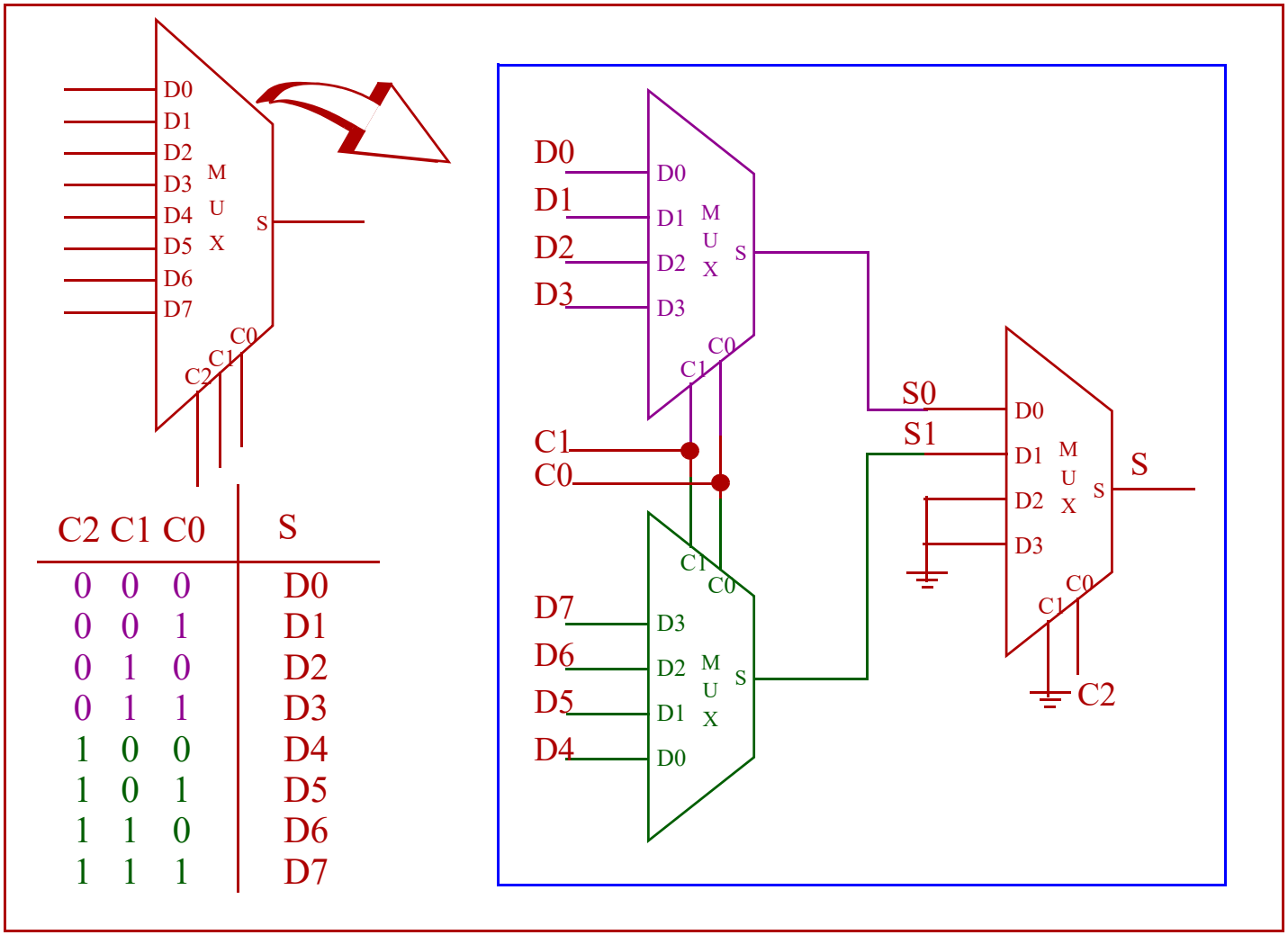


Bloques para el procesamiento de datos. Multiplexores

Expansión modular de bloques multiplexores. Asociación en serie

Aumenta el nº de señales de control, y por tanto el nº de entradas de dato

Ejemplo: Asociación de multiplexores 4x1 (MUX2) para formar un Multiplexor 8x1 (MUX3)



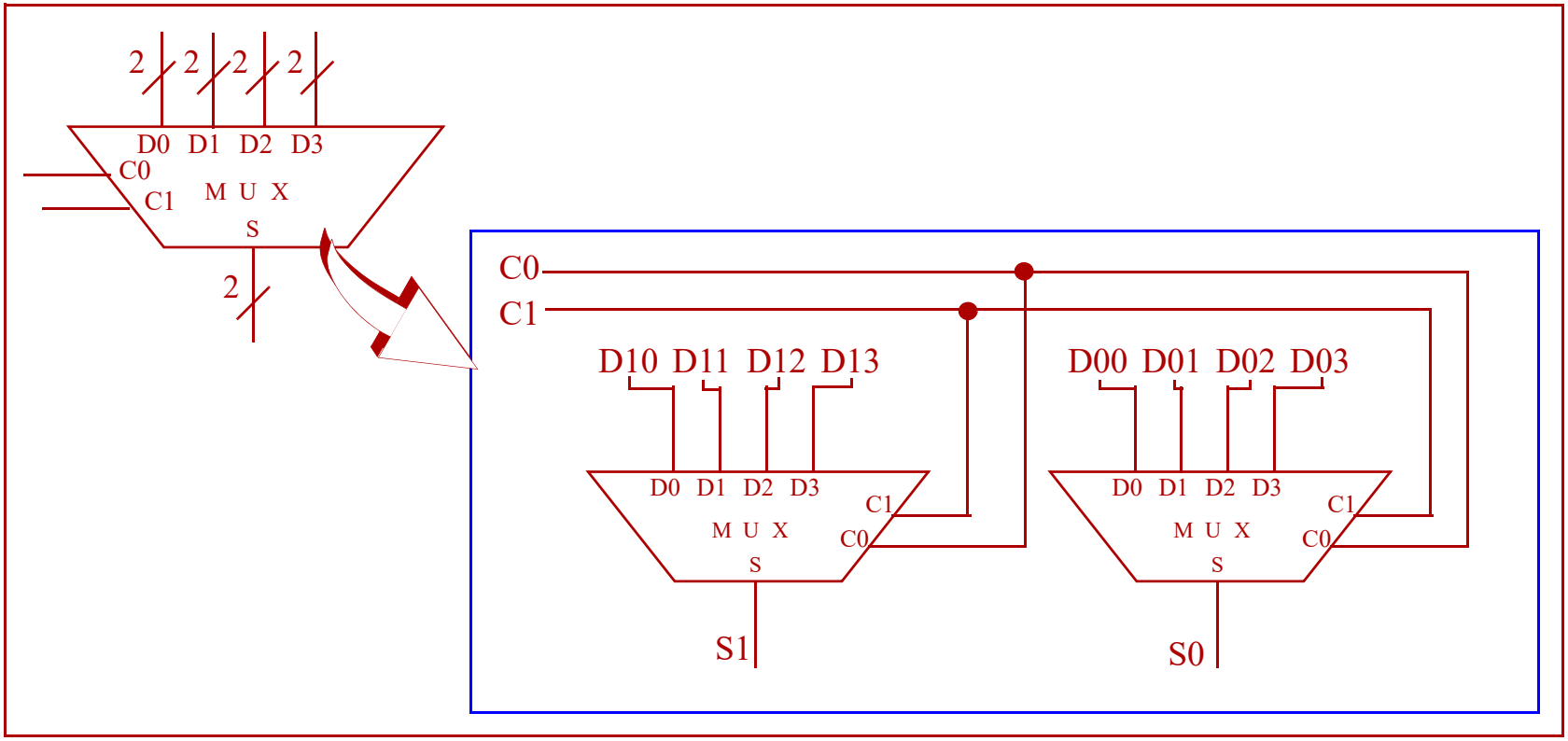
Con esta estructura es directo construir un MUX4, multiplexor 16x1

Bloques para el procesamiento de datos. Multiplexores

Expansión modular de bloques multiplexores. Asociación en paralelo

Aumenta el tamaño de palabra en las líneas de entrada de dato

Ejemplo: Asociación de multiplexores 4x1 (MUX2) para formar un MUX2 para palabras de dos bits



Con esta estructura es directo construir un MUX2, para palabras de n bits

Bloques para el procesamiento de datos. Multiplexores

Implementación de funciones booleanas

La estructura general de un **Multiplexor nx1** se corresponde con una AND-OR en dos niveles, con **n** puertas AND de **c+1** entradas y una puerta OR de **n** entradas.

Esta estructura se corresponde con la forma canónica SDP de cualquier función booleana.

$$F(x_1, x_2, \ldots, x_n) = \sum_{0}^{2^n - 1} (a_i \cdot m_i) \qquad F(x_1, x_2, \ldots, x_N) = \sum_{N} (i, j, k, \ldots, p)$$

Por tanto un multiplexor de ***c*** señales de control (MUXc) es un bloque muy apropiado para realizar cualquier función booleana de **c** variables.

Para ello:

Cada variable de entrada se asocia a una de las entradas de control, de esta forma, cada combinación de control selecciona un minitérmino m_i de **c** variables.

El valor de la función para cada una de estas combinaciones se concreta asignando el valor 1 ó 0 a la entrada de dato seleccionada, según indica su tabla de verdad o forma canónica (a_i).

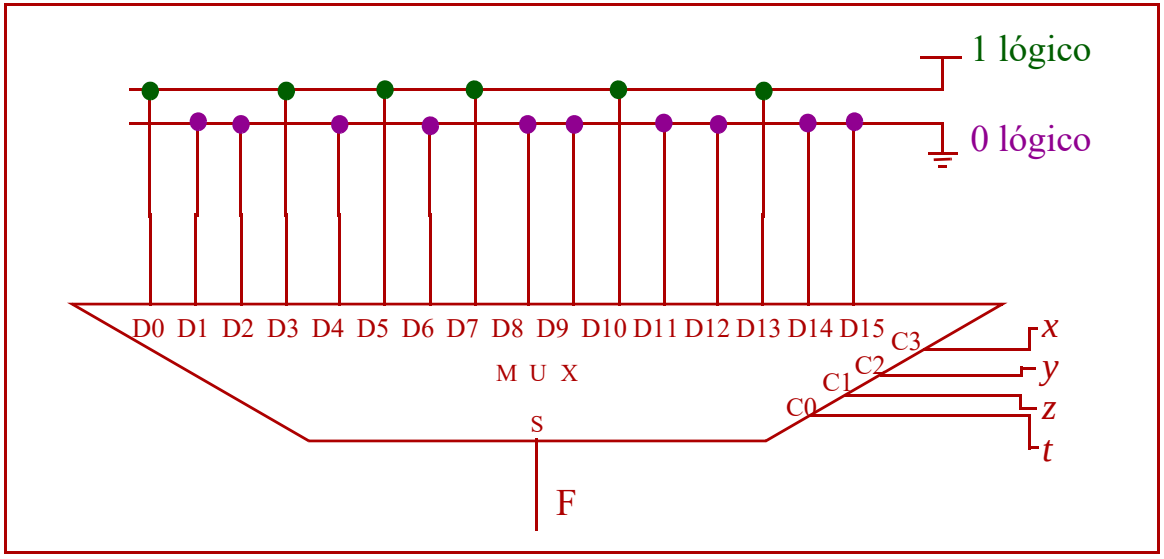
Bloques para el procesamiento de datos. Multiplexores

Implementación de funciones booleanas

Ejemplo: Implementa utilizando el multiplexor adecuado la función booleana:

$$F(x, y, z, t) = \sum_{4}(0, 3, 5, 7, 10, 13)$$

- Será necesario utilizar MUX4, puesto que la función es de cuatro variables
- Conectamos las variables de la función a las entradas de control de modo que la variable más significativa señal (x en este caso) corresponda a la entrada de control más significativa (C3 en este caso) y así sucesivamente.
- Conectamos a 1 lógico las entradas de dato correspondientes a las combinaciones de control que aparecen en la expresión canónica de la función.
- Conectamos a 0 lógico el resto de las entradas de dato.



Bloques para el procesamiento de datos. Multiplexores

Implementación de funciones booleanas

Los bloques multiplexores simples resultan muy útiles como bloques constructivos de funciones booleanas complejas (de un nº de entradas elevado), junto con alguna puerta lógica adicional.

Ejemplo: Implementa utilizando un MUX3 y menor cantidad posible de puertas lógicas la función booleana:

$$F(x, y, z, t, u) = \sum_{5}(1, 3, 5, 9, 10, 11, 12, 13, 14, 15, 18, 19, 21, 23, 25, 26, 27, 28, 29, 30, 31)$$

En el mapa de Karnaugh las combinaciones de variables ztu seleccionan una columna (D0 a D7).

$$S = \bar{z}\cdot\bar{t}\cdot\bar{u}\cdot D0 + \bar{z}\cdot\bar{t}\cdot u\cdot D1 + \bar{z}\cdot t\cdot\bar{u}\cdot D2 + \bar{z}\cdot t\cdot u\cdot D3 + z\cdot\bar{t}\cdot\bar{u}\cdot D4 + z\cdot\bar{t}\cdot u\cdot D5 + z\cdot t\cdot\bar{u}\cdot D6 + z\cdot t\cdot u\cdot D7$$

Cada columna es a su vez una función booleana de las variables x e y.

Podemos usar un MUX 3 controlado por las variables ztu, para seleccionar una de dichas funciones y conectar en cada una de las entradas de dato D_i , la salida de una función de las variables D_i (x,y)

	D0	D1	D3	D2	D6	D7	D5	D4
ztu / xy	*000*	*001*	*011*	*010*	*110*	*111*	*101*	*100*
00	0 (0)	1 (1)	1 (3)	0 (2)	0 (6)	0 (7)	1 (5)	0 (4)
01	0 (8)	1 (9)	1 (11)	1 (10)	1 (14)	1 (15)	1 (13)	1 (12)
11	0 (24)	1 (25)	1 (27)	1 (26)	1 (30)	1 (31)	1 (29)	1 (28)
10	0 (16)	0 (17)	1 (19)	1 (18)	0 (22)	1 (23)	1 (21)	0 (20)

$$D0(x, y) = 0 \qquad D4(x, y) = y$$
$$D1(x, y) = \bar{x} + y \qquad D5(x, y) = 1$$
$$D2(x, y) = x + y \qquad D2(x, y) = y$$
$$D3(x, y) = 1 \qquad D7(x, y) = x + y$$

Bloques para el procesamiento de datos. Multiplexores

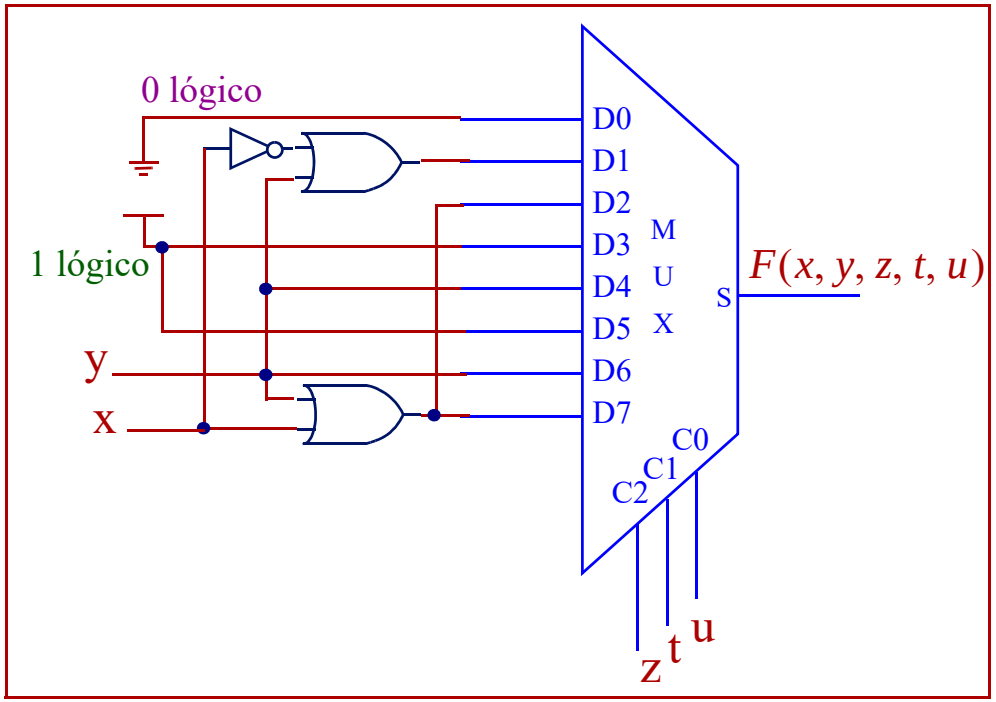
Implementación de funciones booleanas

Ejemplo: (continuación

$$S = \bar{z}\cdot\bar{t}\cdot\bar{u}\cdot D0 + \bar{z}\cdot\bar{t}\cdot u\cdot D1 + \bar{z}\cdot t\cdot\bar{u}\cdot D2 + \bar{z}\cdot t\cdot u\cdot D3 + z\cdot\bar{t}\cdot\bar{u}\cdot D4 + z\cdot\bar{t}\cdot u\cdot D5 + z\cdot t\cdot\bar{u}\cdot D6 + z\cdot t\cdot u\cdot D7$$

	D0	D1	D3	D2	D6	D7	D5	D4
xy \ *ztu*	*000*	*001*	*011*	*010*	*110*	*111*	*101*	*100*
00	0 (0)	1 (1)	1 (3)	0 (2)	0 (6)	0 (7)	1 (5)	0 (4)
01	0 (8)	1 (9)	1 (11)	1 (10)	1 (14)	1 (15)	1 (13)	1 (12)
11	0 (24)	1 (25)	1 (27)	1 (26)	1 (30)	1 (31)	1 (29)	1 (28)
10	0 (16)	0 (17)	1 (19)	1 (18)	0 (22)	1 (23)	1 (21)	0 (20)

$$D0(x,y) = 0 \qquad D4(x,y) = y$$
$$D1(x,y) = \bar{x} + y \qquad D5(x,y) = 1$$
$$D2(x,y) = x + y \qquad D6(x,y) = y$$
$$D3(x,y) = 1 \qquad D7(x,y) = x + y$$



Ejercicio: Implementa esta función con un Mux4 y un inversor.

Bloques para el procesamiento de datos. Demultiplexores

- Un demultiplexor es un circuito combinacional que posee una entrada de dato, ***m*** salidas de datos y ***c*** entradas de control, tal que $2^c \geq m \geq 2^{c-1}$, y que seleccionan una sola salida de dato para presentar el valor de la entrada.

  Un demultiplexor realiza la operación contraria al multiplexor encauzando los datos de una fuente común a diversos destinos.

  Se nombran demultiplexor (1 x m), o bien DEMUXc.

- Pueden incorporar también una entrada adicional de habilitación, **Enable**.

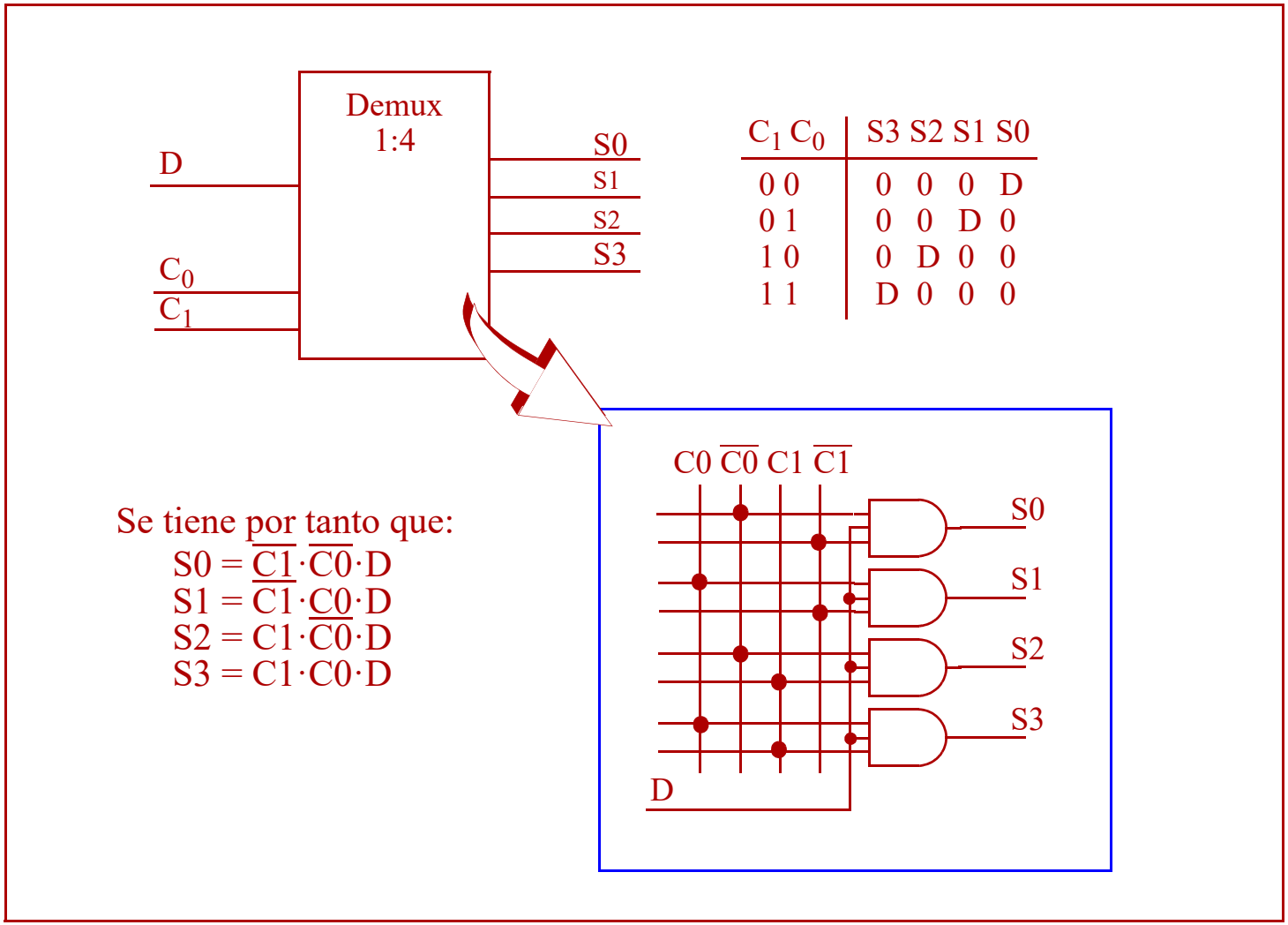
- Los bloque multiplexores pueden asociarse para realizar expansiones bien en serie, aumentando el nº de señales de control, y por tanto el nº de líneas de salida de dato; bien en paralelo, conservando el nº de señales de control y aumentando el tamaño de la palabra en cada línea de entrada/salida de dato.

- Las aplicaciones básicas son:

  - Selección de datos,
  - Interconexión o enrutado de datos
  - Implementación de decodificadores

Bloques para el procesamiento de datos. Demultiplexores

- Ejemplo: Demultiplexor 1x4, o DEMUX2

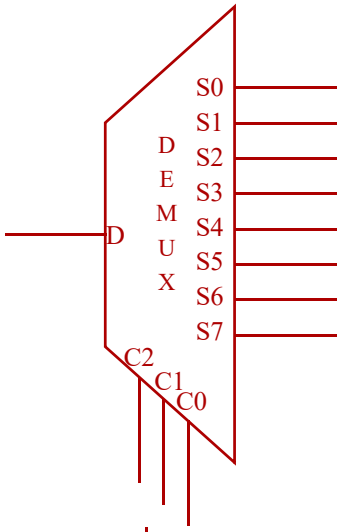


Bloques para el procesamiento de datos. Demultiplexores

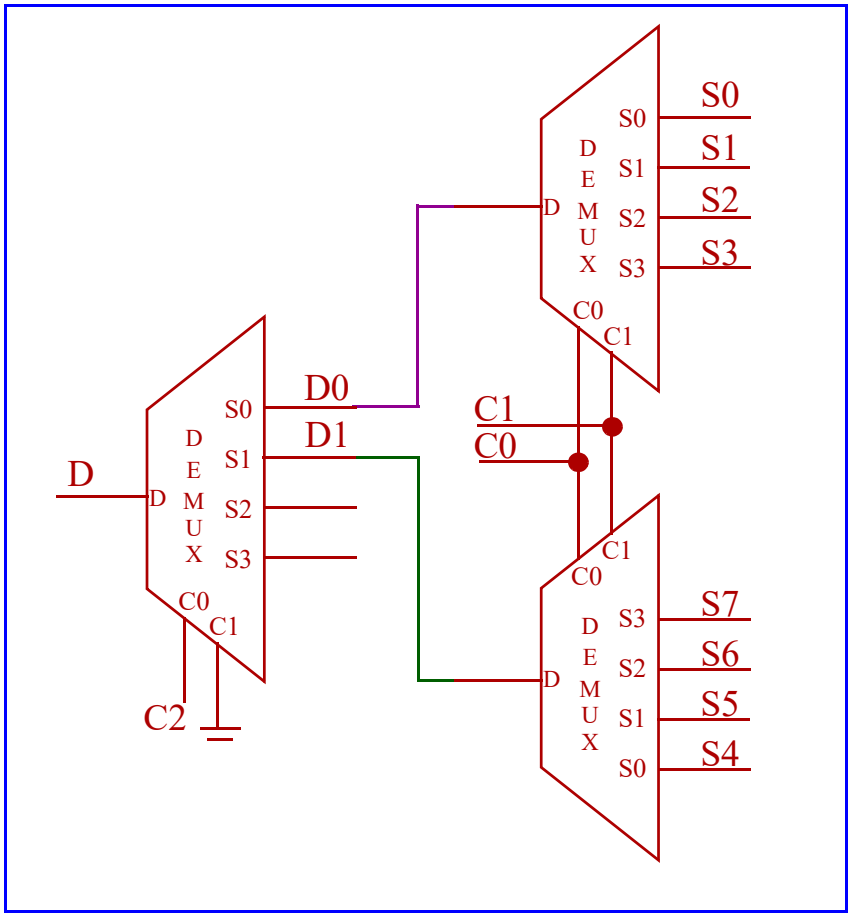
Expansión modular de bloques demultiplexores. Asociación en serie

Aumenta el nº de señales de control, y por tanto el nº de salidas de dato

Ejemplo: Asociación de demultiplexores 1x4 (DEMUX2) para formar un Demultiplexor 1x8 (DEMUX3)



C2	C1	C0	S0	S1	S2	S3	S4	S5	S6	S1
0	0	0	D	0	0	0	0	0	0	0
0	0	1	0	D	0	0	0	0	0	0
0	1	0	0	0	D	0	0	0	0	0
0	1	1	0	0	0	D	0	0	0	0
1	0	0	0	0	0	0	D	0	0	0
1	0	1	0	0	0	0	0	D	0	0
1	1	0	0	0	0	0	0	0	D	0
1	1	1	0	0	0	0	0	0	0	D



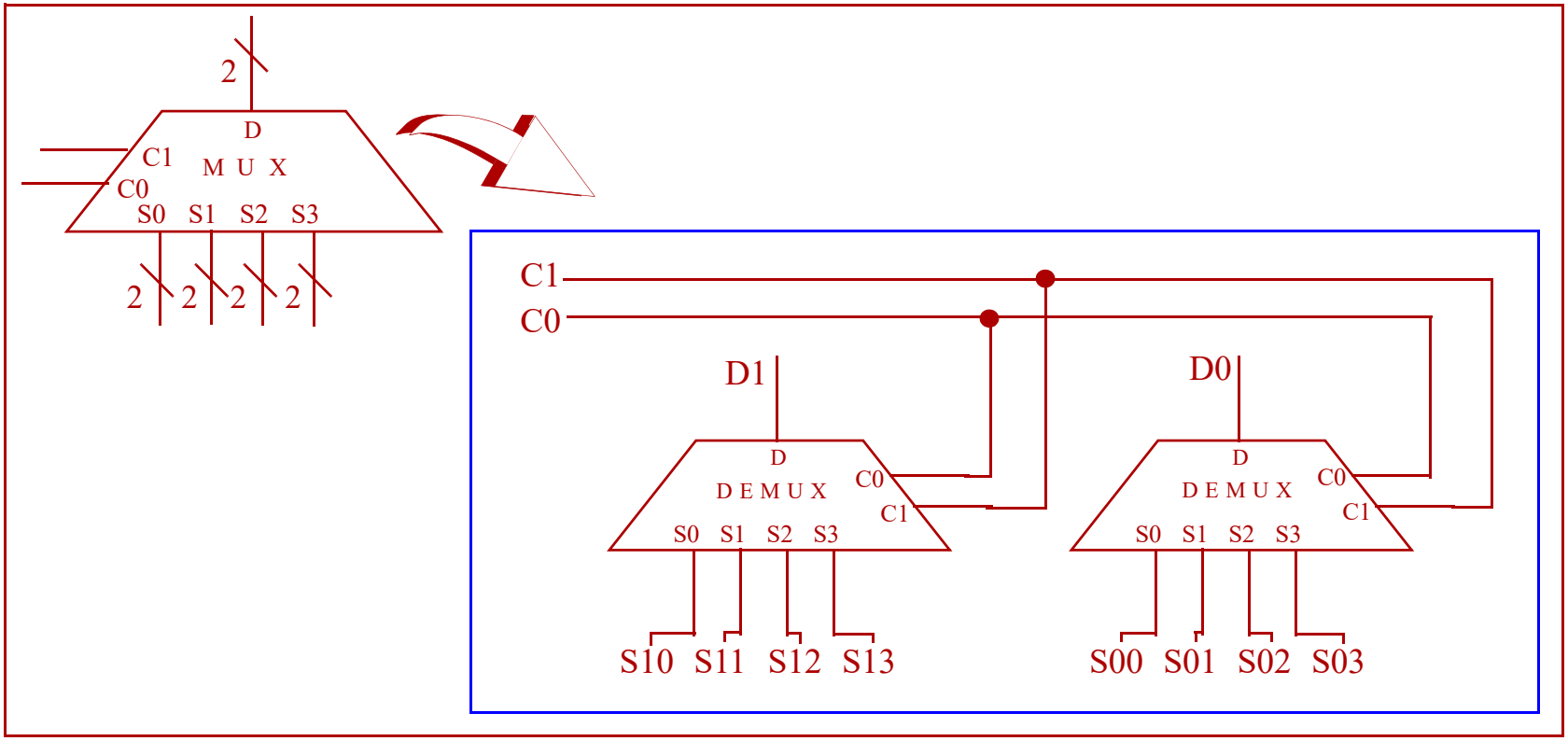
Con esta estructura es directo construir un DEMUX4, multiplexor 1x16

Bloques para el procesamiento de datos. Demultiplexores

Expansión modular de bloques demultiplexores. Asociación en paralelo

Aumenta el tamaño de palabra en las líneas de entrada/salida de dato

Ejemplo: Asociación de demultiplexores 1x4 (MUX2) para formar un DEMUX2 para palabras de dos bits



Con esta estructura es directo construir un DEMUX2, para palabras de n bits