Tema 2

SISTEMAS COMBINACIONALES

En este tema se estudiarán algunas de las funciones combinacionales más utilizadas, las cuales se implementan en chips comerciales. Como estas funciones son relativamente complejas, el chip deberá contener más de 10 puertas lógicas y, por lo tanto, estos circuitos integrados pertenecerán a la escala MSI.

Las funciones que vamos a estudiar son: sumadores, comparadores, unidades aritmético–lógicas (ALUs), multiplexores, demultiplexores, decodificadores, codificadores y conversores de código.

2.1. SUMADORES

2.1.1. Semisumador

La suma de dos dígitos binarios (PLUS) es similar a la suma de dos números decimales, pero teniendo en cuenta que la salida también es un número binario. Esto es importante cuando sumo, por ejemplo, 1 y 1, ya que para codificar el resultado (2 en decimal) necesito dos bits (10). En este caso, el bit menos significativo lo llamaremos suma, mientras que el bit más significativo lo llamaremos acarreo (*carry* en inglés). En total, existen 4 posibilidades de sumar dos números binarios de 1 bit:

	0	0	1	1
PLUS	0	1	0	1
	—	—	—	—
	0	1	1	10

El circuito que implementa esta función se denomina semi-sumador (HA o *half–adder*). Por lo tanto, un HA es el circuito que realiza la suma de dos bits. Como es obvio, precisa dos entradas (que vamos a llamar A y B) y dos salidas: la suma propiamente dicha (S

ENTRADAS		SALIDAS	
A	B	C_{out}	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Figura 2.1: Tabla de verdad y símbolo de un semisumador (HA).

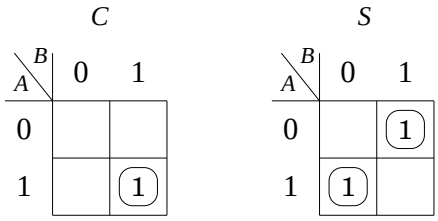


Figura 2.2: Minimización de las funciones suma y acarreo de un semisumador.

o $\sum$) y el acarreo C. En la figura 2.1 se puede ver la tabla de verdad de las funciones de salida y el símbolo del HA. De los diagramas de Karnaugh (figura 2.2) obtenemos sus expresiones mínimas:

$$\begin{aligned} C &= AB \\ S &= A\overline{B} + \overline{A}B \equiv A \oplus B \end{aligned}$$

La función de acarreo C es 1 únicamente cuando las dos entradas son 1. Además de la expresión en suma de productos, existen otras formas de expresar la función suma, aunque todas ellas se pueden deducir de la anterior aplicando las Leyes de De Morgan y la propiedad distributiva del álgebra de Boole. La expresión más sencilla es la EXOR de las entradas: la suma es 1 cuando en las entradas tenemos un número impar de 1's, y es 0 en caso contrario. En la figura 2.3 se pueden ver algunas implementaciones de un HA.

2.1.2. Sumador completo

Si además de sumar dos dígitos, también queremos sumar un acarreo de entrada, entonces el HA es insuficiente. Para sumar 3 dígitos de 1 bit necesitamos lo que se conoce como sumador completo (*full–adder* o FA). Si a los bits de entrada les llamamos A y B, y al acarreo de entrada lo denominamos C_{in}, entonces la tabla de verdad de las salidas del FA (el bit de la suma, S, y el acarreo de salida, C_{out}) la tenemos en la figura 2.4.

La minimización de las funciones de salida del sumador completo se puede ver en la figura 2.5. El resultado de la minimización son las expresiones:

$$\begin{aligned} C_{out} &= AB + AC_{in} + BC_{in} = AB + (A+B)C_{in} \\ S &= \overline{A}B\overline{C}_{in} + A\overline{B}\,\overline{C}_{in} + \overline{A}\,\overline{B}C_{in} + ABC_{in} = A \oplus B \oplus C_{in} \end{aligned}$$

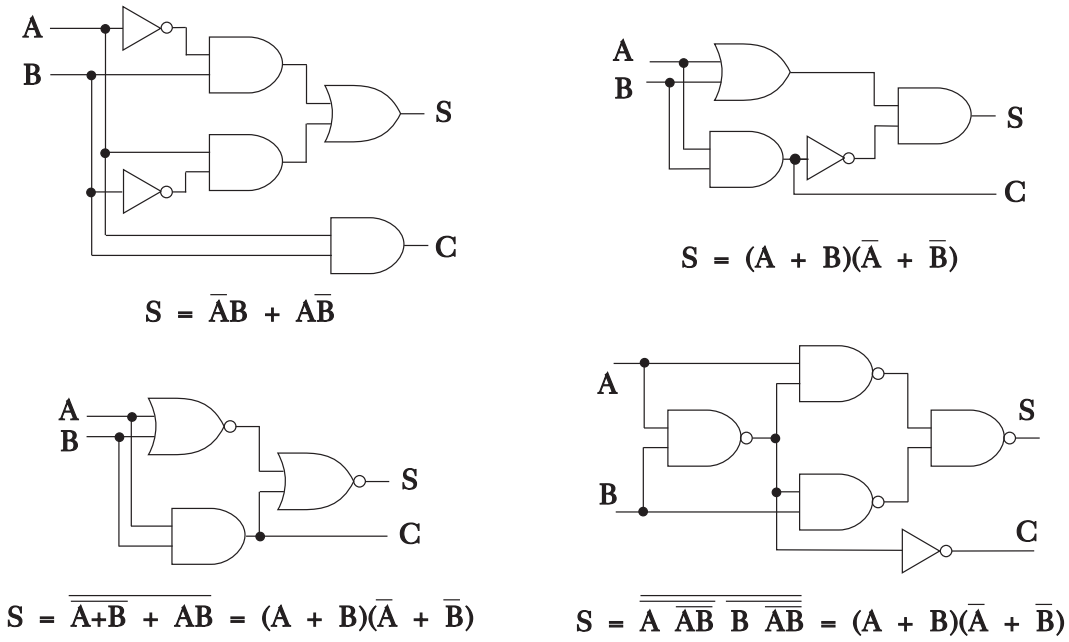


Figura 2.3: Distintas implementaciones de un semisumador (HA).

ENTRADAS			SALIDAS	
C_{in}	A	B	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

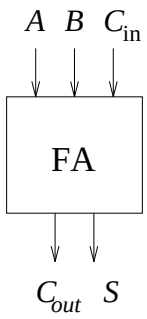


Figura 2.4: Tabla de verdad y símbolo de un sumador completo (FA).

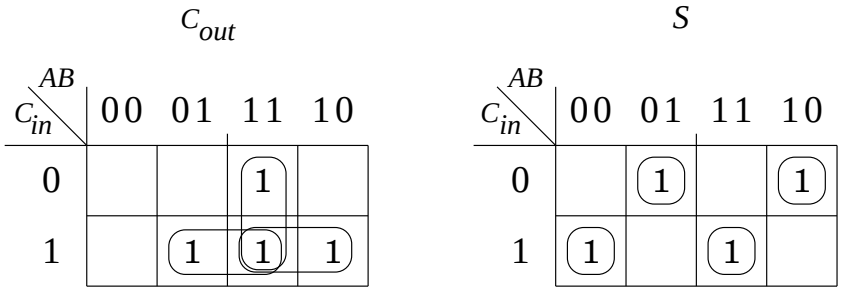


Figura 2.5: Diagramas de Karnaugh de las salidas de un sumador completo.

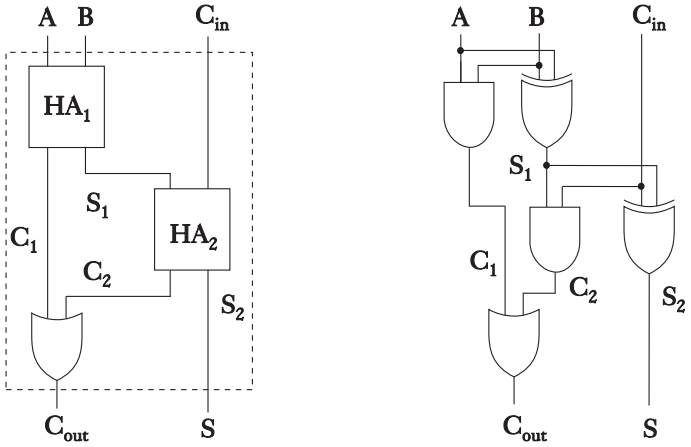


Figura 2.6: Implementación de un FA a partir de HA.

S es la función EXOR de las 3 entradas, es decir, su valor será 1 cuando en las entradas haya un número impar de unos. C_{out} será 1 cuando en las entradas haya dos o tres unos.

Construcción de un Sumador Completo con Semisumadores

Las funciones del sumador completo se pueden implementar directamente utilizando las expresiones mínimas de S y C_{out}. Sin embargo, también se pueden implementar utilizando como módulo básico el semisumador y puertas lógicas adicionales.

Sabemos que el bit de suma del FA es la EXOR de las tres entradas. El HA solo puede hacer EXOR de 2 entradas, por lo tanto necesito, como mínimo, 2 HA conectados en serie, tal que la salida S_1 sea también entrada al segundo HA. De esa forma S_2 será igual a la suma del FA (ver figura 2.6).

$$S_2 = S_1 \oplus C_{in} = A \oplus B \oplus C_{in}$$

Queda calcular el acarreo de salida del FA. Para ello observamos que el acarreo del primer HA es: $C_1 = AB$. El acarreo del segundo HA es $C_2 = C_{in}S_1 = C_{in}(A \oplus B)$. Si hacemos la OR de ambos obtendremos la siguiente expresión:

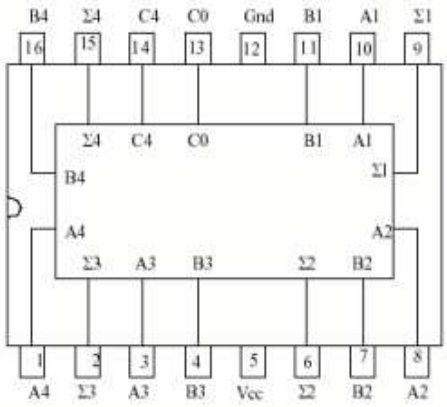


Figura 2.7: Diagrama de pines del C.I. 7483.

$$C_{out} = C_1 + C_2 = AB + (A \oplus B)C_{in}$$

Se puede demostrar que esta expresión es equivalente a la expresión mínima que obtuvimos con el diagrama de Karnaugh. Para ello debemos utilizar las leyes y propiedades del álgebra de Boole.

2.1.3. Sumadores de palabras

Ya sabemos sumar tres números binarios de 1 bit, pero nos interesa poder sumar cantidades mayores, es decir, palabras o números de varios bits que puedan codificar números mayores. En el mercado podemos encontrar chips como el de la figura 2.7, que nos muestra el diagrama de pines del C.I. 7483, sumador de números binarios de 4 bits. Para el diseño de estos circuitos existen dos opciones. La primera consiste en aplicar el mismo método que hemos estado usando, a saber, definir la tabla de verdad de la función u operación que nos interesa implementar y minimizarla. Obviamente, este método resulta poco práctico en el caso de tener números de varios bits. Por poner un ejemplo, la tabla de verdad de un sumador de palabras de 4 bits posee 8 entradas, es decir, 256 combinaciones. El segundo método consiste en hacer un diseño modular, es decir, diseñar un circuito básico que iremos repitiendo las veces que necesitemos. Este método solo es aplicable en funciones que posean un cierto grado de regularidad. Si nos fijamos, la suma aritmética de palabras de n bits cumple dicha condición. Sumemos dos números en binario a la manera "tradicional":

Acarreos	1 0 1 1 1 0 1 1 1 0	
Primer sumando	1 0 1 1 1 0 0 1 1 0	742
Segundo sumando	1 0 0 1 1 0 1 1 1 1	623
Suma	1 0 1 0 1 0 1 0 1 0 1	1365

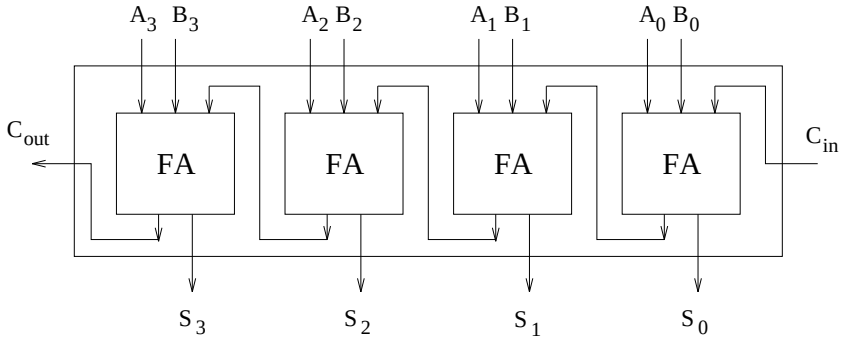


Figura 2.8: Sumador de acarreo enlazado.

Como podemos observar, para calcular el bit i-ésimo del resultado solo necesitamos conocer los bits i-ésimos de las entradas y el acarreo resultado de calcular el bit anterior $(i-1)$. Por lo tanto, el módulo básico es un sumador de 3 bits: un sumador completo. Para sumar palabras de n bits será preciso utilizar n FA. Nos queda por resolver cómo y cuándo calcular el acarreo de cada bit. Para ello existen varias alternativas o soluciones, pero aquí solo veremos la conocida como **sumador de acarreo enlazado**.

En el sumador de acarreo enlazado, el acarreo del sumador completo i se conecta al acarreo de entrada del sumador completo $i+1$. De esta forma, a pesar de que todos los FA trabajan en paralelo, el resultado final (correcto), no se obtendrá hasta que todas las salidas sean estables, es decir, hasta que un acarreo generado en el primer bit (el bit 1) se propague hasta el bit más significativo (el bit n). Resulta evidente que la velocidad del sumador de acarreo enlazado es baja, pues cada etapa o FA ha de esperar al cómputo de los acarreos por parte de todos los sumadores situados a su derecha (bits menos significativos), es decir, el retardo será n veces el tiempo de retardo de un FA.

En la figura 2.8 se muestra un sumador de acarreo enlazado construido con 4 sumadores completos de un bit. El primer acarreo C_0 es un acarreo de entrada al circuito y podemos denotarlo por C_{in}. Los 3 acarreos siguientes C_1, C_2 y C_3 son acarreos generados y usados exclusivamente por el circuito, y, por último, el acarreo C_4 es un acarreo de salida y podemos denotarlo por C_{out}. Las expresiones de cada señal son:

$$\begin{aligned} S_i &= A_i \oplus B_i \oplus C_i, \quad \forall i = 0, ..., 3 \\ C_{i+1} &= A_i B_i + (A_i \oplus B_i) C_i, \quad \forall i = 0, ..., 3 \qquad \text{con} \quad C_0 = C_{in} \quad \text{y} \quad C_4 = C_{out} \end{aligned}$$

El resultado final necesita un total de cinco bits para codificar el resultado, es decir, $C_{out}S_3S_2S_1S_0$. Existe la posibilidad de conectar más sumadores de palabras en cascada para ampliar el tamaño de las palabras a sumar. Para ello se debe conectar el acarreo de salida C_{out} de cada circuito al acarreo de entrada C_{in} del circuito situado a su izquierda, tal y como se ve en la figura 2.9.

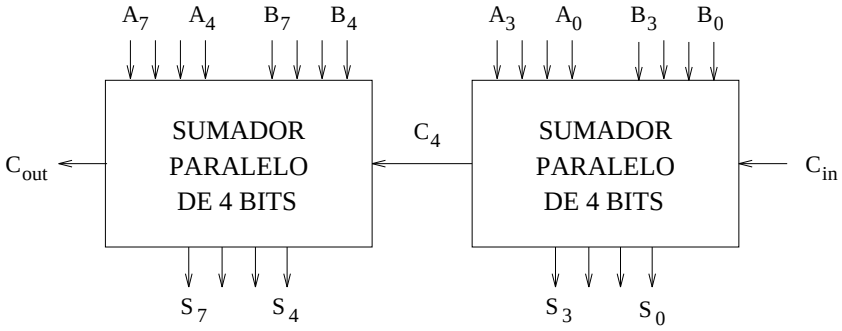


Figura 2.9: Suma de palabras de 8 bits con sumadores paralelos de 4 bits.

2.1.4. Suma y resta de números con signo

La sustracción, que vamos a denotar por MINUS, se puede implementar de muchas formas. Se pueden definir las tablas de verdad para cada uno de los bits de salida y minimizar las funciones con los diagramas de Karnaugh. También se puede seguir los mismos pasos que en la suma construyendo un semirrestador, un restador completo y, finalmente, un restador de palabras. Otra opción, más eficiente, consiste en calcular la resta a partir de la suma. Para ello, solo se necesita calcular el opuesto del sustraendo. Esta operación depende del tipo de representación elegida para codificar los números negativos. Debido a esto, a continuación vamos a ver cómo se realizan las sumas y las restas en Signo–Magnitud, Complemento a 1 y en Complemento a 2.

Formato Signo–Magnitud (S–M)

La suma y la resta son complejas ya que implican conocer el signo de ambos números para realizar bien una suma verdadera, bien una resta verdadera con sumadores y restadores paralelos de n bits, respectivamente. Si queremos restar (sumar) haremos tal resta (suma) si ambos operandos son del mismo signo, y haremos una suma (resta) si poseen signos opuesto. Debido a que en Signo–Magnitud es necesario implementar un restador binario, esta representación no es la más utilizada. Una alternativa es convertir los números a Complemento a 1 o Complemento a 2 y realizar las operaciones en estos formatos, que, como veremos a continuación, resulta mucho más sencillo.

Complemento a 1 (C'1)

Para sumar dos números en C'1 se suman todos sus bits, incluido el de signo. Si existe un acarreo de salida entonces se le suma al resultado. El proceso se puede ver en la figura 2.10. Para restar dos números necesitamos calcular el opuesto del sustraendo, es decir, calcular el Complemento a 1 del sustraendo (figura 2.10) y realizar una suma. Con esta representación solo necesitamos un sumador paralelo de n bits.

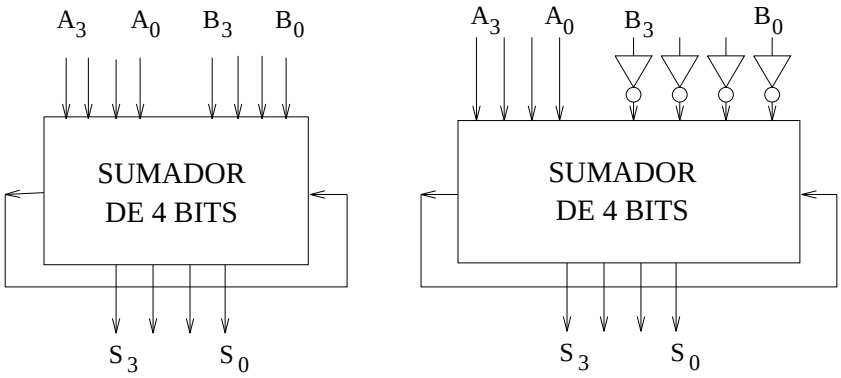


Figura 2.10: Suma y resta de dos números en Complemento a 1.

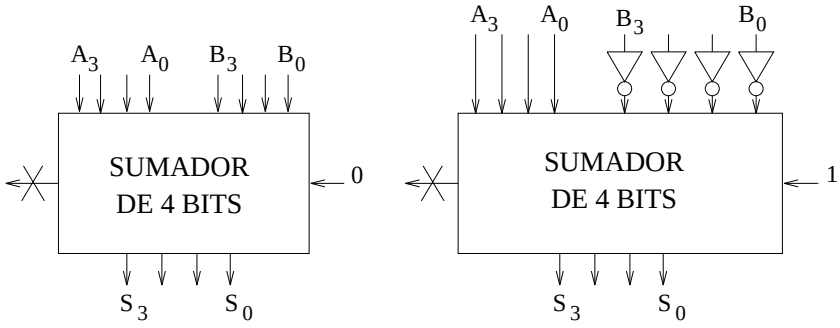


Figura 2.11: Suma y resta de dos números en Complemento a 2.

Complemento a 2 (C'2)

La suma en C'2 es la más sencilla de todas: se suman todos los bits del formato y se desprecia o no se tiene en cuenta el acarreo final de salida (figura 2.11). Para calcular el opuesto de un número necesitamos negar todos los bits del mismo y sumar 1 al final (es decir, calcular el C'2). La resta sería igual que en C'1 (ver figura 2.11), pero con el acarreo de entrada igual a 1 y **despreciando el acarreo de salida**.

Desbordamiento u *overflow*

La suma de dos números con formato (número fijo de bits) puede producir como resultado un número que no es posible representar en el formato de partida. En este caso se dirá que existe desbordamiento (*overflow*) y el resultado de la suma (o resta) será incorrecto. Por ejemplo, en un formato de 4 bits en C'2 la operación (-7) plus (-6)= (-13) producirá *overflow*, pues -13 no es representable en 4 bits.

El *overflow* solo puede producirse cuando los dos sumandos son o bien ambos positivos o bien ambos negativos, pues obviamente cuando un sumando es positivo y el otro negativo, el resultado siempre será menor que uno de los operandos y podrá representarse con el formato de partida. En el caso de la representación Signo–Magnitud el desbordamiento

Cuadro 2.1: Tabla de verdad de una puerta EXOR.

$\overline{S}/R$	B_i	EXOR
0	0	0
0	1	1
1	0	1
1	1	0

se detecta cuando en la suma de magnitudes existe acarreo de salida. En el caso del C'1 y C'2 el *overflow* se detecta comprobando el signo del resultado: si éste es correcto, es decir, coincide con el signo de ambos operandos, se puede asegurar que no hay desbordamiento.

En el caso de la suma, en C'1 y C'2 el resultado es incorrecto (existe desbordamiento) cuando $A_{n-1} = B_{n-1} \neq S_{n-1}$, es decir, ($A_{n-1} = B_{n-1} = 0$ y $S_{n-1} = 1$) ó ($A_{n-1} = B_{n-1} = 1$ y $S_{n-1} = 0$). Por lo tanto:

$$overflow = \overline{A}_{n-1}\overline{B}_{n-1}S_{n-1} + A_{n-1}B_{n-1}\overline{S}_{n-1}$$

Circuito sumador/restador

Usando las propiedades de la función EXOR (cuadro 2.1) podemos construir un circuito para sumar o restar números en C'1 o C'2. Introducimos una señal denominada $\overline{S}/R$, tal que si esta señal es 0 ($\overline{S}$) se realizará una suma *A PLUS B* y si es 1 (R) se realizará una resta *A MINUS B*.

Para ello, si $\overline{S}/R = 0$ los bits de B se propagan tal cuál ($B'_i = 0 \oplus B_i = B_i$), pero si $\overline{S}/R = 1$ entonces se propagan $B'_i = 1 \oplus B_i = \overline{B}_i$. En C'1 debemos conectar C_{out} con C_{in} para completar la operación. Sin embargo, en C'2 para negar un número además de negar todos sus bits (C'1), necesitamos sumarle 1. Para ello aplicamos también la señal $\overline{S}/R$ al C_{in} del sumador, de tal forma que si se realiza una suma $C_{in} = 0$ (no afecta), mientras que en la resta $C_{in} = 1$. En este caso C_{out} no formará parte del resultado y no se usa para nada. En la figura 2.12 podemos ver, como ejemplo, el sumador/restador en Complemento a 2 para números de 4 bits.

Como el formato es fijo y el mismo para las entradas y la salida, existirá desbordamiento cuando:

$$\begin{aligned} overflow &= \overline{A}_{n-1}\overline{B}_{n-1}S_{n-1}\overline{\overline{S}/R} + A_{n-1}B_{n-1}\overline{S}_{n-1}\overline{\overline{S}/R} \\ &+ A_{n-1}\overline{B}_{n-1}\overline{S}_{n-1}\overline{S}/R + \overline{A}_{n-1}B_{n-1}S_{n-1}\overline{S}/R \\ &= \overline{A}_{n-1}(\overline{B_{n-1} \oplus \overline{S}/R})S_{n-1} + A_{n-1}(B_{n-1} \oplus \overline{S}/R)\overline{S}_{n-1} \\ &= \overline{A}_{n-1}\overline{B'}_{n-1}S_{n-1} + A_{n-1}B'_{n-1}\overline{S}_{n-1} \end{aligned}$$

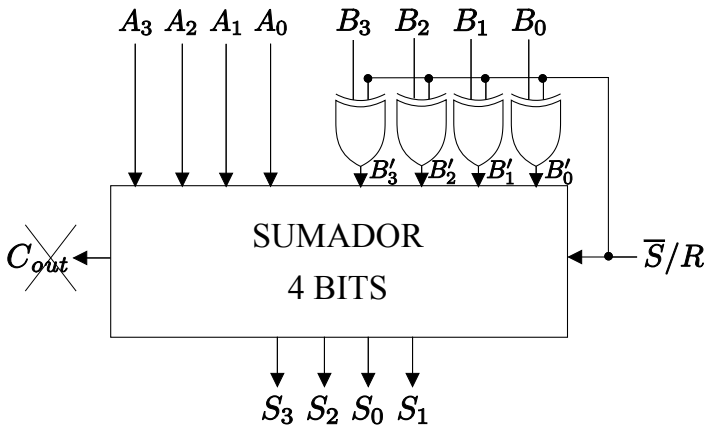


Figura 2.12: Sumador/restador de dos números en Complemento a 2.

2.2. COMPARADORES

2.2.1. Comparador binario

La comparación entre números es la operación que determina si uno de ellos es mayor, menor o igual que el otro. Un comparador de magnitud es un circuito combinacional que compara dos números positivos A y B y proporciona tres salidas $(A < B)$, $(A = B)$ y $(A > B)$. Como son mutuamente excluyentes, conociendo dos de estas funciones es posible determinar la tercera, con lo que realmente solo necesitamos implementar dos. Por ejemplo, para obtener la función $(A < B)$ a partir de las otras dos:

$(A > B)$	$(A = B)$	$(A < B)$
0	0	1
0	1	0
1	0	0
1	1	–

Donde en la última combinación de entradas se ha puesto una indiferencia pues no podrá darse en la práctica. Existen dos posibilidades para expresar $(A < B)$. Si la indiferencia se toma como 0, entonces $(A < B) = \overline{(A{>}B) + (A{=}B)}$ y si la indiferencia se toma como 1, entonces $(A < B) = \overline{(A{>}B)} \oplus (A{=}B)$. Las expresiones de $(A = B)$ y $(A > B)$ en función de las otras dos se obtienen del mismo modo.

Para implementar las funciones $(A > B)$, $(A = B)$ y $(A < B)$ existen dos posibilidades. La primera de ellas es partir de los diagramas de Karnough. Así, por ejemplo, si A y B son números de dos bits $A = A_1A_0$ y $B = B_1B_0$, entonces las funciones tendrán las siguientes expresiones mínimas (ver figura 2.13):

$$\begin{aligned}
(A{>}B) &= A_1\overline{B_1} + A_0\overline{B_1}\,\overline{B_0} + A_1A_0\overline{B_0} \\
(A{=}B) &= \overline{A_1}\,\overline{A_0}\,\overline{B_1}\,\overline{B_0} + \overline{A_1}A_0\overline{B_1}B_0 + A_1\overline{A_0}B_1\overline{B_0} + A_1A_0B_1B_0 \\
(A{<}B) &= \overline{A_1}B_1 + \overline{A_1}\,\overline{A_0}B_0 + \overline{A_0}B_1B_0
\end{aligned}$$

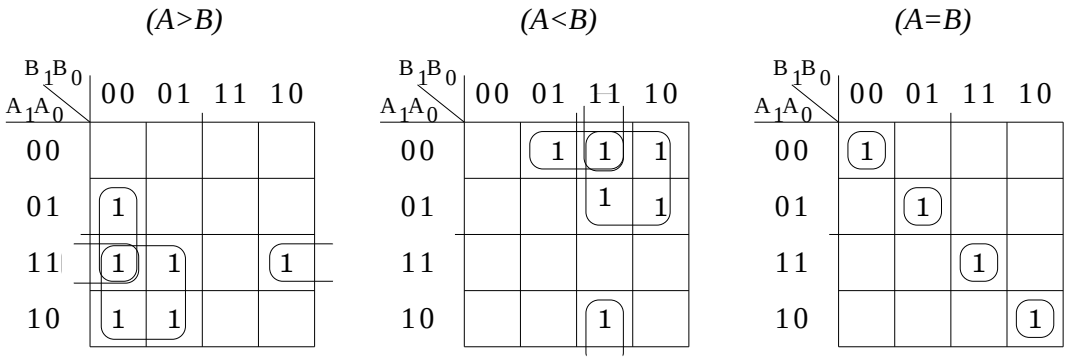


Figura 2.13: Diagramas de Karnaugh de las salidas de un comparador de 2 bits.

Cuadro 2.2: Tablas de verdad de un comparador de 1 bit.

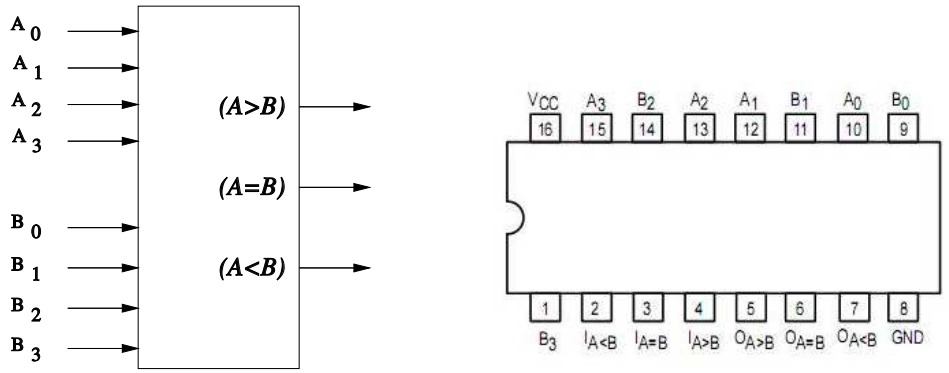
A_i	B_i	$(A_i{>}B_i)$	$(A_i{=}B_i)$	$(A_i{<}B_i)$
0	0	0	1	0
0	1	0	0	1
1	0	1	0	0
1	1	0	1	0

Sin embargo, si utilizamos este método, el diseño se complica si el número de bits de las palabras a comparar es grande. La segunda posibilidad es encontrar una forma simple de diseñar comparadores de cualquier número de bits a partir de un circuito que compare un solo bit, es decir, usar una estrategia modular. Si el bit más significativo de A es mayor (menor) que el bit más significativo de B, entonces A es mayor (menor) que B. Si $A_{n-1}{=}B_{n-1}$ entonces seguimos comparando el siguiente bit más significativo y así sucesivamente. Por último, dos números son iguales si todos sus bits son iguales, es decir, $A_i = B_i, \quad \forall i = 0, ..., n-1$.

El primer paso consiste pues en diseñar un circuito comparador de un bit, es decir, las funciones $(A_i{>}B_i)$, $(A_i{=}B_i)$ y $(A_i{<}B_i)$. Las tablas de verdad de cada una de ellas se pueden ver en el cuadro 2.2, de donde deducimos que sus expresiones mínimas son:

$$\begin{aligned} (A_i{>}B_i) &= A_i\overline{B_i} \\ (A_i{=}B_i) &= \overline{A_i \oplus B_i} \\ (A_i{<}B_i) &= \overline{A_i}B_i \end{aligned}$$

A partir de este bloque, se pueden implementar comparadores de n bits. El caso $n = 2$ es trivial y se puede comprobar fácilmente que los resultados coinciden plenamente con los obtenidos al minimizar las funciones completas utilizando los diagramas de Karnaugh. Vamos a construir ahora un comparador de 4 bits (figura 2.14a). Sean $A = A_3A_2A_1A_0$ y $B = B_3B_2B_1B_0$ los números a comparar. Definimos $x_i = (A_i{=}B_i) = \overline{A_i \oplus B_i}, \quad i = 0, ..., 3$ (función de igualdad de los bits i).



(a) (b)

Figura 2.14: (a) Comparador binario de 4 bits; (b) diagrama de pines del comparador binario 7485.

A y B serán iguales si se verifica que los cuatro bits son iguales, o lo que es lo mismo, si $(A_3 = B_3)$ **y** $(A_2 = B_2)$ **y** $(A_1 = B_1)$ **y** $(A_0 = B_0)$. En el álgebra de Boole esto es equivalente a la función:

$$(A{=}B) = (A_3{=}B_3)(A_2{=}B_2)(A_1{=}B_1)(A_0{=}B_0) = x_3x_2x_1x_0$$

A será mayor que B en si: $A_3 > B_3$ **o** $(A_3 = B_3$ **y** $A_2 > B_2)$ **o** $(A_3 = B_3$ **y** $A_2 = B_2$ **y** $A_1 > B_1)$ **o** $(A_3 = B_3$ **y** $A_2 = B_2$ **y** $A_1 = B_1$ **y** $A_0 > B_0)$. Entonces:

$$\begin{aligned}(A{>}B) &= (A_3{>}B_3) + (A_3{=}B_3)(A_2{>}B_2) + (A_3{=}B_3)(A_2{=}B_2)(A_1{>}B_1) \\ &\quad +(A_3{=}B_3)(A_2{=}B_2)(A_1{=}B_1)(A_0{>}B_0) \\ &= A_3\overline{B_3} + x_3A_2\overline{B_2} + x_3x_2A_1\overline{B_1} + x_3x_2x_1A_0\overline{B_0}\end{aligned}$$

Del mismo modo, A será menor que B si: $A_3 < B_3$ **o** $(A_3 = B_3$ **y** $A_2 < B_2)$ **o** $(A_3 = B_3$ **y** $A_2 = B_2$ **y** $A_1 < B_1)$ **o** $(A_3 = B_3$ **y** $A_2 = B_2$ **y** $A_1 = B_1$ **y** $A_0 < B_0)$. Entonces:

$$\begin{aligned}(A{<}B) &= (A_3{<}B_3) + (A_3{=}B_3)(A_2{<}B_2) + (A_3{=}B_3)(A_2{=}B_2)(A_1{<}B_1) \\ &\quad +(A_3{=}B_3)(A_2{=}B_2)(A_1{=}B_1)(A_0{<}B_0) \\ &= \overline{A_3}B_3 + x_3\overline{A_2}B_2 + x_3x_2\overline{A_1}B_1 + x_3x_2x_1\overline{A_0}B_0\end{aligned}$$

En la figura 2.14b mostramos la configuración de pines del C.I. 7485, que se corresponde con un comparador binario de 4 bits.

2.2.2. Comparación de un mayor número de bits

Para comparar palabras de un mayor número de bits, podemos utilizar el comparador para palabras de 4 bits que acabamos de diseñar. En la figura 2.15 se muestran dos ejemplos. La idea es comparar los números en bloques de 4 bits: si la comparación de los 4 bits más significativos nos indica que un número es mayor o menor que otro entonces no nos hace falta seguir comparando; si, por el contrario, los 4 bits más significativos son iguales entonces necesitamos seguir comparando el o los siguientes bloques de 4 bits.

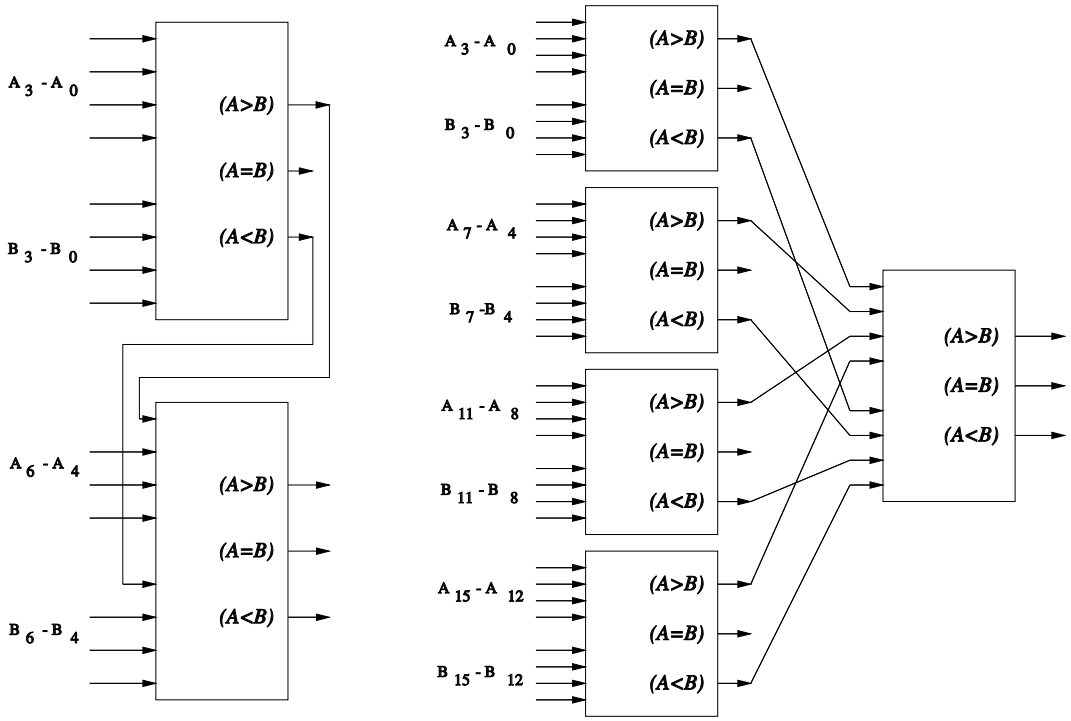


Figura 2.15: Ejemplos de comparación sobre palabras de más de 4 bits.

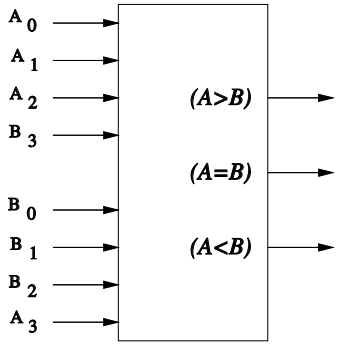


Figura 2.16: Comparador de números en Complemento a 2.

2.2.3. Comparación de números con signo

Para diseñar el comparador de dos números con signo debemos tener en cuenta en que formato está representado. En cualquier caso, dos números son iguales si todos sus bits son iguales, excepto en Signo-Magnitud y en Complemento a 1 en los cuales el cero posee dos representaciones. Si no tenemos en cuenta esa peculiaridad, entonces la función $(A = B)$ es la misma para todas las representaciones.

Para calcular cuando un número en Complemento a 1 ó en Complemento a 2 es mayor o menor que otro, podemos utilizar un comparador binario. Para ello debemos de intercambiar los bits más significativos tal y como se muestra en la figura 2.16 para 4 bits.

Supongamos que queremos comparar un número A negativo ($A_3 = 1$) con otro B positivo ($B_3 = 0$). El comparador compara los números suponiendo que están en formato binario puro. Al intercambiar los bits de signo, estamos haciendo que el bit más significativo de B sea 1, mientras que el más significativo de A es 0. Por lo tanto, B es mayor que A. El caso de A positivo y B negativo es exactamente igual.

Supongamos que ambos números son positivos. En ese caso estamos intercambiando dos ceros y comparamos los números tal y como estaban. El comparador nos dirá cual de los dos es mayor (menor) o si son iguales (recordemos que los números positivos se codifican igual que en binario puro).

El único caso que nos falta es cuando los dos números son negativos. Al igual que antes, no tiene sentido intercambiar los signos puesto que ambos son 1. El comparador hará la comparación suponiendo que los números están codificados en binario puro. El resultado será correcto porque tanto en Complemento a 1 como en Complemento a 2, el orden (de mayor a menor) de los números negativos se mantiene si se considera que los números están codificados en binario puro.

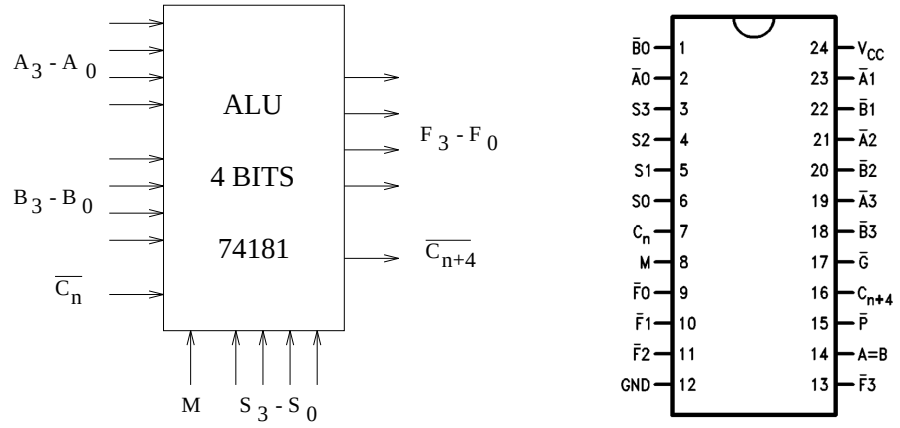
Por ejemplo, -5 es mayor que -7 pero menor que -3. En Complemento a 1 con 4 bits, -5 es 1010, -7 es 1000 y -3 es 1100. Como podemos comprobar, en binario puro 1010 es mayor que 1000 pero menor que 1100, y eso es precisamente lo que nos dirá el comparador binario. Por otra parte, en Complemento a 2 -5 es 1011, -7 es 1001 y -3 es 1101. También se puede ver que en binario puro, 1011 es mayor que 1001 y menor que 1101.

El circuito de la figura 2.16 compararía cualquier pareja de números en Complemento a 2, pero para Complemento a 1 harían falta puertas lógicas adicionales para tener en cuenta la doble representación del cero en este formato. En el caso de números en Signo–Magnitud no se puede implementar un comparador utilizando únicamente un comparador binario, sino que se necesitan puertas lógicas u otros elementos adicionales debido a que también hay doble representación del cero y a que la representación de los números negativos no mantiene el orden si se consideran codificados en binario puro.

2.3. UNIDAD ARITMÉTICO–LÓGICA (ALU)

Las unidades aritmético–lógicas (ALU) constituyen dispositivos útiles y versátiles que implementan diferentes operaciones lógicas y aritméticas, generalmente en un solo circuito integrado. Para estudiar las ALUs vamos a ver un ejemplo: el C.I. 74181 (ver figura 2.17). Funcionalmente, este chip acepta como datos dos palabras de cuatro bits $A = A_3A_2A_1A_0$ y $B = B_3B_2B_1B_0$, produciendo como resultado otra palabra de 4 bits $F = F_3F_2F_1F_0$. Además de estas líneas posee un acarreo de entrada $\overline{C}_n$ y un acarreo de salida $\overline{C}_{n+4}$, activos a nivel bajo.

La operación que se realiza sobre estos datos está determinada por las entradas de selección $S = S_3S_2S_1S_0$ y la entrada de modo M. Cuando $M =$ L las operaciones son aritméticas (suma, resta, etc.), mientras que cuando $M =$ H las operaciones son lógicas



(a) Diagrama lógico (b) Diagrama de pines

Figura 2.17: Unidad aritmético–lógica C.I. 74181.

Cuadro 2.3: Operación de la ALU 74181.

SELECTION S_3 S_2 S_1 S_0	ACTIVE-HIGH DATA		
	M = H LOGIC FUNCTIONS	M = L : ARITHMETICH OPERATIONS	
		$C_n = 0$ $\overline{C_n} = 1 = H$	$C_n = 1$ $\overline{C_n} = 0 = L$
L L L L	$F = \overline{A}$	F = A	F = A PLUS 1
L L L H	$F = \overline{A + B}$	F = A + B	F = (A + B) PLUS 1
L L H L	$F = \overline{A}B$	$F = A + \overline{B}$	$F = (A + \overline{B})$ PLUS 1
L L H H	F = 0	F = MINUS 1 (2's COMPL)	F = ZERO
L H L L	$F = \overline{AB}$	$F = A$ PLUS $A\overline{B}$	$F = A$ PLUS $A\overline{B}$ PLUS 1
L H L H	$F = \overline{B}$	$F = (A + B)$ PLUS $A\overline{B}$	$F = (A + B)$ PLUS $A\overline{B}$ PLUS 1
L H H L	$F = A \oplus B$	F = A MINUS B MINUS 1	F = A MINUS B
L H H H	$F = A\overline{B}$	$F = A\overline{B}$ MINUS 1	$F = A\overline{B}$
H L L L	$F = \overline{A} + B$	F = A PLUS AB	F = A PLUS AB PLUS 1
H L L H	$F = \overline{A \oplus B}$	F = A PLUS B	F = A PLUS B PLUS 1
H L H L	F = B	$F = (A + \overline{B})$ PLUS AB	$F = (A + \overline{B})$ PLUS AB PLUS 1
H L H H	F = AB	F = AB MINUS 1	F = AB
H H L L	F = 1	F = A PLUS A*	F = A PLUS A PLUS 1
H H L H	$F = A + \overline{B}$	$F = (A + B)$ PLUS A	$F = (A + B)$ PLUS A PLUS 1
H H H L	F = A + B	$F = (A + \overline{B})$ PLUS A	$F = (A + \overline{B})$ PLUS A PLUS 1
H H H H	F = A	F = A MINUS 1	F = A

* Each bit is shifted to the next more significant position.

(AND, OR, etc.). Los acarreos de entrada y de salida solo tienen sentido cuando se trata de operaciones aritméticas. La tabla 2.3 ilustra las distintas operaciones que se realizan en términos del valor de las entradas S y M.

Sea por ejemplo S = HLLH, M = L, A = LHHL, B = LLHH y $\overline{C}_n$ = L. La operación a realizar está determinada por M (L: operación aritmética) y S (HLLH: A PLUS B o A PLUS B PLUS 1, sin acarreo y con acarreo, respectivamente). Al ser $\overline{C}_n$ = L (existe acarreo de entrada) entonces la operación que se realiza es $F = A$ PLUS B PLUS 1.

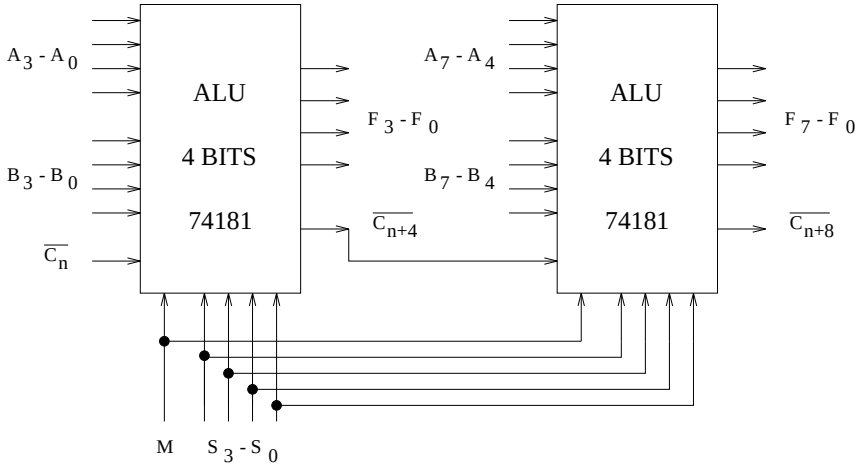


Figura 2.18: ALU para operandos de 8 bits.

Ampliación de la longitud de los datos

A pesar de que las ALUs tienen un número muy limitado de bits en cuanto a la longitud de las palabras sobre las que opera, es posible conectarlas en cascada para realizar operaciones aritmético–lógicas con palabras de un número de dígitos considerablemente superior. Ello se consigue conectando el acarreo de salida $\overline{C}_{n+4}$ de un chip con el acarreo de entrada $\overline{C}_n$ del siguiente que maneja los bits más significativos y puenteando todas las entradas M y S de cada uno de los chips, tal y como se ve en la figura 2.18.

2.4. FUNCIONES DE RUTA DE DATOS

En esta sección veremos los siguiente dispositivos:

- **Multiplexor (MUX)**: selecciona una de entre 2^n entradas en función de n líneas de control.
- **Demultiplexor (DEMUX)**: lleva la entrada a una de las 2^n salidas en función de n líneas de control.

2.4.1. Multiplexor (MUX)

Es un circuito selector de datos, es decir, la operación de este dispositivo es seleccionar una de entre varias entradas y llevar su valor a la salida. Para realizar esta selección son precisas líneas de control que nos indiquen cual de las entradas es la seleccionada. Si disponemos de 2^n entradas necesitaremos n líneas de control para hacer referencia a cada una de ellas. Por tanto, podemos definir el MUX 2^n a 1 como aquel dispositivo con 2^n entradas, una salida y n variables de control, de forma que el código binario contenido

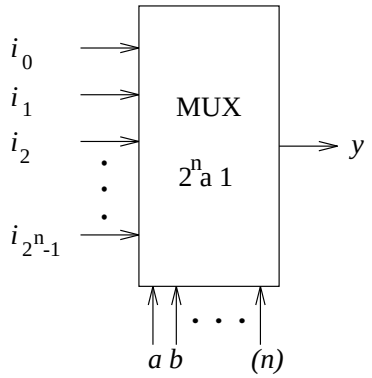


Figura 2.19: Representación de un MUX.

Cuadro 2.4: Tabla de funcionamiento de un MUX 8 a 1.

Líneas de control			Selección
a	b	c	
0	0	0	entrada i_0
0	0	1	entrada i_1
0	1	0	entrada i_2
0	1	1	entrada i_3
1	0	0	entrada i_4
1	0	1	entrada i_5
1	1	0	entrada i_6
1	1	1	entrada i_7

en las líneas de control indica cual de las entradas es la que se conecta a la salida. En el cuadro 2.4 vemos un ejemplo para un MUX con $n = 3$, donde tenemos 3 líneas de control (a, b, c) y 8 entradas (desde i_0 hasta i_7).

Construcción de un MUX

En el cuadro 2.5 presentamos las tablas de verdad del MUX 4 a 1 y del MUX 8 a 1. Se puede observar que solamente se trasmite a la salida el valor (0 o 1) de la entrada i seleccionada, no influyendo en la misma las demás entradas, donde hemos puesto "x". Por ejemplo, para el MUX 4 a 1 si $ab = 00$ a la salida el valor de y será el que haya en i_0, independientemente de los valores de i_1, i_2 e i_3, es decir, para $ab = 00$ e $i_0 = 0$ la salida será siempre 0 para cualquier combinación de valores de las otras tres entradas $i_1 i_2 i_3$ desde 000 hasta 111.

Las expresiones lógicas de las salidas son las siguientes:

MUX 4 a 1: $y = \overline{a}\,\overline{b} i_0 + \overline{a} b i_1 + a\overline{b} i_2 + ab i_3$

MUX 8 a 1: $y = \overline{a}\,\overline{b}\,\overline{c} i_0 + \overline{a}\,\overline{b} c i_1 + \overline{a} b \overline{c} i_2 + \overline{a} b c i_3 + a\overline{b}\,\overline{c} i_4 + a\overline{b} c i_5 + ab\overline{c} i_6 + abc i_7$

Cuadro 2.5: Tablas de verdad para MUX 4 a 1 y MUX 8 a 1.

a	b	i_0	i_1	i_2	i_3	y
0	0	0	x	x	x	0
0	0	1	x	x	x	1
0	1	x	0	x	x	0
0	1	x	1	x	x	1
1	0	x	x	0	x	0
1	0	x	x	1	x	1
1	1	x	x	x	0	0
1	1	x	x	x	1	1

a	b	c	i_0	i_1	i_2	i_3	i_4	i_5	i_6	i_7	y
0	0	0	0	x	x	x	x	x	x	x	0
0	0	0	1	x	x	x	x	x	x	x	1
0	0	1	x	0	x	x	x	x	x	x	0
0	0	1	x	1	x	x	x	x	x	x	1
0	1	0	x	x	0	x	x	x	x	x	0
0	1	0	x	x	1	x	x	x	x	x	1
0	1	1	x	x	x	0	x	x	x	x	0
0	1	1	x	x	x	1	x	x	x	x	1
1	0	0	x	x	x	x	0	x	x	x	0
1	0	0	x	x	x	x	1	x	x	x	1
1	0	1	x	x	x	x	x	0	x	x	0
1	0	1	x	x	x	x	x	1	x	x	1
1	1	0	x	x	x	x	x	x	0	x	0
1	1	0	x	x	x	x	x	x	1	x	1
1	1	1	x	x	x	x	x	x	x	0	0
1	1	1	x	x	x	x	x	x	x	1	1

La obtención de estas ecuaciones a partir de la tabla de verdad del MUX es similar a la construcción de funciones en forma de suma de minterm, pero esta vez las variables de entrada que son "x" no intervienen en la formación de los términos producto.

Normalmente se suele incluir una señal de *enable* o *strobe* (s) para la inhibición del dispositivo, con el siguiente funcionamiento:

- $s = 0$: El circuito está inhibido y la salida es siempre cero ($y = 0$).
- $s = 1$: Funcionamiento normal, la salida es igual a la entrada seleccionada.

La inclusión de esta entrada en las expresiones lógicas se realiza simplemente multiplicando cada término producto por s:

MUX 4 a 1: $y = s\overline{a}\,\overline{b}i_0 + s\overline{a}bi_1 + sa\overline{b}i_2 + sabi_3$

MUX 8 a 1: $y = s\overline{a}\,\overline{b}\,\overline{c}i_0 + s\overline{a}\,\overline{b}ci_1 + s\overline{a}b\overline{c}i_2 + s\overline{a}bci_3 + sa\overline{b}\,\overline{c}i_4 + sa\overline{b}ci_5 + sab\overline{c}i_6 + sabci_7$

La construcción de estos multiplexores a partir de puertas lógicas se muestra en la figura 2.20. En la figura 2.21 mostramos el diagrama de pines del C.I. 74153, que contiene dos multiplexores 4 a 1 con dos líneas de selección comunes y entradas de *strobe* separadas, junto con su tabla de funcionamiento.

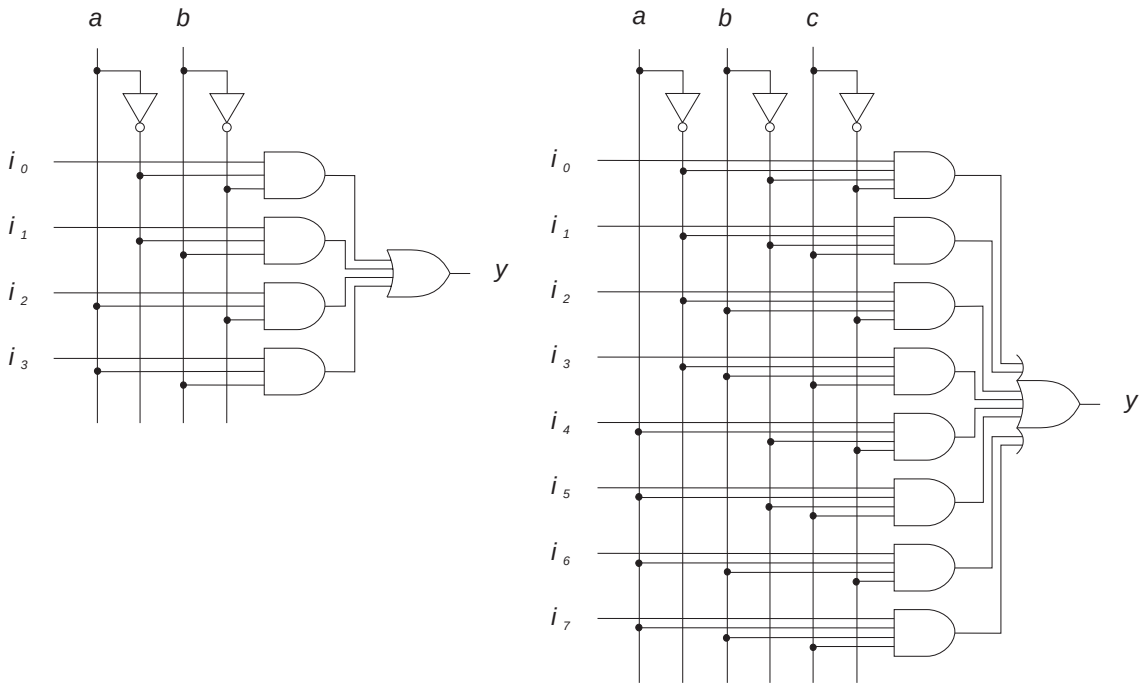


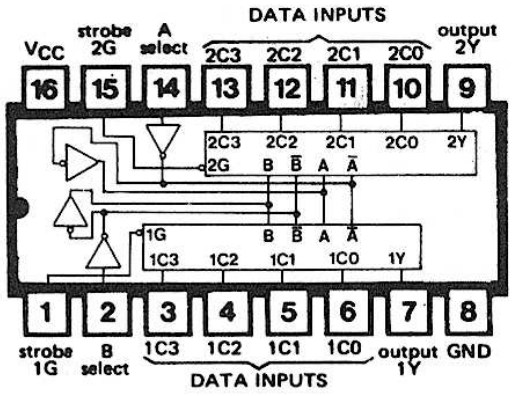
Figura 2.20: Construcciones del MUX 4 a 1 y del MUX 8 a 1.

Function Table

Select Inputs		Data Inputs				Strobe	Output
B	A	C0	C1	C2	C3	G	Y
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.
H = High Level, L = Low Level, X = Don't Care

(a) Tabla de funcionamiento



(b) Diagrama de pines

Figura 2.21: Multiplexor 74153.

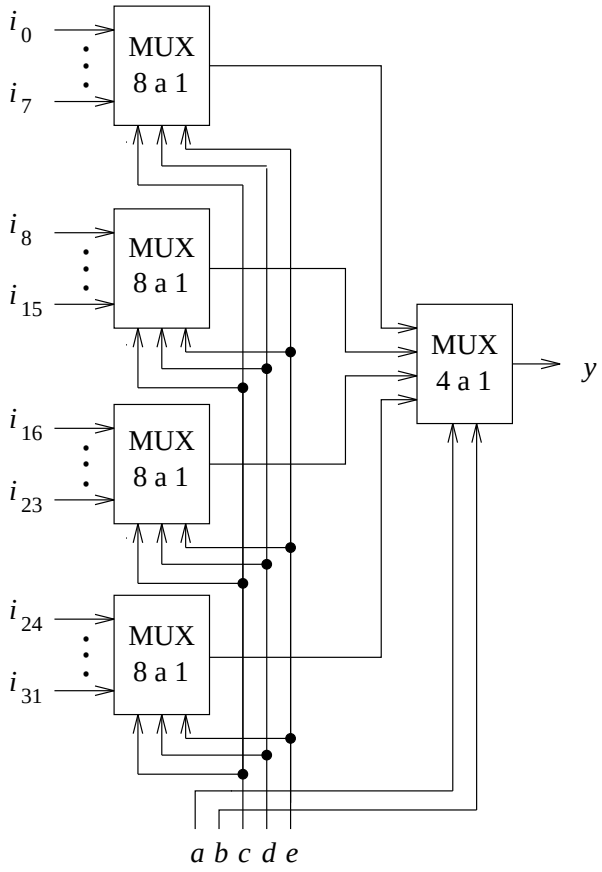


Figura 2.22: Árbol multiplexor 32 a 1.

Árboles multiplexores

El mayor MUX comercial disponible en forma de chip es de tamaño 16 a 1, pero podemos construir MUXes de cualquier tamaño interconectando varios MUX en una estructura de árbol. Por ejemplo, podemos realizar un MUX 32 a 1 a partir de cuatro MUX 8 a 1 y un MUX 4 a 1, tal como se muestra en la figura 2.22.

Cada MUX del primer nivel selecciona una de sus 8 entradas dependiendo de los bits de control comunes c, d y e. El MUX del segundo nivel selecciona una de las salidas de los MUXes del primer nivel en función de los bits de control a y b. El resultado final es que la salida toma el valor de una de las 32 entradas en función de las cinco líneas de control a, b, c, d y e. Notar que al MUX del segundo nivel (MUX 4 a 1) van las líneas de control más significativas.

El tamaño del MUX global se obtiene multiplicando los tamaños de los MUXes de los dos niveles. En este caso, MUXes 8 a 1 y un MUX 4 a 1 dan lugar a un MUX 8×4 a 1 (MUX 32 a 1). Se pueden construir árboles multiplexores de cualquier número de entradas sin más que añadir niveles de MUXes.

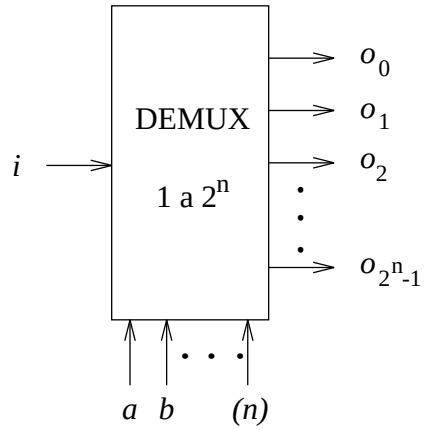


Figura 2.23: Representación de un DEMUX.

2.4.2. Demultiplexor (DEMUX)

Un demultiplexor es un circuito distribuidor de datos, es decir, la operación de este dispositivo consiste en tomar la única entrada, seleccionar una de entre varias salidas y conectarla a la entrada. Para realizar esta selección son precisas líneas de control que nos indiquen cual de las salidas es la seleccionada. Si disponemos de 2^n salidas son precisas n líneas de control para hacer referencia a cada una de ellas (DEMUX 1 a 2^n). Básicamente realiza la función inversa del multiplexor. Por tanto podemos definir el DEMUX 1 a 2^n como aquel dispositivo con 1 entrada, 2^n salidas, y n variables de control, de forma que el código binario contenido en las líneas de control indica cual de las salidas es la que se conecta a la entrada. El resto de las salidas toman un valor inactivo ("0" si son activas a tensión alta o "1" si son activas a tensión baja).

Construcción de un DEMUX

En el cuadro 2.6 presentamos las tablas de verdad del DEMUX 1 a 4 y del DEMUX 1 a 8 (activación a nivel alto). Las expresiones lógicas de las salidas son:

DEMUX 1 a 4: $o_0 = \overline{a}\,\overline{b}i$, $o_1 = \overline{a}bi$, $o_2 = a\overline{b}i$, $o_3 = abi$.

DEMUX 1 a 8: $o_0 = \overline{a}\,\overline{b}\,\overline{c}i$, $o_1 = \overline{a}\,\overline{b}ci$, $o_2 = \overline{a}b\overline{c}i$, $o_3 = \overline{a}bci$, $o_4 = a\overline{b}\,\overline{c}i$, $o_5 = a\overline{b}ci$, $o_6 = ab\overline{c}i$ y $o_7 = abci$.

Al igual que en el caso del MUX, normalmente se suele incluir una señal de *enable* o *strobe* (s) para la inhibición del dispositivo, con el siguiente funcionamiento:

- $s = 0$: El circuito está inhibido y todas las salidas son siempre cero ($o_i = 0$, para todo i).
- $s = 1$: Funcionamiento normal, la salida seleccionada es igual a la entrada.

Cuadro 2.6: Tablas de verdad para DEMUX 1 a 4 y DEMUX 1 a 8.

a	b	i	o_0	o_1	o_2	o_3
0	0	0	0	0	0	0
0	0	1	1	0	0	0
0	1	0	0	0	0	0
0	1	1	0	1	0	0
1	0	0	0	0	0	0
1	0	1	0	0	1	0
1	1	0	0	0	0	0
1	1	1	0	0	0	1

a	b	c	i	o_0	o_1	o_2	o_3	o_4	o_5	o_6	o_7
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	1	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	0	0	0
0	0	1	1	0	1	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	0	0
0	1	0	1	0	0	1	0	0	0	0	0
0	1	1	0	0	0	0	0	0	0	0	0
0	1	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	0	0	0	0
1	0	1	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	0	0	0
1	1	0	1	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0	1

La inclusión de esta entrada en las expresiones lógicas se realiza simplemente multiplicando cada término producto por s:

DEMUX 1 a 4: $o_0 = s\overline{a}\,\overline{b}i$, $o_1 = s\overline{a}bi$, $o_2 = sa\overline{b}i$, $o_3 = sabi$.

DEMUX 1 a 8: $o_0 = s\overline{a}\,\overline{b}\,\overline{c}i$, $o_1 = s\overline{a}\,\overline{b}ci$, $o_2 = s\overline{a}b\overline{c}i$, $o_3 = s\overline{a}bci$, $o_4 = sa\overline{b}\,\overline{c}i$, $o_5 = sa\overline{b}ci$, $o_6 = sab\overline{c}i$ y $o_7 = sabci$.

La construcción de estos DEMUXes a partir de puertas lógicas es la que se puede ver en la figura 2.24. En la figura 2.25 mostramos el diagrama de pines del C.I. 74154, que corresponde a un DEMUX 1 a 16 con la salida activa a nivel bajo, junto con su tabla de funcionamiento.

Árboles demultiplexores

El mayor DEMUX comercial disponible en forma de chip es de tamaño 1 a 16, pero podemos construir DEMUXes de cualquier tamaño interconectando varios DEMUX en una estructura de árbol. Por ejemplo, podemos implementar un DEMUX 1 a 32 a partir de un DEMUX 1 a 4 y cuatro DEMUXes 1 a 8, tal como se muestra en la figura 2.26.

El DEMUX del primer nivel lleva la entrada a una de sus cuatro salidas dependiendo de los bits de control a y b. Los DEMUXes del segundo nivel llevan cada una de las salidas del DEMUX del primer nivel a la salida seleccionada en función de los bits de control comunes c, d y e. El resultado final es que la entrada se lleva a una de las 32 salidas en función de las cinco líneas de control a, b, c, d y e. Notar que al DEMUX del primer nivel (DEMUX 1 a 4) van las líneas de control más significativas.

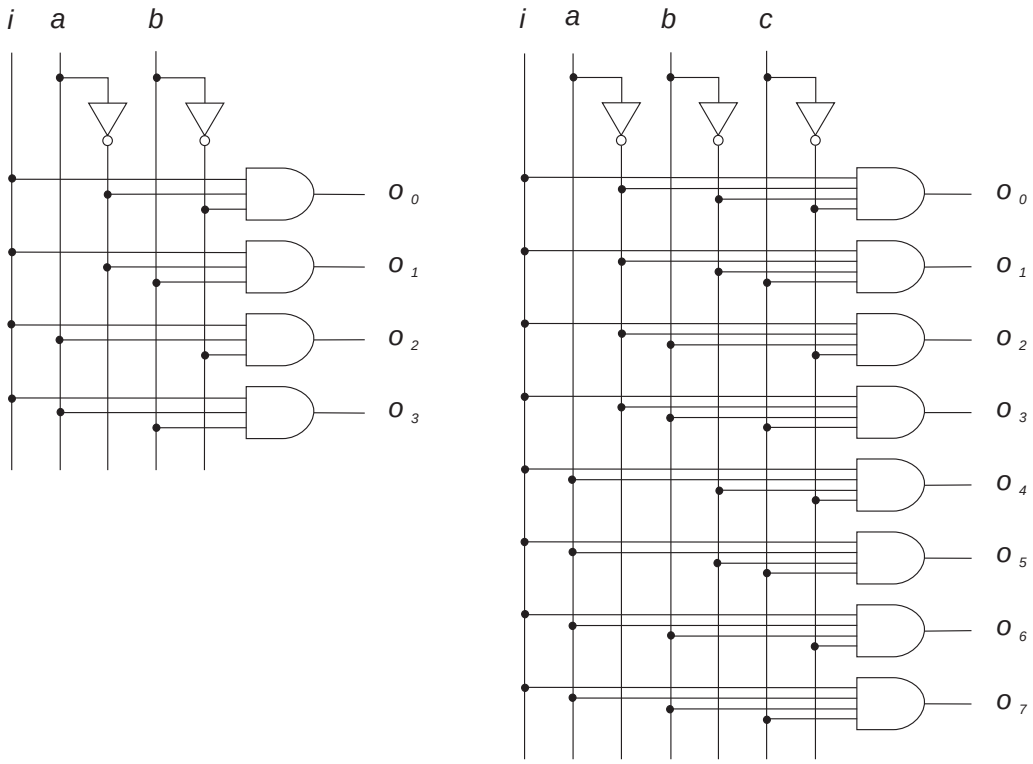


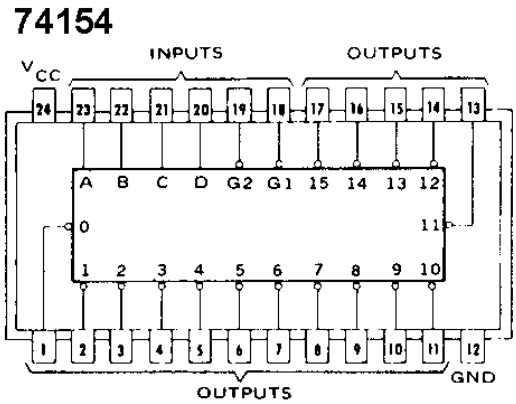
Figura 2.24: Construcciones del DEMUX 1 a 4 y del DEMUX 1 a 8.

El tamaño del DEMUX global se obtiene multiplicando los tamaños de los DEMUX de los dos niveles. En este caso, el DEMUX 1 a 4 y los DEMUXes 1 a 8 dan lugar a un DEMUX 1 a 4×8 (DEMUX 1 a 32). Se pueden construir árboles demultiplexores de cualquier número de salidas sin más que añadir niveles de DEMUXes.

2.5. MANIPULADORES DE CÓDIGO

En esta sección veremos los siguiente dispositivos:

- **Codificador binario**: con 2^n entradas, de las cuales solo una de ellas es activa, genera en las n salidas el código binario asociado a esa línea (código de n bits).
- **Decodificador binario**: el código binario generado por las n entradas activa una de entre 2^n salidas.
- **Conversor de código**: Con un número arbitrario de entradas y salidas transforma las entradas de un código en salidas de otro.



(a) Diagrama de pines

Function Table

Inputs						Outputs														
G1	G2	D	C	B	A	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
L	L	L	L	L	L	L	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H
L	L	L	L	L	H	H	L	H	H	H	H	H	H	H	H	H	H	H	H	H	H
L	L	L	L	H	L	H	H	L	H	H	H	H	H	H	H	H	H	H	H	H	H
L	L	L	L	H	H	H	H	H	L	H	H	H	H	H	H	H	H	H	H	H	H
L	L	L	H	L	L	H	H	H	H	L	H	H	H	H	H	H	H	H	H	H	H
L	L	L	H	L	H	H	H	H	H	H	L	H	H	H	H	H	H	H	H	H	H
L	L	L	H	H	L	H	H	H	H	H	H	L	H	H	H	H	H	H	H	H	H
L	L	L	H	H	H	H	H	H	H	H	H	H	L	H	H	H	H	H	H	H	H
L	L	H	L	L	L	H	H	H	H	H	H	H	H	L	H	H	H	H	H	H	H
L	L	H	L	L	H	H	H	H	H	H	H	H	H	H	L	H	H	H	H	H	H
L	L	H	L	H	L	H	H	H	H	H	H	H	H	H	H	L	H	H	H	H	H
L	L	H	L	H	H	H	H	H	H	H	H	H	H	H	H	H	L	H	H	H	H
L	L	H	H	L	L	H	H	H	H	H	H	H	H	H	H	H	H	L	H	H	H
L	L	H	H	L	H	H	H	H	H	H	H	H	H	H	H	H	H	H	L	H	H
L	L	H	H	H	L	H	H	H	H	H	H	H	H	H	H	H	H	H	H	L	H
L	L	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	L
L	H	X	X	X	X	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H
H	L	X	X	X	X	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H
H	H	X	X	X	X	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H

H = High Level, L = Low Level, X = Don't Care

(b) Tabla de funcionamiento

Figura 2.25: Demultiplexor 74154.

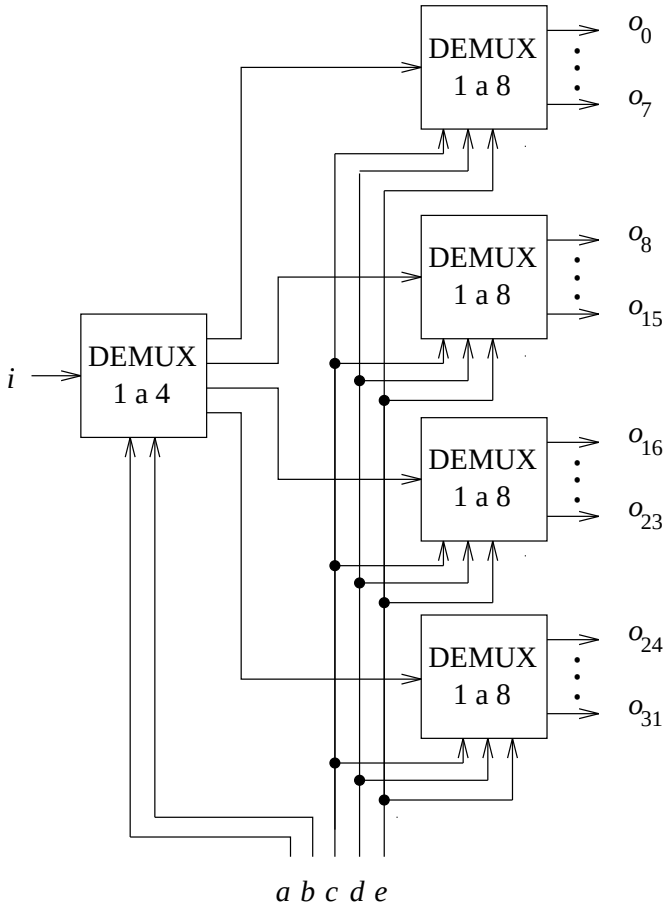


Figura 2.26: Árbol demultiplexor 1 a 32.

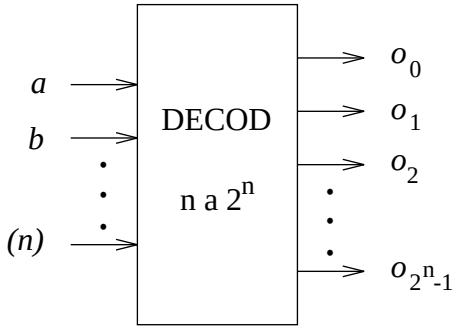


Figura 2.27: Representación de un decodificador.

Cuadro 2.7: Tablas de verdad para un decodificador 2 a 4 y para un decodificador 3 a 8.

a	b	o_0	o_1	o_2	o_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

a	b	c	o_0	o_1	o_2	o_3	o_4	o_5	o_6	o_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

2.5.1. Decodificadores binarios

La función de un decodificador binario es recibir el código binario de la entrada y activar (poner a 1 si la activación es a nivel alto o a cero si es a nivel bajo) la línea de salida que corresponde a ese código binario, dejando el resto de las salidas inactivas (proceso que se denomina decodificación). Un decodificador n a 2^n presentará n entradas y 2^n salidas (ver figura 2.27).

Las tablas de verdad del decodificador binario 2 a 4 y del decodificador binario 3 a 8 las mostramos en el cuadro 2.7. Las expresiones lógicas de las salidas son:

Decodificador 2 a 4: $o_0 = \overline{a}\,\overline{b}$, $o_1 = \overline{a}b$, $o_2 = a\overline{b}$, $o_3 = ab$.

Decodificador 3 a 8: $o_0 = \overline{a}\,\overline{b}\,\overline{c}$, $o_1 = \overline{a}\,\overline{b}c$, $o_2 = \overline{a}b\overline{c}$, $o_3 = \overline{a}bc$, $o_4 = a\overline{b}\,\overline{c}$, $o_5 = a\overline{b}c$, $o_6 = ab\overline{c}$ y $o_7 = abc$.

Estas expresiones son exactamente iguales a las de los DEMUX, pero con la diferencia de que no incluyen la entrada i. Por tanto los decodificadores binarios no se suelen construir como tales; lo que se hace es partir de un DEMUX y hacer la entrada dato $i = 1$. También se puede considerar un DEMUX como un decodificador con señal de *strobe*, donde la entrada i estaría haciendo esta función. Teniendo en cuenta esto, también podemos concluir que decodificadores mayores de 4 a 16 pueden ser construidos a partir de árboles demultiplexores poniendo la primera entrada a uno ($i = 1$).

2.5.2. Codificadores binarios

Un codificador binario es el dispositivo inverso a un decodificador. La función de este dispositivo es generar el código binario de la única línea de entrada que está activa en cada instante de un conjunto de varias entradas (proceso denominado codificación). Un codificador 2^n a n presentará 2^n entradas y n salidas. En principio solo se podrá poner a 1 una de las 2^n entradas. Por ejemplo, en el cuadro 2.8 mostramos las tablas de un codificador binario 4 a 2 y de un codificador binario 8 a 3, donde solo hemos hemos

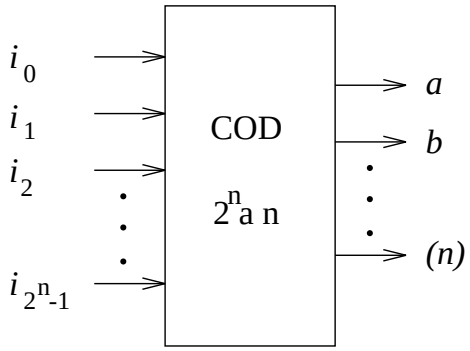


Figura 2.28: Representación de un codificador.

Cuadro 2.8: Tablas de verdad para un codificador 4 a 2 y para un codificador 8 a 3.

i_0	i_1	i_2	i_3	a	b
1	0	0	0	0	0
0	1	0	0	0	1
0	0	1	0	1	0
0	0	0	1	1	1
otras comb.				–	–

i_0	i_1	i_2	i_3	i_4	i_5	i_6	i_7	a	b	c
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1
otras combinaciones								–	–	–

incluido las combinaciones de entrada permitidas. Las expresiones lógicas de las salidas son las siguientes:

Codificador 4 a 2: $a = i_2 + i_3$, $b = i_1 + i_3$.

Codificador 8 a 3: $a = i_4 + i_5 + i_6 + i_7$, $b = i_2 + i_3 + i_6 + i_7$, $c = i_1 + i_3 + i_5 + i_7$.

Como puede observarse las expresiones de las salidas son la suma lógica de los términos de las líneas de entrada a 1 que ponen dicha salida a 1. Estas expresiones sencillas se deben al gran número de indiferencias que presentan las salidas. En la figura 2.29 mostramos el diagrama lógico del codificador binario 4 a 2, según las ecuaciones anteriores.

Codificadores con prioridad

Cabe preguntarse qué sucede en el diseño anterior cuando se ponen varias de las líneas de entrada a 1, cuál de los códigos binarios asociados a cada una de esas líneas de entrada es el que se tomará como salida. Tal como hemos diseñado el dispositivo (poniendo indiferencias en las salidas no permitidas) no podemos decir nada sobre esta cuestión.

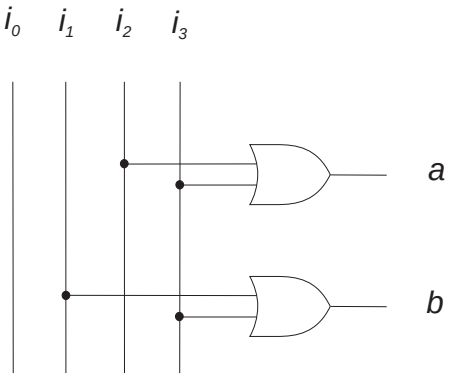


Figura 2.29: Codificador 4 a 2.

Cuadro 2.9: Codificador 4 a 2 con prioridad.

i_0	i_1	i_2	i_3	a	b
x	x	x	1	1	1
x	x	1	0	1	0
x	1	0	0	0	1
1	0	0	0	0	0
0	0	0	0	0	0

$$a = i_3 + i_2\bar{i}_3 = i_3 + i_2$$

$$b = i_3 + i_1\bar{i}_2\bar{i}_3 = i_3 + i_1\bar{i}_2$$

Es posible imponer prioridades a las líneas de entrada, de tal forma que si varias de ellas están activas el codificador solo tendrá en cuenta a la más prioritaria. En el cuadro 2.9 mostramos la tabla de verdad de un codificador 4 a 2 con prioridad y las expresiones lógicas de sus salidas. Hemos supuesto que las líneas de mayor peso son las más prioritarias: $i_3 > i_2 > i_1 > i_0$. El orden $i_3 > i_2 > i_1 > i_0$ es el orden de prioridad más usual y éstos van a ser los circuitos codificadores que se encuentren en el mercado.

En la tabla de verdad solo ha de tenerse en cuenta la línea más prioritaria a uno. Así, por ejemplo, si $i_2 = 1$ e $i_3 = 0$ sabemos que la salida ha de ser 2, independientemente de los valores de las líneas i_0 e i_1 (segunda fila de la tabla). Por otro lado, la obtención de estas ecuaciones a partir de la tabla de verdad es similar a la construcción de funciones en forma de suma de minterm, pero teniendo en cuenta que las variables de entrada que son "x" no intervienen en la formación del término producto. A diferencia de un codificador sin prioridad, en un codificador con prioridad todas las combinaciones de entrada tienen definido un valor de salida y, por lo tanto, no hay indiferencias en las funciones de salida del codificador.

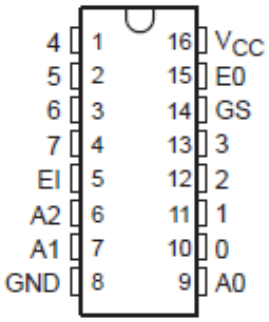
En la figura 2.30 mostramos el diagrama de pines del C.I. 74148 junto con su tabla de funcionamiento. Este chip corresponde a un codificador 8 a 3 con prioridad. Como comentario final, indicar que los codificadores comerciales binarios pueden llegar a ser de 16 a 4. Para diseñar codificadores mayores no es posible construir árboles de decodificadores siguiendo el mismo método empleado para MUX y DEMUX y habría que estudiar cada caso en particular.

FUNCTION TABLE – '148, 'LS148

INPUTS									OUTPUTS				
EI	0	1	2	3	4	5	6	7	A2	A1	A0	GS	EO
H	X	X	X	X	X	X	X	X	H	H	H	H	H
L	H	H	H	H	H	H	H	H	H	H	H	H	L
L	X	X	X	X	X	X	X	L	L	L	L	L	H
L	X	X	X	X	X	X	L	H	L	L	H	L	H
L	X	X	X	X	X	L	H	H	L	H	L	L	H
L	X	X	X	X	L	H	H	H	L	H	H	L	H
L	X	X	X	L	H	H	H	H	H	L	L	L	H
L	X	X	L	H	H	H	H	H	H	L	H	L	H
L	X	L	H	H	H	H	H	H	H	H	L	L	H
L	L	H	H	H	H	H	H	H	H	H	H	L	H

H = high logic level, L = low logic level, X = irrelevant

(a) Tabla de funcionamiento



(b) Diagrama de pines

Figura 2.30: Codificador 8 a 3 con prioridad 74148.

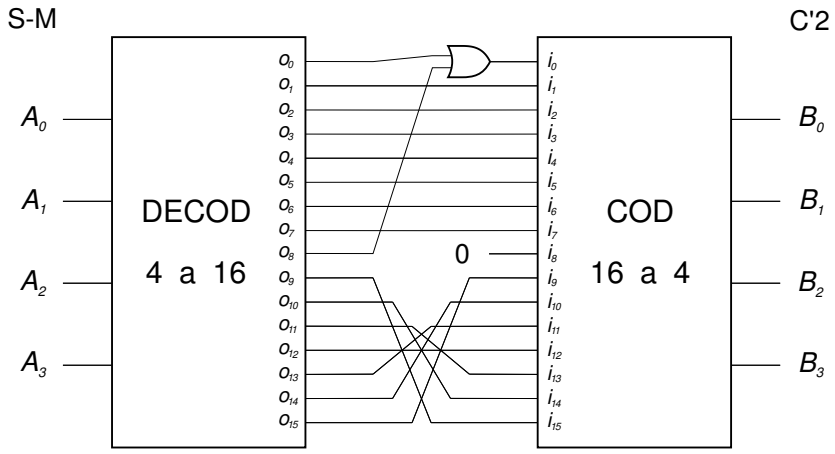


Figura 2.31: Diseño de un conversor S–M a C'2 para números de 4 bits.

2.5.3. Conversores de código

Un conversor de código es un dispositivo que genera la traducción entre dos códigos diferentes. Los números de bits en la entrada y la salida de este dispositivo vienen dados respectivamente por la longitud del código de partida y del código traducido. La construcción de estos dispositivos es particular para cada tipo de conversión de códigos elegida. Una forma sencilla de realizar conversores es partiendo de decodificadores y codificadores. En la figura 2.31 podemos ver un ejemplo de un conversor de números de 4 bits en formato signo–magnitud a formato complemento a 2, utilizando un decodificador binario 4 a 16 y un codificador binario 16 a 4. Sin embargo, este método tiene el inconveniente de la limitación del tamaño de los codificadores, con lo cual si el código de salida es de más de cuatro bits, ya habría que estudiar una implementación específica para el conversor.

Un ejemplo interesante es la conversión BCD a siete segmentos. Un visualizador (*display*) de siete segmentos consta de siete segmentos etiquetados a, b, c, d, e, f y g (figura 2.32), que pueden ser iluminados individualmente mediante LEDs. El visualizador incluye una entrada de control para cada segmento de forma que si, por ejemplo, la en-

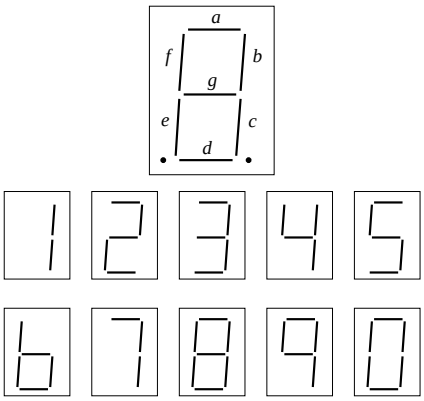


Figura 2.32: Visualizador de siete segmentos.

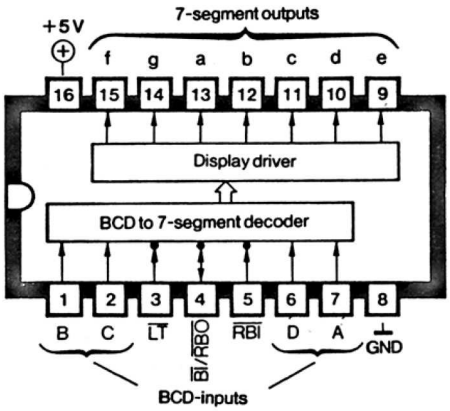
Cuadro 2.10: Conversión BCD a 7 segmentos.

ENTRADAS				SALIDAS							
D	C	B	A	*a*	*b*	*c*	*d*	*e*	*f*	*g*	DISPLAY
0	0	0	0	1	1	1	1	1	1	0	0
0	0	0	1	0	1	1	0	0	0	0	1
0	0	1	0	1	1	0	1	1	0	1	2
0	0	1	1	1	1	1	1	0	0	1	3
0	1	0	0	0	1	1	0	0	1	1	4
0	1	0	1	1	0	1	1	0	1	1	5
0	1	1	0	0	0	1	1	1	1	1	6
0	1	1	1	1	1	1	0	0	0	0	7
1	0	0	0	1	1	1	1	1	1	1	8
1	0	0	1	1	1	1	0	0	1	1	9
1	0	1	0	–	–	–	–	–	–	–	–
1	0	1	1	–	–	–	–	–	–	–	–
1	1	0	0	–	–	–	–	–	–	–	–
1	1	0	1	–	–	–	–	–	–	–	–
1	1	1	0	–	–	–	–	–	–	–	–
1	1	1	1	–	–	–	–	–	–	–	–

trada correspondiente al segmento *a* está activa éste se iluminará, mientras que si está inactiva el segmento permanecerá apagado. Igual para los seis segmentos restantes. La activación de un segmento puede ser con un valor alto (HIGH) cuando el visualizador es de cátodo común, o con un valor bajo (LOW) cuando el visualizador es de ánodo común.

El código BCD (código binario decimal) consta de 4 bits en los cuales las combinaciones posibles son las que generan los números binarios 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, que son precisamente los dígitos que se emplean en el sistema decimal. La conversión BCD a 7 segmentos viene dada por el cuadro 2.10.

En la figura 2.33 mostramos el diagrama de pines y la tabla de funcionamiento del C.I. 7448, que corresponde al conversor de código BCD a 7 segmentos para visualizadores de cátodo común.



(a) Diagrama de pines

TRUTH TABLE
SN54/74LS48

	INPUTS						OUTPUTS							
DECIMAL OR FUNCTION	$\overline{LT}$	$\overline{RBI}$	D	C	B	A	$\overline{BI/RBO}$	a	b	c	d	e	f	g	NOTE
0	H	H	L	L	L	L	H	H	H	H	H	H	H	L	1
1	H	X	L	L	L	H	H	L	H	H	L	L	L	L	1
2	H	X	L	L	H	L	H	H	H	L	H	H	L	H	
3	H	X	L	L	H	H	H	H	H	H	H	L	L	H	
4	H	X	L	H	L	L	H	L	H	H	L	L	H	H	
5	H	X	L	H	L	H	H	H	L	H	H	L	H	H	
6	H	X	L	H	H	L	H	L	L	H	H	H	H	H	
7	H	X	L	H	H	H	H	H	H	H	L	L	L	L	
8	H	X	H	L	L	L	H	H	H	H	H	H	H	H	
9	H	X	H	L	L	H	H	H	H	H	L	L	H	H	
10	H	X	H	L	H	L	H	L	L	L	H	H	L	H	
11	H	X	H	L	H	H	H	L	L	H	H	L	L	H	
12	H	X	H	H	L	L	H	L	H	L	L	L	H	H	
13	H	X	H	H	L	H	H	H	L	L	H	L	H	H	
14	H	X	H	H	H	L	H	L	L	L	H	H	H	H	
15	H	X	H	H	H	H	H	L	L	L	L	L	L	L	
BI	X	X	X	X	X	X	L	L	L	L	L	L	L	L	2
RBI	H	L	L	L	L	L	L	L	L	L	L	L	L	L	3
LT	L	X	X	X	X	X	H	H	H	H	H	H	H	H	4

NOTES:
(1) BI/RBO is wired-AND logic serving as blanking input ($\overline{BI}$) and/or ripple-blanking output (RBO). The blanking out (BI) must be open or held at a HIGH level when output functions 0 through 15 are desired, and ripple-blanking input ($\overline{RBI}$) must be open or at a HIGH level if blanking of a decimal 0 is not desired. X=input may be HIGH or LOW.
(2) When a LOW level is applied to the blanking input (forced condition) all segment outputs go to a LOW level, regardless of the state of any other input condition.
(3) When ripple-blanking input ($\overline{RBI}$) and inputs A, B, C, and D are at LOW level, with the lamp test input at HIGH level, all segment outputs go to a HIGH level and the ripple-blanking output (RBO) goes to a LOW level (response condition).
(4) When the blanking input/ripple-blanking output ($\overline{BI/RBO}$) is open or held at a HIGH level, and a LOW level is applied to lamp-test input, all segment outputs go to a LOW level.

(b) Tabla de funcionamiento

Figura 2.33: C.I. 7448: conversor BCD a 7 segmentos.