Preguntas Más Frecuentes Tema 2

Contenido

CONVERTIDORES de CÓDIGO

P.2.1: ¿Cómo se obtienen las expresiones del Convertidor de Código S-M a C-1 de la fig. 5.4 del texto?

R.2.1: El problema que nos planteamos es el de diseñar el convertidor de números positivos y negativos representados con palabras de tres bits en S-M a los mismos números representados en C-1 (*fig. 5.4, pag 269 del texto base*).

Para resolver el problema tenemos que calcular las expresiones lógicas de las salidas del circuito convertidor, y_2, y_1 e y_0, que representan en C-1 a los mismos números de entrada los cuales están representados en S-M mediante las variables de entrada x_2, x_1 y x_0. Es decir, tenemos que diseñar un circuito en el que, cuando en su entrada se le presente una palabra de tres bis que en S-M representa un determinado nº decimal, presente en su salida una palabra también de tres bits y que sea la correspondiente al mismo número decimal, pero ahora representado en C-1. Por ejemplo, si al circuito le entra la palabra 110 que en S-M representa al número -2 debe presentar en su salida la palabra 101 que también representa -2, pero ahora en C-1, Por tanto, el circuito debe convertir la palabra de entrada 110 en la de salida 101 y, de igual forma debe realizar la conversión para todas y cada una de las palabras de entrada.

Lo primero que tenemos que hacer es construir la tabla de verdad en la que figuren todos y cada uno de los números positivos y negativos que podemos representar con palabras de tres bits. En este caso, con 3 bits sólo tenemos 2^3=8 configuraciones posibles, así que usaremos 4 para representar los números positivos y otras 4 para los números negativos, incluido el cero. Esto significa que, tanto en S-M como en C-1, con 3 bits sólo podemos representar los números decimales comprendidos entre -3 y +3. Por tanto, las tablas que aparecen en la *figura 5.4 del texto*, y que repetimos aquí unidas, presentan las equivalencias entre las tres formas de representar los correspondientes números positivos y negativos en DECIMAL, S-M y C-1.

Así, la tabla de verdad del convertidor a diseñar es:

Nº DECIMAL equivalente	ENTRADAS Representación en S-M del nº DECIMAL			SALIDAS Representación en C-1 del nº DECIMAL		
	x_2 (MSB) **signo**	x_1	x_0 (LSB)	y_2 (MSB) **signo**	y_1	y_0 (LSB)
+3	0	1	1	0	1	1
+2	0	1	0	0	1	0
+1	0	0	1	0	0	1
+0	0	0	0	0	0	0
-0	1	0	0	1	1	1
-1	1	0	1	1	1	0
-2	1	1	0	1	0	1
-3	1	1	1	1	0	0

Como podemos observar, **los números positivos coinciden en ambas representaciones**, mientras que ***los números negativos en C-1 son los complementados de la representación en S-M*** (*ver figura 5.1, 5.2 y 5.3 del texto*).

Dado que tenemos que obtener las expresiones lógicas de y_2, y_1 e y_0 en función de x_2, x_1 y x_0, lo que hacemos es sumar los términos mínimos correspondientes a las variables de entrada, x_2, x_1 y x_0, que hacen que cada una de las y_i tome el valor "1".

Si empezamos por el bit más significativo, y_2, observamos que coincide con el bit más significativo de la entrada, x_2. Por tanto, ya podemos poner que $y_2 = x_2$ lo que supone que en el circuito debemos poner un hilo que una dichas variables (*ver circuito de la figura 5.4 del texto*).

A continuación vamos a calcular las expresiones de las otras dos variables. Así, si en la columna encabezada por y_1 seleccionamos los unos que son los valores que hacen que se verifique y_1, y sumamos los correspondientes términos mínimos de las variables de entrada x_2, x_1 y x_0 resulta:

$$y_1 = \overline{x}_2\, x_1\, x_0 + \overline{x}_2\, x_1\, \overline{x}_0 + x_2\, \overline{x}_1\, \overline{x}_0 + x_2\, \overline{x}_1\, x_0$$

Sacando factor común $\overline{x}_2\, x_1$ de los dos primeros términos mínimos y $x_2\, \overline{x}_1$ de los dos últimos, obtenemos:

$$y_1 = \overline{x}_2\, x_1 + x_2\, \overline{x}_1 = x_2 \oplus x_1$$ (Ver la función XOR en la pag 55).

De forma análoga se obtiene la expresión de y_0. Resultando: $y_0 = x_2 \oplus x_0$

Resumiendo, las expresiones lógicas correspondientes a las variables de salida de un circuito que convierte un número representado con tres bits en S-M a C-1 son:

$$\boldsymbol{y_2 = x_2}\ ,\quad \boldsymbol{y_1 = x_2 \oplus x_1}\ ,\quad \boldsymbol{y_0 = x_2 \oplus x_0}$$

El circuito que las implementa es el que aparece en la parte inferior de la *figura 5.4 del texto*.

✦✦✦

P.2.2: Si para obtener un nº representado en C-2 basta con complementar los números representados en binario y después sumarle 1, es decir, obtener su representación en C-1 y sumarle aritméticamente un 1. ¿Podemos obtener un convertidor de S-M a C-2 añadiéndole un sumador al convertidor de S-M a C-1 con el que le sumemos un "1"?

R.2.2: No, sobre todo si el número de bits de los números que vamos a manejar es reducido. Recordemos que la representación de los números positivos coinciden en las tres representaciones (S-M, C-1 y C-2) y que la afirmación de la cuestión sólo es válida para la representación de los números negativos. Esto implica que habría que diseñar un circuito que, además de sumarle un "1", detectara si el nº a convertir es positivo o negativo y le sumara el "1" sólo en este último caso. Por tanto, el circuito constaría de un convertidor de S-M a C-1, un sumador y un circuito encargado de detectar que el bit más significativo del número a convertir vale "1".

La solución planteada en la cuestión es válida cuando manejamos números muy grandes en los que, para su representación, hacen falta muchos bits y esto complica mucho el diseño.

En la pregunta anterior hemos explicado paso a paso cómo diseñar un convertidor de números de 3 bits representados S-M a C-1. Veamos ahora de forma simplificada el diseño y la simulación de los convertidores de S-M a C-1 y a C-2. Así, la Tabla de verdad es:

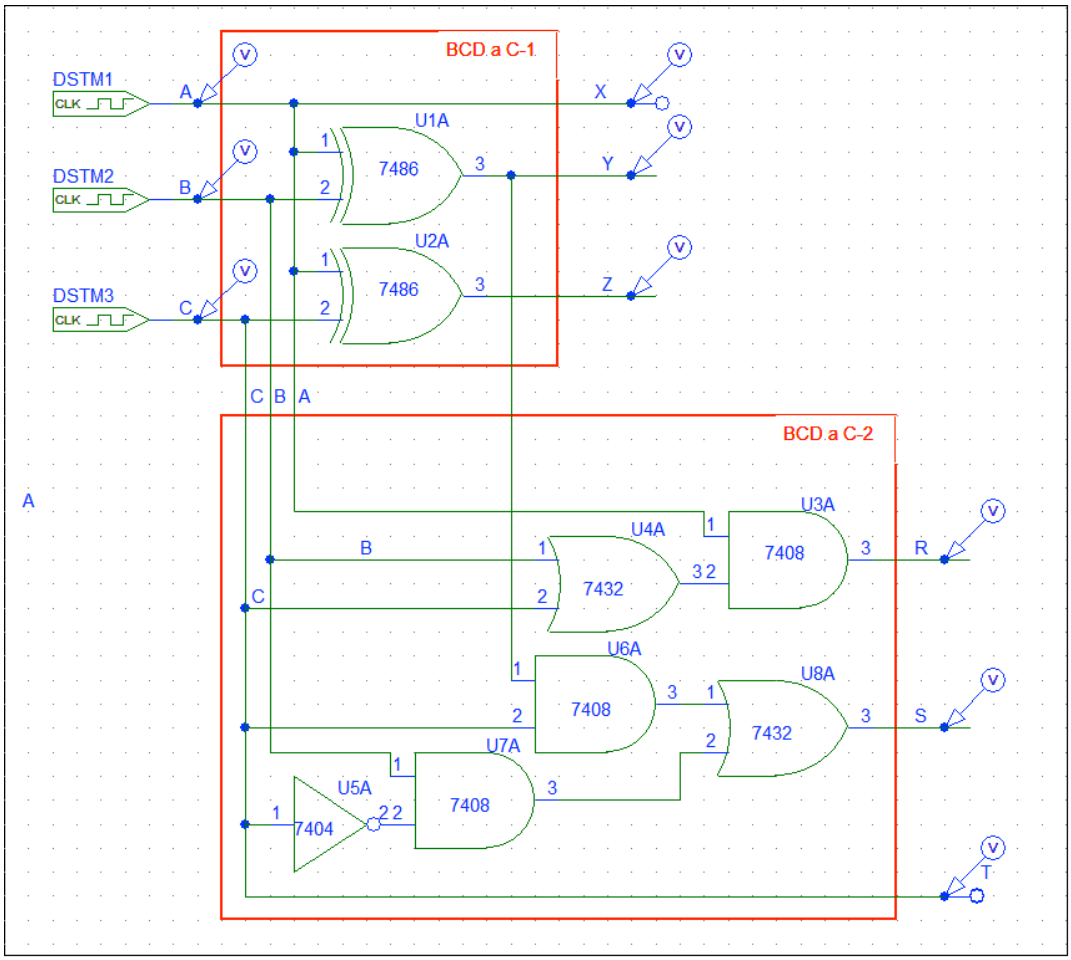
Decimal	S-M			C-1			C-2		
	A (MSB)	B	C	X (MSB)	Y	Z	R (MSB)	S	T
+0	0	0	0	0	0	0	0	0	0
+1	0	0	1	0	0	1	0	0	1
+2	0	1	0	0	1	0	0	1	0
+3	0	1	1	0	1	1	0	1	1
-0	1	0	0	1	1	1	0	0	0
-1	1	0	1	1	1	0	1	1	1
-2	1	1	0	1	0	1	1	1	0
-3	1	1	1	1	0	0	1	0	1

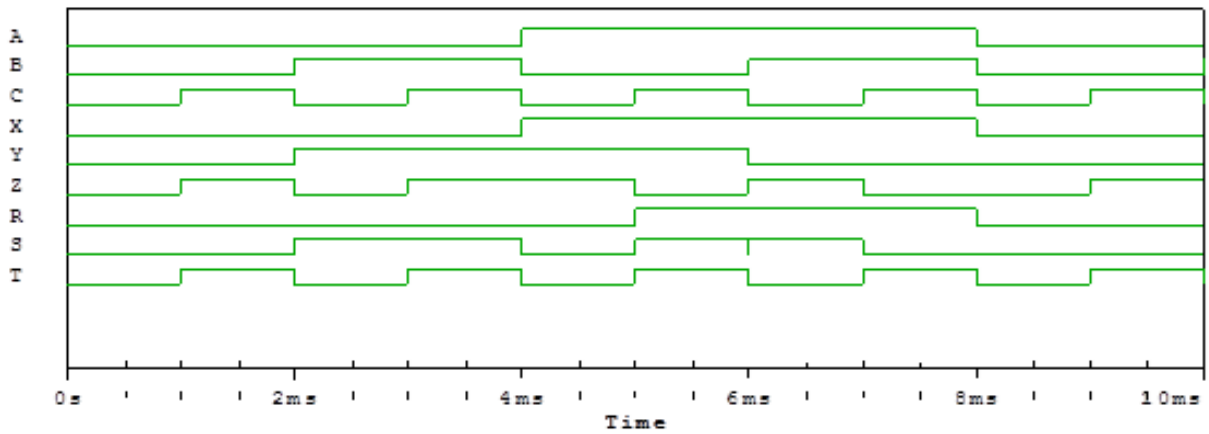
Obsérvese que en S-M y en C-1 existen el +0 y el -0 mientras que en C-2 no, por lo que en C-2 representamos ambos, +0 y -0, por la palabra 000.

Las expresiones de X, Y, Z (C-1) y R, S, T (C-2) en función de A, B, C (S-M) son, tras su minimización:

$$S\text{-}M\ a\ C\text{-}1\begin{cases} X = A \\ Y = \overline{A}B + A\overline{B} = A \oplus B \\ Z = \overline{A}C + A\overline{C} = A \oplus C \end{cases}$$

$$S\text{-}M\ a\ C\text{-}2\begin{cases} R = A(B + C) \\ S = \overline{A}B + B \oplus C \\ T = C \end{cases}$$



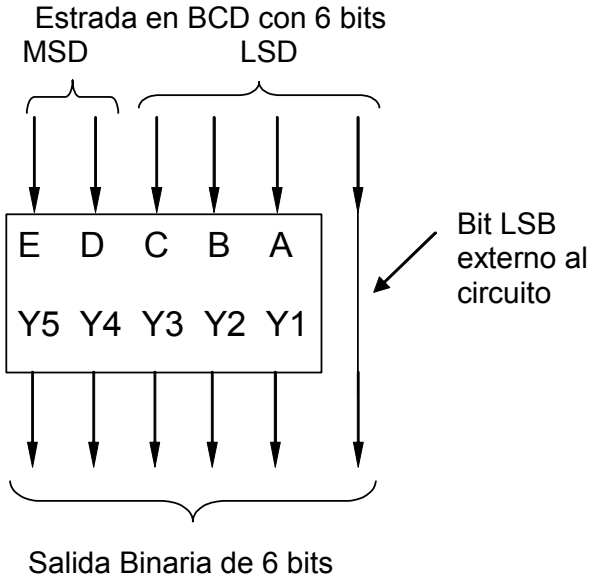


Para verificar que el diseño es correcto debemos construimos la tabla de verdad práctica a partir de este cronograma, leyendo las señales de arriba hacia abajo y viendo que ésta coincide con la tabla del inicio, punto de partida del diseño. No lo vamos a hacer, y lo dejmos como ejercicio para que lo hagáis vosotros.

P.2.3: ¿Podría explicar cómo usar en el simulador el circuito integrado SN74184, convertidor de código BCD a Binario, para comprobar su funcionamiento?

R.2.3: Cuando usamos un circuito integrado es importante ver previamente la información que en las hojas de características proporciona el fabricante. En estas hojas de características el fabricante da información que va desde, cómo están distribuidas las entradas, las salidas y la alimentación en el circuito integrado, hasta distintos tipo de aplicaciones, pasando por su descripción funcional, distintas formas de funcionamiento (si las tiene, como es este caso), tablas de verdad, características eléctricas, etc... Esta información es importante para que el diseñador seleccione el componente que más le interesa no sólo en términos de la función que realiza, sino de otras características que pueden ser de su interés.

El circuito integrado SN74184 además de funcionar como un convertidor de BCD a Binario, también funciona como convertidor de BCD a Complemento a 9 y a 10, por eso tiene 8 salidas. Para la conversión de BCD a Binario sólo se usan las 5 menos significativas (*y1* a *y5*) y las salidas *y6, y7* e *y8*, junto con las 5 anteriores, se usan para la conversión a C-9 y C-10. Además tiene otra característica que hay que tener en cuenta y es que, como el bit menos significativo de todas las palabras en BCD coincide con el bit menos significativo de todas las palabras en binario, este bit no viene integrado en el circuito sino que es un hilo que se pone externamente y que pasa directamente de la entrada a la salida como se muestra en la figura.



Así, las palabras de salida son de 6 bits, y las de entrada también son de 6 bits, lo que nos lleva a considerar que en BCD podemos representar desde 0 (00 0000) hasta 39 (11 1001). Es decir, la tabla de verdad que es:

	Señal de Control	Entrada en BCD	Salida en Binario

Número en decimal	(nG)	E D	C B A	bit externo LSB	Y5 Y4 Y3 Y2 Y1	bit externo LSB
0	0	0 0	0 0 0	0	0 0 0 0 0	0
1	0	0 0	0 0 0	1	0 0 0 0 0	1
2	0	0 0	0 0 1	0	0 0 0 0 1	0
3	0	0 0	0 0 1	1	0 0 0 0 1	1
...	...	...	...	...	...	...
9	0	0 0	1 0 0	1	0 0 1 0 0	1
10	0	0 1	0 0 0	0	0 0 1 0 1	0
11	0	0 1	0 0 0	1	0 0 1 0 1	1
...	...	...	...	...	...	...
19	0	0 1	1 0 0	1	0 1 0 0 1	1
20	0	1 0	0 0 0	0	0 1 0 1 0	0
21	0	1 0	0 0 0	1	0 1 0 1 0	1
...	...	...	...	...	...	...
29		1 0	1 0 0	1	0 1 1 1 0	1
30	0	1 1	0 0 0	0	0 1 1 1 1	0
31	0	1 1	0 0 0	1	0 1 1 1 1	1
	...	...	...	...		...
38	0	1 1	1 0 0	0	1 0 0 1 1	0
39	0	1 1	1 0 0	1	1 0 0 1 1	1
ninguno	1	x x	x x x	x	1 1 1 1 1	x

Para interpretar los resultados debemos recordar que en BCD los 4 bits menos significativos representan las unidades y los 4 bits más representativos las decenas. En este caso, el conversor SN74184, sólo usa dos bits para las decenas y es por eso por lo que sólo se puede representar en BCD hasta el número decimal 39.

Todo lo dicho para el circuito SN74184 es válido para el SN74185.

SUMADORES y RESTADORES

P.2.4: ¿Qué diferencia hay entre la suma lógica y la suma aritmética?

R.2.4: La ***suma lógica*** de 2 números representados en binario es la función OR bit a bit sobre dichos números. Es decir, es la función OR entre las parejas de bits que tienen la misma significancia en cada una de las palabras que representan dichos números. Por ejemplo, dadas dos palabras, A y B, que toman los valores:

$$A = A_3\, A_2\, A_1\, A_0 = 0010$$

$$B = B_3\, B_2\, B_1\, A_0 = 0110$$

Su suma lógica es:

$$A + B = A \text{ OR } B = (A_3 \text{ OR } B_3),(A_2 \text{ OR } B_2),(A_1 \text{ OR } B_1),(A_0 \text{ OR } B_0) = 0110$$

Donde hemos aplicado directamente a cada par de bit el valor correspondiente en la tabla de verdad de la función OR que aparece en la columna encabezada por ($x_1 + x_2$) de la *fig. 1.3, del tema 1 del texto*.

La ***suma aritmética*** es la suma que estamos acostumbrados a hacer cuando sumamos números en decimal, sólo que en vez de representar los números en base 10 (cada cifra puede tomar un valor de 0 a 9) los representamos en base 2 (cada cifra sólo puede tomar el valor 0 ó 1). Por tanto, al igual que hacemos al sumar en decimal, que cuando el resultado de sumar las unidades supera a la base menos 1, es decir, supera a 10-1=9, pasamos a las decenas (una cifra más por la izquierda) y nos llevamos uno, para sumar en base dos cuando superamos la base menos 1 (2-1=1) tenemos que usar un dígito más y también nos llevarnos "1". Así surgen las tablas de verdad del semisumador y del sumador completo en las que aparecen los acarreos, cosa que no ocurre con la función OR o suma lógica.

Resumiendo, en el caso de ***suma aritmética*** **hacemos realmente una suma,** como la hacemos cuando sumamos en decimal, y el resultado de esta suma tiene que coincidir con el de la suma de los sumandos equivales en decimal, lógicamente, teniendo en cuenta el tipo de representación que estamos usando.

Por ejemplo, si queremos hacer la suma aritmética de las mismas palabras anteriores, $A = A_3A_2A_1A_0=0010$ y $B = B_3B_2B_1B_0 = 0110$ deberemos ir sumando los bits desde la derecha hacia la izquierda, como hacemos cuando sumamos los números en base 10, y tener en cuenta que el acarreo que se produce cada vez que se suman dos bits hay que sumarlo a los siguiente bit hacia la izquierda. Así: si consideramos que las palabras representan números en binario puro y, por tanto, sin signo resulta:

Acarreo	1	1	0		
A	0	0	1	0	(=2 en su representación en decimal)
B	0	1	1	0	(=6 en su representación en decimal)
	1	0	0	0	(=8 en su representación en decimal)

Es importante tener en cuenta que al realizar la suma aritmética de números en las distintas representaciones (S-M, C-1, C-2, …) el resultado de la suma en la

representación binaria usada se tiene que corresponder con el número decimal resultante de realizar la suma de los equivalentes en decimal de los sumandos correspondientes. Es decir, si los números a sumar están representados en C-1 y sus equivalentes en decimal son, por ejemplo, 011=+3 y 101=-2, al realizar la suma aritmética en C-1 debe darnos un número en C-1 cuyo valor en decimal es +1, o sea debe dar 001. Esto es válido para cualquier tipo de representación y nos sirve para comprobar que la operación es correcta.

En la ALU la **suma aritmética** se representa por la palabra ***PLUS***, mientras que para la **suma lógica** se usa el signo **+** con independencia de que esté programada para realizar funciones lógicas o aritméticas.

✦✦✦

P.2.5: ¿Cómo se realiza la suma en el circuito de la figura 5.11 (página 276) del texto? No veo claro cómo se obtiene el resultado S=11000 a partir de la suma de A=01011 y B=01101

R.2.5: Al igual que cuando sumamos los números representados en decimal (en base 10), la suma se realiza desde la derecha hacia la izquierda y teniendo en cuenta el acarreo que se va produciendo conforme se van realizando las sumas parciales de los sucesivos dígitos (*ver P2.2 de las P+F de este tema*)

Así, inicialmente suponemos que $C_n=0$ y, además, tenemos como entradas de datos a sumar en el sumador completo: $A_0=1$, $B_0=1$. Esto genera a la salida $S_0= 0$ y $C_{n+1}=C_1=1$. Este valor de C_{n+1} se retarda un Δt (un pulso de reloj de sincronismos del circuito) y es el valor que toma C_n tras ese Δt, que es el tiempo que tarda la siguiente pareja de bits (A_1 y B_1) en presentarse en las entradas del sumador. Es decir, el acarreo actual que hay que sumar a cada pareja de bits coincide con el acarreo resultante de la suma de los dos bits anteriores, $C_n(t)= C_{n+1}(t-\Delta t)$.

El siguiente paso es la suma de $A_1 =1$ y $B_1 =0$ y el acarreo del bit anterior $C_n=C_1=1$, que da lugar a $S_1= A_1 + B_1 + C_1 = 1+0+1= 0$ y $C_{n+1}= C_2=1$ (que se sumará a los siguientes bits). Las siguientes entradas al sumador son $A_2 =0$ y $B_2 =1$ y el acarreo del bit anterior $C_2=1$, resultando $S_2=0+1+1= 0$ y $C_{n+1}= C_3=1$ (que se sumará a los siguientes bits).

Análogamente, para $A_3 =1$, $B_3 =1$ y $C_3=1$, su suma es $S_3=1+1+1= 1$ y su acarreo es $C_{n+1}= C_4=1$ y para $A_4 =0$, $B_4 =0$ y $C_4= 1$ su suma es $S_4=0+0+1= 1$ y $C_{n+1}=C_5=0$.

Por tanto:

$$\boldsymbol{A(01011) + B(01101) = S(11000)} \text{ y } \boldsymbol{C_{n+1}= C_5=0}$$

Una forma de comprobar que la suma es correcta es realizar la suma en decimal. Así, el número *A=01011* representado en binario puro equivale al número *11* representado en decimal. Análogamente, *B=01101* equivale a *13*. Por tanto, *A+B=11+13=24* que, representado en binario puro, es ***11000***. Como podemos comprobar coincide con el valor obtenido al realizar la suma en binario puro.

✦✦✦

P.2.6: ¿Se podría describir de forma algorítmica el procedimiento usado para realizar la suma de dos palabras de n bits en C-1?

R.2.6: Si que se puede describir mediante un algoritmo. Para ello vamos a usar la misma nomenclatura del texto. Se puede usar otra cualquiera, pero hay que definirla previamente de forma clara y precisa.

Una posible descripción algorítmica del procedimiento seguido en el texto para la suma en C-1 de palabras de n bits es:

1. Se suman bit a bit de las dos palabras A y B, → *S=A+B*. Esta suma da lugar a la siguiente palabra de n+1 bits:

   $$C_n\, S_{n-1}\, S_{n-2} \ldots S_0$$

   Siendo:

   Cn = An-1+Bn-1+Cn-1. El acarreo resultante de la suma de los dos bits más significativos de las palabras A y B y del acarreo resultante de la suma del bit anterior.

   Sn-1 Sn-2 … S0 = Palabra de n bits resultante de la suma aritmética de A+B y, por tanto, teniendo en cuenta los acarreos intermedios.

2. Si Cn = 0

   2.1. Si $\overline{A_{n-1}}\,\overline{B_{n-1}}\,S_n + A_{n-1}\,B_{n-1}\,\overline{S_{n-1}} = 0$ → RESULTADO: *S= Sn-1 Sn-2 … S0*, Rebose=0 → FIN

   2.2. Si $\overline{A_{n-1}}\,\overline{B_{n-1}}\,S_n + A_{n-1}\,B_{n-1}\,\overline{S_{n-1}} = 1$ → RESULTADO: S= Sn-1 Sn-2 … S0, Rebose=1 → FIN

3. Si Cn = 1 → S=A+B+1

   3.1. Si $\overline{A_{n-1}}\,\overline{B_{n-1}}\,S_n + A_{n-1}\,B_{n-1}\,\overline{S_{n-1}} = 0$ → RESULTADO: S= Sn-1 Sn-2 …S0, Rebose=0 → FIN

   3.2. Si $\overline{A_{n-1}}\,\overline{B_{n-1}}\,S_n + A_{n-1}\,B_{n-1}\,\overline{S_{n-1}} = 1$ → RESULTADO: S= Sn-1 Sn-2 …S0, Rebose=1 → FIN

✦✦✦

P.2.7: ¿Por qué, en el circuito sumador paralelo de 4 bits con lógica de acarreo adelantado de la figura 5.12, página 278 del texto, se ponen las puertas XOR para generar las salidas S0, S1, S2, S3?.

R.2.7: El circuito es una réplica de las funciones obtenidas en el diseño (*ecuaciones 5.9, 5.10 y 5.11*). Así, en la *expresión 5.10 del texto* se obtiene la expresión general de la suma de cada bit. Es decir,

$$S_i = P_i \oplus C_i \qquad \text{siendo} \qquad P_i = A_i \oplus B_i$$

Como el sumador es de 4 bits, para obtener la suma de cada bit tenemos que particularizar y darle los valores 0, 1, 2, y 3 al subíndice "*i*" de las expresiones anteriores. Por tanto, obtenemos:

$$S_0 = P_0 \oplus C_0$$
$$S_1 = P_1 \oplus C1$$
$$S_2 = P_2 \oplus C_2$$
$$S_3 = P_3 \oplus C_3$$

Como en la salida de la parte recuadrada y matizada de la *figura 5.12 del texto* obtenemos las distintas P_i y C_i, tendremos que realizar la función XOR de cada una de las parejas P_i y C_i para obtener las S_i correspondientes.

✦✦✦

P.2.8: ¿Cuál es el significado de las variables que aparecen en la cabecera de la tabla de la figura 5.14, página 282, del texto?

R.2.8: La nomenclatura que se sigue en esta tabla es la misma que se usa en la parte a) de la figura 5.6. Es decir, la suma de los bits menos significativos, A0 y B0, da lugar a la suma, S0, y al acarreo C1 (que se debe sumar a los siguientes bits más significativos). La suma de A1 y B1 y del acarreo C1 da lugar a S1 y al acarreo C2.

✦✦✦

P.2.9: No entiendo la expresión del rebose que aparece en la página 282 del texto:

$rebose = \overline{A}_1\ \overline{B}_1\ S_1 + A_1\ B_1\ \overline{S}_1$ **?**

R.2.9: Recuerde que existe rebose siempre que para representar el resultado de la suma de dos números nos hace falta un bit más y no lo tenemos porque el número de bits con el que estamos trabajando es menor que el número de bits con el que tenemos que representar el resultado. Por tanto, debemos generar una señal para advertir que la solución no es la correcta.

Si tenemos sólo dos bits para representar números positivos y negativos en C-1 resulta que sólo podemos representar los siguientes números:

C-1	00	01	10	11
Equivalente en Decimal	+0	+1	-1	-0

Si realizamos la suma de dos números positivos el resultado debe ser un número también positivo y si realizamos la suma de dos números negativos el resultado deberá ser también un número negativo.

Pues bien, veamos el rebose a través de un ejemplo. Supongamos que sumamos la palabra $A=A_1$(signo)A_0=01 (+1 en decimal) con la palabra $B=B_1$(signo)B_0=01 (+1 en decimal) el resultado es 010 (+2 en decimal). Este resultado es correcto si tuviéramos 3 bits para su representación, pero como sólo tenemos dos bits, el resultado obtenido, ~~0~~10, queda truncado y sólo obtenemos como resultado de la suma la palabra 10. Si

verificamos este resultado en la tabla vemos que no es correcto ya que 10 corresponde a -1 en decimal. Obsérvese que, además, hemos sumado dos números positivos (01+01) y hemos obtenido un número negativo (10), cosa que es imposible. Por tanto, la operación no es correcta, ***hay rebose***.

En resumen, podemos decir que hay rebose cuando los bit de signo de los dos sumando son 0 (números positivos) y el bit de signo del resultado, S_1, es 1 (número negativo). Esto da lugar al sumando de la *expresión [5.12]*: $\overline{A}_1\ \overline{B}_1\ S_1$

Análogamente, aplicando el mismo razonamiento nos encontramos que *también hay rebose* cuando sumamos dos números negativos y el resultado es un número positivo. Por ejemplo, si sumamos la palabra $A=A_1$(signo)A_0=10 (-1 en decimal) con la palabra $B=B_1$(signo)B_0=10 (-1 en decimal) el resultado es 100 (aquí, además existe el problema del resultado erróneo que se resuelve sumándole 1, por ser en C-1, explicado en la *página 280*). Por tanto, ***también hay rebose*** cuando los bits de signo de los sumandos son 1 (números negativos) y el bit de signo del resultado, S_1, es 0 (número positivo). Esto da lugar al otro sumando de la *expresión [5.12]*: $A_1\ B_1\ \overline{S}_1$.

Las sumas del resto de las posibles combinaciones de los cuatro números no presentan rebose.

Por tanto, la expresión del rebose de un sumador de 2 bits en C-1 es:

$$rebose = \overline{A}_1\ \overline{B}_1\ S_1 + A_1\ B_1\ \overline{S}_1$$

✦✦✦

P.2.10: ¿Qué sentido tiene el diseño modular y en qué consiste?

R.2.10: Como norma general, a la hora de realizar el diseño de un circuito electrónico se intenta que sea modular con el fin de que si hay necesidad de ampliar el diseño, por ejemplo, aumentando el número de bits o de palabras en el caso de que el diseño sea de un circuito digital, no haya necesidad de hacer un nuevo diseño sino que baste con unir dos o más módulos iguales.

Para ello, a la hora de realizar dicho diseño o de implementar el circuito hay que ver qué señales participan en dicho módulo, además de las señales de entrada y de control propias de la función a realizar, y cómo lo hacen con el fin de tenerlas en cuenta para que la operación resultante de la ampliación sea la correcta.

Un caso sencillo de diseño modular es el del sumador completo en el que tenemos en cuenta, a la hora de realizar su diseño, el arrastre de entrada procedente de la suma del bit anterior (C_i en la tabla de verdad del *apartado (b) la figura 5.6 del texto*). Así, una vez que hemos obtenido el circuito del sumador completo para dos bits (circuito del *apartado (c) de la figura 5.6*) podemos ampliarlo a más bits sin más que ir uniendo varios módulos en serie, tantos como bits tienen las palabras que queremos sumar, de forma que, el acarreo de salida del sumador del bit menos significativo se conecta a la entrada de acarreo del bit inmediato superior y así sucesivamente, como se muestra en la *figura 5.7*.

Una vez que se conectan los módulos para expandir la función a más bits es importante

que las entradas de conexión de los bloques correspondientes al ***bit menos significativo*** se pongan a los valores adecuados con el fin de que no afecten a la operación a realizar. Así, si nos centramos en la *figura 5.7*, todos los bloques tienen su entrada de acarreo conectada a la salida de acarreo del bloque anterior, ***menos la entrada de acarreo del primer bloque***, el del bit menos significativo, que deberemos poner a "0" para que no se le sume a este bit un acarreo.

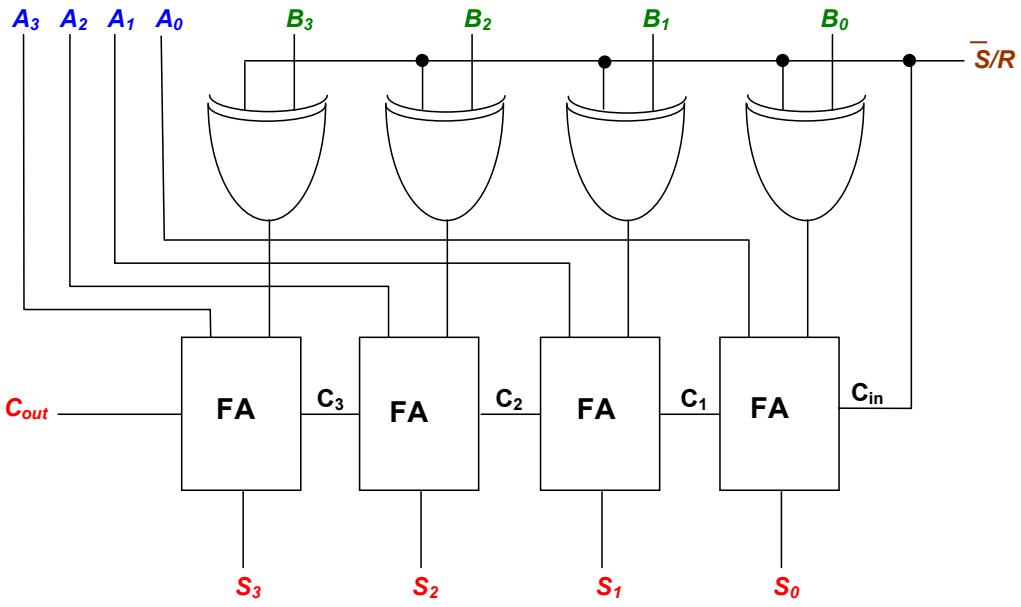
Esto que se ve bastante claro en el circuito sumador por ser el acarreo de la suma de un bit, también se hace con el resto de los circuitos en los que, a veces, no parece tan evidente y hay razonar acerca del valor adecuado (0 ó 1) para que no modifique la función a realizar por este primer bloque.

P.2.11: ¿Cuál es el circuito completo de la fig. 5.3.4, pag. 158, del ejercicio E.5.3 del texto de problemas?.

R.2.11: El principio de funcionamiento de este circuito es que la resta la podemos reducir a una suma si cambiamos de signo al sustraendo, es decir A-B=A+ (- B). Por tanto, para realizar la resta debemos complementar cada uno de los bits de B y sumarle 1 al bit menos significativo, para representar B en C-2 y así poder sumar directamente cada uno de los bits usando tantos sumadores completos de un bit como bits tienen las palabras sobre las que vamos a operar.

Para complementar cada uno de los bits de la palabra B usamos la función XOR entre la señal de control $\overline{S}/R$, que para la resta toma el valor "1", y cada uno de los bits de *B*, obtenida a partir de la tabla de verdad de la *figura 5.3.3*. Además, hay que sumarle "1" al bit menos significativo y esto lo hacemos a través del acarreo de entrada del sumador correspondiente al bit menos significativo, al que le introducimos también la misma señal $\overline{S}/R$ que, como hemos visto anteriormente, toma el valor "1" para la resta.

Así el circuito completo es:



Finalmente, comprobemos el funcionamiento del circuito.

1. Supongamos que queremos hacer una ***suma***. En este caso, la señal de control será $\overline{S} / R = 0$, lo que supone que en las salidas de las puertas XOR se obtienen las expresiones $0 \oplus B_i = B_i$ y que $C_{in}=0$. Por tanto, realiza la función *A+B.*

2. En el caso de que queramos hacer una ***resta*** la señal de control será $\overline{S} / R = 1$, lo que supone que en las salidas de las puertas XOR se obtienen las expresiones $1 \oplus B_i = \overline{B}_i$ (si $B=0$, $1 \oplus 0 = 1$ y si $B=1$, $1 \oplus 1 = 0$) y que $C_{in}=1$. Por tanto, se complementan todos los bits de la palabra *B* y se la suma "1" a través del acarreo de entrada C_{in}.. Es decir, se realiza la operación *A-B* mediante la suma en C-2 de *A* y *-B*.

COMPARADORES

P.2.12: ¿Por qué en el texto se usa para diseñar el comparador de palabras de 4 bits un razonamiento lógico en vez de usar la tabla de verdad?

R.2.12: Por dos razones. La primera es porque la tabla de verdad es muy grande. Recuerde que la tabla de verdad es la descripción en extenso de la función, lo que supone que tienen que aparecer todas y cada una de las posibles combinaciones de las configuraciones de las variables de entrada. Por tanto, en este caso hay que comparar cada una de las palabras A con todas y cada una de las palabras B. Como tenemos palabras de 4 bits, el número de palabras posibles es de 24=16. Esto supone que la tabla de verdad tendrá 16x16 = 256 filas.

La segunda es porque, si hemos conseguido hacer la tabla de verdad, el siguiente paso es la minimización de la función que, como podemos comprobar está formada por términos mínimos de 8 bits (2^8=256 términos mínimos), luego no podemos aplicar Karnaugh directamente. Hay que minimizarla aplicando directamente los teoremas y postulados del Algebra de Boole y operando, lo cual resulta bastante laborioso y, además la probabilidad de cometer algún error es alta.

P.2.13: ¿Podrían explicar de forma más detallada cómo se implementa el razonamiento seguido para obtener las expresiones lógicas del comparador de 2 palabras de n bits?

R.2.13: Para hacerlo más sencillo y sin pérdida de generalidad, vamos a explicar la comparación entre 2 palabras de 3 bits. Al final generalizaremos las expresiones obtenidas y obtendremos las expresiones generales para n bits.

Partimos de la descripción en lenguaje natural de las condiciones que deben cumplirse para que una palabra de 3 bits ($A=A_2A_1A_0$) sea mayor que otra ($B=B_2B_1B_0$) teniendo en cuenta que A_2 y B_2 son los bits mas significativos y A_0 y B_0 los menos significativos: Para ello, vamos a analizar y a especificar todas las posibilidades que se nos pueden presentar.

A continuación transcribimos estas expresiones descritas en lenguajes natural a funciones lógicas. Para ello recordemos que las variables o condiciones unidas mediante la conectiva "***y***" pasan a ser variables o funciones unidas por la función lógica ***AND*** (una cosa ***y*** la otra es la ***intersección*** de ambas) y las unidas por "***o***" pasan a representarse mediante la función lógica ***OR*** (una cosa ***o*** la otra es la ***unión***).

Así, decimos que A>B si se verifica que:

1: *A_2 es mayor que B_2 (con independencia de lo que ocurra con el resto de los bits, puestos que son menos significativos).* Por ejemplo, A=**1**00 siempre será mayor que B=**0**11. Observemos que esto es lo mismo que cuando comparamos números decimales. Por ejemplo, sabemos que **2**xx es mayor **1**zz porque 2>1.

   Como las variables A y B sólo pueden tomar el valor 0 ó el 1, para que A sea mayor que B tiene que ser A=1 ***y*** B=0, de los contrario A nunca sería mayor que B.

   Por tanto, esta condición la representamos por:

$$A_2 > B_2 = A_2\ \overline{B}_2$$

También resulta A>B si se verifica que:

2: *A_2 es igual a B_2* ***y*** *A_1 es mayor que B_1* (por ejemplo, A=1**1**0 y B=1**0**1*).*

Esta condición tiene dos partes. *$A_2 = B_2$* ***y*** *$A_1 > B_1$*

$A_2 = B_2$ se cumple cuando ambas toman el valor "1" (A_2=1 ***y*** B_2=1) ***o*** cuando ambas toman el valor "0" (A_2=0 ***y*** B_2=0). Por lo que la representamos mediante $A_2\ B_2 + \overline{A}_2\ \overline{B}_2$.

$A_1 > B_1$ se cumple cuando *A_1*=1 ***y*** *B_1* =0 y la representamos por $A_1\ \overline{B}_1$

Por tanto, esta condición la podemos expresar por:

$$(A_2 = B_2)\ \ \boldsymbol{y}\ (A_1 > B_1) = (A_2\ B_2 + \overline{A}_2\ \overline{B}_2) + A_1\ \overline{B}_1$$

Finalmente, ***también*** se verifica que A> B si:

3: *A_2 es igual a B_2* ***y*** *A_1 es igual a B_1* ***y*** *A_0 es mayor que B_0 (*por ejemplo, A=11**1** y B=11**0**).

Razonando de la misma forma que lo hemos hecho anteriormente podemos represntar esta condición mediante la expresión lógica

$$(A_2 = B_2)\ \boldsymbol{y}\ (A_1 = B_1)\ \boldsymbol{y}\ (A_0 > B_0) = (A_2\ B_2 + \overline{A}_2\ \overline{B}_2)(A_1\ B_1 + \overline{A}_1\ \overline{B}_1)(A_0\ \overline{B}_0)$$

Por tanto podremos decir que la función que expresa que $A > B$ es:

$$A > B = (A_2 > B_2)\ \boldsymbol{o}\ [(A_2 = B_2)\ \boldsymbol{y}\ (A_1 > B_1)]\ \boldsymbol{o}\ [(A_2 = B_2)\ \boldsymbol{y}\ (A_1 = B_1)\ \boldsymbol{y}\ (A_0 > B_0)] =$$
$$= A_2\ \overline{B}_2 + (A_2\ B_2 + \overline{A}_2\ \overline{B}_2)(A_1\ \overline{B}_1) + (A_2\ B_2 + \overline{A}_2\ \overline{B}_2)(A_1\ B_1 + \overline{A}_1\ \overline{B}_1)(A_0\ \overline{B}_0)$$

Si simplificamos esta expresión sustituyendo las condiciones de igualdad por la variable E de forma que: $E_i = (A_i = B_i) = (A_i\ B_i + \overline{A}_i\ \overline{B}_i)$ resulta:

$$A > B = A_2\ \overline{B}_2 + E_2\ A_1\ \overline{B}_1 + E_2\ E_1\ A_0\ \overline{B}_0$$

Si generalizamos esta expresión a palabras de n bits, obtenemos la de la comparación de 2 palabras de n bits. Así,

$$A > B = A_{n-1}\ \overline{B}_{n-1} + E_{n-1}\ A_{n-2}\ \overline{B}_{n-2} + E_{n-1}\ E_{n-2}\ A_{n-3}\ \overline{B}_{n-3} + \cdots + E_{n-1}\ E_{n-2}\ E_{n-3} \cdots E_2\ E_1\ A_0\ \overline{B}_0 \cdots$$

Si particularizamos para palabras de 4 bits podemos ver que coincide con la que se obtiene en la salida C del circuito de la *figura 5.17 del texto*.

✦✦✦

P.2.14: He diseñado un comparador de dos bits (dos palabras de un bit cada una) y quiero diseñar un comparador de dos palabras de 2 bits usando el diseño anterior como bloque funcional básico. ¿Cómo puedo hacerlo?

R.2.14: Este problema está resuelto de forma recursiva en el *ejercicio E.5.4 del Libro de Problemas*. Sin embargo, si no queremos hacer un diseño recursivo, el diseño resulta más sencillo.

Así, partimos de 2 bloques como el de la *figura 5.16* que vamos a usar como bloque funcional básico de diseño. Al primer bloque lo vamos a llamar M_0 . Sus entradas son las correspondientes a los dos bits menos significativos, A_0 y B_0 ,y sus salidas son los resultados de la comparación de dichos bits de entrada, D_0 (A_0>B_0), E_0 (A_0=B_0) y C_0 (A_0<B_0). Análogamente, las entradas y salidas del bloque M_1 correspondiente a los bits más significativos son A_1, B_1 , D_1 (A_1>B_1), E_1 (A_1=B_1) y C_1 (A_1<B_1). Por tanto, las entradas del circuito resultante de la unión de ambos serán A_0, B_0, A_1, B_1 y las salidas las llamaremos D (A>B), E (A=B) y C (A<B).

Veamos la forma de realizar el diseño mediante el razonamiento lógico:

Uno de los puntos más importante a tener en cuenta es el significado de las variables de salida de cada uno de los bloques. Este es el motivo de ponerlas entre paréntesis. Así, para el bloque correspondiente a la comparación de los bits más significativos tenemos:

$$D_1\,(A_1>B_1),\ E_1\,(A_1=B_1)\ \text{y}\ C_1\,(A_1<B_1).$$

Y para el bloque correspondiente a la comparación de los bits menos significativos:

$$D_0\,(A_0>B_0),\ E_0\,(A_0=B_0)\ \text{y}\ C_0\,(A_0<B_0).$$

El otro punto a tener en cuenta es que, para cada bloque, estas salidas son *mutuamente exclusivas*. Es decir, si una de las salidas de un bloque toma el valor "1" las otras dos tienen que valer "0", no pueden tomar el valor "1". Esto supone que podemos trabajar con la igualdad y una de las desigualdades, pues la otra desigualdad es una consecuencia inmediata.

El razonamiento lógico a seguir es análogo al que se realiza en el texto para comparar palabras de 4 bits, pero adaptándolo a las salidas de los dos bloques. Así, *la palabra A es mayor que la palabra B* ***si*** *el bit más significativo de la palabra A (A_1) es mayor que el bit más significativo de la palabra B (B_1)* ***o bien****, los dos bits más significativos de ambas palabras son iguales* ***y*** *el bit menos significativo de la palabra A (A_0) es mayor que bit menos significativo de la palabra B (B_0).* Si traducimos esto a las variables lógicas de salida de los bloques resulta:

$$D(A>B)=1 \quad \textbf{si} \quad D_1\,(A_1>B_1)=1 \quad \textbf{ó} \quad E_1\,(A_1=B_1)=1 \quad \textbf{y} \quad D_0\,(A_0>B_0)=1$$

Para pasar de la descripción en *lenguaje natural* a la descripción en *lenguaje formal* tenemos que sustituir las conjunciones del lenguaje natural "**ó**" e "**y**" por las expresiones del lenguaje formal **OR** (unión) y **AND** (intersección) respectivamente, y la palabra "**si**" por el "**=**" .

$$D(A>B) \;=\; D_1(A_1>B_1) \;\textbf{OR}\; E_1\,(A_1=B_1) \;\textbf{AND}\; D_0(A_0>B_0)$$

Por tanto: $\boldsymbol{D = D_1 + E_1 \cdot D_0}$

Actuando de forma análoga para la variable E, obtenemos:

$$E(A=B)=1 \;\textbf{si}\; E_1(A_1=B_1)=1 \;\textbf{y}\; E_0\,(A_0=B_0)=1$$

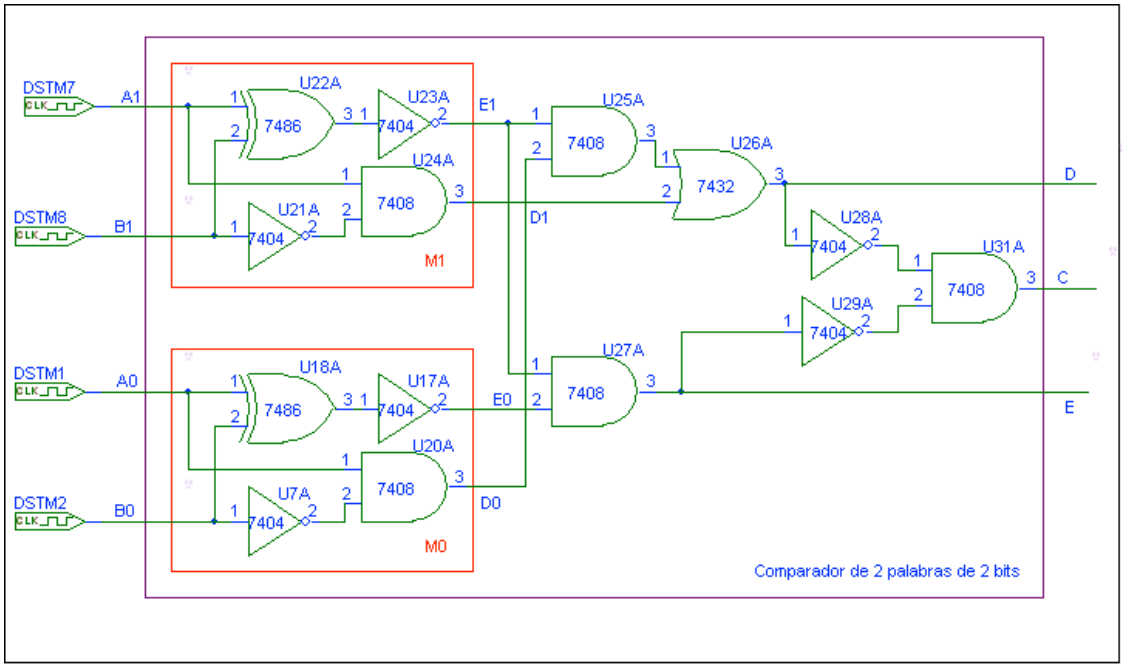
Es decir: $\boldsymbol{E = E_1 \cdot E_0}$

La expresión de C(*A<B*)=*1* la obtenemos de forma inmediata ya que, al ser las variables de salida mutuamente exclusivas, podemos razonar que A es menor que B cuando no es ni mayor ni igual. Por tanto,

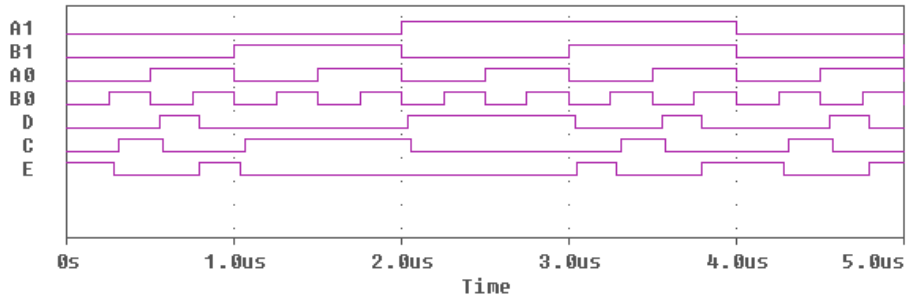
C(A<B)= 1 **si** D(A>B) =0 **y** E(A=B) =0.

O sea: $\boldsymbol{C = \overline{D} \cdot \overline{E}}$

Por tanto, el circuito que implementa al comparador de 2 palabras de 2 bits es:



Y el cronograma resultante de su simulación es:



Como podemos ver en el cronograma, no hay ninguna respuesta en la que dos de las salidas tomen el valor 1 a la vez, con lo que verificamos que estas señales son mutuamente exclusivas. Además, se verifica que la variable de salida *D(A>B)* toma el valor 1 ante las siguientes configuraciones de entrada:

(A=01, B=00), (A=10, B=00), (A=10, B=01), (A=11, B=00), (A=11, B=01) y *(A=11, B=10)*

Como podemos comprobar, estas configuraciones son todas las configuraciones de dos bits en las que se verifica que A>B.

Análogamente, *E(A=B)* toma el valor 1 ante las 4 configuraciones de entrada en las que ambas palabras son iguales, Es decir:

(A=00, B=00), (A=01, B=01), (A=10. B=10) y (A=11, B=11)

Finalmente, C(A<B) toma el valor 1 ante el resto de las configuraciones de entrada que son aquellas en las que, como podemos comprobar, A es menor que B y son:

(A=00, B=01), (A=00, B=10), (A=00, B=11), (A=01, B=10), (A=01, B=11) y (A=10, B=11).

Otra forma de realizar el mismo diseño es mediante la tabla de verdad de los bloques previamente diseñados. Partimos de las mismas señales:

D_1 (A_1>B_1), E_1 (A_1=B_1) y C_1 (A_1<B_1)

D_0 (A_0>B_0), E_0 (A_0=B_0) y C_0 (A_0<B_0).

Al ser las salidas de los bloques mutuamente exclusivas nos encontramos con bastantes términos mínimos que no se van a presentar nunca, por lo que obtenemos una tabla de verdad con muchos *términos mínimos indiferente* que nos van a facilitar la minimización. Así, la tabla de verdad es:

Salidas de los bloques		**Salida del circuito**
D_0 E_0	***D_1 E_1***	***D E C***
0 0	0 0	0 0 **1**
"	0 1	0 0 **1**
"	1 0	**1** 0 0
"	1 1	x x x
0 1	0 0	0 0 **1**
"	0 1	0 **1** 0
"	1 0	**1** 0 0
"	1 1	x x x
1 0	0 0	0 0 **1**
"	0 1	**1** 0 0
"	1 0	**1** 0 0
"	1 1	x x x
1 1	0 0	x x x
"	0 1	x x x
"	1 0	x x x
"	1 1	x x x

El diagrama de Karnaugh para la minimización de la variable de salida *D* es:

D_1E_1 / D_0E_0	00	01	11	10
00			x	1
01			x	1
11	x	x	x	x
10		1	x	1

Así, la expresión minimizada de *D* es: $D = D1 + E1\ D0$

Análogamente para E obtenemos:

D_1E_1 / D_0E_0	00	01	11	10
00			x	
01		1	x	
11	x	x	x	x
10			x	

Por tanto, $E = E_1\ E_0$

Finalmente, para *C obtenemos*:

D_1E_1 / D_0E_0	00	01	11	10
00	1	1	x	
01	1		x	
11	X	x	x	x
10	1		x	

$$C = \overline{D}_1\ \overline{E}_1 + \overline{D}_0\ \overline{E}_0\ E_1 = \overline{D}\ \overline{E}$$

En resumen las expresiones de salida del comparador de 2 palabras de 2 bits son:

$$D = D1 + E1\ D0$$
$$E = E_1\ E_0$$
$$C = \overline{D}\ \overline{E}$$

Que, como podemos comprobar, coinciden con las calculadas previamente.

✦✦✦

P.2.15: Estudiando el comparador de la figura 5.17 me surge la duda acerca de las variables "E". En cada puerta AND entra un bit de la palabra A, el bit correspondiente de la palabra B y una E. ¿Qué señales son las "Ei" y de dónde proceden?

R.2.15: Usando la misma nomenclatura del comparador de un bit de la *figura 5.16*, la "*E*" significa que ambos bits son iguales. Es decir, *E=1* si los correspondientes bits de las dos palabras

toman el mismo valor y *E=0* si toman diferentes valores. Por tanto, *E* procede de la salida de una puerta XNOR cuyas entradas son los bits a comparar.

El circuito de la *fig.5.17* es sólo el circuito de extensión del circuito comparador a palabras de 4 bits, como se indica en su pie de figura. Por lo tanto, a este circuito hay que anteponerle los detectores de igualdad entre cada pareja de bits de las palabras A y B, así como los inversores de la palabra B. Es decir, a este circuito hay que anteponerle 4 puertas XNOR y cuatro inversores. A cada una de las puertas XNOR le entra un bit de la palabra A y el equivalente de la B dando lugar a las correspondientes E_i como consecuencia de que A_i es igual a B_i. Así, a la primera XNOR le debe entrar A_3 y B_3 y a su salida generará E_3, que uniremos a la entrada E_3 del circuito de la *figura 5.17*. De igual forma ocurre con el resto de los bits de las palabras *A* y *B*, generando así las entradas E_2, E_1 y E_0.

De esta forma tenemos un comparador de 2 palabras de 4 bits y, como el diseño es modular, lo podemos ampliar a palabras de 8 bits con sólo unir en serie dos circuitos iguales a este, de forma que las salidas C y E del comparador correspondiente a las palabras de los 4 bits menos significativos las conectamos a las entradas C' y E' del comparador correspondiente a las palabras de los 4 bits más significativos.

✦✦✦

P.2.16: No entiendo cómo funciona el circuito comparador de la figura 5.17.

R.2.16: Si observamos la parte superior correspondiente a las 4 primeras puertas AND del circuito de la *figura 5.17***,** vemos que el número de entradas de las puertas AND se va incrementando de forma que, en cada una van entrando todas las "E" de los bits superiores. Así, en la AND superior sólo entran los bits más significativos, A_3 y $\overline{B}_3$. En la siguiente puerta entra A_2, $\overline{B}_2$ y E_3. En la siguiente A_1, $\overline{B}_1$ E_3 y E_2 y en la cuarta entran los bits menos significativos y las señales correspondientes a la igualdad del resto de los bits, es decir, A_0, $\overline{B}_0$, E_3, E_2. y E_1. Esto supone que en la salida de cada una de estas puertas AND obtenemos un sumando de la expresión de la función que cumple con la condición de que A es mayor que B (*expresión [5.15], pag 286 del texto*). El resultado de estas puertas AND se suma en la puerta OR de salida obteniéndose a partir de estas cuatro puertas AND la expresión:

$$A_3\,\overline{B}_3 + A_2\,\overline{B}_2\,E_3 + A_1\,\overline{B}_1\,E_3\,E_2 + A_0\,\overline{B}_0\,E_3\,E_2\,E_1$$

En la parte inferior de la figura vemos que hay otras dos puertas AND de 5 entradas.

La salida de primera de estas puertas AND también se suma con las salidas de las puertas anteriores y sus entradas son todas las E_i *y* una entrada adicional correspondiente a $C'\left(A_n' > B_n'\right)$. Por tanto, su salida es: $E_3\,E_2\,E_1\,E_0\,C'\left(A_n' > B_n'\right)$ cuyo significado es que todos los bits de las dos palabras que compara este comparador coinciden pero, además, el resultado de la comparación de los bits de la palabra anterior ha sido $A_n>B_n$. Así, la salida del circuito, $C\left(A_n > B_n\right)$ corresponde a que la palabra *A* que entra en este bloque es mayor que la palabra *B* o que, siendo ambas palabras iguales, en

la comparación de las palabras del bloque anterior el resultado fue que *A* era mayor que *B*. Por tanto:

$$C\left(A_n > B_n\right) = A_3\,\overline{B}_3 + A_2\,\overline{B}_2\,E_3 + A_1\,\overline{B}_1\,E_3\,E_2 + A_0\,\overline{B}_0\,E_3\,E_2\,E_1 + C'E_3\,E_2\,E_1E_0$$

Veamos el significado físico.

Esta expresión, *C*, consta de cinco sumandos, de forma que basta con que uno de ellos tome el valor "1" para que la función sea "1". Así, el primer sumando $A_3\,\overline{B}_3$ es igual a "1" cuando $A_3 = 1\ y\ B_3 = 0$ ($A_3\,\overline{B}_3 = 1$), o sea, cuando el bit más significativo de la palabra *A* de 4 bits es mayor que el más significativo de la palabra *B*, por lo tanto, en este caso ya podemos afirmar que la palabra *A* es mayor que la *B*. El segundo sumando es $A_2\,\overline{B}_2\,E_3$ que toma el valor de 1 cuando $A_2 = 1\,,\,B_2 = 0$ y $E_3 = 1$, o sea, cuando los bits más significativos de ambas palabras son iguales ($E_3 = 1$) y el segundo bit de la palabra *A,* A_2*,* es mayor que el correspondiente de la palabra *B,* B_2*,* por lo tanto, ahora también podemos afirmar que la palabra *A* es mayor que la *B*. De igual forma operan los otros dos sumandos. Es decir, con esto queda implementado el razonamiento lógico que se presenta en el texto (*página 286*). Finalmente, el último sumando toma el valor uno cuando todos los bits de la palabra *A* son iguales que los de la palabra *B*, o sea, que la palabra *A* y la *B* coinciden, pero el bit de entrada *C'* que procede de la comparación del bloque anterior, en el caso de diseño modular y expansión de la comparación a más bits, toma el valor "1".

Por último, existe una última puerta AND en la que le entran todas las *E*, incluida la del resultado de la comparación del supuesto bloque anterior, $E'\left(A_n' = B_n'\right)$, por lo que en su salida tenemos la expresión correspondiente a $E\left(A_n = B_n\right)$. Es decir:

$$E\left(A_n = B_n\right) = E'E_3\,E_2\,E_1E_0$$

Así, a expresión *E(A=B)* de la salida del comparador se hace "1" cuando los bits de las palabras A y B son iguales dos a dos y, además, en el caso de usar el bloque para la expansión de dicha función, el resultado de la comparación de los bits menos significativos también es "1", es decir, los bits menos significativos también coinciden dos a dos.

En resumen, este es un módulo comparador de palabras de 4 bits que tiene la posibilidad de expandirse para comparar palabras de más bits sin más que conectar varios módulos como este en serie de forma que las salidas D y E de un módulo se conectan a las entradas D' y E' del siguiente.

En el caso de que usemos este módulo como bloque único porque las palabras a comparar sean sólo de 4 bits o si lo usamos para expandir la comparación a palabras de más de 4 bits y este bloque es el correspondiente a los 4 bits menos significativos, la entrada *C'* la debemos poner a "0" para hacer que el sumando $C'E_3E_2E_1E_0$ tome el valor "0" y no afecte al resultado de la comparación. Obsérvese que si *C'=1* y *A=B*, la salida del

comparador es *C=1*, que significa que *A>B*, lo cual es falso. Así mismo, la entrada *E'* la debemos poner a "1".

✦✦✦

P.2.17: ¿Qué significan las variables de entrada que aparecen en la parte inferior del circuito comparador de la figura 5.17? La que están representadas como: C' (A'n > B'n) y E' (A'n=B'n)

R.2.17: De acuerdo con la explicación dada en la pregunta anterior estas señales corresponden a las entradas que hacen que el circuito comparador de palabras de 4 bits se pueda expandir a palabras de más bits (8, 12, 16, …). Es decir, son las líneas con las que vamos a enlazar en serie los distintos bloques en el proceso de expansión.

✦✦✦

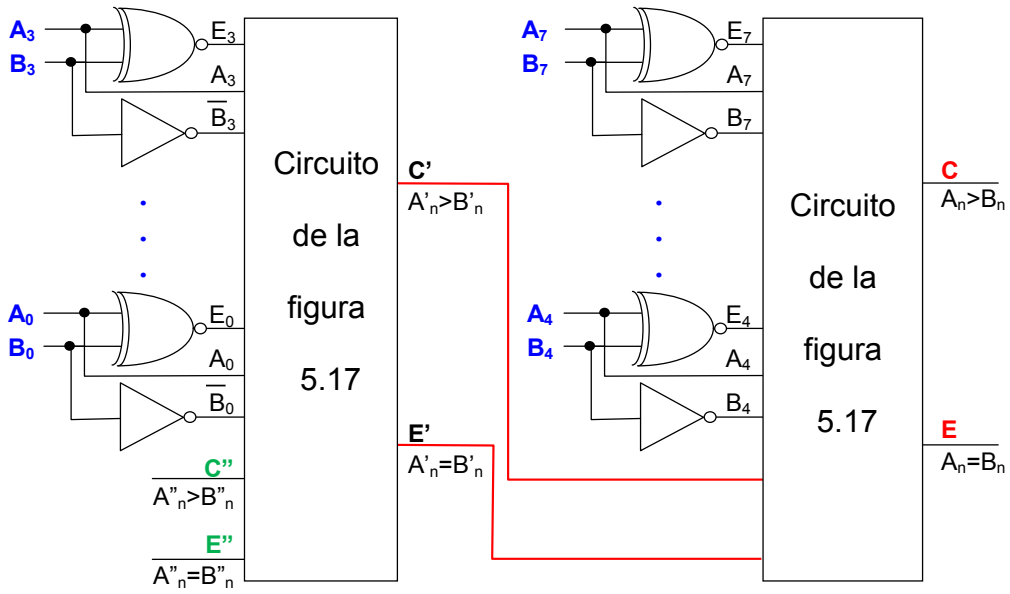
P.2.18: ¿Cómo sería el circuito completo de un comparador de 2 palabras de 8 bits usando el circuito de expansión de la figura 5.17?

R.2.18: Generalmente, en Electrónica Digital se intenta que los diseño de los bloque funcionales sean modulares, lo que supone que hay que ver la forma de que determinadas salidas de un módulo se puedan unir a determinadas entradas de otro módulo igual, con el fin de que se pueda ampliar la función a palabras de más bits y sin necesidad de hacer un diseño propósito especial cada vez que se cambia el número de bits. Así, por ejemplo, con el modulo del sumador completo hemos construido un sumador de n bits (*ver figura 5.7 pag 273*) sin más que conectar el acarreo de salida de un módulo con la entrada del acarreo del módulo del bit inmediato superior. Pues bien, en este módulo del comparador la conexión entre dos módulos se hace a través de las entradas $C'(A'_n>B'_n)$ y $E'(A'_n=B'_n)$.

Si queremos pasar de un comparador de dos palabras de 4 bits a un comparador de dos palabras de 8 bits usamos dos módulos de este tipo, de forma que el resultado de la comparación de los cuatro bits menos significativos ($A'_n>B'_n$ y $A'_n=B'_n$) se introduce en las entradas *C'* y *E'* del módulo correspondiente a los 4 bits más significativos. Así, si los cuatro bits son todos iguales, pero el resultado de los cuatro bits menos significativos es que *A>B* entonces el comparador de 8 bits debe producir un "1" en la salida $A_n>B_n$, por eso se suma en la puerta OR de salida donde se suman de forma lógica todas las expresiones que hacen que la palabra *A* sea mayor que la *B*.

Análogamente, si el resultado de la comparación de la palabra de 4 bits menos significativos es que $A'_n = B'_n$ y todos los bits de la palabra de los 4 bits más significativos también son iguales, entonces ambas palabras de 8 bits son iguales y aparece un "1" en la salida $E\ (A_n=B_n)$.

Por tanto, el circuito para palabras de 8 bits es:



Lógicamente, las entradas de *C"* y *E"* de la palabra menos significativa hay que ponerlas a valores que no modifiquen la función a realizar, es decir, que no modifiquen *C'* y *E'* (salidas del primer módulo en la figura anterior). Por tanto, habrá que poner *C"* a "0", ya que así la salida de la segunda puerta AND empezando por abajo de la *figura 5.17*, *pag. 285* es: *E3 E2 E1 E0 C"=0* y no modifica el resultado de la puerta OR que genera las salida *C'* del primer módulo. Por el contrario, hay que poner *E"* a "1" para que tampoco afecte al resultado de la última puerta AND que genera la salida *E'*.

✦✦✦

P.2.19: ¿Qué diferencia hay entre la comparación lógica y la aritmética?

R.2.19: La diferencia está en que la comparación lógica tiene lugar entre dos palabras cuya representación es en binario puro (sin signo). Por tanto, la comparación se realiza bit a bit. Así, por ejemplo, si comparamos las palabras *A= 101* y *B= 011*, el resultado es *A>B*. En efecto, el equivalente en decimal de estas dos palabras en binario puro es: A=9 y B =3.

Sin embargo, la comparación aritmética tiene lugar entre números positivos y negativos cuyo valor equivalente en decimal depende de que el tipo de representación binaria sea en S-M, C-1 ó C-2. Como en estas representaciones el bit más significativo toma el valor "1" para los números negativos y "0" para los positivos, la comparación no se puede hacer bit a bit, ya que nos daría como resultado que los números negativos son mayores que los positivos, cosa que no es cierta. Por ejemplo, si suponemos que las palabras A y B anteriores corresponden a números representados en S-M, el resultado correcto es que *B>A* ya que, *A=101* representa en S-M al número decimal -1 y *B=011* representa al +3. Por tanto, el algoritmo de comparación es distinto y depende del tipo de representación que estemos considerando.

✦✦✦

P.2.20: No entiendo la tabla de la figura 5.19.

R.2.20: La figura 5.19 es la tabla de verdad que da el fabricante del circuito SN74866.

El significado de las distintas columnas de la tabla de verdad es el siguiente:

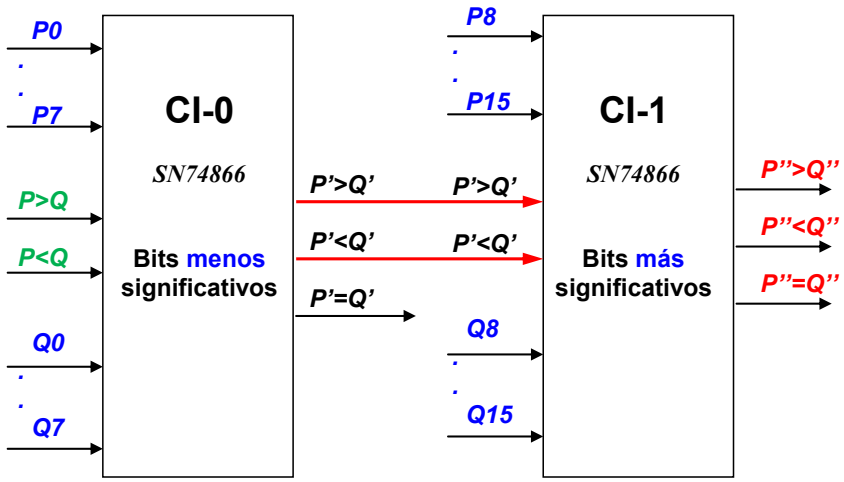
Primera columna, encabezada con "*Comparación*". Especifica el tipo de comparación que realiza y que puede ser lógica (entre 2 palabras binarias de 8 bits) o aritmética (2 palabras de 8 bits en complemento a 2).

Segunda columna, encabezada con "*L/noA"*. Esta entrada es la señal que controla el tipo de operación que realiza, de forma que, si la ponemos a "1" (*H*) el circuito realiza la comparación lógica entre las dos palabras de entrada, P y Q. Si por el contrario la ponemos a "0" (*L*) realiza la comparación aritmética de dichas palabras. Si miramos el símbolo lógico de la *figura 5.18* vemos que es la primera línea de entrada de la izquierda (terminal 2 del dispositivo), que está también nombrada como "*L/noA"* y que se desdobla en dos, una con el epígrafe *M(logic)* y la otra *M(arith, 2s comp).* Esta última está negada (esto se pone de manifiesto con el triángulo que aparece en esta entrada del símbolo lógico[1]). Es decir, cada una está nombrada con la función que realiza.

Tercera columna, encabezada con "*Datos de entrada. P0-P7, Q0-Q7"*. En esta columna se especifican todas las posibilidades que existen en la comparación de las dos palabras de entrada, P y *Q*. En la *figura 5.18* se corresponden con las 8 líneas de entrada, *P0*(terminal 18) a *P7*(terminal 25), una línea para cada bit de la palabra *P* y *Q0*(terminal 12) a *Q7*(terminal 5) una para cada bit de la palabra *Q*. A continuación justificaremos la repetición de *P*=*Q*.

Cuarta columna, encabezada con "*Entradas. P>Q, P<Q"*. En esta columna se especifican los posibles valores de las entradas procedentes de la comparación realizada por un comparador previo al que estamos considerando. Estas entradas nos permiten ampliar el circuito comparador a palabras de más bits, por ejemplo, podemos comparar palabras de 16 bits (*P0-P15* y *Q0-Q15*, siendo *P15* y *Q15* los bits más significativos), sin más que conectar los dos circuitos en cascada, o sea, conectando las salidas del primer comparador a estas entradas del segundo, como se muestra en la siguiente figura.

[1] Ver Apéndice B: Símbolos Lógicos de texto base, *pags. 725-734*



En este caso, los bits de las palabras del primer comparador serán menos significativos que los de las palabras del segundo comparador.

Analicemos los valores que van tomando las filas de estas columnas. En las dos primeras filas aparece "*x*". Esto significa que es indiferente el valor que toma. O sea, que P'>Q' puede tomar el valor "1" (quiere decir que *P'* es mayor que *Q'*) ó "0" (quiere decir que *P'* no es mayor que *Q'*) porque no modifica el resultado de la comparación entre *P8-P15* y *Q9-Q15*. Observe que en estos casos, si la palabra formada por *P8-P15* es mayor que *Q8-Q15* podemos afirmar que *P>Q* con independencia de si *P0-P7* es mayor o menor que *Q0-Q7*. El razonamiento es análogo para *Q>P*.

Siguiendo con los valores de esta columna, vemos que aparece repetido 4 veces *P=Q*, pero en las siguientes columnas, las encabezadas con *P>Q* y *P<Q* como *entradas*, aparecen las cuatro posibilidades que se pueden presentar en la comparación de la palabra previa (los bits menos significativos). Es decir, que no sea ni mayor ni menor, o sea, que sea igual, que sea *P'<Q'*, que sea *P'>Q'* o que sea *P'>Q'* y *P'<Q'* a la vez. Esta última situación, es incoherente ya que una cosa no puede ser, a la vez, mayor y menor que otra. Pero como en el circuito existen las entradas *P>Q* y *P<Q* que se pueden poner fijas a valores "0" ó "1", está claro que se pueden poner ambas señales a 1, es una posibilidad física que tiene el circuito y, por tanto la tiene que contemplar, aunque no tenga sentido al interpretarla, dando como resultado la misma salida *P'>Q'* y *P'<Q'* para el primer comparador y *P''>Q''* y *P''<Q''* para el segundo. Además, estas señales que no tienen sentido físico, pero que se pueden presentar y se presentan, se pueden usar, por ejemplo, como señales de detección de error o como señales de control de otro circuito que forme parte de un sistema más complejo y del que también forme parte el comparador.

Quinta columna, encabezada con "*Salidas. P>Q, P<Q, P=Q"*. Son las tres salidas del circuito en las que aparecerá un "0" ó un "1" dependiendo del resultado de la comparación.

En el símbolo lógico del circuito aparecen otras señales de control que actúan facilitando los biestables D de entrada y de salida de datos, pero que no vamos a explicar porque queda fuera del alcance de este tema. Lo estudiaremos más adelante, en la lógica secuencial.

Para el que tenga interés en saber más acerca de este circuito integrado, adjuntamos como anexo al final de este documento las hojas de características del dispositivo fabricado por Texas Instruments. Es interesante ver el circuito interno que proporciona dicho fabricante.

Características del comparador integrado SN74AS866A de Texas Instruments

DETECTOR DE PARIDAD

P.2.21: ¿Qué función realiza un detector/generador de paridad?

R.2.21: Un detector/generador de paridad/imparidad es un circuito que presenta en su salida un "1" cuando la suma de los "1" de los bits de la palabra de entrada es par/impar. Es decir, si tenemos palabras de 2 bits, las palabras que tienen un número de "1" par son: 00 y 11. El resto, 01 y 10, tiene un número de "1" impar. Por tanto, el circuito detector de paridad debe producir un "1" ante las entradas 00 y 11 y producir un "0" ante las entradas 01 y 10. Recordemos que la función lógica ***anticoincidencia*** la implementamos con una puerta XNOR y que, inversamente, la función ***coincidencia*** la implementamos con la puerta XOR. Por tanto, el circuito que detecta la paridad en una palabra de 2 bits es una puerta XNOR de 2 entradas, una para cada bit, y el que detecta la imparidad es la puerta XOR.

Comprobémoslo a través de las Tablas de verdad:

Entradas		Salidas	
A	**B**	**F_P (par)**	**F_I (impar)**
0	0	1	0
0	1	0	1
1	0	0	1
1	1	1	0

A partir de la Tabla de verdad obtenemos:

$$F_P = \overline{A}\,\overline{B} + A\,B = \overline{A \oplus B}$$
$$F_I = \overline{A}\,B + A\,\overline{B} = A \oplus B$$

P.2.22: ¿Cómo funciona el circuito detector de paridad de la figura 5.21, página 290 del texto?

R.2.22: Para ver el funcionamiento del circuito detector de paridad vamos a analizar los distintos valores y los significados de las señales en los distintos puntos del circuito.

Está claro que la función XOR es la función *anticoincidancia*. Por lo tanto, a su salida presenta un 1 cuando *A* es distinto de B, es decir, presenta a la salida un "1" cuando *A=1* y *B=0* y cuando *A=0* y *B=1*. Dicho de otra forma, cuando a su salida presenta un "*1*" es porque el número de unos en sus entradas es impar. Luego podemos decir que ***la puerta XOR detecta la imparidad***. Por el contrario, presenta "*0*" cuando ambas entradas coinciden. Es decir, cuando *A=B=0* y cuando *A=B=1*

Si llamamos *X* a la salida de (*A XOR B*) e *Y* a la salida de (*C XOR D*), tendremos:

X= A XOR B = 1 significa que el número de unos es IMPAR

X= A XOR B = 0 significa que el número de unos es PAR

Lo mismo ocurre con *C* y *D*:

Y= C XOR D =1 significa que el número de unos es IMPAR

Y= C XOR D =0 significa que el número de unos es PAR

Veamos lo que ocurre a la salida de la XOR marcada como Z.

Hay que tener en cuenta que, si unimos una palabra cuyo número de unos es PAR (o IMPAR) con otra cuyo número de unos también es PAR (o IMPAR) el resultado es una palabra cuyo número de unos PAR. Mientras que, si unimos una palabra cuyo número de unos es PAR (o IMPAR) con otra cuyo número de unos es IMPAR (o PAR) el resultado es una palabra cuyo número de unos es IMPAR. Es decir, podemos construir la tabla siguiente:

Palabra-1	Palabra-2	Palabra-resultante
PAR	PAR	PAR
IMPAR	PAR	IMPAR
PAR	IMPAR	IMPAR
IMPAR	IMPAR	PAR

La señal *Z*, de nuevo, es la función XOR de sus entradas (que hemos llamado *X* e Y). Así, los valores de la columna encabezada con *Z* será la que indicamos en la siguiente tabla y el significado que tiene que tener, en función de la paridad de sus entradas, es el que aparece en la última columna:

X=A XOR B	Y= C XOR D	Z=X XOR Y	significado
0 (par)	0 (par)	0	PAR
0 (par)	1 (impar)	1	IMPAR
1 (impar)	0 (par)	1	IMPAR
1 (impar)	1 (impar)	0	PAR

Luego,

Z=0 significa que la palabra de entrada tiene un número PAR de unos.

Z=1 significa que la palabra de entrada tiene un número IMPAR de unos.

Hasta aquí todo es coherente y lo que tenemos es un detector de paridad de palabras de 4 bits de tal forma que a su salida presenta un 1 cuando el número de unos es IMPAR y presenta un 0 cuando es PAR. Por tanto, si consideramos que el “1” hace referencia a la función que realiza podemos decir que lo que tenemos es un detector de imparidad (pero esto no tiene gran importancia) si los conceptos están claros.

Pasemos ahora a construir un módulo para que podamos usarlo en el diseño modular, de forma que podamos ampliar el número de bits sin necesidad de hacer un nuevo diseño "ad hoc".

Para ello, de nuevo, usamos una función XOR (la que genera a su salida P).

Vamos a hacer el diseño. Partimos de una señal Z que arrastra un significado, que es:

Z=0 significa PAR

Z=1 significa IMPAR

Queremos que el circuito detecte la PARIDAD, luego la salida deberá ser:

P=1 significa PAR

P=0 significa IMPAR

Veamos los valores que debe tomar *P'*, teniendo en cuenta que *P= Z XOR P'*. Es decir, vamos a calcular qué valores debe tomar *P'* para que al realizarse la operación XOR, entre ella (*P'*) y la entrada *Z*, produzca la *P* de salida. Así:

Z	P	P' tal que: P=Z XOR P'
0 (par)	0 (impar)	0
0 (par)	1 (par)	1
1 (impar)	0 (impar)	1
1 (impar)	1 (par)	0

Ahora ya tenemos los valores de *P'*, pero no conocemos su significado aunque a partir de la tabla los podemos determinar. En efecto,

- Si la función XOR de *Z=0(PAR)* y de *P'=0* tiene que dar como resultado *P=0(IMPAR),* deberemos interpretar que ***P'=0*** significa ***IMPAR***, ya que para que la salida sea IMPAR las entradas tiene que ser (PAR, IMPAR).
- Si la función XOR de *Z=0(PAR)* y de *P'=1* tiene que dar como resultado *P=1(PAR),* deberemos interpretar ***P'=1*** como ***PAR***.
- Si la función XOR de *Z=1(IMPAR)* y de *P'=1* tiene que dar como resultado *P=0(IMPAR),* deberemos interpretar ***P'=1*** como ***PAR***.
- Si la función XOR de *Z=1 (IMPAR)* y de *P'=0* tiene que dar como resultado *P=1(PAR)*, deberemos interpretar ***P'=0*** *como* ***IMPAR***.

Luego:

P'=1 (PAR)

P'=0 (IMPAR)

Por último, si comparamos los valores y los significados de *P'* y *P* vemos que ambas tiene el mismo significado para los mismos valores, luego podemos unir dos módulos como el de la figura 5.21 sin necesidad de añadirle ningún circuito adicional.

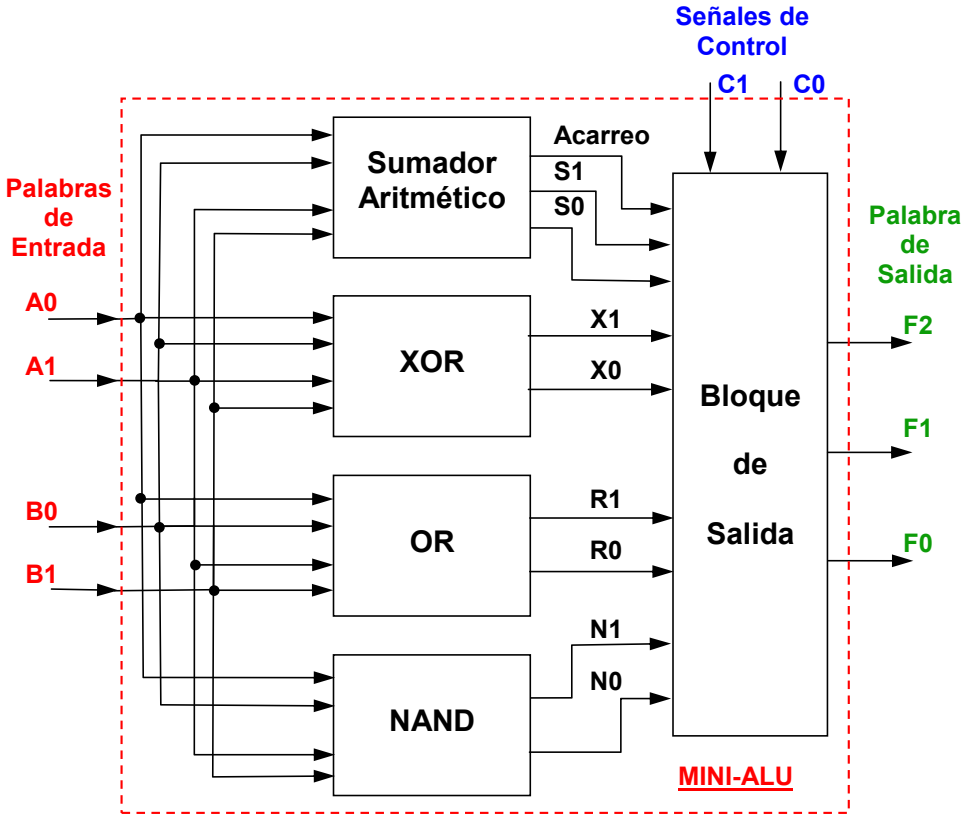
UNIDA ARITMÉTICO-LÓGICA (ALU)

P.2.23: ¿Cómo podemos diseñar una Mini-ALU que realice 4 operaciones sobre palabras de 2 bits?

R.2.23: Vamos a responder a esta cuestión realizando el diseño de una Mini-ALU que realice 4 operaciones sobre palabras de 2 bits. Hemos seleccionado una operación aritmética, la SUMA, y tres operaciones lógicas XOR, NOR y NAND,

La mini-ALU tiene que tener 4 bloques para que cada uno haga una de las 4 operaciones sobre las 2 palabras de entrada de 2 bits. Además, tiene que tener un bloque de salida que presente en su salida el resultado de una de las 4 operaciones en función del valor de la señal de control, S1 y S0.

Así, el diagrama de bloques de la mini-ALU es:



Vamos a diseñar cada uno de los bloques funcionales de este diagrama de bloques.

PRIMER BLOQUE: Para el primer bloque vamos que diseñar un **sumador** paralelo de palabras de 2 bits y de acarreo enlazado.

Los pasos a seguir para diseñar este bloque son:

1º ***Diseño de un Semisumador*:** Debemos empezar por construir la tabla de verdad del semisumador y a partir de esta tabla obtener las expresiones de la suma y del acarreo (*ver apartado 5.2.1 del texto*).

2º ***Diseño de un Sumador Completo*:** De igual forma, partiendo de la tabla de verdad correspondiente, obtenemos las expresiones de la suma de 2 palabras de 1 bit y con acarreo (*ver apartado 5.2.2 del texto*).

3º ***Obtención del Sumador de palabras de 2 bits*****:** Como ya tenemos un sumador de palabras de 1 bit lo que tenemos que hacer es unir en serie dos bloques, como los diseñados en el apartado anterior, en la forma que se indica en la *figura 5.7 del texto*.

De esta forma tenemos un sumador aritmético que realiza la suma de dos palabras (*A* y *B*) de 2 bits cada una, (*A1 A0*) y (*B1 B0*), generando una palabra de salida, S, también de 2 bits (*S1 S0*) más una salida adicional que es el acarreo o arrastre producido por la suma aritmética de los 2 bits más significativos (*A1 PLUS B1*).

SEGUNDO BLOQUE: El diseño de este bloque que implementa la función XOR es muy sencillo porque, así como en las operaciones aritméticas hay que tener en cuenta los sucesivos acarreos que se van produciendo al operar con los distintos bits, en el caso de las operaciones lógicas no hay acarreo, ya que las operaciones se realizan **bit a bit.** Así, las entradas de este bloque son *A1, A0, B1, B0* y la salida es la palabra *X* de dos bits, *X1 X0*, tal que

$$X=A \text{ XOR } B= (A1\ A0) \text{ XOR } (B1\ B0) = (A1 \text{ XOR } B1)\ (A0 \text{ XOR } B0) = (X1\ X0)$$

Nota: Los paréntesis debemos interpretarlos como palabras. Es decir, (*A1 A0*) es la palabra *A* en la que *A1* es el bit más significativo (MSB) de la palabra *A* y *A0* el bit menos significativo (LSB)

Por tanto, este bloque estará formado por dos puertas XOR de 2 entradas de forma que, a la puerta que opera sobre el bit más significativo entran los bits *A1* y *B1* y sale *X1* y a la que opera sobre los bits menos significativos le entra *A0* y *B0* y sale *X0*, dando lugar a la palabra de salida (*X1 X0*).

TERCER y CUARTO BLOQUE: Sus diseños son análogos al anterior y sólo tenemos que cambiar las puertas XOR por NOR en el tercer bloque y por NAND en el cuarto.

QUINTO BLOQUE: Este es el bloque de salida cuyas entradas son los resultados de las operaciones realizadas por los 4 bloques anteriores, su salida es una palabra de 3 bits y la función a realizar es presentar en su salida el resultado de una u otra operación en función del valor que toma la señal de control.

La forma de diseñar este bloque es la de un circuito combinacional en el que partimos de la tabla de verdad de la función a realizar y, a partir de ella, obtener las expresiones de las variables de salida en función de las variables de entrada y de las señales de control.

La tabla de verdad es:

C1	C0	F2	F1	F0
0	0	Acarreo	S1	S0
0	1	--	X1	X0
1	0		R1	R0
1	1		N1	N0

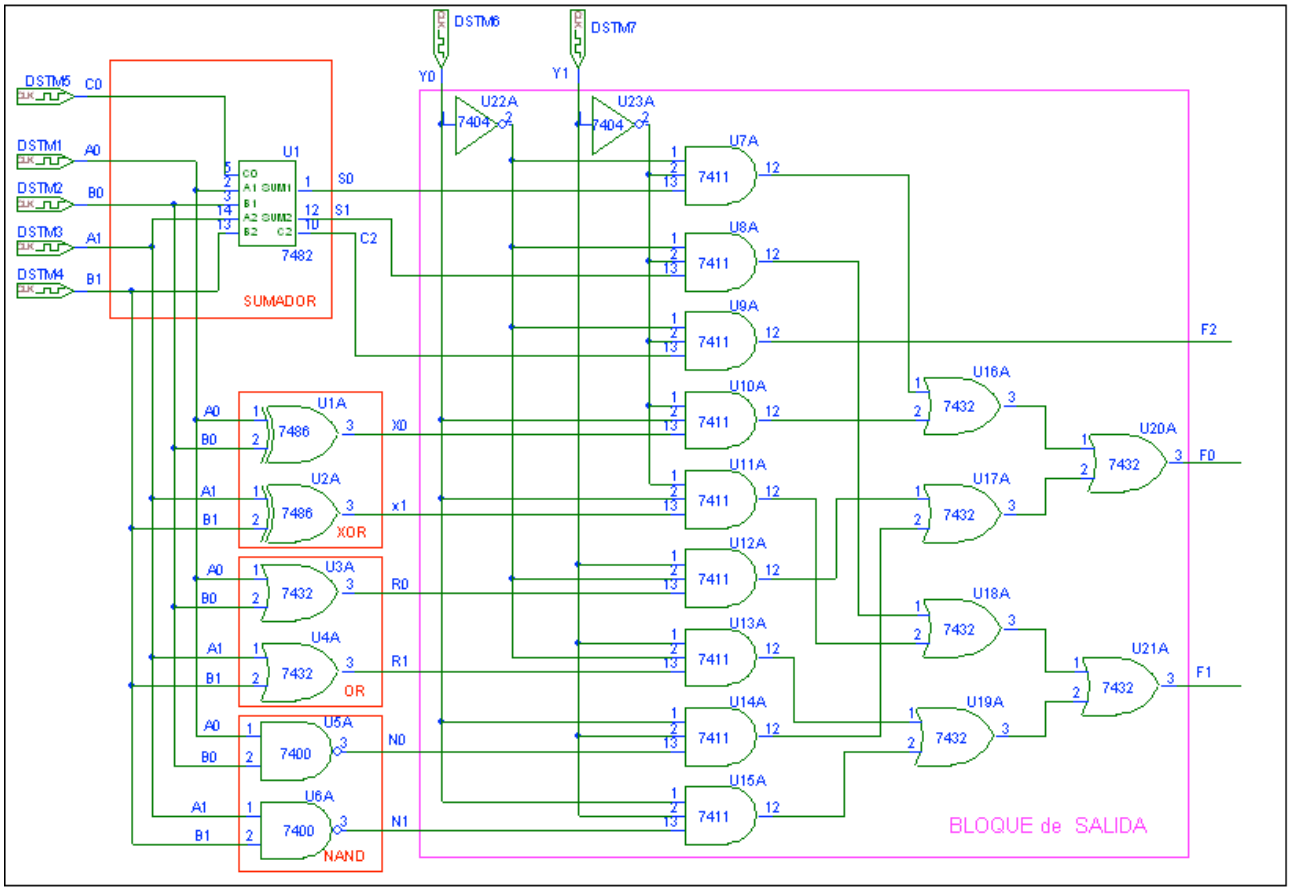
A partir de esta tabla obtenemos las expresiones de *F2*, *F1* y *F0*. Así,

$$F2 = \overline{C1}\,\overline{C0}\,Acarreo$$

$$F1 = \overline{C1}\,\overline{C0}\,S1 + \overline{C1}\,C0\,X1 + C1\,\overline{C0}\,R1 + C1\,C0\,N1$$

$$F0 = \overline{C1}\,\overline{C0}\,S0 + \overline{C1}\,C0\,X0 + C1\,\overline{C0}\,R0 + C1\,C0\,N0$$

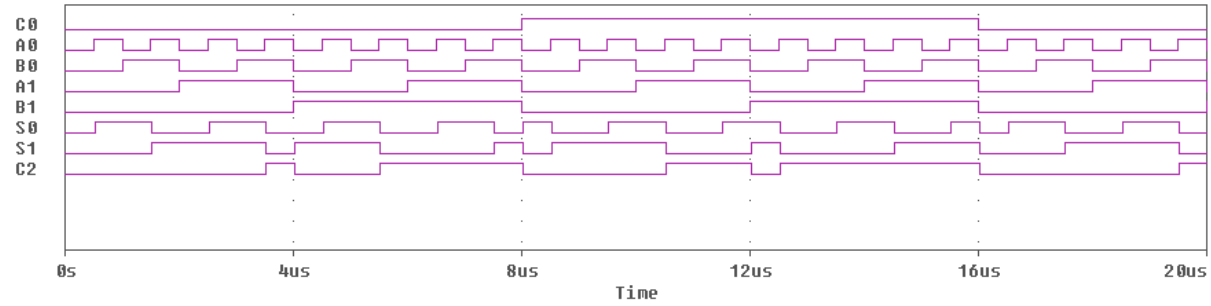
El circuito completo resultante es el siguiente:



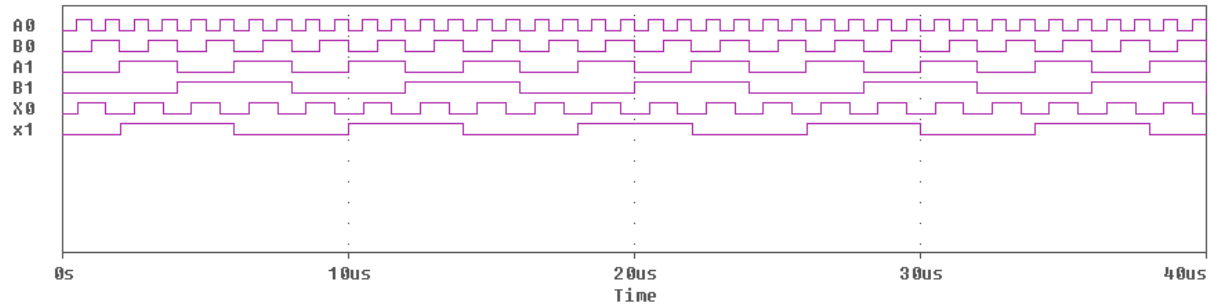
El bloque de salida lo hemos implementado con puertas, pero lo podríamos haber hecho mediante dos MUX de 4 a 1, uno para las señales *S0, X0, R0* y *N0* que dará lugar a la salida *F0* en función de los valores de *Y1* e *Y0* y otro para las señales *S1, X1, R1* y *N1* que dará lugar a la salida *F1*.

Para la simulación del sumador hemos usado un sumador integrado con el fin de simplificar el circuito. En el cronograma siguiente mostramos las señales de entrada y de salida del sumador aritmético en el que:

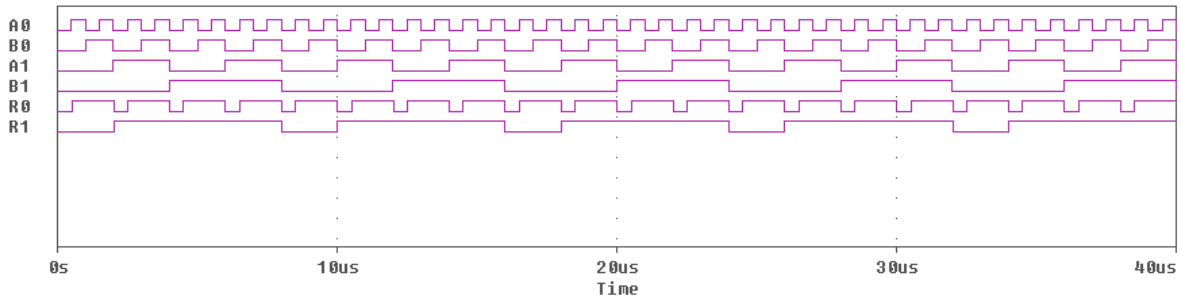
- *C0* es el acarreo de entrada
- *A0* y *B0* son los bits menos significativos de las dos palabras de entrada
- *A1* y *B1* son los bits más significativos
- *S0* es la suma de *A0, B0* y *C0*
- *S1* es la suma de *A1, B1* y el acarreo resultante de la suma anterior
- *C2* es el acarreo de salida resultante de la suma aritmética de *A1* y *B1*.

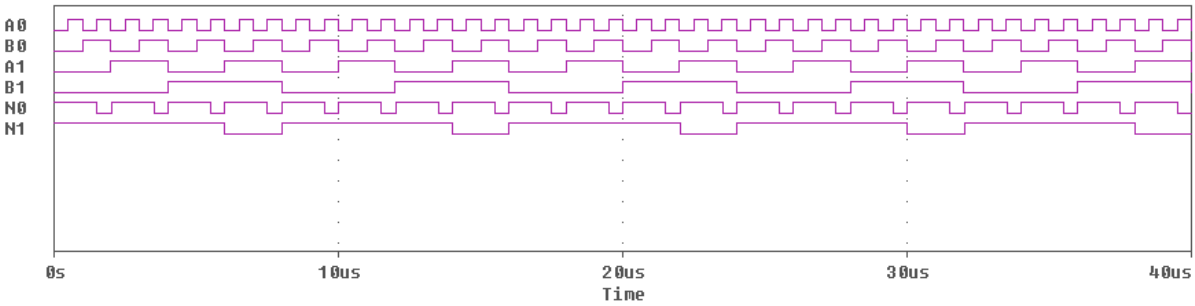


En el siguiente cronograma mostramos las entradas y las salidas del bloque que realiza la función XOR bit a bit

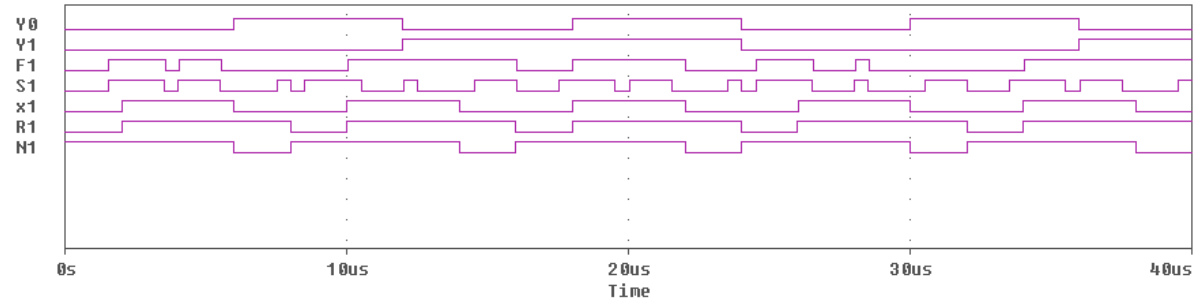


Análogamente, los cronogramas de los bloques OR y NAND son:

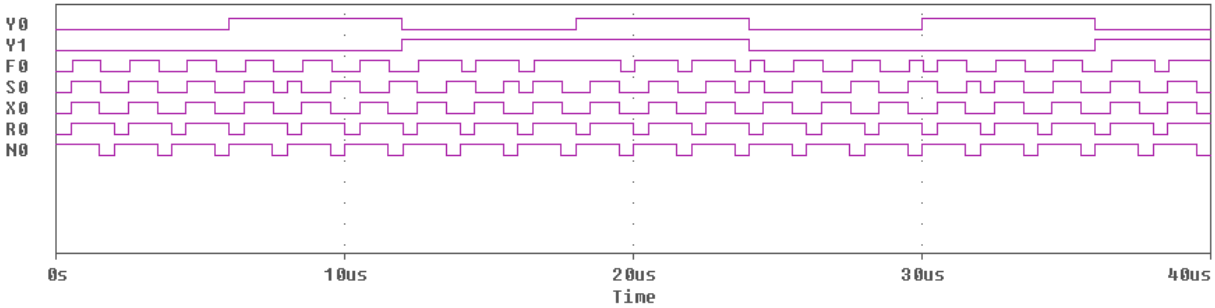




En el siguiente cronograma representamos la salida *F1* en función de los valores de las señales de control. Como podemos ver, cuando ambas señales de control, *Y1* e *Y0*, toman el valor “0”, la salida *F1* coincide con la señal *S1*. Cuando toman el valor *Y1Y0=01*, *F1* coinciden co *X1*, Cuando toman el valor *Y1Y0=10, F1* coincide con *R1* y, finalmente, cuando *Y1Y0=11* entonces *F1=N1*.



De igual forma, *F0* coincide con *S0, X0, R0* y *N0* cuando los valores de las señales de control toman los valores correspondientes a cada una.



En la salida *F2* se presenta el acarreo de la *suma aritmética* cuando la palabra de control toma el valor *Y1Y0=00, en el resto de las operaciones siempre estará a "0".*

P.2.24: ¿Cómo puedo usar la ALU SN74181 y aprovechar todas las posibilidades que ofrece?

R.2.24: La ALU es un bloque funcional que realiza operaciones aritméticas y lógicas y para ello necesita de una serie de circuitos adicionales para su programación a través de las señales de control *S3, S2, S1, S0* y *M*.

Si ponemos estas señales de control a valores fijos, entonces realizará la función seleccionada sobre las 2 palabras de entrada, *A* y *B*, cuyos bits también pondremos a "0" ó a "1" en función de los valores concretos sobre los que queremos que realice la operación. Con esto el funcionamiento de la ALU es, por decirlo de alguna forma, "estático". Sólo hace una operación determinada y sobre dos palabras concretas.

Sin embargo, su uso suele ser más flexible ya que puede ir cambiando de función e ir realizando distintas funciones sobre cualquier par de las posibles palabras de entrada, que también pueden ir cambiando. Para ello basta con que usemos trenes de pulsos para las señales de control, de forma que en cada intervalo de tiempo (semiperiodo de la señal de menor periodo) la palabra de control (S3, S2, S1, S0) toma un valor distinto y, por tanto, realizará una operación distinta, la que corresponda a la palabra *S* en la tabla de verdad de dicha ALU. Además, también podemos usar generadores de pulsos para generar los datos correspondientes a las palabras sobre las que va a operar, palabras *A* y *B*.

Por tanto, a la hora de ver el funcionamiento de la ALU en el simulador es conveniente usar para las distintas entradas (datos y control) generadores de trenes de pulsos del tipo DigClock.

Conviene que los periodos de las señales de control sean lo suficientemente grandes como para que abarquen todas las posibles palabras sobre las que va a operar y así podamos comprobar en el cronograma que, ante una determinada configuración de las señales de control, realiza la operación seleccionada de forma adecuada sobre todas las posibles palabras.

Finalmente, cualquiera de los circuitos que estudiamos en esta asignatura va a funcionar integrado en un sistema digital mucho más complejo, como por ejemplo, un ordenador. Por tanto, en cada momento realizará la función marcada por sus señales de control y sobre los datos que le llegue y que va a depender del programa que se esté ejecutando en ese momento.

✦✦✧

P.2.25: ¿Qué significa el acarreo de salida Cn+4 en la ALU?

R.2.25: Es el acarreo producido al operar con el cuarto bit (A_3, B_3) cuando la ALU está programada para realizar operaciones aritméticas.

Es importante comprobar que la salida C_{n+4} de la ALU está complementada. Es decir, si al realizar una operación aritmética no se genera acarreo, C_{n+4} debería ser "0" y, sin embargo, la ALU presenta en C_{n+4} un "1". Por ejemplo, si los bits más significativos de las palabras de entrada son A_3=B_3=*0* y estamos sumando, verá que en esa salida (C_{n+4}) aparece un "1", cuando debería ser "0".

Esto es debido a que está construida así para estar de acuerdo con la señal de entrada C_n negada de la tabla de verdad. Así, si queremos ampliar la ALU para que opere sobre palabras de 8 bits lo que hacemos es unirlas directamente. Es decir, unimos la salida C_{n+4} de la ALU que opera sobre los 4 bits menos significativos de las palabras *A* y *B* a la entrada C_n de la ALU que opera sobre los 4 bits más significativos, para que el resultado sea correcto y que este acarreo se sume a los bits menos significativos de la segunda ALU, A_4 y B_4.

✦✦✦

P.2.26: ¿Qué significa que la ALU se pone a 0 y qué diferencia hay entre la F=0 de la función lógica y F=cero de la función aritmética de la ALU?

R.2.26: Poner la ALU a "0" es hacer que la ALU presente a la salida "0". Es decir, que la palabra *F=*F_3 F_2 F_1 F_0 *= 0 0 0 0.* De acuerdo con la tabla de verdad de la ALU, para ponerla a cero deberemos programarla con la palabra *S=0011 y M = 1* ó con *S=0011* y *M = 0* y C_n=*L*.

No hay ninguna diferencia entre *F=0* (lógica) y *F=cero* (aritmética), la palabra de salida es la misma, *F=0000*.

Como se puede comprobar en la tabla de verdad, en la ALU hay funciones repetidas para la realización de funciones lógicas y aritméticas con el fin de facilitar la programación. Así, por ejemplo, la suma lógica o función OR existe en las tres columnas de la tabla de verdad y en todas está representada por el signo +, mientras que la suma aritmética sólo

aparece en las columnas de las funciones aritméticas y está representada por la palabra "PLUS".

✦✦✦

P.2.27: ¿Qué valor hay que poner en la entrada de acarreo, Cn, al programar la ALU para que realice las distintas funciones aritméticas con acarreo y sin acarreo?

R.2.27: La ALU puede programarse para que opere tanto en *activa en alta* como *activa en baja*.

En esta asignatura todos los circuitos los diseñamos y usamos considerando *activa en alta* porque nos parece la forma de trabajo más intuitiva.

La tabla de verdad de la ALU que aparece en el texto base *es correcta* y se corresponde con la que proporciona Texas Instruments y cuya hoja de característica se adjunta con este documento.

El problema está en que las tablas de verdad del funcionamiento que proporcionan los distintos fabricantes dependen de la *interpretación* de las señales, pero las señales de entrada y salida a la ALU son las que le entran y salen al circuito, sólo que se interpretan de una forma u otra dependiendo de si se considera activa en alta o en baja.

La tabla siguiente (proporcionada por el fabricante) muestra la forma en la que se pueden interpretar dichas señales en función de si se considera activa en alta o en baja.

PIN NUMBER	2	1	23	22	21	20	19	18	9	10	11	13	7	16	15	17
Active-low data (Table 1)	$\overline{A}_0$	$\overline{B}_0$	$\overline{A}_1$	$\overline{B}_1$	$\overline{A}_2$	$\overline{B}_2$	$\overline{A}_3$	$\overline{B}_3$	$\overline{F}_0$	$\overline{F}_1$	$\overline{F}_2$	$\overline{F}_3$	C_n	C_{n+4}	$\overline{P}$	$\overline{G}$
Active-high data (Table 2)	A_0	B_0	A_1	B_1	A_2	B_2	A_3	B_3	F_0	F_1	F_2	F_3	$\overline{C}_n$	$\overline{C}_{n+4}$	X	Y

Observen que las señales de activa en baja son negadas respecto de las de activa en alta.

Como hemos dicho anteriormente, nosotros vamos a trabajar en *activa en alta* tanto en los diseños como en las simulaciones. Por tanto, para activa en alta, tanto las palabras de entrada, $A_3\ A_2\ A_1\ A_0$ y $B_3\ B_2\ B_1\ B_0$, como las señales de control, $S_3\ S_2\ S_1\ S_0$, M, $\overline{C_n}$, las interpretamos tal cual, o sea, son directamente (sin negar) los datos que salen de los generadores de pulsos y que entran en la ALU y que se corresponden con la cabecera de la tabla de verdad de la ALU para activa en alta. Estos serán los valores sobre los que deberemos operar a la hora de comprobar el funcionamiento de dicha ALU.

Observen que las únicas señales que aparecen negadas son los acarreos o arrastres de entrada y salida, $\overline{C_n}$ y $\overline{C_{n+4}}$.

El significado de $\overline{C_n}$ en la tabla de verdad (de activa en alta, la del texto) es:

$\overline{C_n}$ = H (sin Acarreo). Por tanto, su significado es C_n = 0. O sea, Acarreo = 0

$\overline{C_n}$ = L (con Acarreo). Por tanto, su significado es C_n = 1. O sea, Acarreo = 1

Si miramos la *página 3-712* de las hojas de características adjuntas verán que esta señal está negada, esta es la razón por la que a este terminal lo *nombran* con $\overline{C_n}$. De nuevo,

nosotros en el terminal marcado con $\overline{C_n}$ ponemos directamente la señal, sin negarla. Es decir, a la ALU no hay que añadirle ningún inversor. Se programa directamente con las señales tal cual y se interpretan también tal cual (salvo C_{n+4}, como veremos más adelante). Así, si queremos programar la ALU para que realice la función aritmética sin acarreo "*A PLUS B*" deberemos poner los siguientes valores a las señales de control:

S_3 S_2 S_1 S_0 = H L L H, M = L, terminal C_n (olvidaros del negado) = *H*.

Si por el contrario, queremos que realice la operación "*A PLUS B PLUS 1*" deberemos poner:

S_3 S_2 S_1 S_0 = H L L H, M = L, terminal C_n (olvidaros del negado) = L.

La señal de acarreo de salida, C_{n+4}, sale negada. Es decir, cuando no hay acarreo de salida en C_{n+4} se obtiene un "1" y cuando hay acarreo se obtiene "0". Aquí si que al comprobar el funcionamiento en el simulador hay que tener en cuenta que está negada.

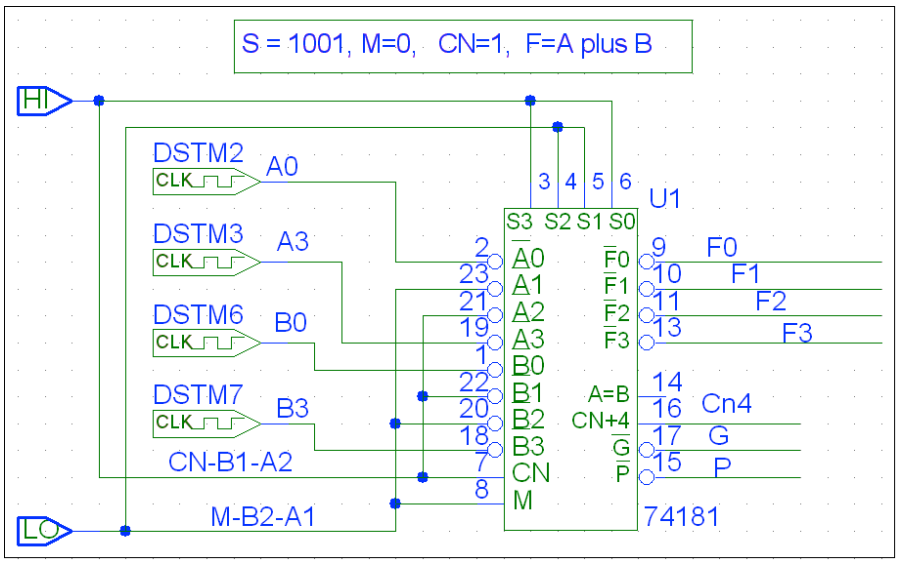
La ALU está diseñada para que se pueda ampliar su funcionamiento a palabras de más bits, permitiendo su conexión en cascada o serie, uniendo directamente la salida $\overline{C_{n+4}}$ de la primera ALU (4 bits menos significativos) a la entrada $\overline{C_n}$ de la segunda (4 bits más significativos) y así sucesivamente cada vez que ampliemos, en 4 bits, las palabras sobre las que opera. Esto justifica el hecho de que sea $\overline{C_n}$ = H (sin acarreo) y $\overline{C_n}$ = L (con acarreo) ya que se va a unir directamente al $\overline{C_{n+4}}$ y cuyo significado es: $\overline{C_{n+4}}$ = H cuando no hay acarreo y $\overline{C_{n+4}}$ = L cuando hay acarreo. El significado se mantiene y no hay que introducir ningún inversor para su ampliación, sino que se une una ALU a la siguiente directamente.

Esta forma de conexión en cascada implica que tiene lugar un acarreo encadenado o enlazado de la misma forma que hemos visto en los sumadores. El problema que tiene este tipo de conexión es el retardo en la presentación del resultado final debido a que hasta que no se consiga el acarreo de la primera ALU, no se puede obtener el resultado de la segunda ALU y así sucesivamente.

Para subsanar este problema en la rapidez del cálculo, el circuito tiene la posibilidad de poderse usar con acarreo adelantado. Para ello, se usa la ALU junto con el circuito SN74182 que es un generador de acarreo adelantado del tipo del circuito de la *figura 5.12 del texto*. Para su conexión utiliza las salidas $\overline{P}$ y $\overline{G}$, siendo $\overline{P}$ = señal de Propagación del Acarreo y $\overline{G}$ = señal de Generación de Acarreo, al igual que consideramos en la *página 277 del texto*. Así, para ampliar la ALU a palabras de más bits, por ejemplo a 16 bits y con acarreo adelantado, se conectan las salidas $\overline{P}$ y $\overline{G}$ de cada una de las ALU (SN74181) a una de las parejas de entradas $\overline{P_i}$ y $\overline{G_i}$ del generador de acarreo adelantado (SN74182).

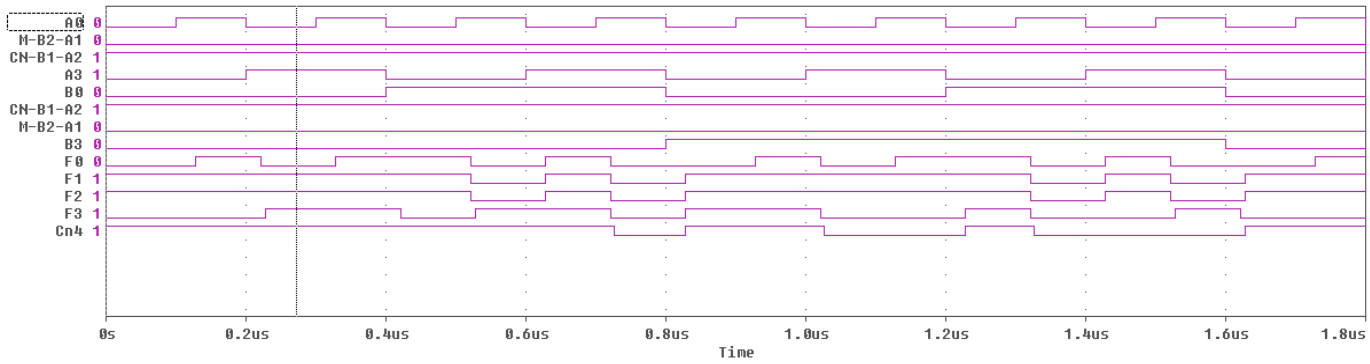
Para concretar y para que después de esta explicación no quede ninguna duda acerca de la interpretación de las señales de entrada y salida de la ALU, vamos a realizar el siguiente ejemplo:

Vamos a programar la ALU para que, según la Tabla de Verdad del texto, realice la función aritmética sin acarreo, *A PLUS B*. Los valores de las señales de entrada son las que se muestran en el siguiente circuito:



Con el fin de simplificar la tabla de verdad experimental es recomendable el uso de relojes en dos de los bits de cada una de las palabras de entrada, mientras que el resto de los bits conviene ponerlos a un valor fijo de “0” ó “1”. Así, para ver los valores que va tomando C_{n+4} y comprobar que realmente sale negado, una buena solución es usar los generadores de pulsos para los bits menos significativos y para los bits más significativos de ambas palabras, y los bits intermedios ponerlos fijos, uno a “0” y el otro a “1”, por ejemplo. Con estos valores los resultados que se obtienen en el cronograma y, por tanto, en la tabla de verdad (sólo contiene 16 términos mínimos) son bastantes significativos como para comprobar que el circuito funciona correctamente.

El cronograma que se obtiene es:



A continuación mostramos la tabla de verdad experimental obtenida a partir del cronograma y la teórica obtenida realizando la operación *A PLUS B* bits a bit (S^*_0, S^*_1,

S^*_2, S^*_3) y teniendo en cuenta el acarreo de cada bit (C_1, C_2, C_3, C_4) en la suma del bit de orden superior:

Tablas de Verdad Práctica				**Tabla de Verdad Teórica**							
$A_3\ A_2\ A_1\ A_0$	$B_3\ B_2\ B_1\ B_0$	$F_3\ F_2\ F_1\ F_0$	C_{n+4}	C_4	S^*_3	C_3	S^*_2	C_2	S^*_1	C1	S^*_0
0100	0010	0110	1	0	**0**	0	**1**	0	**1**	0	**0**
0101	0010	0111	1	0	**0**	0	**1**	0	**1**	0	**1**
1100	0010	1110	1	0	**1**	0	**1**	0	**1**	0	**0**
1101	0010	1111	1	0	**1**	0	**1**	0	**1**	0	**1**
0100	0011	0111	1	0	**0**	0	**1**	0	**1**	0	**1**
0101	0011	1000	1	0	**1**	1	**0**	1	**0**	1	**0**
1100	0011	1111	1	0	**1**	0	**1**	0	**1**	0	**1**
1101	0011	0000	0	1	**0**	1	**0**	1	**0**	1	**0**
0100	1010	1110	1	0	**1**	0	**1**	0	**1**	0	**0**
0101	1010	1111	1	0	**1**	0	**1**	0	**1**	0	**1**
1100	1010	0110	0	1	**0**	0	**1**	0	**1**	0	**0**
1101	1010	0111	0	1	**0**	0	**1**	0	**1**	0	**1**
0100	1011	1111	1	0	**1**	0	**1**	0	**1**	0	**1**
0101	1011	0000	0	1	**0**	1	**0**	1	**0**	1	**0**
1100	1011	0111	0	1	**0**	0	**1**	0	**1**	0	**1**
1101	1011	1000	0	1	**1**	1	**0**	1	**0**	1	**0**
0100	0010	0110	1	0	**0**	0	**1**	0	**1**	0	**0**

Como podemos comprobar, las columnas encabezadas por S*3, S*2, S*1, S*0, obtenidas sumando las palabras A y B, coinciden con la columna encabezada por F3 F2 F1 F0 obtenida del cronograma. El acarreo total, C4, coincide con la columna $\overline{C}_{n+4}$ 4. Por tanto, como hemos podido comprobar el circuito funciona correctamente.

Para el que tenga interés en saber más acerca de este circuito integrado, adjuntamos como anexo al final de este documento las hojas de características del dispositivo fabricado por Texas Instruments. Es interesante ver el circuito interno que proporciona dicho fabricante.

Características de la ALU integrada SN74181 de Texas Instruments

✦✦✦

P.2.28: ¿Es necesario saberse la figura 5.22 (pag. 291)?

R.2.28: No, no hace falta aprenderla de memoria. En esta figura presentamos el símbolo con el que se representa el circuito de acuerdo con la norma de "IEEE Standard 91-1984" y que muestra la relación entre cada entrada de un circuito lógico y cada salida, sin necesidad

de mostrar explícitamente el circuito interno. Lo que se suele hacer es consultarla cuando nos hace falta, como cualquier norma.

En el *Apéndice B (pag. 725)* se puede ver un resumen de dicha norma en el que hemos seleccionado y explicado el significado de los símbolos usados en los distintos temas del texto.

✦✦✦

P.2.29: ¿Son correctas las funciones de las dos primeras filas de la tabla de la figura 5.6.13 de la pag. 177 del libro de problemas?.

R.2.29: 1: Veamos primero la función $A + \overline{B}$ (arit.)

Debemos recordar que en la ALU el + implica realizar la función OR, con independencia de que esté programada para realizar operaciones lógicas o aritméticas.

Para representar la suma aritmética se usa siempre la palabra "PLUS".

Así, realizando la función OR bit a bit, resulta:

$$A + \overline{B} = 0011 + 1110 = (0011)\text{OR}\,(1110) = 1111$$

Por tanto, en las cuatro últimas columnas (F_3, …F_0) de la primera fila [$A + \overline{B}$ (arit.)] de la tabla de la *figura 5.6.13,* debe decir 1111.

2. Veamos ahora la función: ***(A + $\overline{B}$) plus AB* (arit.).**

Calculamos *AB* que es la función AND bit a bit entre A y B. Es decir,

AB = (0011) AND (0001) = 0001

La función $A + \overline{B}$ la hemos calculado anteriormente, $A + \overline{B} = 1111$

Así: *(A + $\overline{B}$) plus AB = (1111) plus (0001) = (1) 0000*

Por tanto, en las cuatro últimas columnas (F_3, F_0) de la tercera fila [*(A + $\overline{B}$) plus AB* (arit.)] de la tabla debe decir 0000.

Así, hay que corregir las dos primeras filas de la tabla de la figura 5.6.13, pero además las filas están mal ordenadas. Se ha cambiado el orden de las filas segunda y tercera al pasar de la tabla de la figura 5.6.12 a la de la figura 5.6.13. Por tanto la tabla correcta para la figura 5.6.13 es:

Función	A_3	A_2	A_1	A_0	B_3	B_2	B_1	B_0	F_3	F_2	F_1	F_0
$A + \overline{B}$ (arit.)	0	0	1	1	0	0	0	1	1	1	1	1
$A \oplus B$ (log.)	0	0	1	1	0	0	0	1	0	0	1	0
($A + \overline{B}$) PLUS AB (arit.)	0	0	1	1	0	0	0	1	0	0	0	0
$A + B$ (log.)	0	0	1	1	0	0	0	1	0	0	1	1

Esto hace que las formas de onda mostradas en la *figura 5.6.14* tampoco sean correctas. Así, la figura 5.6.14 correcta es:

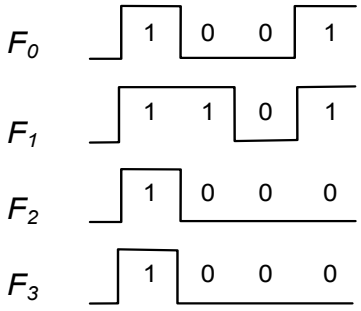


Fig. 5.6.14

ANEXO: HOJAS DE CARACTERÍSTICAS