Grado en Ingeniería de Tecnologías de Telecomunicación.
Electrónica Digital I. Problemas resueltos. Tema IIIb.

Página 1. Diseñar un circuito multiplexor con prioridad de 4 bits. El circuito tiene 4 entradas de datos (I3-I0), 4 entradas de selección (S3-S0) y dos salidas Z y G. Cuando una o más de las entradas S están a 1, Z toma el valor de la entrada Ii, siendo i es el índice más alto de las entradas Si que están a 1; si todas las entradas S3-S0 están a 0, entonces Z toma el valor 0. La salida G se fija a 1 si al menos alguna entrada Si está a 1, en caso contrario se fija a 0.
a) Mostrar en una tabla el comportamiento lógico del circuito. Encontrar las ecuaciones lógicas de las salidas Z y G expresándolas en dos niveles y en forma factorizada.
b) Implementar la expresión factorizada de Z utilizando multiplexores de 2 entradas.
c) Diseñar un multiplexor con prioridad de 16 bits en base a los multiplexores con prioridad de 4 bits diseñados.
d) Realizar una descripción VHDL del multiplexor con prioridad.

a) El multiplexor con prioridad sigue esta tabla, donde se muestran las condiciones en las entradas S para que cada una de las entradas I pase a la salida.

Función	**S3**	**S2**	**S1**	**S0**	**Z**	**G**
$S3\ I3$	1	X	X	X	I3	1
$\overline{S3}\ S2\ I2$	0	1	X	X	I2	1
$\overline{S3}\ \overline{S2}\ S1\ I1$	0	0	1	X	I1	1
$\overline{S3}\ \overline{S2}\ \overline{S1}\ S0\ I0$	0	0	0	1	I0	1
$\overline{S3}\ \overline{S2}\ \overline{S1}\ \overline{S0} \bullet 0$	0	0	0	0	0	0

De la tabla genero Z y la factorizo:

$$Z = S3\ I3 + \overline{S3}\ S2\ I2 + \overline{S3}\ \overline{S2}\ S1\ I1 + \overline{S3}\ \overline{S2}\ \overline{S1}\ S0\ I0$$
$$Z = S3\ I3 + \overline{S3}\ (S2\ I2 + \overline{S2}\ S1\ I1 + \overline{S2}\ \overline{S1}\ S0\ I0)$$
$$Z = S3\ I3 + \overline{S3}\ [S2\ I2 + \overline{S2}\ (S1\ I1 + \overline{S1}\ S0\ I0)]$$

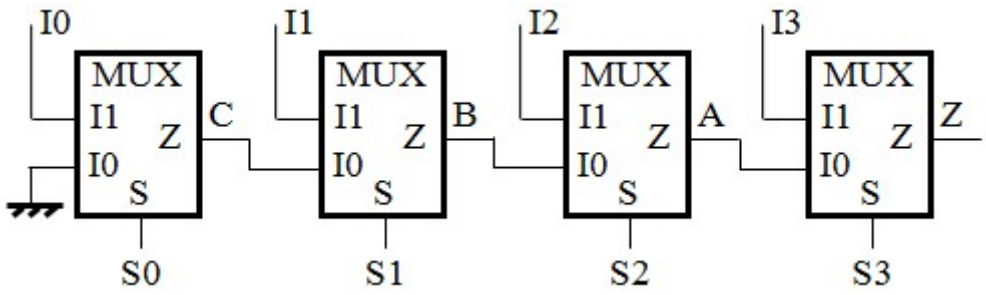
También se ve intuitivamente de la tabla (G es 1 si alguna de las entradas S es 1, y 0 si no lo es ninguna) que:

$$G = S3 + S2 + S1 + S0$$

b) Los multiplexores de 2 entradas siguen la expresión $Z = S\ I1 + \overline{S}\ I0$. De la expresión factorizada de Z todos los paréntesis guardan la relación de un multiplexor de dos entradas, salvo el paréntesis más interno, que se puede modificar para que lo cumpla.

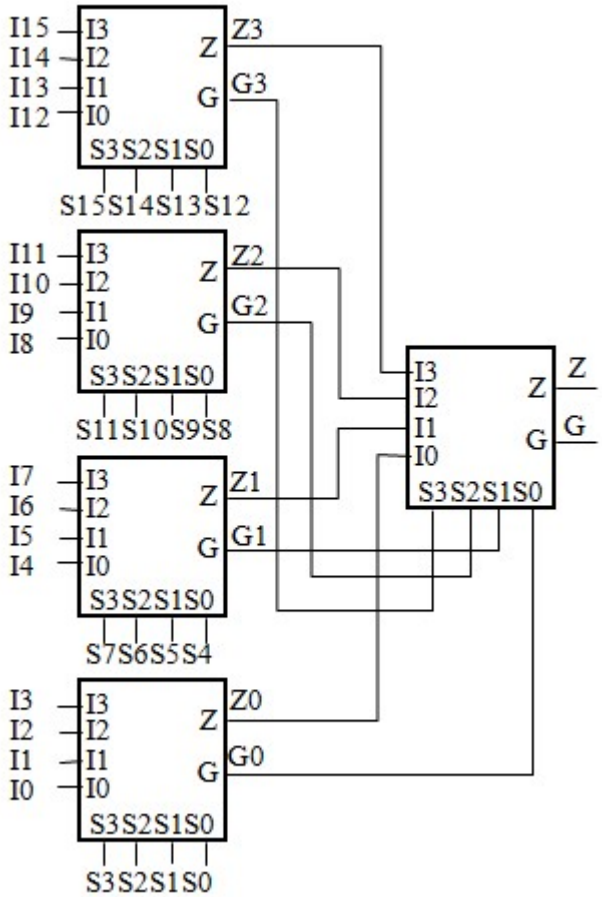
$$Z = S3\ I3 + \overline{S3}\ A$$
$$A = S2\ I2 + \overline{S2}\ (S1\ I1 + \overline{S1}\ S0\ I0) = S2\ I2 + \overline{S2}\ B$$
$$B = S1\ I1 + \overline{S1}\ S0\ I0 = S1\ I1 + \overline{S1}\ C$$
$$C = S0\ I0 = S0\ I0 + \overline{S0} \bullet 0$$

Con lo que se puede diseñar este circuito:



c) Para generar un circuito multiplexor con prioridad de 16 bits con multiplexores con prioridad de 4 bits, hay que usar un circuito como el de la figura. En la figura los 16 bits se dividen en grupos de 4 ordenados de mayor a menor, tanto las entradas I como las entradas S, con lo que se usan 4 multiplexores con prioridad de 4 bits. Se necesita un quinto multiplexor para general la salida final, el multiplexor de la derecha en la figura.

Si alguna de las entradas S15-S12 está a 1 entonces su G (G3) es 1, con lo que el quinto multiplexor tiene su S3 a 1, por lo que Z es su I3, que es Z3, donde Z3 es una de las entradas I15-I12, dependiendo de la entrada de índice más alto a 1 en S15-S12. Si ninguna de las entradas S15-S12 están a 1, entonces G3 es 0, y habría que mirar el resto de las entradas, empezando por S11-S8, repitiendo el funcionamiento anterior con Z2 y G2. Similarmente si S11-S8 son 0, se comprueban S7-S3 (Z1 y G1), y si estas son 0 se comprueba S3-S0 (Z0 y G0). Si todas las entradas S son 0, en el quinto multiplexor todas las entradas S son 0s, por lo que Z es 0.



d) Primero hago una descripción VHDL específica para el problema de 4 bits basada en una sentencia del tipo *if-elsif-else*. Hago una segunda descripción genérica para cualquier número de bits, donde uso sentencias VHDL no vistas en las clases teóricas: la sentencia *exit* en un *for*

permite abandonar el lazo sin recorrerlo entero. Esta descripción genérica podría hacerse también sin recurrir a estas sentencias, usando variables extras de control. En la descripción genérica inicializo primero las salidas Z y G a 0, y luego las modifico si alguna de las entradas S está a 1.

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

entity muxconpri4 is
port( S, I: in std_logic_vector(3 downto 0);
      Z, G: out std_logic );
end muxconpri4;

architecture uno of muxconpri4 is
begin
process(S, I)
begin
if ( S(3) = '1' ) then
   Z <= I(3); G <= '1';
elsif ( S(2) = '1' ) then
   Z <= I(2); G <= '1';
elsif ( S(1) = '1' ) then
   Z <= I(1); G <= '1';
elsif ( S(0) = '1' ) then
   Z <= I(0); G <= '1';
else Z <= '0'; G <= '0';
end if;
end process;
end uno;
```

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

entity muxconprigen is
generic (N: integer := 4);
port( S, I: in std_logic_vector(N-1 downto 0);
      Z, G: out std_logic );
end muxconprigen;

architecture uno of muxconprigen is
begin
process(S, I)
begin
Z <= '0'; G <= '0';
for j in N-1 downto 0 loop
   if ( S(j) = '1' ) then
      Z <= I(j); G <= '1';
      exit; -- Finaliza el lazo si hay 1 en S
   end if;
end loop;
end process;
end uno;
```

Página 2-1. Construir un multiplexor de 5 entradas
a) utilizando puertas lógicas.
b) utilizando multiplexores de dos entradas.

Supongo un multiplexor de 5 entradas, numeradas de I0 a I4. Para controlar las 5 entradas se necesitan 3 entradas de selección (S2 S1 S0), capaces de codificar de 0 (000) a 4 (100). Inicialmente la tabla del problema queda:

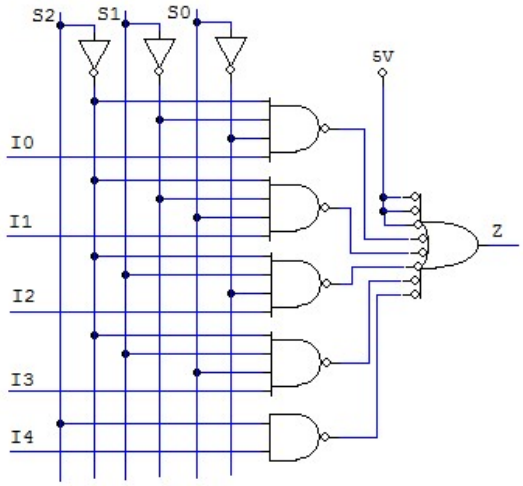
Función	S2	S1	S0	Z
$\overline{S2}\ \overline{S1}\ \overline{S0}$ I0	0	0	0	I0
$\overline{S2}\ \overline{S1}$ S0 I1	0	0	1	I1
$\overline{S2}$ S1 $\overline{S0}$ I2	0	1	0	I2
$\overline{S2}$ S1 S0 I3	0	1	1	I3
S2 $\overline{S1}\ \overline{S0}$ I4	1	0	0	I4

Pero si nos fijamos en que la última entrada es la única fila que tiene S2 a 1, la tabla puedo definirla como:

Función	S2	S1	S0	Z
$\overline{S2}\ \overline{S1}\ \overline{S0}$ I0	0	0	0	I0
$\overline{S2}\ \overline{S1}$ S0 I1	0	0	1	I1
$\overline{S2}$ S1 $\overline{S0}$ I2	0	1	0	I2
$\overline{S2}$ S1 S0 I3	0	1	1	I3
S2 I4	1	X	X	I4

a) La función lógica de Z puede obtenerse directamente de la tabla anterior, sin realizar ninguna simplificación, e implementarla con puertas lógicas (NAND en la figura).

$Z = \overline{S2}\ \overline{S1}\ \overline{S0}\ I0 + \overline{S2}\ \overline{S1}\ S0\ I1 + \overline{S2}\ S1\ \overline{S0}\ I2 + \overline{S2}\ S1\ S0\ I3 + S2\ I4$

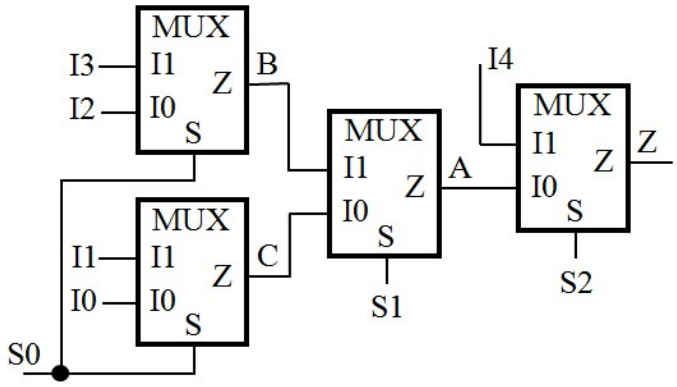


b) La función del apartado a) puede operarse de forma que las expresiones queden del tipo $S\ I1 + \overline{S}\ I0$, que corresponden a multiplexores de dos entradas.

$Z = S2\ I4 + \overline{S2}\ (\overline{S1}\ \overline{S0}\ I0 + \overline{S1}\ S0\ I1 + S1\ \overline{S0}\ I2 + S1\ S0\ I3) =$
$= S2\ I4 + \overline{S2}\ [S1\ (S0\ I3 + \overline{S0}\ I2) + \overline{S1}\ (S0\ I1 + \overline{S0}\ I0)]$

Con lo que uso los siguientes cuatro multiplexores de dos entradas y genero el circuito con multiplexores de dos entradas.

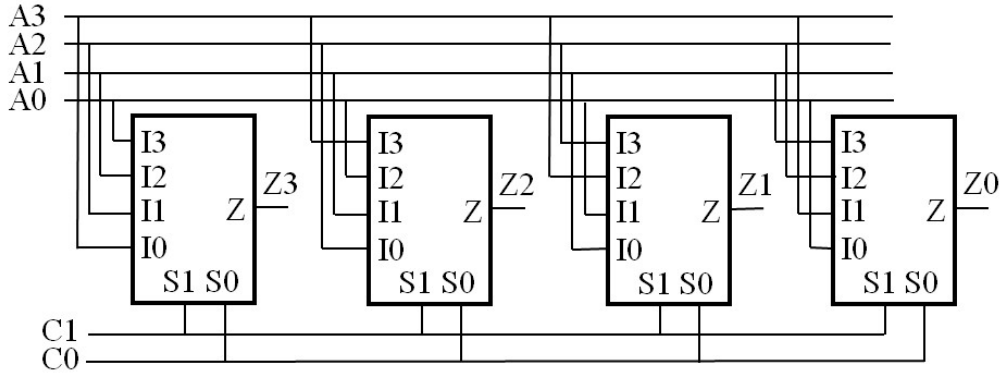
$Z = S2\ I4 + \overline{S2}\ A$
$A = S1\ B + \overline{S1}\ C$
$B = S0\ I3 + \overline{S0}\ I2$
$C = S0\ I1 + \overline{S0}\ I0$



Página 2_2. Un circuito de "desplazamiento en barril" ("barrel-shifter") mueve los datos de entrada de forma que aparezcan en la salida girados el número de posiciones marcados por las señales de control. Construir utilizando multiplexores un "barrel-shifter" de 4 bits de entrada (a3a2a1a0) y 4 bits de salida (z3z2z1z0) con 4 posibles desplazamientos (dos señales de control c1c0):
(c1c0) = 0 => (z3z2z1z0) = (a3a2a1a0),
(c1c0) = 1 => (z3z2z1z0) = (a2a1a0a3),
(c1c0) = 2 => (z3z2z1z0) = (a1a0a3a2),
(c1c0) = 3 => (z3z2z1z0) = (a0a3a2a1).
Realizar la descripción VHDL de este circuito.

Del enunciado se puede obtener la operación de las 4 salidas, comparadas con el funcionamiento de un multiplexor de cuatro entradas, suponiendo que las entradas de control C1 y C0 se conectan directamente a las entradas de selección C1 y C0 del multiplexor. Cada salida Z puede ser implementada con un multiplexor de 4 cuyas correspondientes entradas I0, I1, I2 y I3 se conectan a las entradas A indicadas en la tabla.

S1	S0	Z	C1	C0	Z3	Z2	Z1	Z0
0	0	I0	0	0	A3	A2	A1	A0
0	1	I1	0	1	A2	A1	A0	A3
1	0	I2	1	0	A1	A0	A3	A2
1	1	I3	1	1	A0	A3	A2	A1



La descripción VHDL puede ser:

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

entity barrel is
port(A: in std_logic_vector(3 downto 0);
     C: in std_logic_vector(1 downto 0);
     Z: out std_logic_vector(3 downto 0) );
end barrel;

architecture uno of barrel is
begin
process(A,C)
begin
case C is
   when "00" => Z <= A;
   when "01" => Z <= A(2 downto 0) & A(3);
   when "10" => Z <= A(1) & A(0) & A(3) & A(2);
   when "11" => Z <= A(0) & A(3 downto 1);
end case;
end process;
end uno;
```

Página 3_1. Diseñar un multiplexor de 16 entradas utilizando 4 multiplexores triestado de 4 entradas con habilitador (el circuito deshabilitado queda en alta impedancia) y un decodificador 2 a 4.
Indicar como debería diseñarse el circuito con dos chips 74'153, un chip 74'139 y una puerta lógica.

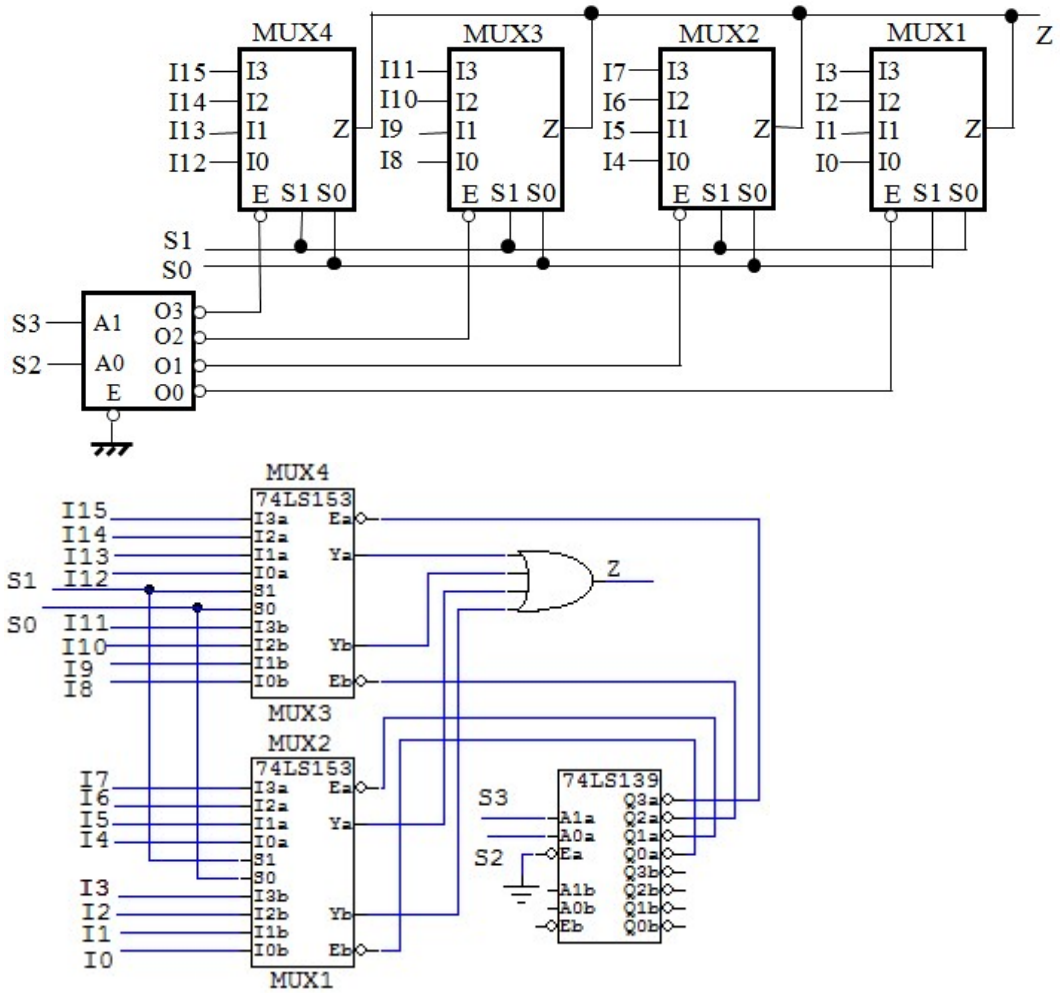
Un multiplexor de 16 entradas tiene, una salida Z, 16 entradas de datos I15-I0 seleccionadas por cuatro entradas de selección S3-S0, y puede tener una entrada de habilitación**.** Para construir un multiplexor de 16 entradas este circuito se necesitan al menos 4 multiplexores. Hay una

versión con cinco multiplexores de 4 entradas en la diapositiva 8 de teoría. Por el enunciado disponemos de cuatro multiplexores de 4 entradas y un decodificador 2 a 4. El método para diseñar el circuito consiste en conectar las salidas del decodificador a las entradas de habilitación de los multiplexores. De esta forma solo habrá cada vez un multiplexor habilitado y el resto deshabilitados. Lo más sencillo es conectar las entradas de selección más significativas a las entradas del decodificador y las menos significativas como entradas de selección de los multiplexores. Como hay cuatro salidas intermedias de cada uno de los multiplexores, y hay que juntarlas finalmente en una única salida, se necesita utilizar lógica adicional. Esta lógica depende de cómo funciona la deshabilitación de los multiplexores: si los deja en alta impedancia, como en el problema inicial, se podrían conectar todas las entradas juntas (en modo *wire*), si los deja a 0 como el 74'153 habría que usar una OR (X + 0 = X), si los deja a 1 habría que usar una AND (X • 1 = X).

El control de los multiplexores queda según esta tabla:

S3	S2	MUX1	MUX2	MUX3	MUX4	Z (F(S1,S0))
0	0	ON	OFF	OFF	OFF	I3,I2,I1,I0
0	1	OFF	ON	OFF	OFF	I7,I6,I5,I4
1	0	OFF	OFF	ON	OFF	I11,I10,I9,I8
1	1	OFF	OFF	OFF	ON	I15,I14,I13,I12

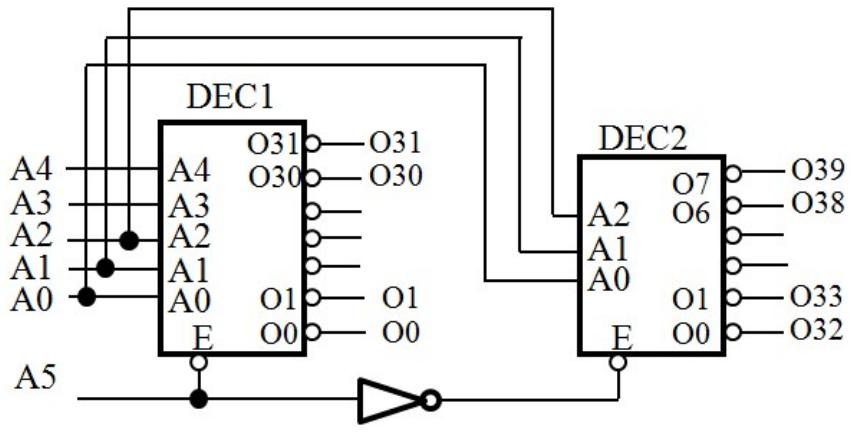
Los circuitos quedarían así:



Página 3_2. Se quiere diseñar un decodificador de 40 direcciones de 0 a 39 utilizando decodificadores binarios (2 a 4, 3 a 8, 4 a 16, etc). Indicar cuál es el número mínimo de decodificadores binarios que hay que utilizar y realizar el diseño del decodificador utilizando los decodificadores binarios y las puertas lógicas que sean necesarias (un inversor).

Hay que conseguir un decodificador con 40 direcciones usando el menor número de decodificadores binarios ideales. El circuito debe tener 6 entradas de direcciones (A5-A0) y 40 salidas. Como 42 = 32 + 8, se puede formar el circuito con dos decodificadores uno 5 a 32 (5 entradas de direcciones A4-A0), y otro de 3 a 8 (3 entradas de direcciones A2-A0). Elijo el decodificador DEC1 de 5 a 32 para las 32 direcciones más bajas (de 0 a 31), y el decodificador DEC2 3 a 8 para las direcciones más altas (de 32 a 39 que son de 0 a 7 en DEC2). Para seleccionar entre DEC1 y DEC2 necesito un decodificador de 1 a 2, pero ese decodificador lo puedo hacer únicamente con un inversor. El decodificador 1 a 2, debe tener como bit de dirección A5, y sus salidas deben ir conectadas a los habilitadores de DEC1 (O0) y DEC2 (O1). Cuando A5 es 0 DEC1 está habilitado y DEC2 está deshabilitado (salidas 32 a 39 a 0), en función de A4-A0 activo una de sus salidas 0-31, mientras que el resto de las salidas están a 0. Cuando A5 es 1 DEC1 está deshabilitado (salidas 0 a 31 a 0) y DEC2 está habilitado y activo una de sus 8 salidas según el valor de A2-A0. Algunos ejemplos de la operación:

32	16	8	4	2	1				
A5	**A4**	**A3**	**A2**	**A1**	**A0**	**DEC1**	**DEC2**	**O31-0**	**O39-32**
0	0	0	0	0	0	ON	OFF	O0 a 1, resto a 0	salidas a 0
0	0	0	1	1	0	ON	OFF	O6 a 1, resto a 0	salidas a 0
0	1	0	1	0	0	ON	OFF	O20 a 1, resto a 0	salidas a 0
0	1	1	1	1	1	ON	OFF	O31 a 1, resto a 0	salidas a 0
1	X	X	0	0	0	OFF	ON	salidas a 0	O32 a 1, resto a 0
1	X	X	0	1	1	OFF	ON	salidas a 0	O35 a 1, resto a 0
1	X	X	1	1	1	OFF	ON	salidas a 0	O39 a 1, resto a 0



Página 3_3. Diseñar un circuito decodificador del código de Hamming capaz de recuperar un error simple en un código (M0M1M2M3) con bits de paridad (P0P1P2) par utilizando puertas EXOR (para determinar F2F1F0 la dirección del bit erróneo, y para complementar dicho bit) y un 3 a 8 DEC (para indicar el bit erróneo en función de F2F1F0).

1	2	3	4	5	6	7	
P0	P1	M0	P2	M1	M2	M3	
X		X		X		X	F0
	X	X			X	X	F1
			X	X	X	X	F2

Para resolver este circuito usaré tres pasos. El primero genera los valores de (F2F1F0) en función de los bits de entrada. Cualquiera de las salidas F debe ser 1 cuando el número de bits de entrada a 1 es impar. Aunque la función lógica y el circuito asociado que realiza esto podría calcularse explícitamente con tablas de verdad y mapas de Karnaugh por ejemplo, se ha indicado en el tema IIa que una puerta EXOR puede considerarse como un circuito que calcula la paridad impar de las salidas (salida a 1 si el número de 1s en las entradas es impar), y hemos visto ejemplo de esto en una puerta EXOR de dos entradas o en la salida de suma de un "full-adder" de 1 bit (EXOR de 3 entradas). Según esto:

$F2 = P2 \oplus M1 \oplus M2 \oplus M3$
$F1 = P1 \oplus M0 \oplus M2 \oplus M3$
$F0 = P0 \oplus M0 \oplus M1 \oplus M3$

El segundo paso calcula el bit incorrecto. Una posibilidad es usar un decodificador 3 a 8. Las entradas de dirección son F2, F1 y F0. Si toman el valor binario 0 (000) no hay error, si toman un valor distinto la salida correspondiente del decodificador se fija a 1 y el resto de las salidas se fija a 0.

El tercer paso es utilizar un circuito entre la salida del codificador y la entrada correspondiente a su valor decimal. Si la salida del decodificador Oi es 0 (sin error), la entrada Ii debe mantenerse, y si es 1 (con error), la entrada debe complementarse. Este circuito ha aparecido en problemas anteriores (tema IIa), y corresponde a una EXOR, como se ve en la tabla.

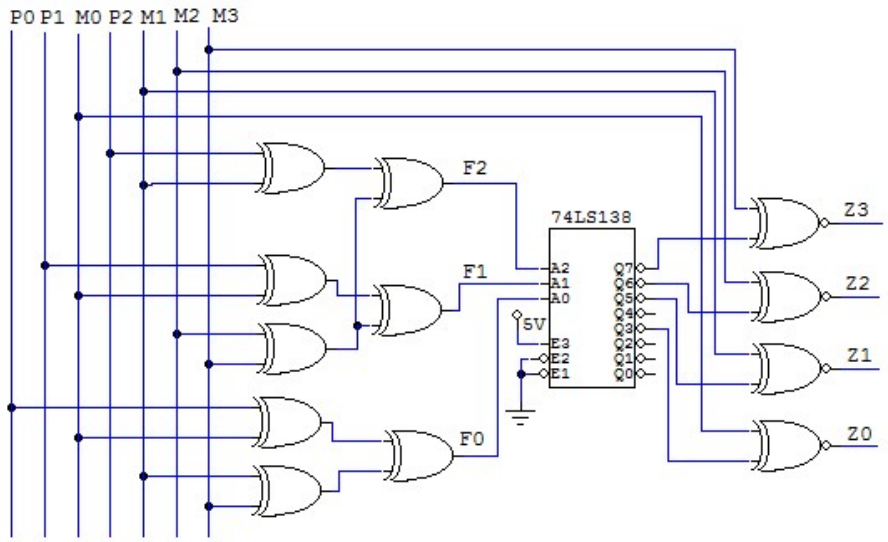
Oi	**Ii**	**Zi**
0	0	0
0	1	1
1	0	1
1	1	0

Como solo hace falta obtener los bits de datos M0, M1, M2 y M3:

$Z0 = O3 \oplus M0$
$Z1 = O5 \oplus M1$
$Z2 = O6 \oplus M2$
$Z3 = O7 \oplus M3$

Si las salidas del decodificador están en lógica negativa, como en los circuitos comerciales de la familia 74, la operación cambia y se debería implementar con puertas EXNOR.

$Z0 = \overline{\overline{O3} \oplus M0} = O3 \oplus M0$
$Z1 = \overline{\overline{O5} \oplus M1} = O5 \oplus M1$
$Z2 = \overline{\overline{O6} \oplus M2} = O6 \oplus M2$
$Z3 = \overline{\overline{O7} \oplus M3} = O7 \oplus M3$



Página 4_1. Realizar un multiplicador para números A (a1a0) y B (b1b0) de 2 bits en complemento-2, utilizando decodificadores 74LS138 y puertas NAND de 4 entradas (74LS20) o de 8 entradas (74LS30). El resultado P (p3p2p1p0) es de 4 bits en complemento-2.

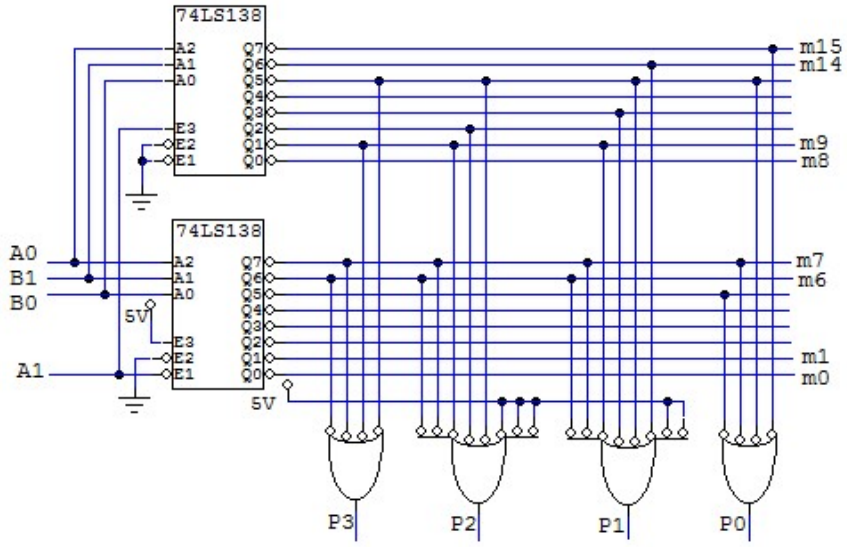
Estos circuitos se desarrollan a partir de la tabla de verdad del problema y se implementan como si fuese una ROM, pero en vez de usar un plano OR programable, hago las líneas OR con puertas lógicas. Como las salidas del decodificador están en lógica negativa las entradas de las puertas OR también deben estarlo, lo que implica que se usan puertas NAND.

		-2	1	-2	1			-8	4	2	1
A	**B**	**A1**	**A0**	**B1**	**B0**	**DEC**	**P**	**P3**	**P2**	**P1**	**P0**
0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	1	1	0	0	0	0	0
0	-2	0	0	1	0	2	0	0	0	0	0
0	-1	0	0	1	1	3	0	0	0	0	0
1	0	0	1	0	0	4	0	0	0	0	0
1	1	0	1	0	1	5	1	0	0	0	1
1	-2	0	1	1	0	6	-2	1	1	1	0
1	-1	0	1	1	1	7	-1	1	1	1	1
-2	0	1	0	0	0	8	0	0	0	0	0
-2	1	1	0	0	1	9	-2	1	1	1	0
-2	-2	1	0	1	0	10	4	0	1	0	0
-2	-1	1	0	1	1	11	2	0	0	1	0
-1	0	1	1	0	0	12	0	0	0	0	0
-1	1	1	1	0	1	13	-1	1	1	1	1
-1	-2	1	1	1	0	14	2	0	0	1	0
-1	-1	1	1	1	1	15	1	0	0	0	1

Según esta tabla:

P3 = F3 (A1, A0, B1, B0) = $\sum$ (6, 7, 9, 13)
P2 = F2 (A1, A0, B1, B0) = $\sum$ (6, 7, 9, 10, 13)
P1 = F1 (A1, A0, B1, B0) = $\sum$ (6, 7, 9, 11, 13, 14)
P0 = F0 (A1, A0, B1, B0) = $\sum$ (5, 7, 13, 15)

Como el problema tiene 4 entradas, la implementación debe hacerse con un decodificador de 4 a 16. El enunciado del problema pide que el circuito a usar sea un 74'138, que es un decodificador 3 a 8, por lo que hay usar dos en la configuración de la diapositiva 15 de teoría. Para las salidas P3 y P2 se pueden usar puertas NAND de 4 entradas, pero para las salidas P2 y P1 se necesitan puertas NAND de 8 entradas; las entradas no usadas se conectan a alto. El circuito queda así:



Página 4_2. Diseñar un circuito que realice simultáneamente:
a) La suma de dos números positivos X (X1.H, X0.H) e Y (Y1.H Y0.H) de dos bits.
b) El producto de dos números positivos M (M1.H, M0.L) e N (N1.L N0.H) de dos bits.
El circuito debe tener solo cuatro bits de entrada y dos grupos diferenciados de salidas para cada una de las operaciones, todas las salidas deben ser de polaridad positiva. El diseño debe realizarse utilizando el menor número de decodificadores 74'138, y el menor número de puertas NAND (suponer que es posible cualquier número de entradas en las puertas).

Este problema es similar al anterior. Hay dos entradas de dos bits, que llamamos A (A1, A0) y (B1, B0), con valores entre 0 y 3, que se comportan como X e Y para hacer la suma, y como M y N para hacer la multiplicación. La suma S de operando de dos bits nos da un resultado de 3 bits (S2, S1, S0, 3+3 = 6), y el producto de operando de dos bits da un resultado de 4 bits (P3, P2, P1, P0, 3*3 = 9). En total son 7 salidas. En principio, como en el problema anterior, el problema puede resolverse con dos decodificadores 74'138 y 7 puertas NAND.

Hay un problema adicional en el producto, ya que en este caso las entradas M0 (A0) y N1 (B1) están en lógica negativa. Para utilizar el mismo circuito hay que suponer que en el producto A0 es $\overline{M0}$ y B1 es $\overline{N1}$, con lo que cambian la posición de los *minterms*, es decir si M = N = 0 (00) y el producto P es 0 (0000), entonces A es 1 (01) y B es 2 (10), con lo que en notación decimal (A1A0B1B0) es 6 (01 10). Las tablas se pueden desarrollar para la suma y el producto. En las tablas aparecen los valores numéricos de X (X1, X0) e Y (Y1, Y0), y M (M1, M0) y N (N1, N0), en decimal y en binario, el resultado de la operación para los valores de X e Y, o M y N en decimal (S, P) y binario (S2S1S0, P3P2P1P0), y cómo X e Y, o M y N se convierten en A y B en binario (A1A0B1B0), y la notación decimal para la suma (DECS) o el producto (DECP), que se toma como base para obtener la función lógica de cada una de las salidas.

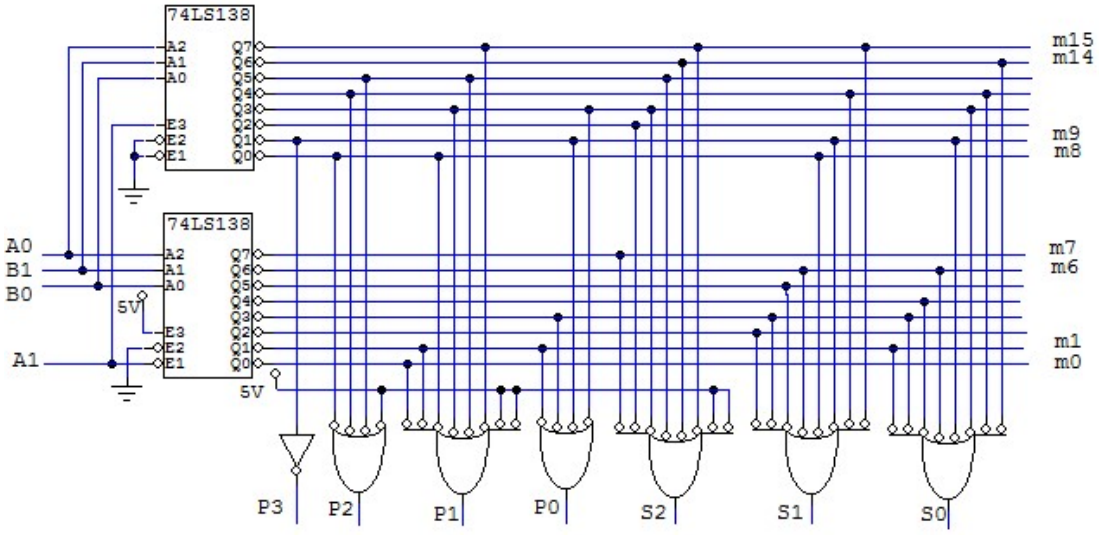
		2	1	2	1		4	2	1	8	4	2	1	
X	**Y**	**X1**	**X0**	**Y1**	**Y0**	**S**	**S2**	**S1**	**S0**	**A1**	**A0**	**B1**	**B0**	**DECS**
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	1	1	0	0	1	0	0	0	1	1
0	2	0	0	1	0	2	0	1	0	0	0	1	0	2
0	3	0	0	1	1	3	0	1	1	0	0	1	1	3
1	0	0	1	0	0	1	0	0	1	0	1	0	0	4
1	1	0	1	0	1	2	0	1	0	0	1	0	1	5
1	2	0	1	1	0	3	0	1	1	0	1	1	0	6
1	3	0	1	1	1	4	1	0	0	0	1	1	1	7
2	0	1	0	0	0	2	0	1	0	1	0	0	0	8
2	1	1	0	0	1	3	0	1	1	1	0	0	1	9
2	2	1	0	1	0	4	1	0	0	1	0	1	0	10
2	3	1	0	1	1	5	1	0	1	1	0	1	1	11
3	0	1	1	0	0	3	0	1	1	1	1	0	0	12
3	1	1	1	0	1	4	1	0	0	1	1	0	1	13
3	2	1	1	1	0	5	1	0	1	1	1	1	0	14
3	3	1	1	1	1	6	1	1	0	1	1	1	1	15

S2 = F2(A1, A0, B1, B0) = ∑(7, 10, 11, 13, 14, 15)
S1 = F1(A1, A0, B1, B0) = ∑(2, 3, 5, 6, 8, 9, 12, 15)
S0 = F0(A1, A0, B1, B0) = ∑(1, 3, 4, 6, 9, 11, 12, 14)

		2	1	2	1		8	4	2	1	8	4	2	1	
M	**N**	**M1**	**M0**	**N1**	**N0**	**P**	**P3**	**P2**	**P1**	**P0**	**A1**	**A0**	**B1**	**B0**	**DECP**
0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	6
0	1	0	0	0	1	0	0	0	0	0	0	1	1	1	7
0	2	0	0	1	0	0	0	0	0	0	0	1	0	0	4
0	3	0	0	1	1	0	0	0	0	0	0	1	0	1	5
1	0	0	1	0	0	0	0	0	0	0	0	0	1	0	2
1	1	0	1	0	1	1	0	0	0	1	0	0	1	1	3
1	2	0	1	1	0	2	0	0	1	0	0	0	0	0	0
1	3	0	1	1	1	3	0	0	1	1	0	0	0	1	1
2	0	1	0	0	0	0	0	0	0	0	1	1	1	0	14
2	1	1	0	0	1	2	0	0	1	0	1	1	1	1	15
2	2	1	0	1	0	4	0	1	0	0	1	1	0	0	12
2	3	1	0	1	1	6	0	1	1	0	1	1	0	1	13
3	0	1	1	0	0	0	0	0	0	0	1	0	1	0	10
3	1	1	1	0	1	3	0	0	1	1	1	0	1	1	11
3	2	1	1	1	0	6	0	1	1	0	1	0	0	0	8
3	3	1	1	1	1	9	1	0	0	1	1	0	0	1	9

P3 = F7(A1, A0, B1, B0) = ∑(9)
P2 = F6(A1, A0, B1, B0) = ∑(8, 12, 13)
P1 = F5(A1, A0, B1, B0) = ∑(0, 1, 8, 11, 13, 15)
P0 = F4(A1, A0, B1, B0) = ∑(1, 3, 9, 11)

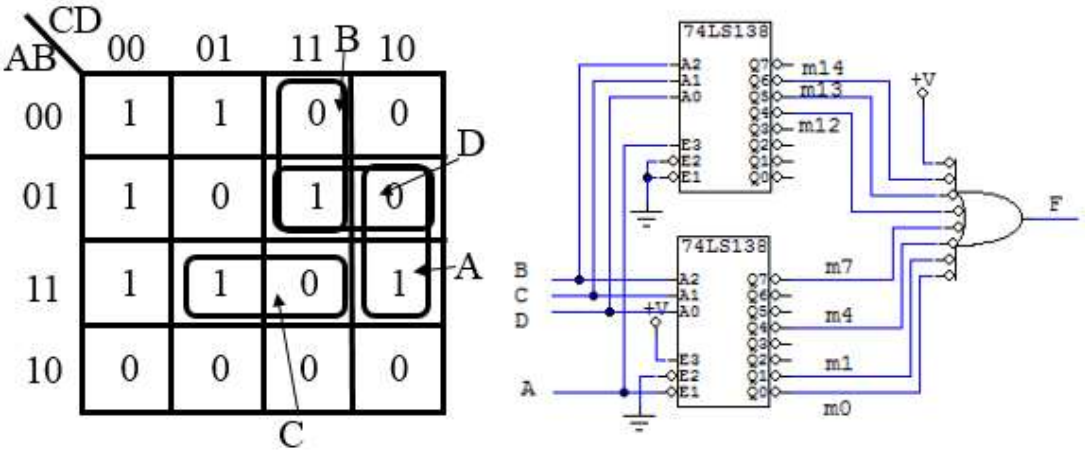
Implemento usando decodificadores y puertas NAND. Aunque, según el enunciado, puedo usar puertas NAND con cualquier número de entradas, en el diseño uso un inversor para P3, y puertas de 4 y 8 entradas conectando las entradas no usadas a 1 lógico.



Página 5_1. Implementar las siguientes funciones utilizando el menor número de decodificadores 3 a 8, y 2 a 4.

a) F(A,B,C,D) = ∑(0,1,4,7,12,13,14) con A.H, B.H, C.H y D.H.

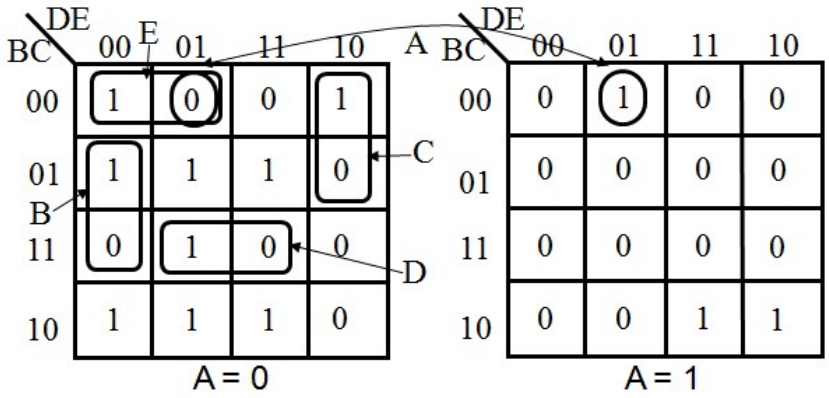
Primero compruebo si hay alguna entrada es redundante. Utilizo el método usado con los multiplexores (diapositivas 25 y 26 de teoría), basado en el mapa de Karnaugh (buscar un 1-cubo con valores 0-1 para cada entrada). No hay entradas redundantes, y como todas las entradas están en lógica positiva implemento directamente con los decodificadores 3 a 8 74'138 y una puerta NAND.



Otra opción posible realiza la implementación usando únicamente decodificadores 2 a 4 como en la diapositiva 16 de teoría. En el mapa se ve que hay 2-cubos de 0s (A $\overline{B}$, por ejemplo) y no hay 2-cubos de 1s. Conectando A a A3, B a A2, C a A1 y D a A0, habría que conectar las salidas del decodificador O0, O1, O4, O7, O12, O13 y O14 a las entradas de una puerta NAND como en la figura anterior. Como no se usan ninguna de las salidas $O_{8\text{-}11}$, el decodificador correspondiente es innecesario y se puede eliminar del diseño. Queda propuesto.

b) F(A,B,C,D,E) = ∑(0,2,4,5,7,8,9,10,13,17,26,27) con A.H, B.L, C.L, D.L y E.H.

Es un circuito de 5 entradas por lo que, en principio, se necesita un decodificador de 5 a 32. Lo primero es usar un mapa de Karnaugh para comprobar si alguna entrada es redundante. Los 1-cubos para la entrada A son la misma casilla de los dos mapas de 4 entradas.



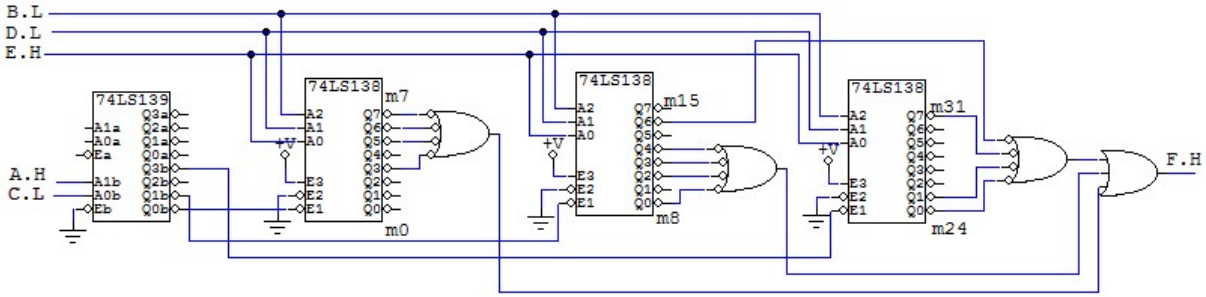
Del mapa de Karnaugh se observa que no hay entradas redundantes. Lo más lógico es hacer una implementación de los decodificadores con dos niveles. El primero es un decodificador 2 a 4 con dos bits de dirección (normalmente A y B), y el segundo cuatro decodificadores 3 a 8, los cuatro con los mismos tres bits de dirección (normalmente C, D y E).

Pero además se observa en el mapa que hay solo un 3-cubo de 0s con (AC) = (11). Eso significa que si reordeno las entradas como (AC) para el decodificador 2 a 4 y (BDE) para los decodificadores 3 a 8, puedo eliminar uno de los 4 decodificadores 3 a 8. Por último, las entradas B, C y D son .L, por lo que se complementan los 1s y 0s para saber los *minterms* que hay que utilizar. Por lo que uso:

F(A, C, B, D, E) = ∑(3, 5, 6, 7, 8, 10, 11, 12, 14, 24, 25, 31)

	16	8	4	2	1	16	8	4	2	1	
DEC	**A**	**B**	**C**	**D**	**E**	**A**	**C**	**B**	**D**	**E**	**DEC**
0	0	0	0	0	0	0	1	1	1	0	14
2	0	0	0	1	0	0	1	1	0	0	12
4	0	0	1	0	0	0	0	1	1	0	6
5	0	0	1	0	1	0	0	1	1	1	7
7	0	0	1	1	1	0	0	1	0	1	5
8	0	1	0	0	0	0	1	0	1	0	10
9	0	1	0	0	1	0	1	0	1	1	11
10	0	1	0	1	0	0	1	0	0	0	8
13	0	1	1	0	1	0	0	0	1	1	3
17	1	0	0	0	1	1	1	1	1	1	31
26	1	1	0	1	0	1	1	0	0	0	24
27	1	1	0	1	1	1	1	0	0	1	25

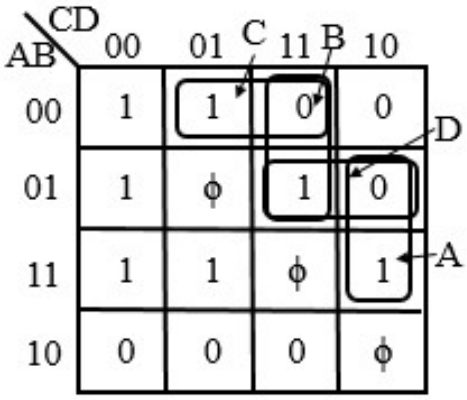
Implemento el circuito con los 3 circuitos 74LS138 (DEC 3 a 8) y un circuito 74LS139 (DEC 2 a 4, uso solo uno de los dos decodificadores). En la diapositiva 15 de teoría se muestra cómo se podría sustituir el DEC 2 a 4 por un único inversor. Hago la OR final con 3 NANDs de 4 entradas y una OR de 3 entradas.



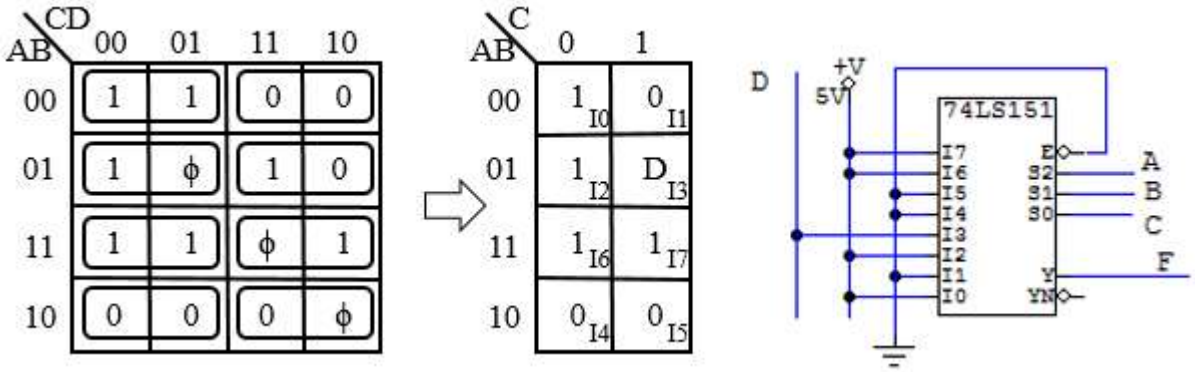
Página 5_1. Implementar las siguientes funciones lógicas utilizando un único multiplexor, lo más pequeño posible.

a) F(A,B,C,D) = $\sum(0,1,4,7,12,13,14) + \sum_{\phi}(5,10,15)$ con A.H, B.H, C.H y D.H, F.H.

Primero compruebo si hay alguna entrada redundante con el mapa de Karnaugh. Como para todas las entradas se puede encontrar por lo menos un 1-cubo con 0 y 1, entonces todas las entradas son necesarias, y el problema es de 4 entradas.



Voy a hacer una implementación de tipo 1, en la que elijo una entrada que aplico a las entradas de datos del multiplexor, y las otras tres entradas a las entradas de selección del multiplexor. Entonces se necesita un multiplexor de 8 entradas, y como máximo un inversor para las entradas de datos. Elijo la variable D aleatoriamente la introduzco en el mapa. Comparo en el mapa de Karnaugh el valor de la función con el valor de D en los márgenes y escribo 0, (dos 0s), 1 (dos 1s), D (valores iguales) o $\overline{D}$ (valores complementarios).

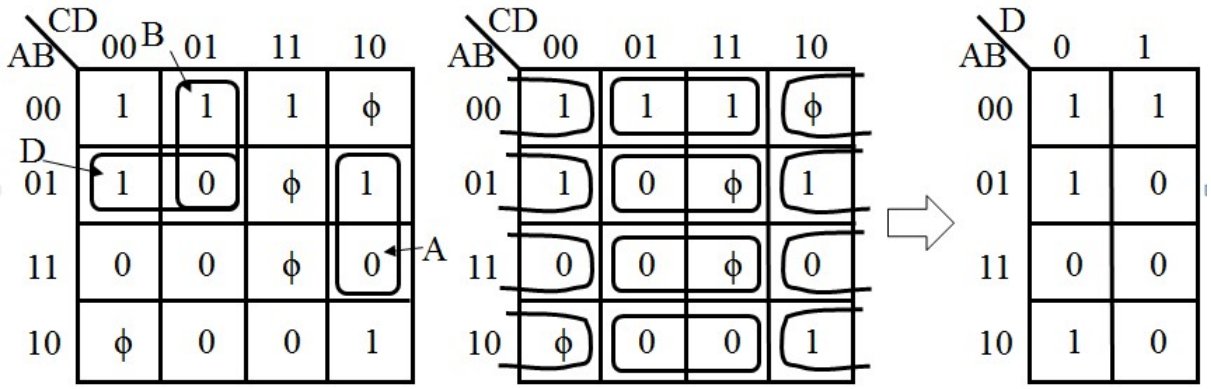


Dentro de cada casilla aparece la entrada de datos del multiplexor a la que se tiene que conectar el valor. Como todas las entradas están en polaridad positiva no hay que hacer más operaciones y genero el circuito.

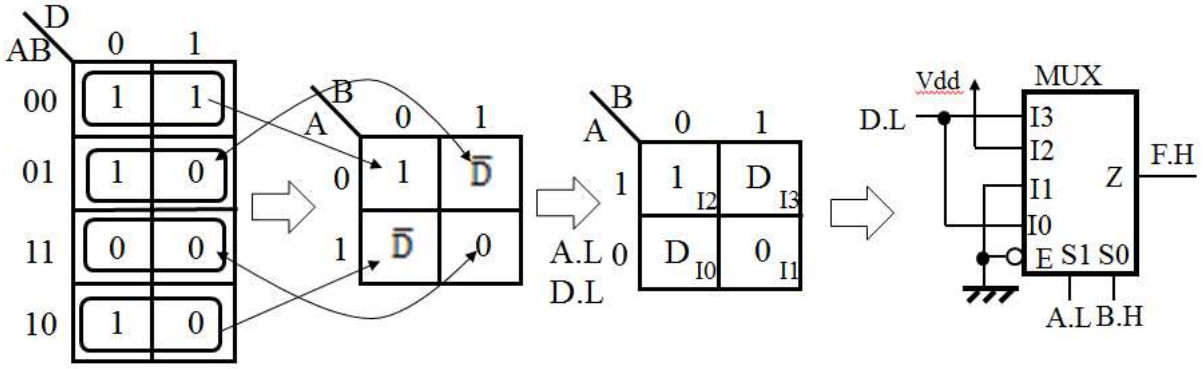
b) F(A,B,C,D) = ∑(0, 1, 3, 4, 6, 10) + ∑ϕ(2, 7, 8, 15) para A.L, B.H, C.H, D.L, F.H.

Repito el procedimiento para esta función. Compruebo si hay alguna entrada redundante. A, B y D no lo son, pero utilizando los "don't cares" y asignándole 0 o 1, según convenga, consigo que la función no dependa de C. Como ahora la función tiene tres entradas, y voy a aplicar una a las entradas de datos, hay que aplicar las otras dos a entradas de selección del multiplexor, por lo que necesito un multiplexor de 4 entradas.

F(A,B,C,D) = ∑(0, 1, 3, 4, 6, 10) + ∑ϕ(2, 7, 8, 15) = F1(A,C,D) = ∑(0, 1, 2, 4)

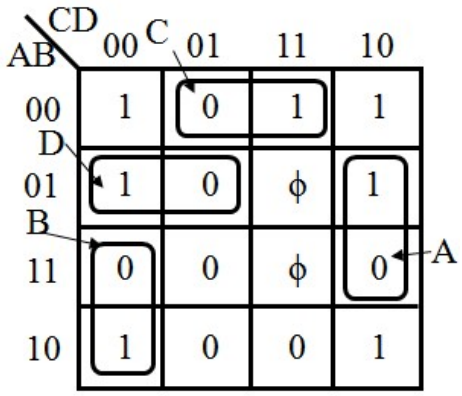


Introduzco una entrada en el mapa. Elijo D. Ajusto la polaridad de las entradas, para lo que complemento las entradas .L: A en los márgenes del mapa, y D en las casillas del mapa. Como en las entradas de selección sitúo A (S1) y B (S0), el valor binario de las entradas (AB) indica el índice de las entradas de datos que se corresponde con cada casilla del mapa. Implemento en un multiplexor de 4 entradas conectando 0 a GND, 1 a la tensión de alimentación, D a la variable de entrada y $\overline{D}$ al inversor de D.

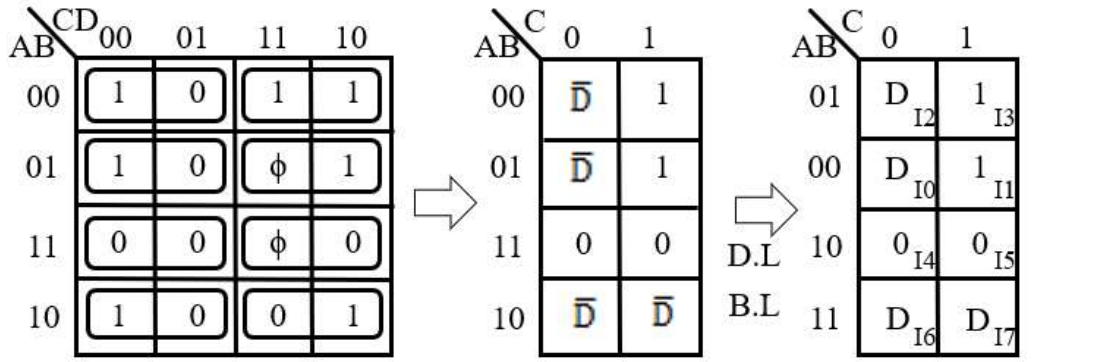


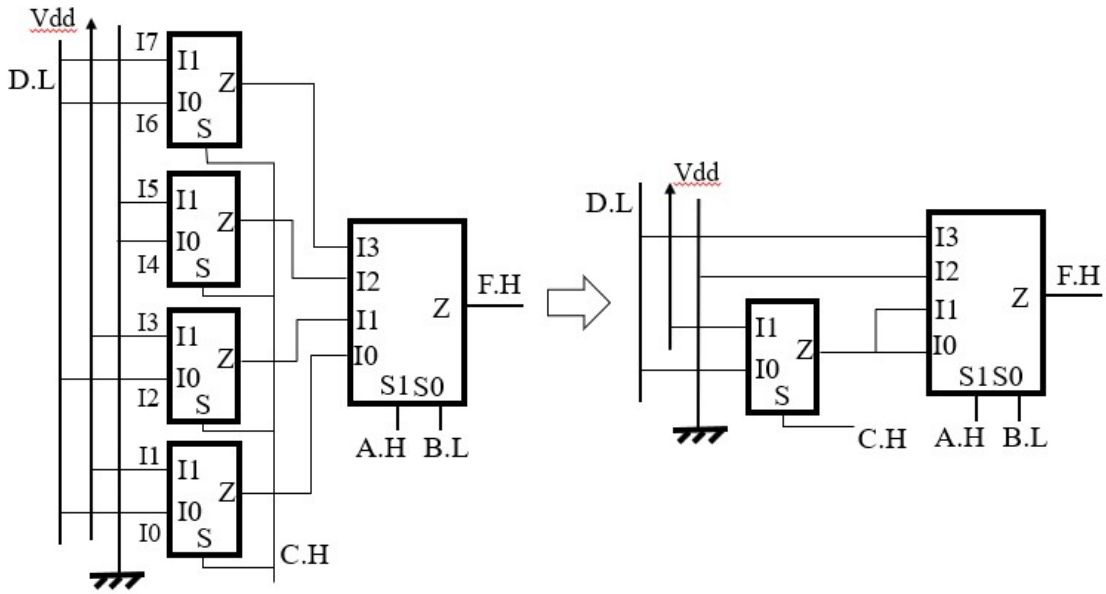
Página 5_3. Implementar utilizando un multiplexor de cuatro entradas y el menor número de multiplexores de dos entradas la función lógica:
F.H = F(A,B,C,D) = ∏(1,5,9,11,12,13,14) • ∏ϕ(7,15), para entradas A.H, B.L, C.H y D.L.

La función tiene cuatro entradas por lo que se necesita, en principio, un multiplexor de 8 entradas, que se puede construir con un multiplexor de 4 entradas y multiplexores de dos entradas, usando la configuración de la diapositiva 7 de teoría. Siguiendo los pasos compruebo si alguna entrada es redundante: no las hay.



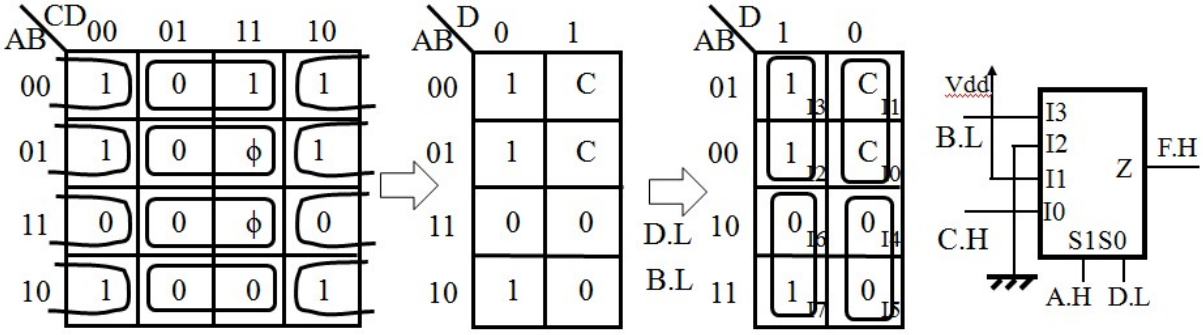
El objetivo es encontrar una solución con el menor número de multiplexores de dos entradas. Para conseguir la solución mínima hay que hacer pruebas sobre qué entradas son de selección del multiplexor de 4 entradas, el valor lógico asociado a los "don't cares", etc. La solución es que no se necesita ninguno. De momento sigo el orden convencional e intento minimizar el número de multiplexores: A y B son las entradas de selección del multiplexor de 4 entradas, C de los de 2 entradas y D en las entradas de datos. Meto D en el mapa y ajusto su polaridad.





El circuito de la izquierda puede reducirse ya que los dos multiplexores inferiores de dos entradas son iguales por lo que se puede eliminar uno, y los otros multiplexores tienen las dos entradas conectadas iguales (0 y D.L), luego puede sustituirse por el valor de las entradas (0 y D.L). Tres multiplexores se pueden eliminar, produciendo el circuito de la derecha.

Para conseguir hacer un circuito sin multiplexores de dos entradas, hay que meter C en el mapa, suponiendo que el *minterm* 7 es 1 y el 15 es 0. Además, las entradas de selección del multiplexor de 4 entradas son A y D, y la de los multiplexores de 2 entradas es B. En la figura, después de meter C en el mapa y ajustar la polaridad ordeno las entradas I0-I7 en función de ADB, cada par de casillas marcadas es un multiplexor: I0-I1 es C y no requiere multiplexor; igualmente I2-I3 son 1, e I4-I5 son 0. Por último, en I6-I7, si B es 0, F es 0, y si B es 1, F es 1, por lo que F es igual a B, y tampoco necesitan multiplexor.



Página 6_1. Obtener las expresiones lógicas minimizadas que permiten encontrar cuál de 7 líneas de entrada A1, A4, A6, A8, A9, A13, A14 está puesta a valor lógico 1, dando como resultado su correspondiente codificación binaria: por ejemplo, A8 daría como resultado 8 en la salida (codificado en binario).
a) Solo puede estar una línea a valor lógico 1.
b) Varias líneas de entrada están simultáneamente a 1, pero la salida tomará el valor binario de la línea de índice más bajo.

a) La descripción del circuito corresponde a un codificador sin prioridad, que se puede implementar con puertas OR. La salida máxima es 14, que se requiere 4 bits de salida Z3, Z2, Z1 y Z0.

A1	A4	A6	A8	A9	A13	A14	Z3	Z2	Z1	Z0
1	0	0	0	0	0	0	0	0	0	1
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	0	1
0	0	0	0	0	1	0	1	1	0	1
0	0	0	0	0	0	1	1	1	1	0

Z3 = A8 + A9 + A13 + A14
Z2 = A4 + A6 + A13 + A14
Z1 = A6 + A14
Z0 = A1 + A9 + A13

b) La descripción ahora corresponde a un codificador con prioridad baja. La tabla de este problema quedaría así:

Función	A1	A4	A6	A8	A9	A13	A14	Z3	Z2	Z1	Z0
A1	1	X	X	X	X	X	X	0	0	0	1
$\overline{A1}$ A4	0	1	X	X	X	X	X	0	1	0	0
$\overline{A1}$ $\overline{A4}$ A6	0	0	1	X	X	X	X	0	1	1	0
$\overline{A1}$ $\overline{A4}$ $\overline{A6}$ A8	0	0	0	1	X	X	X	1	0	0	0
$\overline{A1}$ $\overline{A4}$ $\overline{A6}$ $\overline{A8}$ A9	0	0	0	0	1	X	X	1	0	0	1
$\overline{A1}$ $\overline{A4}$ $\overline{A6}$ $\overline{A8}$ $\overline{A9}$ A13	0	0	0	0	0	1	X	1	1	0	1
$\overline{A1}$ $\overline{A4}$ $\overline{A6}$ $\overline{A8}$ $\overline{A9}$ $\overline{A13}$ A14	0	0	0	0	0	0	1	1	1	1	0
$\overline{A1}$ $\overline{A4}$ $\overline{A6}$ $\overline{A8}$ $\overline{A9}$ $\overline{A13}$ $\overline{A14}$	0	0	0	0	0	0	0	0	0	0	0

Z3 = $\overline{A1}$ $\overline{A4}$ $\overline{A6}$ A8 + $\overline{A1}$ $\overline{A4}$ $\overline{A6}$ $\overline{A8}$ A9 + $\overline{A1}$ $\overline{A4}$ $\overline{A6}$ $\overline{A8}$ $\overline{A9}$ A13 + $\overline{A1}$ $\overline{A4}$ $\overline{A6}$ $\overline{A8}$ $\overline{A9}$ $\overline{A13}$ A14
Z2 = $\overline{A1}$ A4 + $\overline{A1}$ $\overline{A4}$ A6 + $\overline{A1}$ $\overline{A4}$ $\overline{A6}$ $\overline{A8}$ $\overline{A9}$ A13 + $\overline{A1}$ $\overline{A4}$ $\overline{A6}$ $\overline{A8}$ $\overline{A9}$ $\overline{A13}$ A14
Z1 = $\overline{A1}$ $\overline{A4}$ A6 + $\overline{A1}$ $\overline{A4}$ $\overline{A6}$ $\overline{A8}$ $\overline{A9}$ $\overline{A13}$ A14
Z0 = A1 + $\overline{A1}$ $\overline{A4}$ $\overline{A6}$ $\overline{A8}$ A9 + $\overline{A1}$ $\overline{A4}$ $\overline{A6}$ $\overline{A8}$ $\overline{A9}$ A13

Las ecuaciones se pueden reducir con la propiedad distributiva y el teorema de simplificación ($X + \overline{X}\,Y = X + Y$).

Z3 = $\overline{A1}$ $\overline{A4}$ $\overline{A6}$ (A8 + $\overline{A8}$ A9 + $\overline{A8}$ $\overline{A9}$ A13 + $\overline{A8}$ $\overline{A9}$ $\overline{A13}$ A14) =
= $\overline{A1}$ $\overline{A4}$ $\overline{A6}$ (A8 + A9 + $\overline{A9}$ A13 + $\overline{A9}$ $\overline{A13}$ A14) =
= $\overline{A1}$ $\overline{A4}$ $\overline{A6}$ (A8 + A9 + A13 + $\overline{A13}$ A14) = $\overline{A1}$ $\overline{A4}$ $\overline{A6}$ (A8 + A9 + A13 + A14)
Z2 = $\overline{A1}$ (A4 + $\overline{A4}$ A6 + $\overline{A4}$ $\overline{A6}$ $\overline{A8}$ $\overline{A9}$ A13 + $\overline{A4}$ $\overline{A6}$ $\overline{A8}$ $\overline{A9}$ $\overline{A13}$ A14) =
= $\overline{A1}$ (A4 + A6 + $\overline{A6}$ $\overline{A8}$ $\overline{A9}$ A13 + $\overline{A6}$ $\overline{A8}$ $\overline{A9}$ $\overline{A13}$ A14) =
= $\overline{A1}$ (A4 + A6 + $\overline{A8}$ $\overline{A9}$ A13 + $\overline{A8}$ $\overline{A9}$ $\overline{A13}$ A14) =
= $\overline{A1}$ [A4 + A6 + $\overline{A8}$ $\overline{A9}$ (A13 + $\overline{A13}$ A14)] = $\overline{A1}$ [A4 + A6 + $\overline{A8}$ $\overline{A9}$ (A13 + A14)]
Z1 = $\overline{A1}$ $\overline{A4}$ (A6 + $\overline{A6}$ $\overline{A8}$ $\overline{A9}$ $\overline{A13}$ A14) = $\overline{A1}$ $\overline{A4}$ (A6 + $\overline{A8}$ $\overline{A9}$ $\overline{A13}$ A14)
Z0 = A1 + $\overline{A4}$ $\overline{A6}$ $\overline{A8}$ A9 + $\overline{A4}$ $\overline{A6}$ $\overline{A8}$ $\overline{A9}$ A13 = A1 + $\overline{A4}$ $\overline{A6}$ $\overline{A8}$ (A9 + $\overline{A9}$ A13) =
= A1 + $\overline{A4}$ $\overline{A6}$ $\overline{A8}$ (A9 + A13)

Página 6_2. Encontrar las ecuaciones lógicas que permiten definir un circuito codificador con prioridad baja de 8 bits de entrada (I7-I0) y salidas en código Gray (de más a menos significativas: A B C).

Para el problema del enunciado, su tabla de verdad queda como se muestra. Los valores de las salidas cuando ninguna entrada está activa (0s) podrían definirse de otra manera.

Función	I0	I1	I2	I3	I4	I5	I6	I7	A	B	C
I0	1	X	X	X	X	X	X	X	0	0	0
$\overline{I0}$ I1	0	1	X	X	X	X	X	X	0	0	1
$\overline{I0}$ $\overline{I1}$ I2	0	0	1	X	X	X	X	X	0	1	1
$\overline{I0}$ $\overline{I1}$ $\overline{I2}$ I3	0	0	0	1	X	X	X	X	0	1	0
$\overline{I0}$ $\overline{I1}$ $\overline{I2}$ $\overline{I3}$ I4	0	0	0	0	1	X	X	X	1	1	0
$\overline{I0}$ $\overline{I1}$ $\overline{I2}$ $\overline{I3}$ $\overline{I4}$ I5	0	0	0	0	0	1	X	X	1	1	1
$\overline{I0}$ $\overline{I1}$ $\overline{I2}$ $\overline{I3}$ $\overline{I4}$ $\overline{I5}$ I6	0	0	0	0	0	0	1	X	1	0	1
$\overline{I0}$ $\overline{I1}$ $\overline{I2}$ $\overline{I3}$ $\overline{I4}$ $\overline{I5}$ $\overline{I6}$ I7	0	0	0	0	0	0	0	1	1	0	0
$\overline{I0}$ $\overline{I1}$ $\overline{I2}$ $\overline{I3}$ $\overline{I4}$ $\overline{I5}$ $\overline{I6}$ $\overline{I7}$	0	0	0	0	0	0	0	0	0	0	0

A = $\overline{I0}$ $\overline{I1}$ $\overline{I2}$ $\overline{I3}$ I4 + $\overline{I0}$ $\overline{I1}$ $\overline{I2}$ $\overline{I3}$ $\overline{I4}$ I5 + $\overline{I0}$ $\overline{I1}$ $\overline{I2}$ $\overline{I3}$ $\overline{I4}$ $\overline{I5}$ I6 + $\overline{I0}$ $\overline{I1}$ $\overline{I2}$ $\overline{I3}$ $\overline{I4}$ $\overline{I5}$ $\overline{I6}$ I7
B = $\overline{I0}$ $\overline{I1}$ I2 + $\overline{I0}$ $\overline{I1}$ $\overline{I2}$ I3 + $\overline{I0}$ $\overline{I1}$ $\overline{I2}$ $\overline{I3}$ I4 + $\overline{I0}$ $\overline{I1}$ $\overline{I2}$ $\overline{I3}$ $\overline{I4}$ I5
C = $\overline{I0}$ I1 + $\overline{I0}$ $\overline{I1}$ I2 + $\overline{I0}$ $\overline{I1}$ $\overline{I2}$ $\overline{I3}$ $\overline{I4}$ I5 + $\overline{I0}$ $\overline{I1}$ $\overline{I2}$ $\overline{I3}$ $\overline{I4}$ $\overline{I5}$ I6

Al igual que en el problema anterior uso la propiedad distributiva y el teorema de simplificación para reducir las expresiones lógicas:

$A = \overline{I0}\ \overline{I1}\ \overline{I2}\ \overline{I3}\ \left(I4 + \overline{I4}\ I5 + \overline{I4}\ \overline{I5}\ I6 + \overline{I4}\ \overline{I5}\ \overline{I6}\ I7\right) =$
$= \overline{I0}\ \overline{I1}\ \overline{I2}\ \overline{I3}\ \left(I4 + I5 + \overline{I5}\ I6 + \overline{I5}\ \overline{I6}\ I7\right) = \overline{I0}\ \overline{I1}\ \overline{I2}\ \overline{I3}\ (I4 + I5 + I6 + \overline{I6}\ I7) =$
$= \overline{I0}\ \overline{I1}\ \overline{I2}\ \overline{I3}\ (I4 + I5 + I6 + I7)$

$B = \overline{I0}\ \overline{I1}\ (I2 + \overline{I2}\ I3 + \overline{I2}\ \overline{I3}\ I4 + \overline{I2}\ \overline{I3}\ \overline{I4}\ I5) = \overline{I0}\ \overline{I1}\ (I2 + I3 + \overline{I3}\ I4 + \overline{I3}\ \overline{I4}\ I5) =$
$= \overline{I0}\ \overline{I1}\ (I2 + I3 + I4 + \overline{I4}\ I5) = \overline{I0}\ \overline{I1}\ (I2 + I3 + I4 + I5)$

$C = \overline{I0}\ \left(I1 + \overline{I1}\ I2 + \overline{I1}\ \overline{I2}\ \overline{I3}\ \overline{I4}\ I5 + \overline{I1}\ \overline{I2}\ \overline{I3}\ \overline{I4}\ \overline{I5}\ I6\right) =$
$= \overline{I0}\ \left(I1 + I2 + \overline{I2}\ \overline{I3}\ \overline{I4}\ I5 + \overline{I2}\ \overline{I3}\ \overline{I4}\ \overline{I5}\ I6\right) = \overline{I0}\ \left[I1 + I2 + \overline{I3}\ \overline{I4}\ I5 + \overline{I3}\ \overline{I4}\ \overline{I5}\ I6\right] =$
$= \overline{I0}\ \left[I1 + I2 + \overline{I3}\ \overline{I4}\ \left(I5 + \overline{I5}\ I6\right)\right] = \overline{I0}\ [I1 + I2 + \overline{I3}\ \overline{I4}\ (I5 + I6)]$

Página 6_3. Construir un circuito codificador binario de 8 a 3 con prioridad baja tomando como base el circuito codificador 74LS148, y el menor número posible de puertas lógicas que sean necesarias. Se permite definir como mejor convenga la polaridad de las entradas y de las salidas.

La tabla del funcionamiento del circuito 74LS148 aparece en la diapositiva 34 de teoría. Por defecto las entradas y salidas están en lógica negativa, y la prioridad de las entradas es alta: I7 a 1, genera L L L en las salidas A2 A1 A0, que es $(111)_2$ en lógica negativa, o $(7)_{10}$.

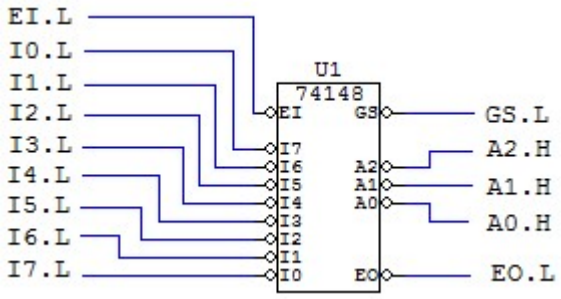
FUNCTION TABLE

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

Si en esta tabla se cambia como se mira la polaridad de las salidas A2, A1 y A0, que pasan a estar en lógica positiva, entonces en la tercera línea se genera la salida (A2 A1 A0) = $(000)_2$, por lo que la línea que fija el valor de las salidas tiene que ser I0. Esto se mantiene en todas las filas de la tabla: si se fijan las salidas $(001)_2$ la entrada prioritaria es I1. Esto significa que las entradas están en prioridad baja conectándolas en orden inverso.; la entrada 7 es I0, la 6 es I1, la 5 es I2, …, la 1 es I6 y la 0 es I7. La tabla lógica del circuito 74'148 sería esta:

E1	**I7**	**I6**	**I5**	**I4**	**I3**	**I2**	**I1**	**I0**	**A2**	**A1**	**A0**	**GS**	**E0**
0	X	X	X	X	X	X	X	X	1	1	1	0	0
1	0	0	0	0	0	0	0	0	1	1	1	0	1
1	X	X	X	X	X	X	X	1	0	0	0	1	0
1	X	X	X	X	X	X	1	0	0	0	1	1	0
1	X	X	X	X	X	1	0	0	0	1	1	1	0
1	X	X	X	X	1	0	0	0	0	1	0	1	0
1	X	X	X	1	0	0	0	0	1	1	0	1	0
1	X	X	1	0	0	0	0	0	1	1	1	1	0
1	X	1	0	0	0	0	0	0	1	0	1	1	0
1	1	0	0	0	0	0	0	0	1	0	0	1	0
1	0	0	0	0	0	0	0	0	0	0	0	1	0

El circuito resultante es este. No hay que tener en cuenta las burbujas de inversión del esquemático, solo la definición .H o .L de las líneas.

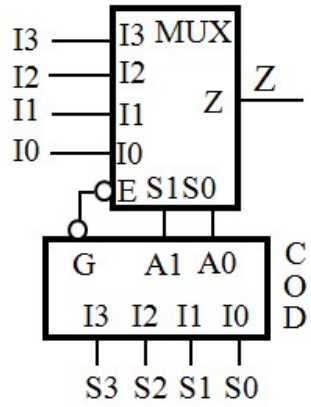


Página 7_1. Diseñar un circuito multiplexor con prioridad de 4 bits. El circuito tiene 4 entradas de datos (I3-I0), 4 entradas de selección (S3-S0) y dos salidas Z y G. Cuando una o más de las entradas S están a 1, Z toma el valor de la entrada Ii, siendo i es el índice más alto de las entradas Si que están a 1; si todas las entradas S3-S0 están a 0, entonces Z toma el valor 0. La salida G se fija a 1 si al menos alguna entrada Si está a 1, en caso contrario se fija a 0. Utilizar en el diseño circuitos MSI convencionales: un 74LS148 (8 a 3 HPRI COD) y un circuito 74LS153 (4-input MUX).

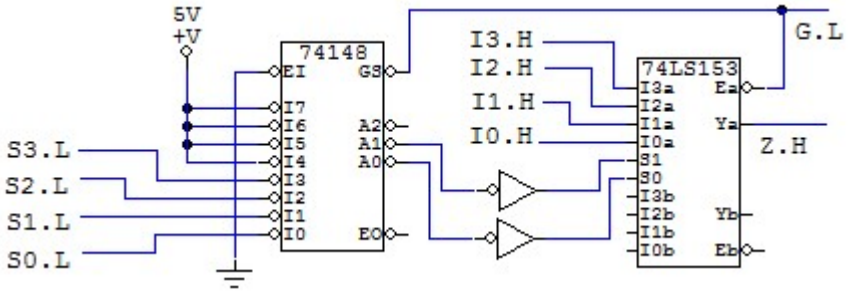
El enunciado es el mismo que el de la página 1. La tabla de ese problema se puede modificar de forma que defina un sistema basado en un codificador con prioridad y un multiplexor, tal que el codificador con prioridad lee las entradas S y genera el índice en binario, del S más alto a 1 en dos salidas A1A0. A1 y A0 son las entradas de un multiplexor convencional de cuatro entradas que pasan la entrada I a la salida Z.

S3	S2	S1	S0	A1	A0	Z	G
1	X	X	X	1	1	I3	1
0	1	X	X	1	0	I2	1
0	0	1	X	0	1	I1	1
0	0	0	1	0	0	I0	1
0	0	0	0	0	0	0	0

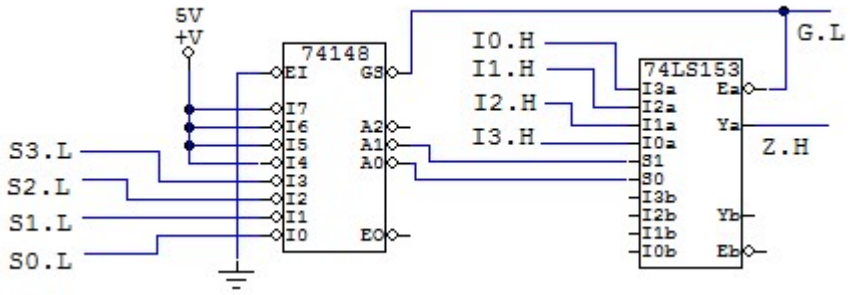
La implementación inicial necesita un codificador con prioridad alta 4 a 2 de entradas S3-S0 conectadas a sus entradas I3-I0; las salidas del codificador (A1A0) se conectan a las entradas de selección (S1S0) de un multiplexor de 4 entradas (I3-I0), y la salida G del codificador a la entrada de habilitación, para dejar la salida del multiplexor a 0 cuando no hay ninguna entrada S a 1.



La implementación se hace con circuitos de la familia 74: codificador con prioridad alta 8 a 3 74LS148 (diapositiva 34 de teoría) y multiplexor de cuatro entradas 74LS153 (diapositiva 5 de teoría). Como el codificador es 8 a 3, le convierto en 4 a 2 fijando las entradas 7-4 a 0, que es tensión alta H, ya que las entradas están en lógica negativa. Igualmente, las entradas S3-S0, tienen que definirse en lógica negativa. El circuito 74'153 tiene dos multiplexores de 4 entradas, de los que solo necesito uno. La salida GS del codificador se conecta a la entrada de habilitación del multiplexor: si todas las entradas del codificador están a 0 (H), GS está a 0 (H), el *enable* del multiplexor está a 0 (H), el multiplexor se deshabilita y su salida se fija a 0 (L). Al conectar las salidas del codificador (.L) a las entradas de selección del multiplexor (.H) hay un problema, ya que la polaridad de las señales es distinta. La solución directa para solventar este problema es usar inversores como en la figura.



Pero existe otra solución mejor, que evita los inversores, realizando las conexiones entre (A1A0) y (S1S0) directamente. Basta con modificar donde se colocan las entradas de datos del multiplexor. Por ejemplo, si la entrada S3 es 1 (L, Z = I3), el codificador fija A1A0 a 11 (LL), y las entradas del multiplexor (S1S0) leen 00 (LL), con lo que su entrada I0 pasa a Z; entonces la entrada I3 del multiplexor con prioridad debe situarse en la entrada I0 del multiplexor. Lo mismo puede comprobarse con el resto de valores en las entradas S3, S2, S1 y S0. El circuito queda:



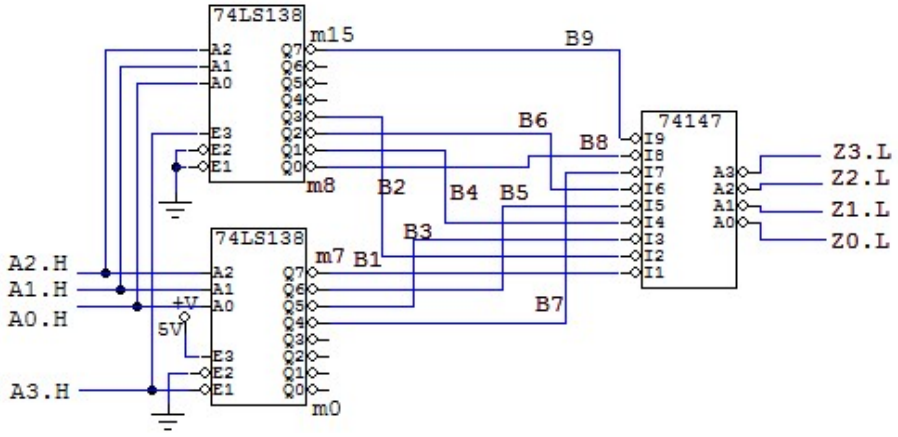
Página 7_2. Realizar un circuito conversor del código BCD con pesos (8, 7, -2, -4) al código NBCD (8, 4, 2, 1) usando únicamente circuitos 74'138 (decodificador 3 a 8) y 74'147 (codificador con prioridad alta 10 a 4).

El circuito tiene una entrada A de 4 bits (A3 A2 A1 A0) que codifica el código BCD de entrada con pesos (8, 7, -2, -4)**,** que ya ha aparecido en problema de otros temas, y una salida Z de cuatro bits (Z3 Z2 Z1 Z0) que codifica el código NBCD. Este problema se podría resolver con mapas de Karnaugh, pero se va a realizar con los circuitos indicados en el enunciado. Los circuitos 74'138 y 74'147 aparecen en las diapositivas 14 y 31 de teoría, respectivamente. El método para generar el circuito es usar un sistema decodificador/codificador. El decodificador convierte A en un valor intermedio B de 10 bits, y el codificador codifica los 10 bits de B en cuatro bits de Z.

	8	7	-2	-4												8	4	2	1
D	**A3**	**A2**	**A1**	**A0**	**O**	**B9**	**B8**	**B7**	**B6**	**B5**	**B4**	**B3**	**B2**	**B1**	**B0**	**A3**	**A2**	**A1**	**A0**
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
1	0	1	1	1	7	0	0	0	0	0	0	0	0	1	0	0	0	0	1
2	1	0	1	1	11	0	0	0	0	0	0	0	1	0	0	0	0	1	0
3	0	1	0	1	5	0	0	0	0	0	0	1	0	0	0	0	0	1	1
4	1	0	0	1	9	0	0	0	0	0	1	0	0	0	0	0	1	0	0
5	0	1	1	0	6	0	0	0	0	1	0	0	0	0	0	0	1	0	1
6	1	0	1	0	10	0	0	0	1	0	0	0	0	0	0	0	1	1	0
7	0	1	0	0	4	0	0	1	0	0	0	0	0	0	0	0	1	1	1
8	1	0	0	0	8	0	1	0	0	0	0	0	0	0	0	1	0	0	0
9	1	1	1	1	15	1	0	0	0	0	0	0	0	0	0	1	0	0	1

Con los circuitos 74'138 se puede formar un decodificador 4 a 16, seleccionando solamente 10 salidas. Cada combinación del código fija a 1 una salida del decodificador, que es uno de los bits B intermedios. Por ejemplo, el dígito 1 se codifica en A por 0111, que activa la salida m7 del decodificador y corresponde a la señal B1; el dígito 6 se codifica en A por 1010, que activa la salida m10 del decodificador y la señal B6. Las entradas A están en lógica positiva.

El convertidor 74'147 genera las salidas directamente en código NBCD. El circuito es un codificador con prioridad, aunque en este caso no es necesaria, ya que solo hay una entrada B a 1 cada vez. Se podría haber hecho con un codificador sin prioridad, hecho con puertas OR. Cada Bi se aplica a la entrada Ii del codificador, con i entre 1 y 9, lo que genera en Z el dígito D en NBCD. Si ninguna de las entradas B9–B1 están a 1, significa que B0 está a 1 y la salida es 0000 (dígito 0, NBCD). Las salidas Z están en lógica negativa. Para pasarlo a lógica positiva habría que usar inversores. El circuito queda así:



Página 7_3. Se quiere realizar un circuito de 8 entradas (I7-I0) y 8 salidas (O7-O0), tal que la salida muestra la entrada, pero eliminando todos los unos menos el más significativo. Por ejemplo, si I = "01101101", O = "01000000"; si I = "00010110", O = "00010000", etc. Si todos los bits de la entrada son 0, los de la salida también: I = "00000000", O = "00000000".

a) Realizar un código VHDL para la descripción del problema.

b) Implementar el circuito con un circuito codificador 8 a 3 74LS148 y un circuito decodificador 3 a 8 74LS138. Suponer las entradas y salidas I7.L, … I0.L; O7.L, … O0.L en polaridad negativa.

a) El código VHDL es similar al del problema 1. Se basa en una sentencia del tipo *if-elsif-else*, mirando los bits del más significativo al menos significativo, y cargando los valores de la salida bajo cada condición. Se puede hacer también una descripción genérica basada en un lazo *for*: se inicializan todos los bits de la salida a O a 0 con *others*, y cuando, en un lazo descendente, se encuentra el primer bit i de la entrada a 1, el bit i de la salida se fija a 1 y se abandona el lazo con la sentencia *exit*, por lo que el resto de los bits quedan a 0.

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

entity problema7_3 is
port (I: in std_logic_vector(7 downto 0);
      O: out std_logic_vector(7 downto 0));
end problema7_3;

architecture uno of problema7_3 is
begin
process(I)
begin
if ( I(7) = '1') then O <= "10000000";
elsif ( I(6) = '1') then O <= "01000000";
elsif ( I(5) = '1') then O <= "00100000";
elsif ( I(4) = '1') then O <= "00010000";
elsif ( I(3) = '1') then O <= "00001000";
elsif ( I(2) = '1') then O <= "00000100";
elsif ( I(1) = '1') then O <= "00000010";
elsif ( I(0) = '1') then O <= "00000001";
else O <= "00000000";
end if;
end process;
end uno;
```

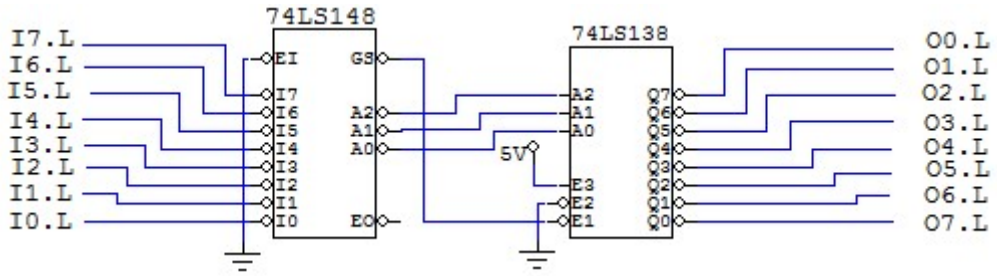
```
library ieee;
use ieee.std_logic_1164.all;

entity problema7_3gen is
generic(N: integer := 8);
port (I: in std_logic_vector(N-1 downto 0);
      O: out std_logic_vector(N-1 downto 0));
end problema7_3gen;

architecture uno of problema7_3gen is
begin
process(I)
begin
O <= (others => '0');
for j in N-1 downto 0 loop
   if ( I(j) = '1' ) then
      O(j) <= '1';
      exit;
   end if;
end loop;
end process;
end uno;
```

b) El funcionamiento del circuito para 8 bits puede verse en una tabla de entradas y salidas, donde defino unas señales intermedias A de tres bits A2, A1 y A0, que codifican en binario el índice más alto de las entradas fijadas a 1.

I 7	**I 6**	**I 5**	**I 4**	**I 3**	**I 2**	**I 1**	**I 0**	**A 2**	**A 1**	**A 0**	**O 7**	**O 6**	**O 5**	**O 4**	**O 3**	**O 2**	**O 1**	**O 0**
1	X	X	X	X	X	X	X	1	1	1	1	0	0	0	0	0	0	0
0	1	X	X	X	X	X	X	1	1	0	0	1	0	0	0	0	0	0
0	0	1	X	X	X	X	X	1	0	1	0	0	1	0	0	0	0	0
0	0	0	1	X	X	X	X	1	0	0	0	0	0	1	0	0	0	0
0	0	0	0	1	X	X	X	0	1	1	0	0	0	0	1	0	0	0
0	0	0	0	0	1	X	X	0	1	0	0	0	0	0	0	1	0	0
0	0	0	0	0	0	1	X	0	0	1	0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

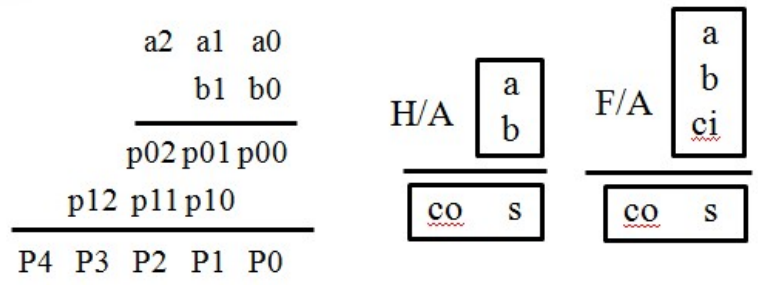
Al estudiar las señales se puede ver que las señales A son la respuesta de un codificador con prioridad alta de 8 a 3. A su vez las salidas O son la respuesta de un decodificador de 3 a 8 con entradas A. Para conseguir que cuando todas las entradas I estén a 0 se puede usar la salida GS de un codificador y conectarlo a la entrada de habilitación del decodificador. Ahora ya se puede construir el circuito con el codificador 74LS148 (diapositiva 34 de teoría) y el decodificador 74LS138 (diapositiva 14 de teoría). El problema al conectar los circuitos es que las salidas del codificador están en lógica negativa, mientras que las entradas de los decodificadores están en lógica positiva. Al igual que en el problema 7_1 hay dos formas de ajustar el circuito, la primera

consiste en situar inversores entre las salidas del codificador y las entradas del decodificador. La segunda consiste en "mover" la posición de las salidas: si el codificador genere el índice 7 (111) en lógica negativa es voltaje bajo (LLL); el decodificador lee (LLL) pero en lógica positiva eso es (000), luego fija a 1 la salida O0. Esto implica que en la salida O0 del codificador está realmente la salida O7 del problema. Si se hace con todas las salidas, complementando los bits de A, se pasa de 0 ⇔ 7, 1 ⇔ 6, 2 ⇔ 5 y 3 ⇔ 4. Con todo esto el circuito, para entradas y salidas .L queda así:

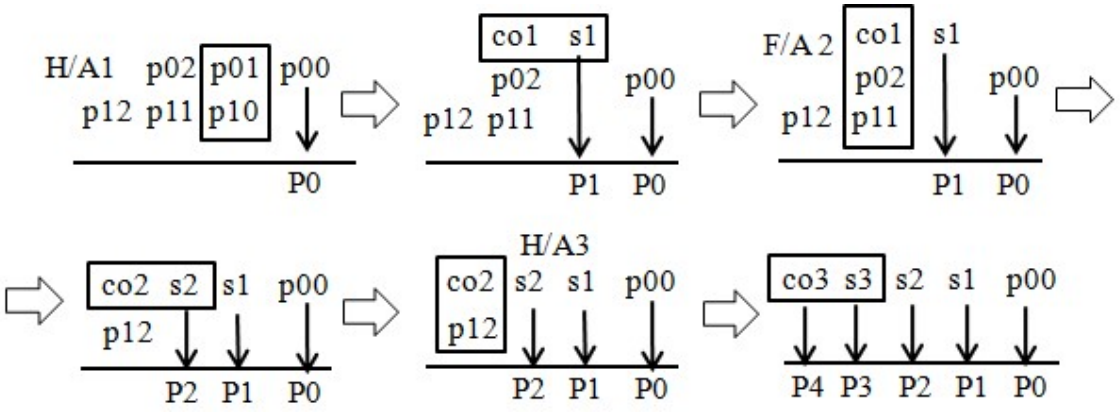


Página 8_1. Diseñar utilizando únicamente semisumadores y sumadores completos un circuito digital que realice la multiplicación de un número binario de dos bits por otro de tres bits.

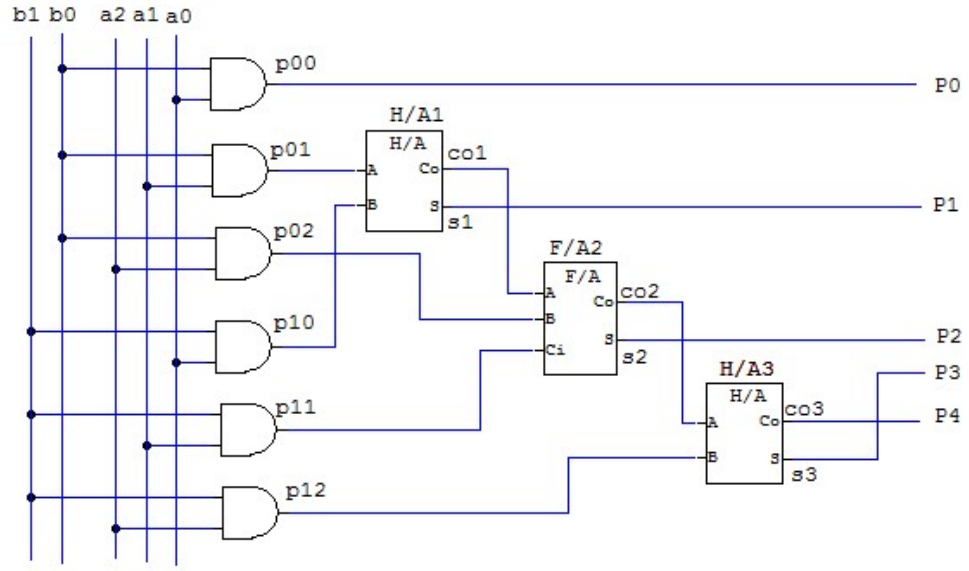
Se dispone de un multiplicando A de tres bits (a2 a1 a0, valores de 0 a 7) y un multiplicador B de dos bits (b1 b0, valores de 0 a 3). El resultado P debe ser de 5 bits (P4 P3 P2 P1 P0, valor máximo 7 * 3 = 21). Planteo el algoritmo de suma basado en productos parciales aritméticos de 1 bit, pij = bi * aj, que ya se sabe que puede hacerse con una puerta AND. En total hay 6 puertas AND. Y esos términos se suman con sumadores completos (F/A) o semisumadores (H/A) de 1 bit.



Para realizar la suma se pueden seguir estos pasos, donde se indican los sumadores completos y los semisumadores que se utilizan:

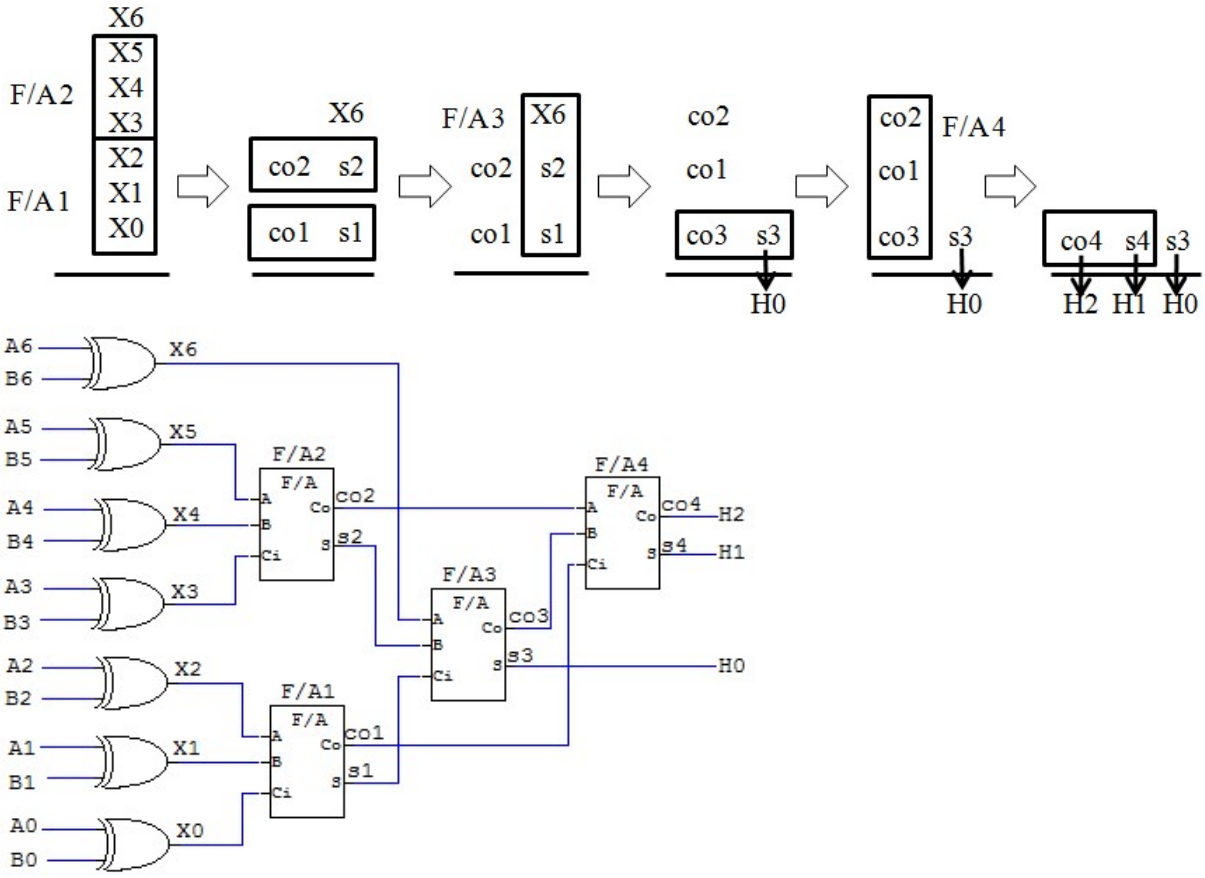


El circuito asociado, indicando las AND y los sumadores queda:



Página 8_2. Diseñar un circuito que calcule la distancia de Hamming de dos palabras A y B de 7 bits usando el menor número posible de puertas lógicas y/o de dispositivos MSI.

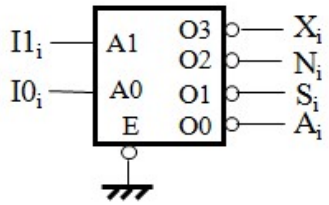
La distancia de Hamming de dos palabras de N bits es el número de posiciones en las que los valores de los bits son distintos. Por tanto, la distancia de Hamming puede tomar valores entre 0 (palabras iguales) y N (todos los bits distintos). En este problema N es 7 (A6-A0 y B6-B0), por lo que la distancia de Hamming se puede codificar en 3 bits (H2 H1 H0).



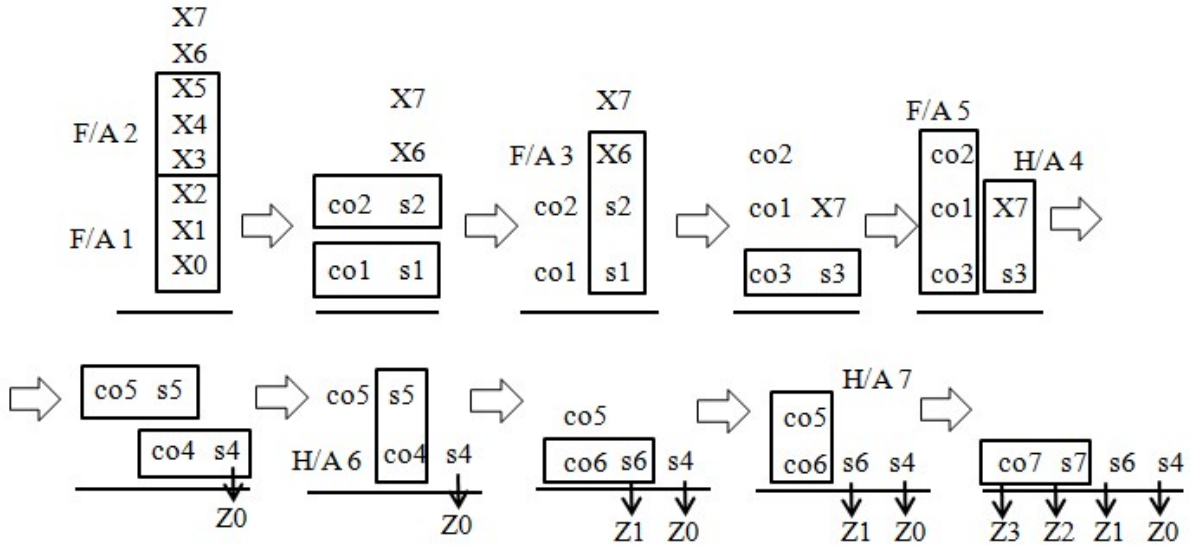
Para realizar el circuito puedo empezar por comprobar si para cada posición i los bits de Ai y Bi son distintos. Ya ha aparecido varias veces esta condición y que Xi = (Ai ≠ Bi) = Ai ⊕ Bi, por lo que el circuito necesita 7 puertas XOR. Para saber el número en binario de posiciones distintas hay que "contar" el número de 1s en las Xi, y la forma de contar es sumar todos los bits usando "full-adders" o "half-adders". Hay que intentar sumar primero de los bits menos significativos a los más significativos, obteniendo un único bit en cada posición que será el bit del resultado. La suma de 7 bits del mismo peso se puede hacer con 4 "full-adders", según el esquema de sumas y el circuito de la figura anterior.

Página 8_3. Diseñar utilizando elementos MSIs (sumadores y decodificadores) un circuito que calcule el resultado de una votación de siete votos. Cada voto aparece codificado mediante dos bits I1I0, de forma que la abstención se representa por 00, 'Si' por 01, 'No' por 10 y los votos nulos aparecen como 11. El resultado de la votación debe darse indicando el número de votos de cada tipo. Realizar lo mismo para una votación de ocho votos.

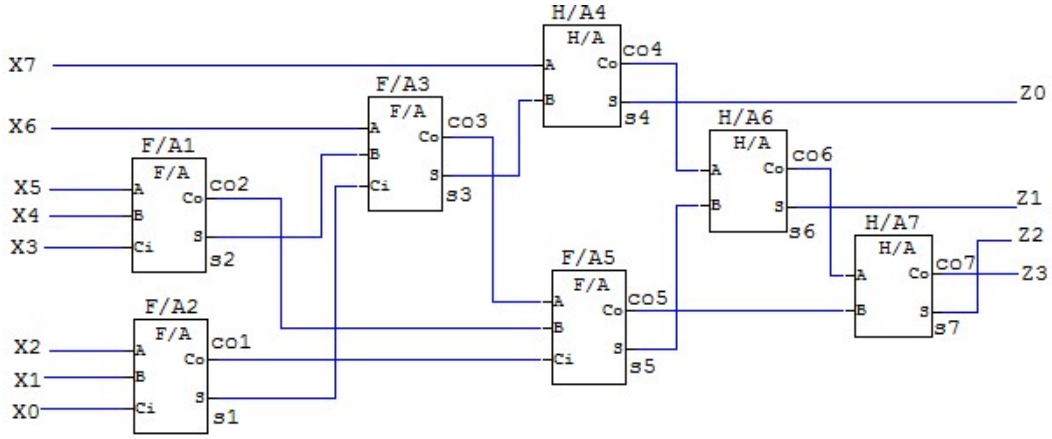
Este problema es muy parecido al problema anterior. Hay siete votantes que tienen cuatro opciones de voto. Para cada votante V_i se usa un decodificador, que tiene como entradas las señales $I1_iI0_i$ que genera en su salida O0 la señal A_i, (Abstención), en su salida O1 la señal S_i (Si), en su salida O2 la señal N_i (No), y en su salida O3 la señal X_i (Nulo).



El número de votos de cada tipo está entre 0 y 7, por lo que requiere tres bits de salida por tipo: Va (abstenciones, Va2 Va1 Va0), Vs (síes, Vs2 Vs1 Vs0), Vn (noes, Vn2 Vn1 Vn0) y Vx (nulos, Vx2 Vx1 Vx0). Para pasar de las siete Ai a las tres Va hay que usar el circuito de suma del problema anterior formado por 4 "full-adders". Para cada uno de los otros tipos de salida (S_i a Vs, N_i a Vn y X_i a Vx) también se requiere otro circuito de suma de 7 bits. El circuito total necesita 7 decodificadores y 16 "full-adders" (cuatro circuitos de suma de 7 bits).



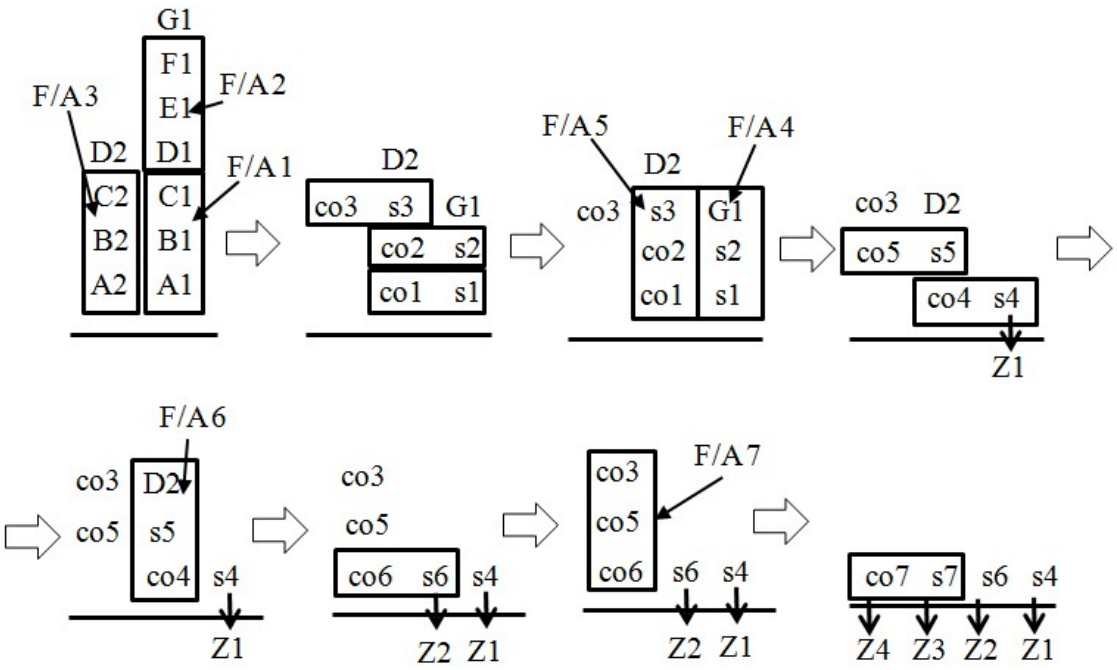
Si la votación fuese de 8 votantes, el esquema es similar al de 7 votantes, pero el valor de las salidas debe estar entre 0 y 8, por lo que la salida Z necesita cuatro bits (Z3 Z2 Z1 Z0) por tipo de voto, y el circuito de suma es algo más complicado. El esquema anterior y el siguiente circuito corresponden al diseño de un sumador de 8 bits (X7-X0) con "full-adders" o "half-adders".



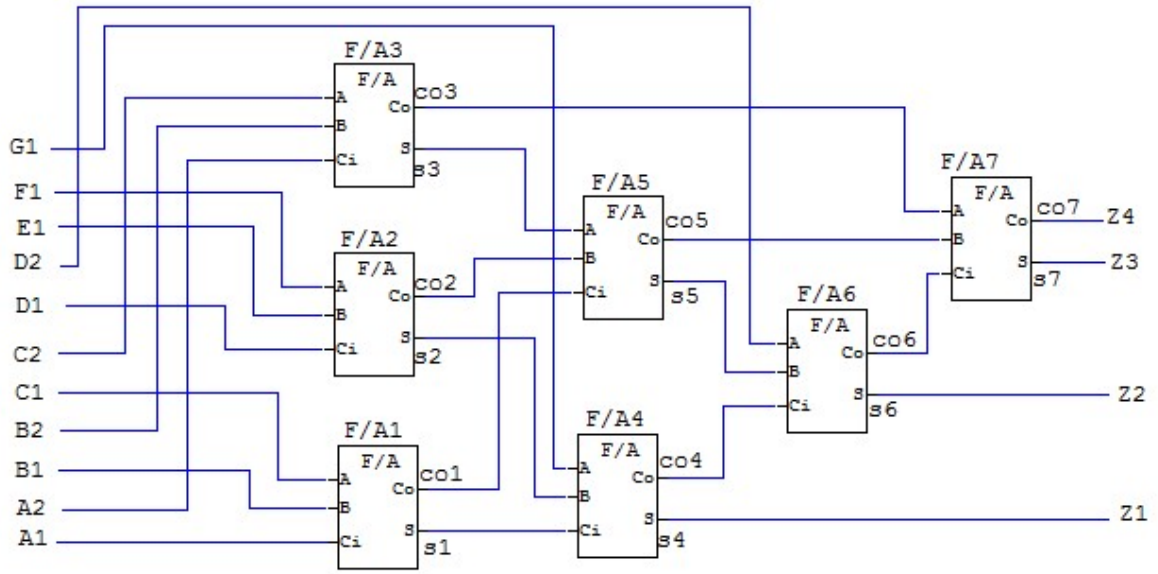
Página 8_4. Realizar la suma de cuatro números de dos bits A (a2a1), B (b2b1), C (c2c1) y D (d2d1) y tres números de 1 bit, E (e1), F (f1) y G (g1) utilizando el menor número posible de sumadores completos ("full-adders").

El problema tiene 11 entradas, según indica y el resultado. Como el valor máximo A, B, C y D es 3 al ser de 2 bits, y el de E, F y G es 1 al ser de 1 bit, el resultado máximo de la suma es 15 (3+3+3+3+1+1+1), por ello la salida Z tiene 4 bits Z4, Z3, Z2 y Z1.

Como en problemas anteriores voy a sumar de tres en tres usando únicamente "full-adders" hasta conseguir que solo haya un bit por posición:



El circuito asociado a este esquema de sumas queda así:



Página 9_1. Diseñar un circuito que realice la operación aritmética:
O = 5 X + 2 Y + Z
para operandos X (x1x0), Y (y1y0) y Z (z1z0) de dos bits, utilizando el menor número posible de semisumadores de dos bits de operandos de entradas A (a1a0) y B (b1b0).

Este problema se resuelve haciendo sumas. Tenemos 6 entradas en X, Y y Z con valores entre 0 y 3; para saber el número de salidas, obtengo el resultado máximo O = 5*3+2*3+3 = 24, y para codificar 24 en binario necesito cinco bits (O4 O3 O2 O1 O0). El sumador que se debe usar es un "half-adder" de dos bits, que suma las entradas de dos bits A, B sin acarreo de entrada, y genera un resultado de tres bits dos de suma y el acarreo de salida: (a1a0) PLUS (b1b0) = (Co S1 S0).

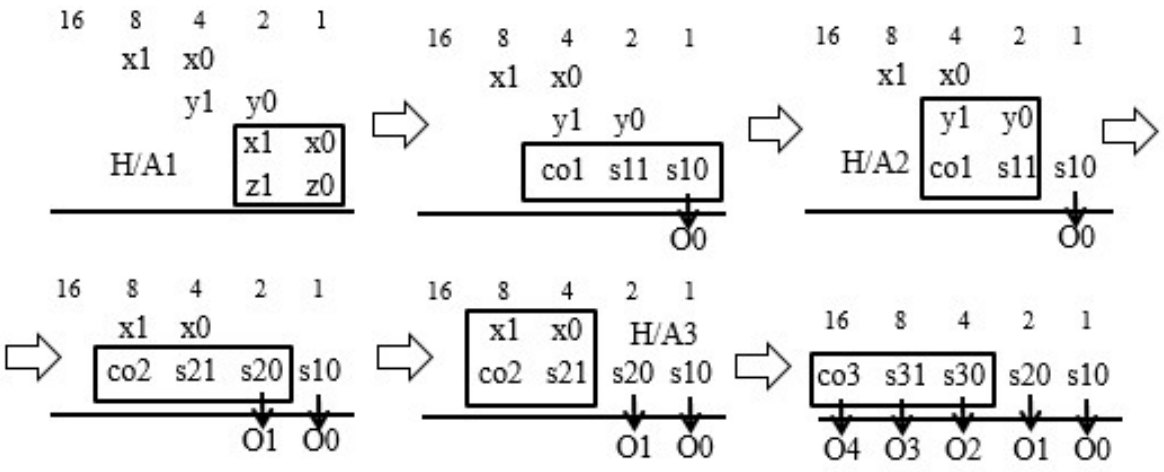
Para sumar los operados un método incorrecto sería sumar 5 veces X, 2 veces Y y una vez Z, ya que el circuito sería muy grande. Para reducir el circuito hay que darse cuenta que si multiplico un dato por una potencia de 2, basta desplazar el dato a la izquierda tantas posiciones como la potencia de 2, al igual que hacemos con las potencias de 10 cuando usamos aritmética decimal. Así:

$O = 5\,X + 2\,Y + Z = 4\,X + X + 2\,Y + Z = 2^2\,X + X + 2^1\,Y + Z = (X00) + X + (Y0) + Z$

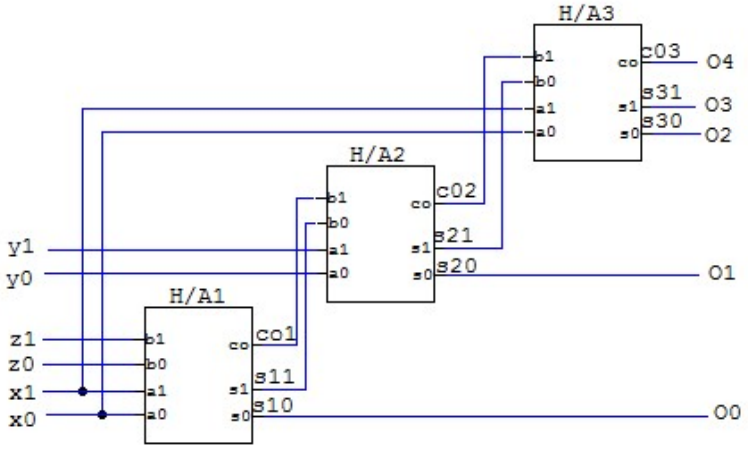
16	8	4	2	1
	x1	x0	x1	x0
		y1	y0	
			z1	z0
O4	O3	O2	O1	O0

	4	2	1
		a1	a0
H/A		b1	b0
	co	s1	s0

Agrupo los bits para sumar con el menor número de "half-adders". Para reducir el circuito hay que sumar los bits menos significativos primero, hasta conseguir un único bit por peso de derecha a izquierda, lo que evita tener que propagar acarreos; los bits únicos en las columnas de la derecha son los bits de la solución. La solución sigue estos pasos:



Del esquema se obtiene el circuito:



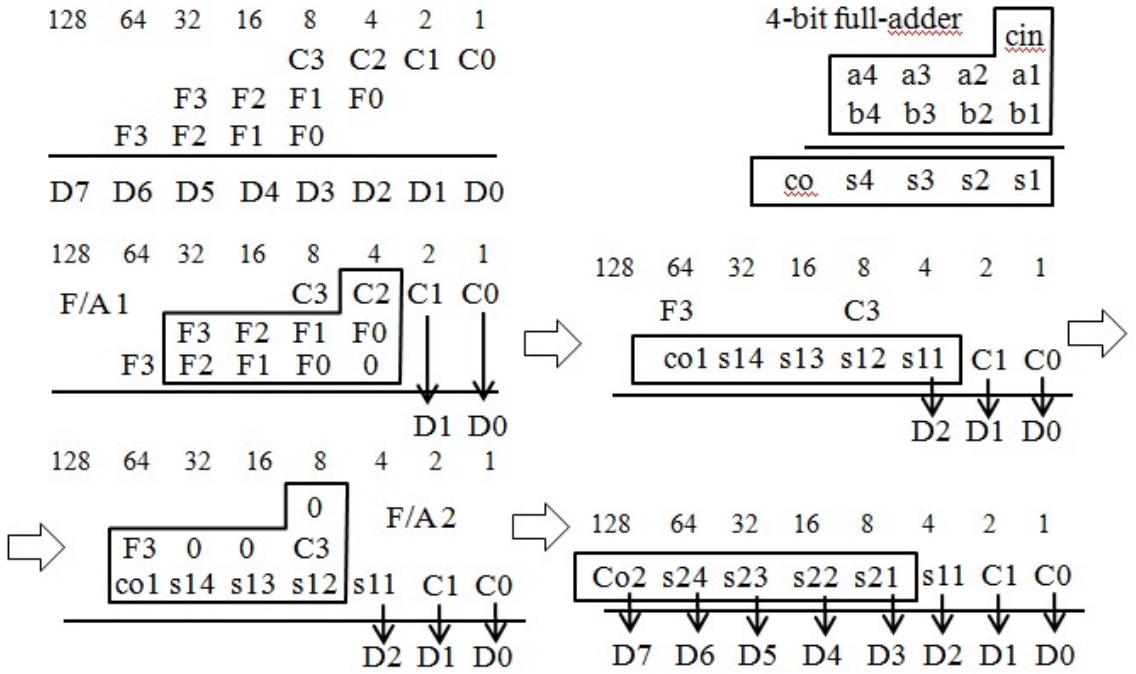
Página 9_2. Un sistema digital accede a los elementos de una matriz de 12*12 (144 elementos) que están almacenados en memoria. Para acceder a un elemento de la matriz el sistema utiliza la posición de filas F y la posición de columnas C, ambas de 4 bits (F3-F0, C3-C0) en código binario con valores entre 0 y 11. Sin embargo, la memoria utilizada solo tiene un bus de direcciones D.
Diseñar un circuito que genere la dirección D (8 bits D7-D0, valores entre 0 y 143) del elemento de la matriz en memoria de la forma D = 12 • F + C. Utilizar el menor número posible de sumadores (preferentemente 74LS83) para realizar la implementación.

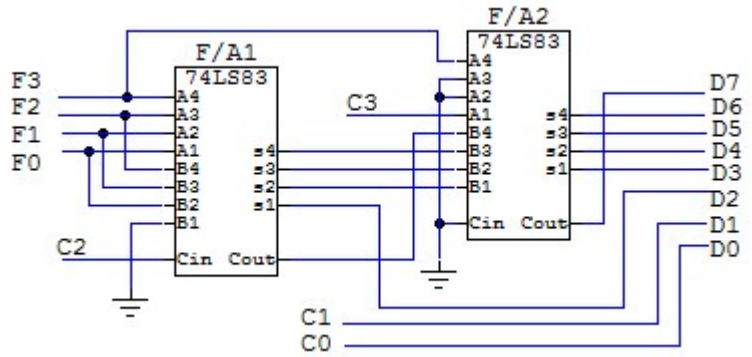
Como se indica en el enunciado la salida es dirección D de 8 bits que toma valores entre 0 y 143 en binario, y las entradas son F y C, con valores entre 0 y 11, codificados en 4 bits en binario. Se calcula mediante la operación aritmética D = 12 F + C, que debe resolverse usando preferentemente sumadores "full-adders" de 4 bits 74LS83, cuya descripción está en la diapositiva 46 de teoría.

Para realizar la operación no se suma F doce veces, sino que se puede plantear unas multiplicaciones por potencias de 2, ya que multiplicar por una potencia N de 2 significa desplazar a la izquierda N bits. Así:

$D = 12\ F + C = 8\ F + 4\ F + C = 2^3\ F + 2^2\ F + C = (F000) + (F00) + C.$

Hago un esquema de los bits a sumar y añado el esquema de la suma "full-adder" de 4 bits. Con estos elementos se puede realizar el problema usando dos sumadores. Relleno con 0s los huecos que quedan en los operandos de los sumadores, y los conecto a GND. Muestro un esquema posible de las sumas y del circuito correspondiente.





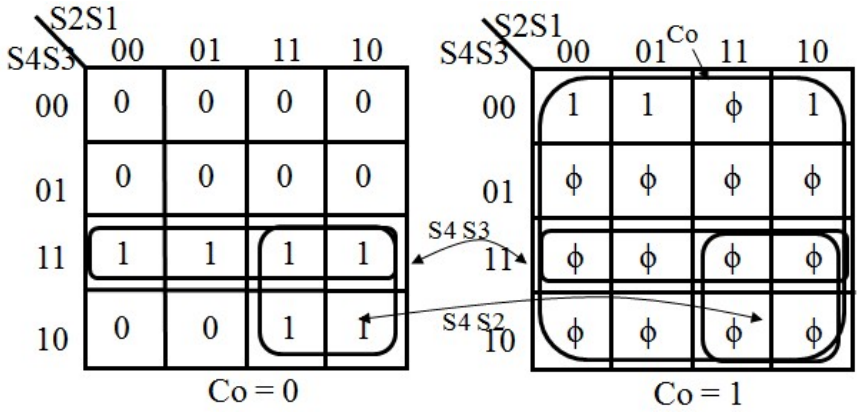
Página 9_3. Realizar el diseño de un circuito que sume dos dígitos NBCD, dando el resultado en código NBCD, utilizando puertas lógicas cuando sea necesario. Indicar como puede utilizarse este circuito para sumar números NBCD de más de un dígito.
Ayuda: Hay que sumar 6.

Este problema se ha usado en un apartado de la práctica 2. Se supone que tenemos dos dígitos NBCD A (A4 A3 A2 A1) y B (B4 B3 B2 B1) de cuatro bits, y se quiere que el resultado de la suma aritmética A PLUS B también se muestre en código NBCD. Para resolver este problema hay que considerar que el resultado de la suma está entre 0 (0+0) y 18 (9+9), y que se codifica en dos dígitos NBCD D2 y D1. D1 puede tomar valores entre 0 y 9, por lo que requiere cuatro bits, mientras que D2 solo toma valores 0 o 1, por lo que solo necesita 1 bit.

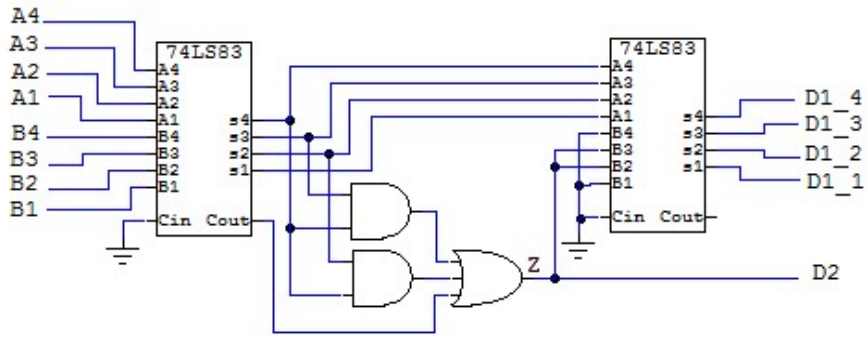
El método de realizar la suma genera primero S = A PLUS B, donde S es el resultado de la suma en 5 bits (Co S4 S2 S3 S1). Hay que obtener una señal Z que indique si S ≥ 10 (Z es 1) o si S < 10 (Z es 0). Si S < 10 el resultado es correcto => D2 = Z = 0; D1 = (S4 S3 S2 S1); pero

si S ≥ 10 se sabe que para D2 = Z = 1, pero para D1 hay que realizar otra suma para obtener Sx = S PLUS 6, si S es 10 (01010), entonces Sx es 16 (10000) que se puede dividir en BCD D2 D1 = 10 (D2 = Z = 1, D1 = 0000); si S es 18, entonces Sx es 24 (11000), que se divide en NBCD D2D1 = 18 (D2 = Z = 1, D1 = 1000).

Este método requiere tres subcircuitos. Primero, un sumador para hacer A PLUS B: un sumador como el 74LS83 con el bit de acarreo a 0. Segundo, un circuito que genere la señal Z (S ≥ 10); la función lógica de Z la calculo mediante un mapa de Karnaugh sabiendo que S está entre 0 y 18 (para el resto de valores puedo considerar Z como "don't care"). Z = Co + S4 S3 + S4 S2. Z también podría generarse con un circuito comparador.



El tercer circuito es un sumador que sume 6 si Z es 0 y 0 si Z es 1. El sumador extra puede hacerse como S PLUS (0 Z Z 0), fijando el bit de acarreo a 0. El sumador podría ser solo de 3 bits ya que el bit menos significativo de D1 es directamente S1. El circuito queda como en la figura. El bit de D2 también puede obtenerse del acarreo de salida del segundo sumador.



Para sumar números de varios dígitos BCD, el circuito de la figura puede considerarse como un semisumador (sin acarreo de entrada), siendo D1 el dígito de suma y D2 el acarreo de salida. Si en el circuito en el sumador de la izquierda se considera que en el acarreo de entrada Cin se puede conectar una entrada de un bit, que actué como acarreo de entrada del sumador BCD se tendría un sumador completo BCD. La suma de un sumador completo BCD estaría entre 0 y 19, pero esto no alteraría el cálculo de Z ya que, si la casilla 19 del mapa de Karnaugh anterior se cambia de ϕ a 1, su función lógica no cambia. Se puede generar una estructura de suma tipo *ripple*, conectando el acarreo de salida (D2) generado por dígito i al acarreo de entrada de la suma del siguiente dígito i+1.

Página 10_1. Realizar el diseño de un comparador de dos números A y B de cuatro bits tomando como base el sumador de números binarios de cuatro bits 74'83, utilizando puertas lógicas cuando sea necesario. El circuito debe generar tres salidas: O1 (A = B), O2 (A > B), O3 (A < B).

La base de este circuito consiste en hacer una resta. Si hago A – B y el resultado es 0, entonces A = B; si es positivo es que A > B, y si es negativo es que A < B. La resta en circuitos digitales se hace operando en complemento-2, y el circuito restador se muestra en la diapositiva 47 de teoría. En ese circuito se utiliza $A - B = A + (B)_{2,c} = A + \overline{B} + 1$.

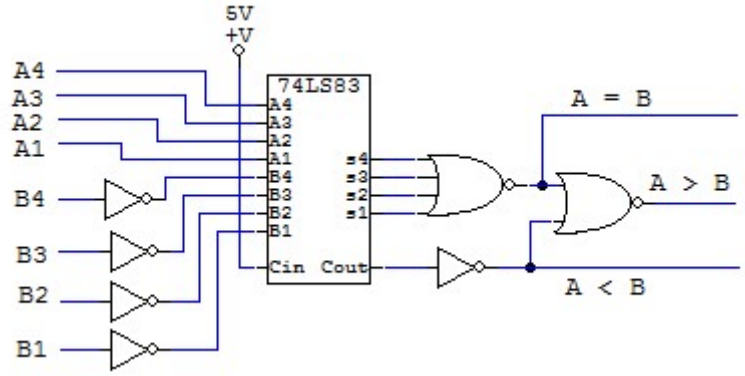
El complemento-2 también se puede definir de otra manera $A - B = A + (2^N - B)$ (diapositiva 21 del tema I). Si uso la expresión $A - B = 2^N + (A - B)$, donde 2^N es el peso de la salida de acarreo del sumador operando como restador. De esta expresión se pueden obtener las salidas de un comparador:

- Si $A < B \Rightarrow 2^N + (A - B) < 2^N \Rightarrow Cout = 0; S \neq 0$
- Si $A = B \Rightarrow 2^N + (A - B) = 2^N \Rightarrow Cout = 1; S = 0$
- Si $A > B \Rightarrow 2^N + (A - B) > 2^N \Rightarrow Cout = 1; S \neq 0$

Comparando con los valores de Cout y de S:

- A < B si Cout es 0 => $(A < B) = \overline{Cout}$
- A = B si S es 0 => $(A = B) = \overline{S4}\ \overline{S3}\ \overline{S2}\ \overline{S1} = \overline{S4 + S3 + S2 + S1}$
- A > B si Cout = 1 y S ≠ 0 => (A > B) = Cout (S4 + S3 + S2 + S1) = Cout ● $\overline{(A = B)}$ = $= \overline{\overline{Cout} + (A = B)}$

El circuito, basado en un restador, queda así:



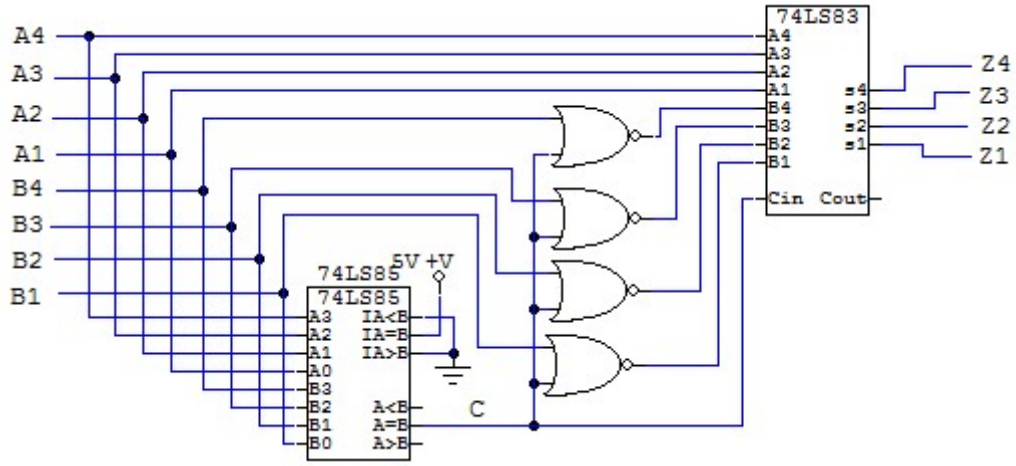
Página 10_2. Diseñar un circuito que realice la operación aritmética Z = A + 1 cuando A es igual B, y la operación aritmética Z = (A - B) - 1 cuando A es mayor que B, donde A y B son números binarios de cuatro bits, siendo siempre A mayor o igual que B. Implementar el circuito utilizando como base el sumador 74'83 y otros elementos MSI y puertas lógicas.

Según el enunciado hay varias operaciones que se deben realizar. Por un lado, hay que comparar A y B para ver si A > B o A = B (A > B) no está permitido. Este comparador genera una señal de control C para realizar una de las operaciones aritméticas. La operación aritmética "A + 1", se puede realizar con un sumador. La operación "A – B – 1", se puede realizar como una suma en complemento-2, haciendo la transformación "A + ($\overline{B}$ + 1) – 1", o "A + $\overline{B}$". Las operaciones por realizar se pueden plantear en una tabla, donde indico los valores que tienen

que tomar cada operando del sumador 74'83 de 4 bits: Asum (4 bits), Bsum (4 bits), Cin (1 bit).

	C	Op.	Asum	Bsum	Cin
A > B	0	A – B – 1	A	$\overline{B}$	0
A = B	1	A + 1	A	0	1

De esta tabla se obtiene que C es la salida "igual" de un comparador de 4 bits (74'85 por ejemplo), y que Asum = A y Cin = C. Para calcular Bsum utilizo, como si fuese un multiplexor seleccionado por C, la expresión Bsum = $\overline{C}\,\overline{B} + C \bullet 0 = \overline{C}\,\overline{B} = \overline{C + B}$, que son en realidad 4 puertas NOR, una por cada bit de B. El circuito queda:



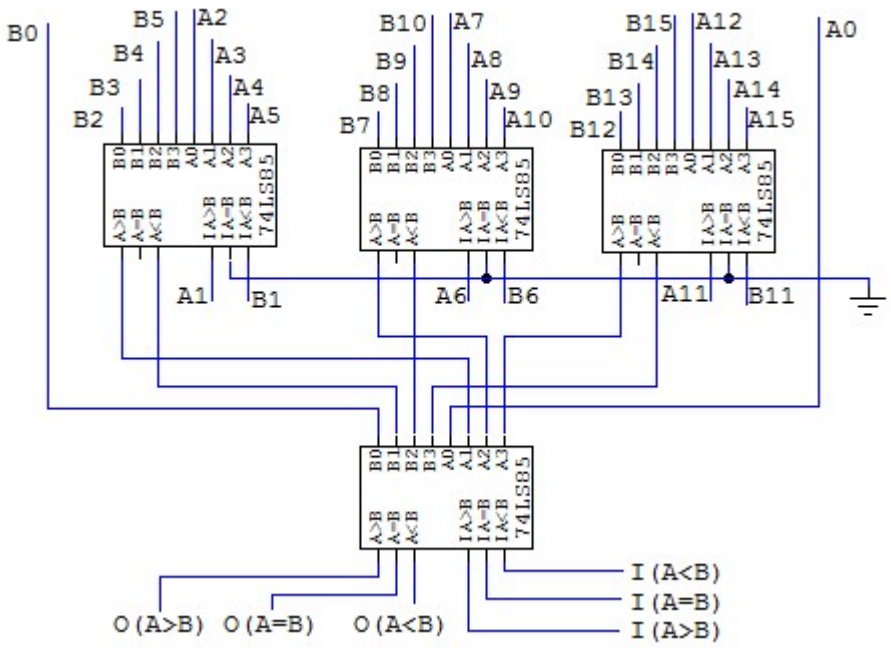
Un par de puntualizaciones sobre el circuito. Primero, en las clases teóricas se han hecho las restas sobre operandos en complemento-2, y no sobre operandos positivos como sucede en este problema. Cuando la resta da siempre resultado positivo, como en este problema, el circuito opera bien. En este caso podríamos suponer que los operandos no son de 4 bits, sino de 5 con un 0 extendido a la izquierda y que el resultado también es de 5 bits, pero como el resultado es siempre positivo, el bit más significativo es también 0, por lo que las operaciones se pueden hacer en 4 bits, y el resultado es correcto.

Segundo, en este circuito he presupuesto que la salida Z es de 4 bits. Esto es válido salvo en un caso A = B = 15, en el que Z debería ser 16, por lo que se necesitarían 5 bits de salida. Para obtener este quinto bit de salida, una solución podría basarse en que A, B y Z son de 6 bits, con A y B extendidos dos bits a la izquierda con 0s, y donde los bits 6 son 0 como en el párrafo anterior y no se opera con ellos. Para los bits 5, se puede suponer un sumador no de 4, sino de 5 bits, donde Z5 = Asum5 $\oplus$ Bsum5 $\oplus$ Cout4 = 0 $\oplus$ $\overline{C + 0}$ $\oplus$ Cout4 = 0 $\oplus$ $\overline{C}$ $\oplus$ Cout4 = = $\overline{C}$ $\oplus$ Cout4 = $\overline{C \oplus Cout}$, ya que Cout4 es la señal Cout del sumador de la figura; esta solución necesita una puerta exnor. Otra solución utiliza una AND de cinco entradas: Z5 es 1 si A = B (C es 1) y A es 15 => Z5 = C A4 A3 A2 A1; simplificable a una puerta AND de dos entradas como Z5 = C ● Cout, (A = B y $Z_{4\text{-}1}$ > 15, es decir Cout = 1).

Página 10_3. Diseñar un circuito comparador COMP16 de números binarios de 16 bits utilizando únicamente cuatro comparadores 74'85 de cuatro bits y un máximo de dos niveles de lógica. El circuito debe tener dos entradas A y B de 16 bits, tres entradas de expansión I(A=B), I(A>B) y I(A<B), y tres salidas O(A=B), O(A>B) y O(A<B), como el circuito 74'85. Indicar razonadamente el número máximo de bits de los números que se pueden comparar mediante un circuito formado por 3 COMP16 y 3 circuitos 74'85, y un

máximo de tres niveles de comparadores 74'85 (sabiendo que en los circuitos COMP16 ya hay dos niveles).

Es posible realizar un comparador de 16 bits (índices de 15 a 0) conectando los 4 comparadores 74’85 en serie como en la diapositiva 54 de teoría. Sin embargo, esa solución tiene 4 niveles de lógica entre las entradas y las salidas, por lo que no es la solución. La solución es situar comparadores en paralelo (diapositivas 53 y 54 de teoría), utilizando además la capacidad de comparar 5 bits con el circuito 74’85 usando sus entradas de expansión. Para comparar 16 bits se puede usar un sistema en paralelo: en el comparador del segundo nivel se conectan las salidas de tres comparadores de 5 bits del primer nivel (bits 15-11, 10-6, 5-1 por ejemplo), queda una entrada más para los bits 0 de entrada, y sus entradas de expansión quedan para las entradas de expansión totales. El circuito queda así:



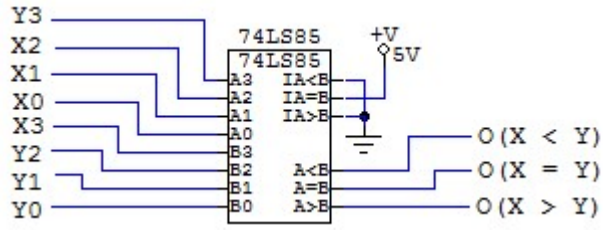
En un circuito con 3 COMP16 y 3 comparadores 74’85, se usaría uno de los 74’85 como circuito del tercer nivel, en sus entradas de comparación y expansión se conectarían las salidas de los otros 5 comparadores, los cuatro con sus salidas (A > B) y (A < B) conectadas a las entradas de datos Ai y Bi podrían usar sus entradas de expansión para comparar un bit más, y el comparador cuyas salidas (A > B), (A = B) y (A < B) se conectan a las entradas de expansión (IA > B), (IA=B) y (IA < B), solo puede comparar los bits de datos, ya que sus entradas de expansión son las entradas de expansión del comparador total. El número de bits es N = 3*17 + 5 + 4 = 60 bits.

Página 11_1. Se quieren diseñar circuitos digitales que realicen la comparación de dos números binarios con signo X e Y. Se deben obtener tres salidas que indiquen cuando X = Y, X > Y o X < Y. Se debe utilizar en lo posible comparadores comerciales como el 74’85, y otros elementos lógicos cuando sea necesario. Indicar en cada caso el razonamiento o las ecuaciones lógicas que llevan al diseño final.
Recordar: los números positivos siempre son mayores que los negativos; entre números positivos el mayor es el de mayor valor absoluto (5 > 3), entre números negativos el mayor es el de menor valor absoluto (-3 > -5).
a) Suponer X (x3x2x1x0) e Y (y3y2y1y0) de 4 bits en complemento-2.

b) Suponer X (Sx x3x2x1x0) e Y (Sy y3y2y1y0) de cinco bits en formato con bit de signo, donde Sx e Sy son los signos de X y de Y: 0 positivo, 1 negativo. Los otros bits contienen los módulos Mx y My de cada número en código binario. Para simplificar un poco el problema suponer que existe el +0 pero no existe el -0 (evita evaluar -0 = +0).

a) Se quiere hacer un comparador para dos operandos X e Y de 4 bits en complemento-2 usando un comparador convencional para número sin signo como es el circuito 74'85 y lógica adicional. Supongo primero que intento conectar X a A, e Y a B. Para realizar el circuito hay que pensar que los datos en complemento-2 tienen pesos (-8, 4, 2, 1), mientras que en circuito 74'85 las entradas tienen pesos (8, 4, 2, 1). Con estos datos se puede ver que los tres bits menos significativos operan en igual en los dos casos, y la diferencia está en el bit más significativo. En este bit más significativo, si los bits de los operandos son iguales operan igual que el peso sea +8 o -8; sin embargo, si los bits son distintos: si X3 es 1 e Y3 es 0, entonces X es negativo e Y es positivo, luego X < Y; y si X3 es 0 e Y3 es 1, entonces X es positivo e Y es negativo, luego X > Y. Esto funciona de forma complementada a cómo operan A3 y B3 en el circuito 74´85.

Hay dos formas rápidas de realizar el circuito. Si X3 e Y3 funcionan de forma complementada se pueden usar dos puertas NOT para hacer A3 = $\overline{X3}$ y B3 = $\overline{Y3}$. Así, si X3 es 1 e Y3 es 0, A3 está a 0 y B3 está a 1, con lo que A < B (X < Y). Pero existe una segunda forma que ahorra los inversores: intercambiar X3 e Y3, conectado X3 a B3, e Y3 a A3; en este caso, si X3 es 1 e Y3 es 0, entonces A3 es 0 y B3 es 1, con lo que A < B (X < Y), y si X3 es 0 e Y3 es 1, entonces A3 es 0 y B3 es 1, con lo que A > B (X > Y). Implemento esta segunda forma y el circuito queda así:



b) En el caso de números en codificación con bit de signo se pueden generar las salidas en función de los bits de signos Sx y Sy, y de los módulos Mx y My. Dos números son iguales si tienen el mismo signo y módulo. En este problema no existe el -0, lo que no hay que comprobar si los dos módulos son 0. Que dos bits sean iguales se resuelve con una puerta XNOR, como ya ha aparecido en problemas previos.

O(X=Y) = (Sx=Sy) ● (Mx=My) = $\overline{Sx \oplus Sy}$ ● (Mx=My)

Si existiese el -0 la condición de igualdad es O(X=Y) = (Mx=My) ● [(Sx=Sy) + (Mx=0)], los módulos son iguales y, los signos son iguales o los módulos son 0.

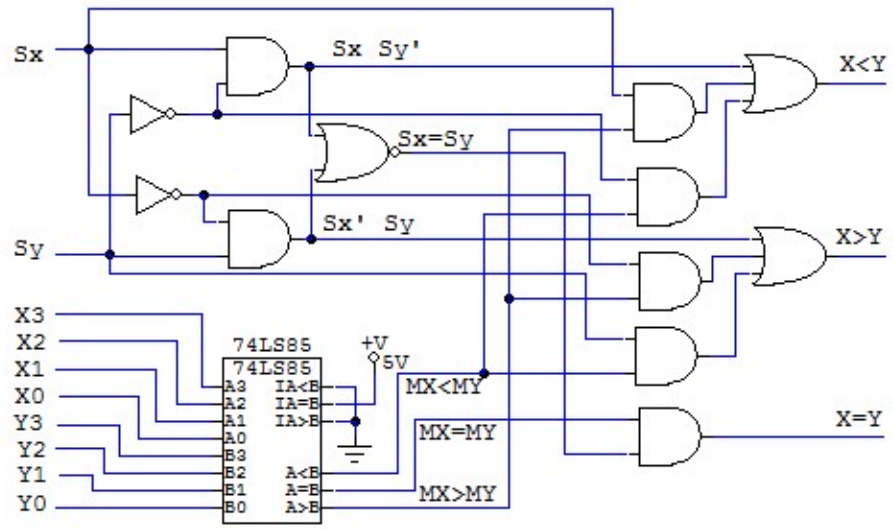
Para que X > Y se necesita que X sea positivo e Y negativo (Sx = 0 y Sy = 1, $\overline{Sx}$ ● Sy, no se comprueba el caso X = +0 e Y = -0, ya que no existe el valor -0), o que los dos sean positivos (Sx = 0 y Sy = 0, $\overline{Sx}$ ● $\overline{Sy}$) y el módulo de X sea mayor que el módulo de Y (Mx > My), o que sean negativos (Sx = 1 y Sy =1, Sx ● Sy) y el módulo de X sea menor que el de Y (Mx < My). Luego:

$$O(X>Y) = \overline{Sx} \bullet Sy + \overline{Sx} \bullet \overline{Sy} \bullet (Mx > My) + Sx \bullet Sy \bullet (Mx < My) =$$
$$= \overline{Sx} \bullet [Sy + \overline{Sy} \bullet (Mx > My)] + Sx \bullet Sy \bullet (Mx < My) =$$
$$= \overline{Sx} \bullet [Sy + (Mx > My)] + Sx \bullet Sy \bullet (Mx < My) =$$
$$= \overline{Sx} \bullet Sy + \overline{Sx} \bullet (Mx > My) + Sx \bullet Sy \bullet (Mx < My) =$$
$$= \overline{Sx} \bullet (Mx > My) + Sy \bullet [\overline{Sx} + Sx \bullet (Mx < My)] =$$
$$= \overline{Sx} \bullet (Mx > My) + Sy \bullet [\overline{Sx} + (Mx < My)] =$$
$$= \overline{Sx} \bullet Sy + \overline{Sx} \bullet (Mx > My) + Sy \bullet (Mx < My)$$

La salida X < Y es similar la salida anterior, cambiando básicamente X por Y: se necesita que X sea negativo e Y positivo (Sx = 1 y Sy = 0, $\overline{Sx} \bullet Sy$, no se comprueba el caso X = -0 e Y = +0, ya que no existe el valor -0), o que los dos sean positivos (Sx = 0 y Sy = 0, $\overline{Sx} \bullet \overline{Sy}$) y el módulo de X sea menor que el módulo de Y (Mx < My), o que sean negativos (Sx = 1 y Sy =1, Sx ● Sy) y el módulo de X sea mayor que el de Y (Mx > My). Luego:

$$O(X<Y) = Sx \bullet \overline{Sy} + \overline{Sx} \bullet \overline{Sy} \bullet (Mx < My) + Sx \bullet Sy \bullet (Mx > My) =$$
$$= \overline{Sy} \bullet [Sx + \overline{Sx} \bullet (Mx < My)] + Sx \bullet Sy \bullet (Mx > My) =$$
$$= \overline{Sy} \bullet [Sx + (Mx < My)] + Sx \bullet Sy \bullet (Mx > My) =$$
$$= Sx \bullet \overline{Sy} + \overline{Sy} \bullet (Mx < My) + Sx \bullet Sy \bullet (Mx > My) =$$
$$= \overline{Sy} \bullet (Mx < My) + Sx \bullet [\overline{Sy} + Sy \bullet (Mx > My)] =$$
$$= \overline{Sy} \bullet (Mx < My) + Sx \bullet [\overline{Sy} + (Mx > My)] =$$
$$= Sx \bullet \overline{Sy} + Sx \bullet (Mx > My) + \overline{Sy} \bullet (Mx < My)$$

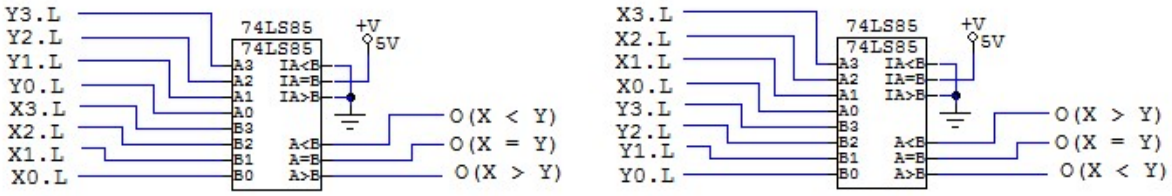
Las señales Mx = My, Mx > My y Mx < My se obtienen con un comparador de 4 bits 74’85 en sus salidas O(A = B), O(A > B), y O(A < B) respectivamente. Con todo esto el circuito queda así:



Página 11_2. Indicar como se puede hacer un comparador de dos palabras de 4 bits X e Y que codifican números binarios sin signo cuando las entradas de datos están en lógica negativa, utilizando el comparador 74’85.

Este problema tiene parecido con el problema 11_1a, ya que de alguna manera hay una complementación entre las entradas del problema X e Y que son .L y las entradas del comparador A y B que son .H. Supongo que conecto X a A, e Y a B en principio. El problema se puede resolver de varias maneras. La forma más directa es usar 8 puertas NOT, una para cada entrada de X e Y, para convertir las señales .L en .H, y usar el comparador directamente.

Las demás formas evitan usar las puertas NOT, se basa en que, primero, si las bits Xi e Yi son iguales la comparación es la misma tanto con señales .H como con señales .L, y que si son distintas: si Xi es H (0) e Yi es L (1), el resultado debería ser Xi < Yi, pero el comparador produciría Ai (H) > Bi (L), ya que A y B son .H; si Xi es L (1) e Yi es H (0), el resultado debería ser Xi > Yi, pero el comparador produciría Ai (L) < Bi (H) , ya que A y B son .H. Este problema se puede solventar de dos formas. La primera forma consiste en cambiar las conexiones de las entradas manteniendo las de la salidas: suponiendo que A es X y que B es Y, conecto X a B e Y a A => si X < Y (o si X > Y) entonces B > A (B < A) y la salida O(A<B) (o O(A<B)) se fija a 1. La segunda forma implica menos cambios ya que cambia las salidas manteniendo las entradas, X a A e Y a B: si X > Y (o X < Y) implica que A < B (A> B), coloco la salida O(X > Y) en la salida del comparador O(A < B), y de la misma forma coloco la salida (X < Y) en la salida del comparador O(A > B).



Página 12_1. Dos tanques de agua A y B tienen cada uno de ellos 8 sensores (entradas A7-A0, B7-B0, el índice 0 es el nivel inferior, el índice 7 el nivel superior) que indican el nivel de agua de cada tanque. El sensor está activado cuando hay agua en el nivel y desactivado en caso contrario. Los tanques tienen unas esclusas que permiten introducir agua (salidas AI, BI) o sacar agua (salidas AO, BO).
Diseñar un circuito para automáticamente sacar agua del tanque (activar AO o BO) que más tiene cuando la suma de los niveles de los tanques pase de 11, o cargar agua en el tanque (activar AI o BI) que menos tiene cuando la suma de los niveles de los tanques sea menor de 5. Usar elementos MSI estándar y/o puertas lógicas.
Calcular el tiempo de propagación máximo del circuito.

Tengo que diseñar un circuito de 16 entradas (A7-A0, B7-B0) y 4 salidas (AI, BI, AO, BO). Voy a ir dividiendo todo el problema en partes, y voy viendo cómo se resuelve cada parte y qué circuitos hacen falta. Primero usaré circuitos genéricos, y luego intentaré construirlo con circuitos de la familia 74 y puertas lógicas.

I7	I6	I5	I4	I3	I2	I1	I0	SA2	SA1	SA0
1	1	1	1	1	1	1	1	1	1	1
0	1	1	1	1	1	1	1	1	1	0
0	0	1	1	1	1	1	1	1	0	1
0	0	0	1	1	1	1	1	1	0	0
0	0	0	0	1	1	1	1	0	1	1
0	0	0	0	0	1	1	1	0	1	0
0	0	0	0	0	0	1	1	0	0	1
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	0	0	0	0

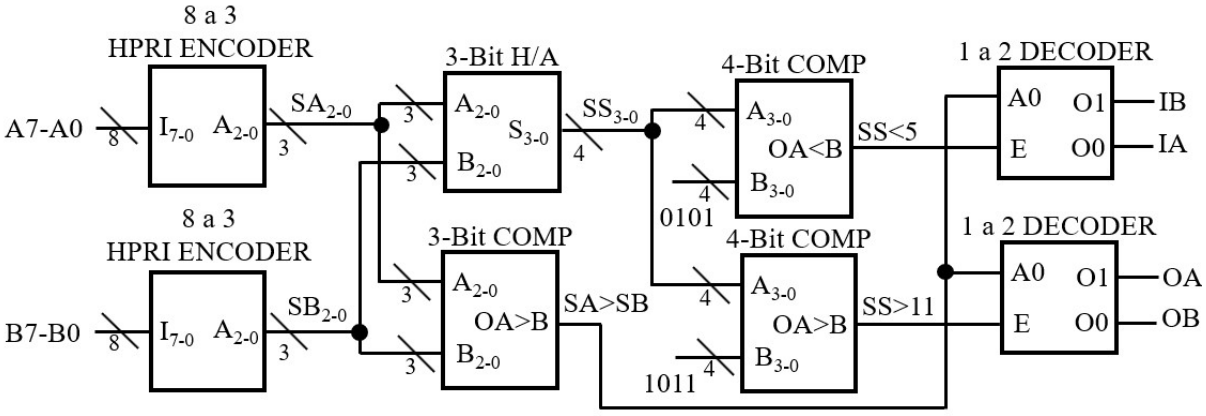
Los circuitos de entrada leen los sensores de los tanques. Debo calcular un valor binario asociado al contenido del tanque para luego poder sumarlo y compararlo. Luego el primer

objetivo es pasar los valores de los sensores de los tanques A7-A0 y B7-B0 a valores SA y SB, donde SA y SB son números binarios de 3 bits que representan números entre 0 y 7. Hay que tener en cuenta que en un tanque si hay agua en un nivel i, todos los sensores con índice menor o igual que i deben estar activados. El funcionamiento del sensor se puede representar en la tabla anterior. De los circuitos conocidos, esta tabla se puede construir con un codificador con prioridad de 8 a 3, para entradas A y salidas SA, y entradas B y salidas SB.

El enunciado nos pide comprobar si la suma de SA y SB es mayor o menor que ciertos valores. Luego necesito un semisumador de 3 bits, que sume SA PLUS SB, generando un resultado SS de 4 bits. Los valores de SS están entre 0 y 14.

Se debe comprobar si el resultado de SS es mayor que 11 o es menor que 5. Intuitivamente se puede hacer con dos comparadores de 4 bits. El primero tiene como operandos SS (A) y 11 (B, 1011) y, utilizando su salida A > B, genera (SS > 11), y el segundo tiene como operandos SS (A) y 5 (B, 0101) y, utilizando su salida A < B, genera (SS < 5). Al tener un operando constante estos circuitos se podrían hacer con puertas lógicas: se calculan las tablas de verdad que nos dicen si un número de 4 bits es mayor que 11, o menor que 5, se generan los mapas de Karnaugh y las funciones SOP asociadas. Para SS de 4 bits (SS3 SS2 SS1 SS0), (SS > 11) = SS3 ● SS2, y (SS < 5) = $\overline{SS3}\ \overline{SS2} + \overline{SS3}\ \overline{SS1}\ \overline{SS0}$. Queda propuesto. El diseño con el comparador es más flexible, ya que si se quieren cambiar los valores de apertura basta con cambiar la entrada del comparador, sin tener que calcular nuevas funciones lógicas.

En la última fase genero las salidas. Para ello se necesita un comparador de tres bits para comparar SA y SB, ya que en función de si uno es mayor o menor que el otro se abrirán las esclusas del tanque A o del tanque B. El comparador tiene como operandos SA (A) y SB (B), y utilizo la salida > (SA > SB). Para que las salidas AO y BO se activen SS tiene que ser mayor que 11, y se activa la salida asociada al mayor valor de SA (AO) o SB (BO). Estas salidas pueden ser resueltas con un decodificador 1 a 2 con habilitación: la señal (SS > 11) se conecta al habilitador, para fijar a 0 las dos salidas del decodificador, que generan AO y BO, si SS ≤ 11. Utilizo como entrada de dirección A0 la señal SA > SB; con el circuito habilitado si SA > SB, A0 es 1, luego se activa la salida 1 del decodificador que debe ser AO (A es el mayor), cuando SA ≤ SB la entrada de dirección A0 es 0, y se activa la salida O0, que debe ser BO (B es el mayor). En este circuito y en el caso de SA y SB sean iguales, se activa BO, aunque esto es opcional, ya que no está definido en el enunciado.

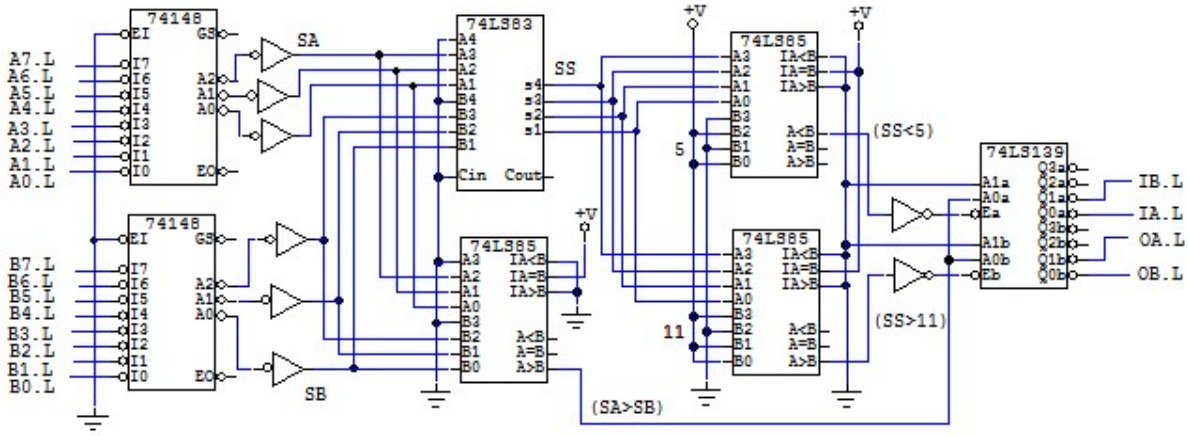


Para las salidas AI y BI, el circuito es similar: un decodificador 1 a 2 con habilitación. La habilitación es la señal (SS < 5), utilizo como bit de dirección A0 la señal (SA > SB) y en las salidas fijo AI y BI. Si SS ≥ 5, el decodificador está deshabilitado y las dos salidas AI y BI se

fijan a 0. Si SS < 5, y si (SA > SB) es 1 se activa O1, que debe ser BI (B es el menor), cuando (SA > SB) es 0 se activa O0, que debe ser AI (A es el menor). Cuando SA y SB son iguales (SA > SB) es 0 y se activa IA también. El caso de igualdad de nivel entre tanques no está definido en el enunciado.

El tiempo de propagación de este circuito podemos evaluarlo por el camino crítico, o camino tiempo más largo desde las entradas a la salida. Ese camino, por ejemplo, de Sa a IA (aunque hay varios caminos similares) consta de un codificador 8 a 3 (de A a SA), un sumador de 3 bits (de SA a SS), un comparador de 4 bits (de SS a (SS<5)) y un decodificador 1 a 2 (E a IA). Luego $Tp = Tp_{encoder}(I\text{-}A) + Tp_{adder}(A\text{-}S) + Tp_{comp}(A\text{-}<) + Tp_{decoder}(E\text{-}O)$.

El circuito puede hacerse con circuitos de la familia 74, haciendo modificaciones para ajustar los números de bits y la polaridad de las señales. Todos los dispositivos aparecen en las diapositivas de teoría. Uso dos codificadores con prioridad 8 a 3 74'148, un sumador completo de 4 bits 74'83 como sumador de 3 bits (las entradas sin usar A4, B4 y Cin se fijan a 0, y no se conecta la salida Cout), tres comparadores 74'85 de 4 bits y un decodificador 2 a 4 74'139 que contiene dos decodificadores en el circuito, que se puede utilizar como decodificador de 1 a 2, conectando A1 a 0 y dejando sin conectar las salidas O3 y O2. Como las entradas de los codificadores y las salidas del decodificador están en lógica negativa, las entradas y las salidas las declaro en lógica negativa (.L). Un último problema es que las salidas de los codificadores están en lógica negativa mientras que, en principio las entradas del sumador y del comparador están en lógica positiva; lo mismo sucede entre la salida del comparador (positiva) y la entrada de los decodificadores (negativa). Para ajustar el circuito incluyo inversores. El circuito queda así:



El tiempo de propagación máximo del circuito queda entonces:

$Tp = Tp_{148}(I\text{-}A) + Tp_{04}(NOT) + Tp_{83}(A\text{-}S) + Tp_{85}(A\text{-}<) + Tp_{04}(NOT) + Tp_{139}(E\text{-}O)$.

Algunos de los tiempos están en las diapositivas de teoría, los otros los saco de hojas de características para la familia 74LS y fabricante ON o Motorola. Uso el mayor de los tiempos tphl, tplh ya que con circuitos como sumadores o comparadores un cambio en una entrada puede hacer cambiar unas salidas LH y otras HL, y utilizo los tiempos máximos.

Tp = 36 ns + 15 ns + 24 ns + 36 ns + 15 ns + 32 ns = 158 ns

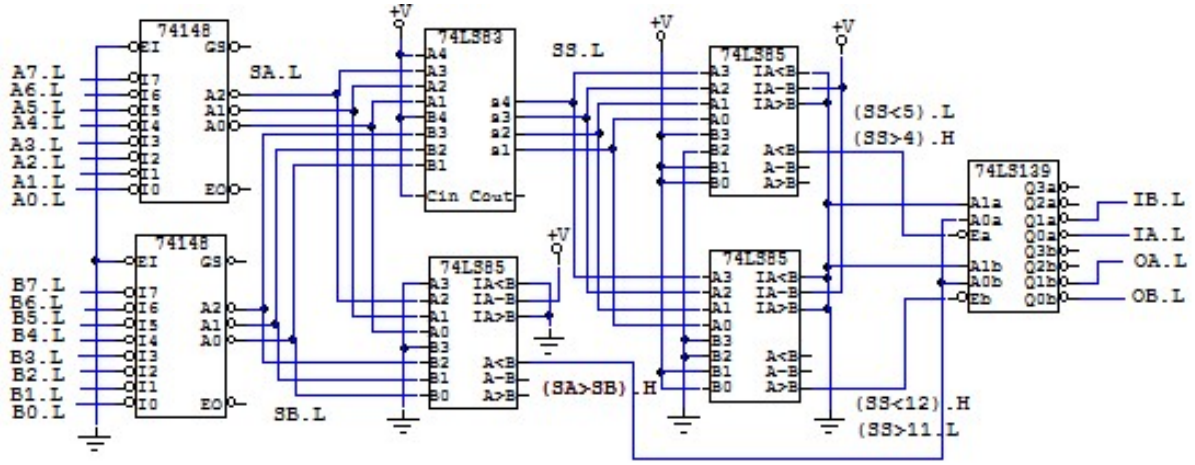
El circuito se puede optimizar ya que todos los inversores se pueden eliminar, aunque es algo complicado de seguir. Los inversores de las salidas de los codificadores se pueden eliminar y

trabajar internamente con señales .L, ya que los sumadores suman igual en lógica positiva como en negativa (todas las entradas y salidas .L, las entradas que no se usan se conectan a alimentación), y los comparadores pueden tener las entradas de datos en lógica negativa (11 es LLHL y 5 es HLHL) según el problema 11_2.

Los inversores que alimentan las entradas de habilitación de los decodificadores también se pueden eliminar. Los comparadores generan las salidas .H y el habilitador utiliza una entrada E.L, pero puedo hacer las siguientes transformaciones:

- Eb.L = (SS > 11).L = $(\overline{SS \leq 11})$. L = $(\overline{SS < 12})$. L = (SS < 12).H
- Ea.L = (SS < 5).L = $\left(\overline{SS \geq 5}\right)$. L = $(\overline{SS > 4})$. L = (SS > 4).H

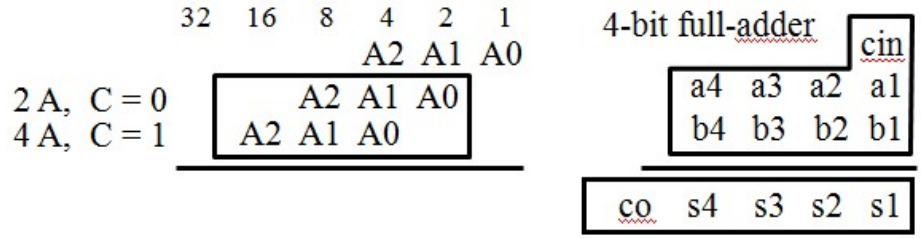
Modificando los comparadores para que trabajen en lógica negativa, en vez de generar S > 11, genero S < 12 en la salida A > B del comparador, ya que opera en lógica negativa, siendo 12 igual a LLHH en lógica negativa. Por otro lado, en vez de generar S < 5, genero S > 4 en la salida A < B del comparador, siendo 4 igual a HLHH en lógica negativa.



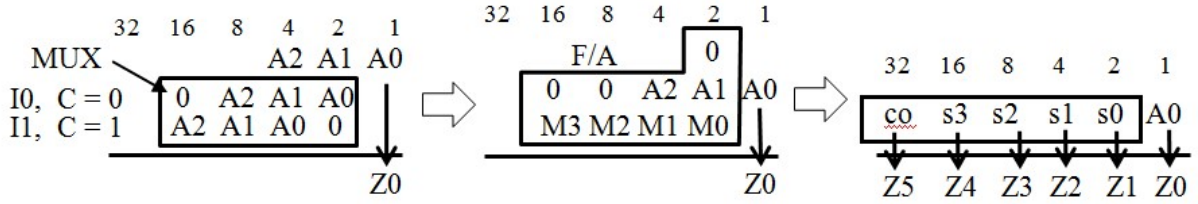
En este circuito se eliminan dos inversores del camino crítico, con respecto al circuito anterior, con lo que Tp = 128 ns.

Página 12_2. Diseñar un circuito que dado como entrada un número binario positivo A de 3 bits realice la operación Z = 3*A si la entrada de control C está a 0 o la operación Z = 5*A si la entrada de control C está a 1, usando 1 sumador completo 74LS83 y el menor número de puertas lógicas u otros circuitos MSI (74LS157). Calcular el tiempo de propagación máximo del circuito.

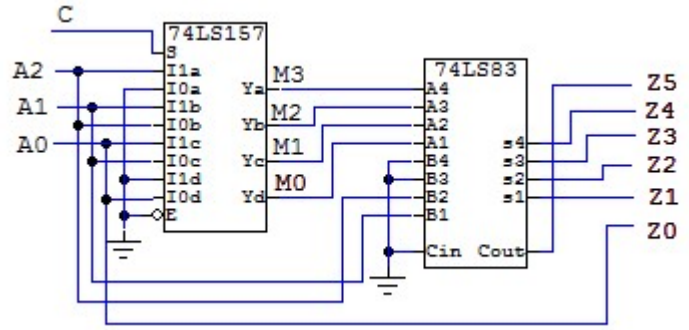
El circuito tiene 3 entradas de datos A (A2 A1 A0) con valores entre 0 y 7. Como el valor máximo de Z = 5 * 7 = 35, que requiere 6 bits (Z5 Z4 Z3 Z2 Z1 Z0). Obviamente, este problema se puede resolver con mapas de Karnaugh y funciones lógicas, pero se va a resolver con circuitos MSI.



Mientras que un operando del sumador es A, de la figura se ve que, dependiendo del valor de C, el segundo operando a utilizar es 2A o 4 A. La selección la podemos hacer con multiplexores de 2 entradas, cuya entrada de selección es C. En la familia 74, el multiplexor 74LS157 tiene cuatro multiplexores de 2 entradas, con entradas de selección comunes. Es el ideal para este circuito.



Para poder hacer las sumas con un único sumador completo de 4 bits 74LS83, hay que darse cuenta que Z0 es directamente A0, y que en la columna de peso 1 no hace falta sumar. Hay que empezar a sumar en la columna de peso 2. Con un multiplexor 74'157 y con un sumador 74'83 el circuito queda así:



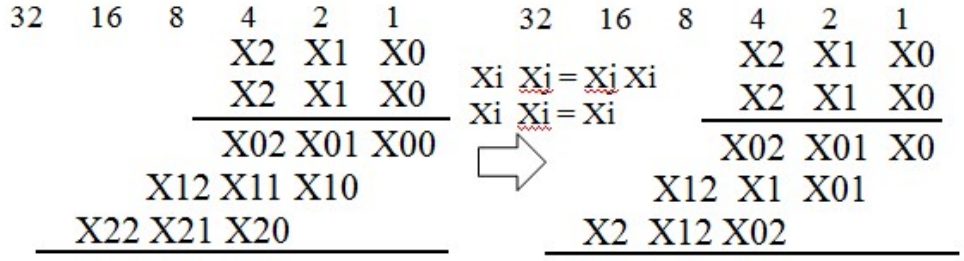
El tiempo de propagación máximo del circuito depende de los tiempos de los dos componentes. Tpmax = Tpmax(74LS157) + Tpmax(74LS85). Los tiempos de propagación de las hojas características están en las diapositivas de teoría (4 del tema IIIb y 49 del tema IIIa).
El multiplexor tiene dos tiempos posibles de S a Y (C a M), o de I a Y (A a M); el más alto es el tphl de S a Y con 27 ns. El sumador tiene tiempos de A-B a S o de A-B a Cout, el más alto es de A, B a S, y tphl = tplh = 24 ns. Normalmente se escoge el más alto de tphl o de tplh. Tp = 24 ns + 27 ns = 51 ns.

Página 13_1. Realizar la operación X^2 (X*X), siendo X un número binario sin signo de tres bits (X2X1X0). Utilizar el menor número posible de puertas AND, y semisumadores y sumadores completos de 1 bit. Para simplificar el circuito se recuerda que se puede utilizar la propiedad conmutativa, el teorema de la idempotencia, y que A PLUS A = 2*A = (A0).
Diseñar un circuito que realice la operación X^2, siendo X de 4 bits (X3X2X1X0) usando el circuito X^2 para X de 3 bits diseñado anteriormente, el menor número de puertas AND y un único sumador 74LS83 (4-bit full-adder).

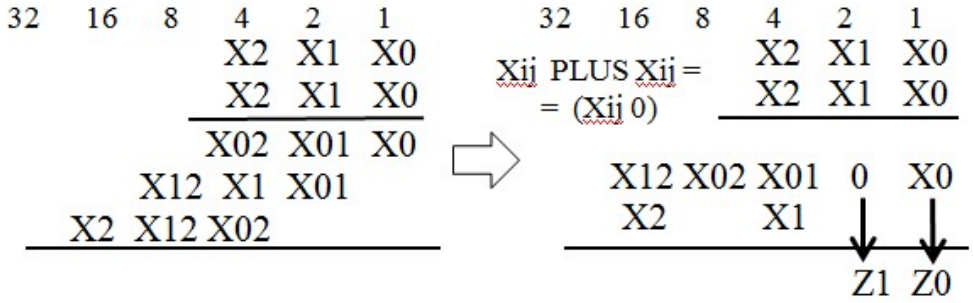
Este es un problema de tres entradas y podría resolverse también mediante mapas de Karnaugh y funciones lógicas. La entrada X está entre 0 y 7, y la salida Z máxima es 49, que requiere 6 bits (Z5 Z4 Z3 Z2 Z1 Z0). Planteo la operación como una multiplicación X * X, y planteo los productos parciales Xij de cada multiplicación Xi ● Xj, que ya sabemos que se resuelve por un producto AND. Las nueve puertas AND iniciales se reducen a 3(X10, X20 y X21) al aplicar las reglas lógicas:

- Conmutativa: Xij = Xji (Xi ● Xj = Xi ● Xj = Xij).
- Idempotencia: Xii = Xi (Xi ● Xi = Xi).

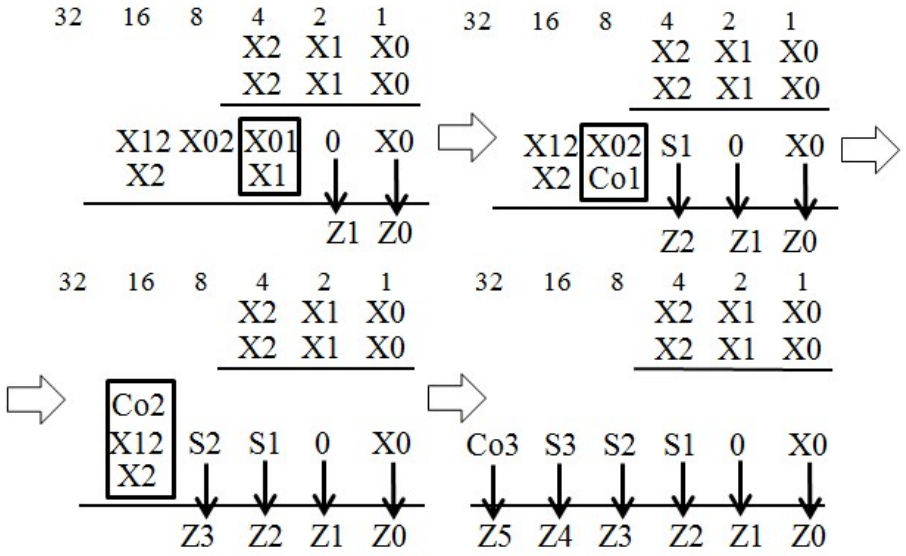
.



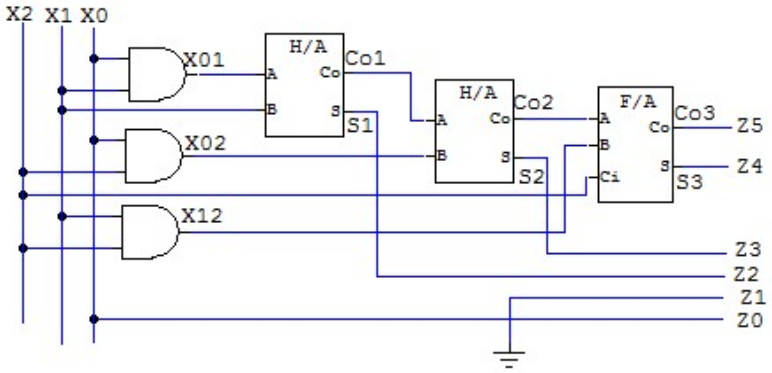
Ahora aplico la regla aritmética A PLUS A = 2 * A = 2^1 * A = (A0)



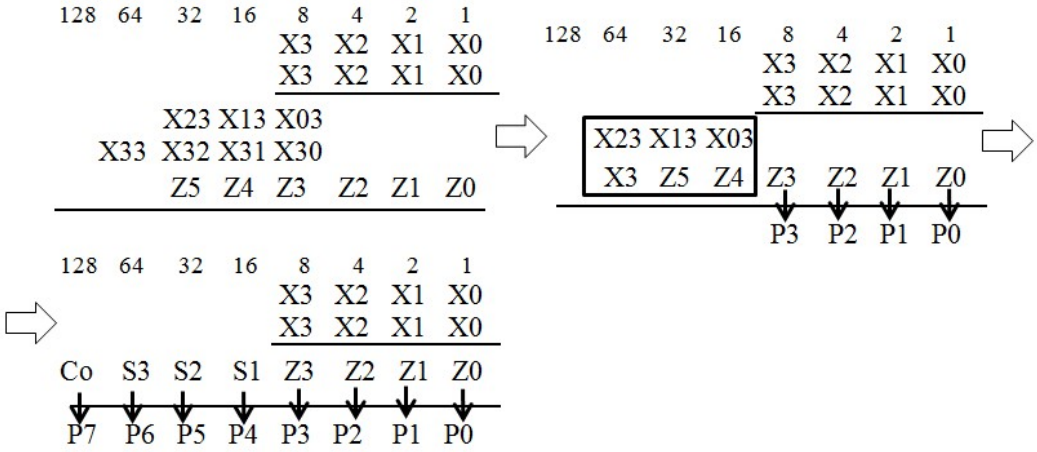
El circuito que queda se puede sumar usando dos semisumadores y un sumador completo:

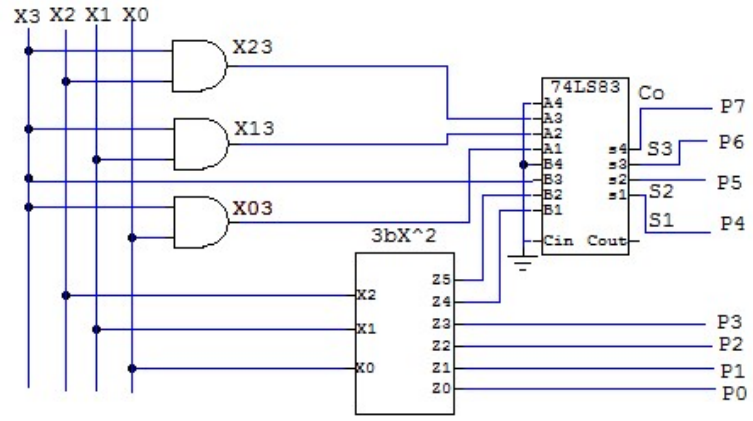


El circuito queda:



Ahora hay que hacer el cuadrado de X, siendo X de 4 bits, sabiendo que se dispone un circuito como el anterior ya ha hecho. El problema tiene 4 entradas y como el mayor cuadrado es 15 * 15 = 225, la salida P debe tener 8 bits (P7 P6 P5 P4 P3 P2 P1 P0). La llamo P para no confundirla con la salida Z que he generado antes. Para generar P, parto de que conozco $X_{2\text{-}0}^2$. Por lo que para $X_{3\text{-}0}^2$ tengo el siguiente esquema de productos parciales, donde todos los productos parciales entre los bits 0, 1 y 2 ya han sido calculados en el resultado parcial Z. Aplicando a X33, X30, X31 y X32 los mismos razonamientos anteriores, solo se necesita añadir 3 AND para generar los productos parciales y un semisumador de 3 bits, que implemento con un sumador completo de 4 bits, fijando a 0 las entradas que no se necesitan (A4, B4 y Cin).





Página 13_2. Se disponen de dos palabras de 3 bits A (Sa a1a0) y B (Sb b1b0) que representan números con signo en notación de bit de signo. Se quiere obtener la suma de A y B, pero mostrando el resultado en complemento-2. La suma se hará convirtiendo los números A y B a complemento-2 usando decodificadores 74LS138 y puertas NAND de 4 entradas, y luego sumando los números convertidos usando un sumador completo de 4 bits como el 74LS83. Mostrar la implementación del circuito.

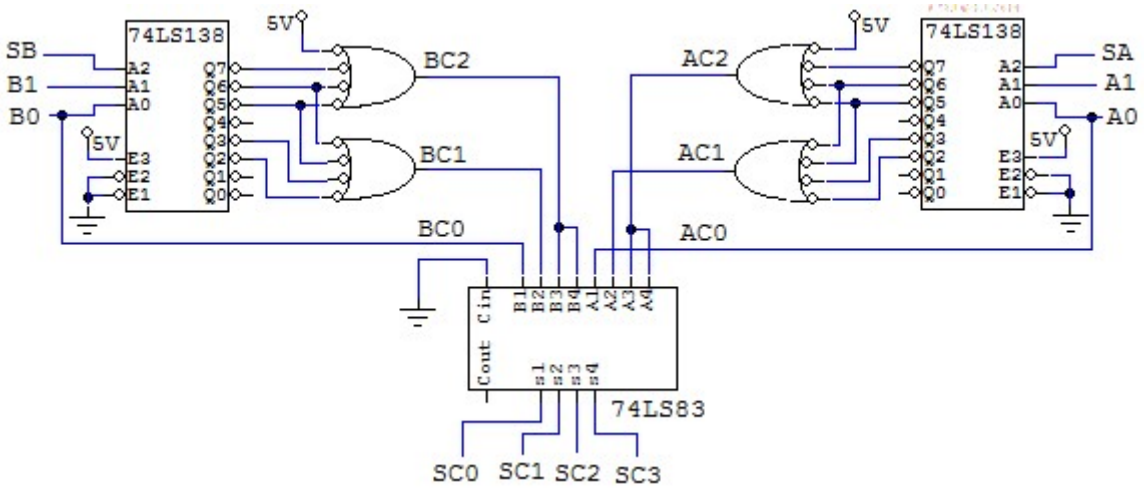
Este es un problema de 6 entradas y 4 salidas. Para realizar el circuito hago una conversión de notación de bit de signo a complemento-2 y luego sumo los números convertidos. La conversión sigue esta tabla, que se puede implementar con un decodificador 3 a 8 74LS138 y puertas NAND.

	+ (0) / - (1)	2	1		-4	2	1
N	**Sx**	**X1**	**X0**	**D**	**Y2**	**Y1**	**Y0**
0	0	0	0	0	0	0	0
1	0	0	1	1	0	0	1
2	0	1	0	2	0	1	0
3	0	1	1	3	0	1	1
-0	1	0	0	4	0	0	0
-1	1	0	1	5	1	1	1
-2	1	1	0	6	1	1	0
-3	1	1	1	7	1	0	1

Y2 = F2 (Sx, X1, X0) = ∑ (5, 6, 7)
Y1 = F1 (Sx, X1, X0) = ∑ (2, 3, 5, 6)
Y0 = F0 (Sx, X1, X0) = ∑ (1, 3, 5, 7)

En la tabla se puede ver que Y0 = X0, por lo que no se necesita puerta NAND para esa salida. El circuito requiere el decodificador y dos puertas NAND de 4 entradas. En Y2 hay que fijar una de las entradas a valor no controlante (1). La conversión podría hacerse también solo con puertas lógicas, incluso solo con puertas NAND (con mapas de Karnaugh Y2 = Sx (X1 + X0), e Y1 = $\overline{Sx}\,X1 + X1\,\overline{X0} + Sx\,\overline{X1}\,X0$).

Los operandos A y B en notación de bit de signo convierten a AC y BC en complemento-2, con dos copias del circuito anterior. AC (AC2 AC1 AC0) y BC (BC2 BC1 BC0) de tres bits se suman en complemento-2, y se obtiene un resultado SC de 4 bits (SC3 SC2 SC1 SC0), ya que SC está en [-6, 6]. La suma en complemento-2 se puede hacer con un sumador convencional como el 74LS83, con la condición adicional de que los operandos de entrada deben extenderse a 4 bits: (AC2 AC2 AC1 AC0) y (BC2 BC2 BC1 BC0). El circuito queda así:



Página 14. Diseñar un circuito sumador para números binarios A y B de cinco bits descritos en código binario con signo, de la forma (Sa a3a2a1a0) y (Sb b3b2b1b0), donde S es el bit de signo (0 positivo, 1 negativo), y (a3-a0), (b3-b0) la codificación binaria del módulo de los números. El circuito tiene que generar un número de seis bits como resultado (Sf f4f3f2f1f0, f4-f0 bits de módulo y Sf de signo).
Utilizar como base del diseño un único circuito sumador/restador (S/R) de 4 bits, y otros circuitos lógicos MSI (comparadores, multiplexores, etc) y puertas lógicas. El circuito S/R utiliza dos operandos X e Y de cuatro bits, una señal de control C del tipo de operación

(X PLUS Y, X MINUS Y), una salida Z de 4 bits y un bit de salida de acarreo Co. Los operandos X e Y son números binarios positivos, con la restricción de que en el modo resta X debe ser mayor o igual que Y.
Explicar en qué se basa el diseño realizado y representar el diseño en modo esquemático con notación de tipo bus para simplificar el dibujo final. Generar un circuito sumador/restador, añadiendo una señal de control K y solo una puerta lógica al circuito anterior.

La organización del circuito depende de la configuración del circuito aritmético, que es un circuito sumador/restador para números sin signo, con la condición de que el resultado de la resta debe ser mayor que 0: el minuendo es mayor o igual, que el sustraendo. Aunque no lo pide explícitamente, luego haré una versión del sumador/restador.

Las operaciones que realiza el circuito S/R dependen de comparación de los módulos MA, MB y del signo de los operandos Sa y Sb. Si los signos son iguales, se debe hacer una suma y el signo del resultado es el mismo que el de los operandos. Si los signos son distintos, la operación es una resta del mayor módulo menos el menor módulo y el signo del resultado es el signo del operando de mayor módulo. Luego las señales de control son:

- SC: comparación de signos. SC es 1 si los dos bits de signo Sa y Sb son distintos y 0 si son iguales. Esta operación ya ha aparecido varias veces (por ejemplo, problema 11_1b) y es la EXOR. SC = Sa $\oplus$ Sb.
- MM: comparación de módulos. MM es 1 si MA < MB, y 0 si MA ≥ MB. Se puede utilizar un comparador o, por lo menos, parte de él.

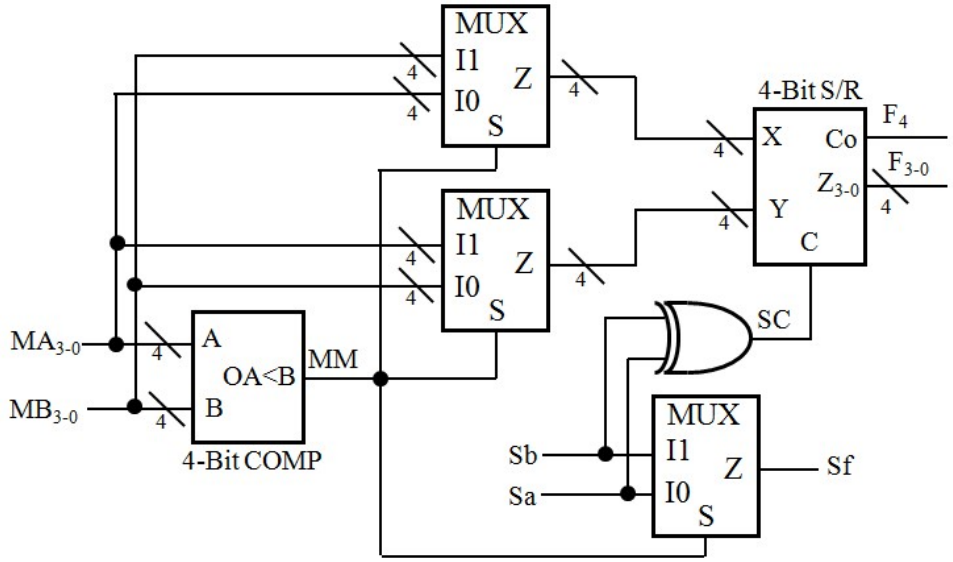
Supongo que la señal de control C del circuito S/R opera de forma que si C es 0 circuito suma y si C es 1 el circuito resta. Los operandos de entrada son X e Y, y las operaciones X PLUS Y, X MINUS Y. El funcionamiento de las operaciones se puede poner en esta tabla (habría otras opciones parecidas a esta), intentando optimizar los circuitos. En algún caso puede aparecer el resultado -0, pero esto no está prohibido por el enunciado del problema.

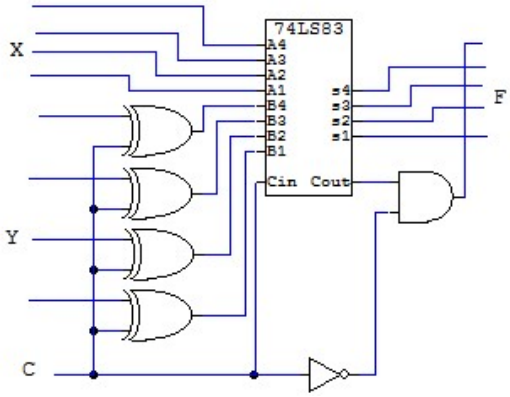
SC	MM	Sf	C	X	Y	Op.
0	0	Sa	0 (+)	MA	MB	MA PLUS MB
0	1	Sb	0 (+)	MB	MA	MB PLUS MA
1	0	Sa	1 (–)	MA	MB	MA MINUS MB
1	1	Sb	1 (–)	MB	MA	MB MINUS MA

Hay que pensar que X e Y son entradas de 4 bits y que, junto con Sf, se pueden implementar con multiplexores. Los multiplexores deberían ser, en principio de 4 entradas con las entradas de selección SC y MM. Tal como se han puesto los valores en las entradas, basta con multiplexores de dos entradas con MM como entrada de selección: si MM es 0 (I0) Sf = Sa, X = MA, Y = MB; si MM es 1 (I1) Sf = Sb, X = MB, Y = MA. Por otro lado, C = SC.

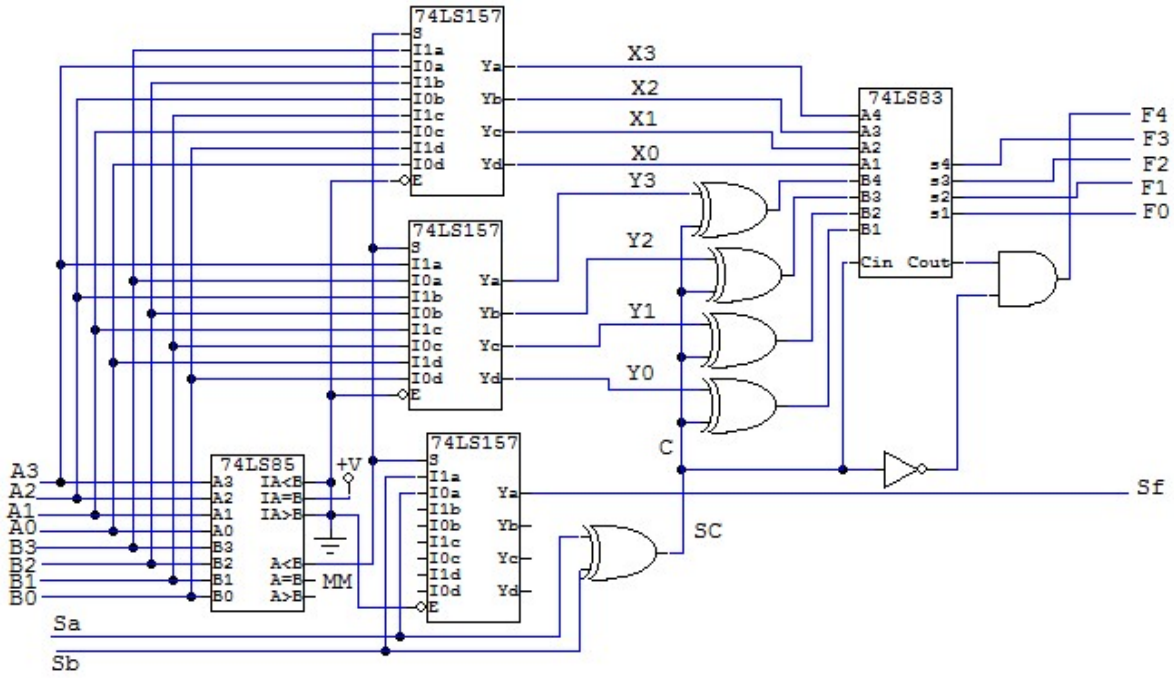
Aunque no es parte del problema el circuito S/R puede conseguirse del sumador/restador de 4 bits en complemento-2 (diapositiva 48 de teoría). Según se ha comentado en los problemas 10_1 y 10_2, el circuito también funciona bien como sumador para números sin signo (obvio) y como restador para números sin signo cuando el minuendo es mayor que el sustraendo. El único problema es el acarreo de salida (F4), en la suma (C=0) debe ser directamente Cout, pero en la resta (C=1) debe ser 0. Luego F4 = $\overline{C} \bullet Cout + C \bullet 0 = \overline{C} \bullet Cout$.

La estructura del circuito y del circuito sumador/restador queda así:





Para el comparador se utiliza el circuito 74'85, y para los multiplexores uso el circuito 74'157.

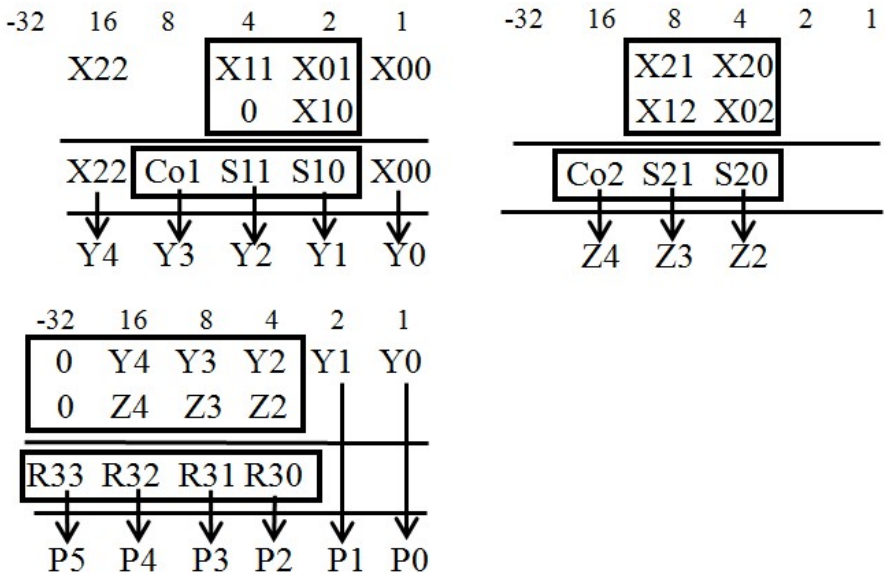


El enunciado pide hacer a partir de este circuito un circuito sumador restador con una señal K que controle la operación: suma (K a 0) o resta (K a 1). Este problema tiene una solución simple: A – B = A + (–B), donde (–B) se consigue complementando el valor de Sb, el signo de B. El efecto de si K es 0, Sb se mantenga, y si K es 1 Sb se complemente se consigue con una puerta EXOR, como ya se ha usado en otros circuitos SBI = K ⊕ Sb.

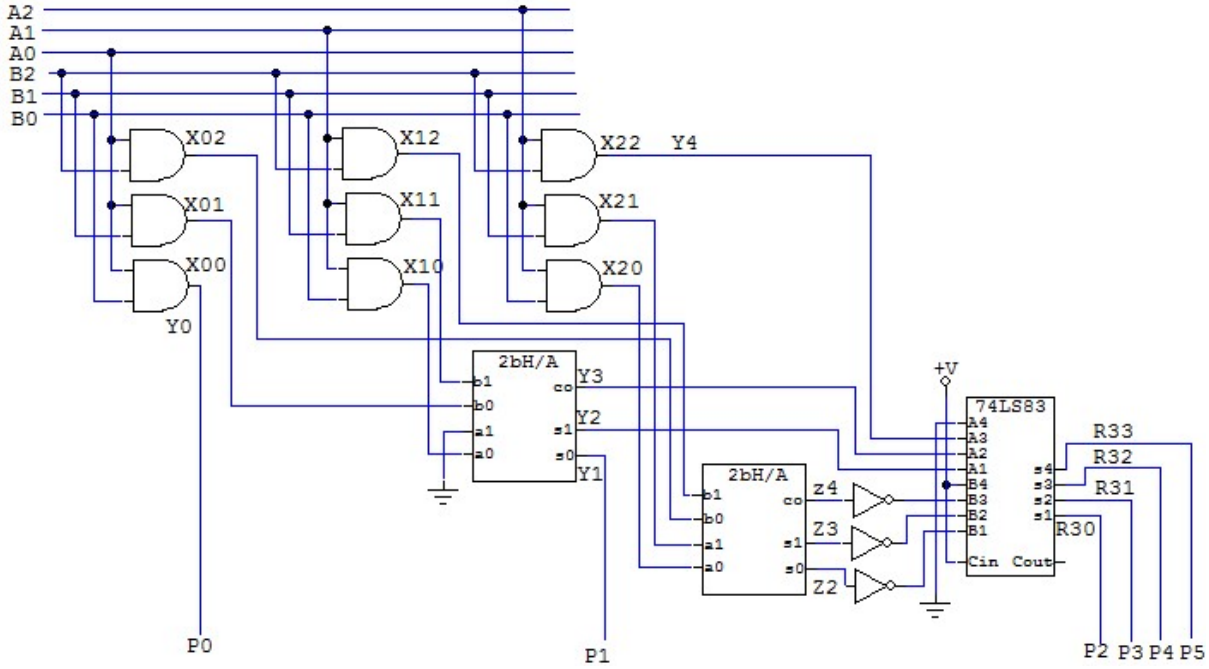
Página 15. La multiplicación de dos números binarios sin signo A y B puede hacerse utilizando puertas AND para generar los términos producto parciales (Xij = ai • bj), para sumar posteriormente estos términos mediante diferentes algoritmos (por filas, por columnas, etc). Cuando se trabaja con números con signo en complemento-2, el proceso se complica ya que los Xij pueden tener peso positivo o negativo en el resultado final por lo que deben sumarse o restarse. En este problema se debe realizar la multiplicación de dos números de tres bits A (a2 a1 a0) y B (b2 b1 b0) en notación en complemento-2, cuyos pesos de más significativo a menos significativo son (-4, 2, 1).
¿Cuántos bits debe tener el resultado P (también en complemento-2) de la multiplicación? Teniendo en cuenta que P = A • B = 16 X22 - 8(X21 + X12) - 4(X20 + X02) + 4X11 + +2(X01 + X10) + X00, y ordenando de forma adecuada las operaciones de suma y resta a realizar, diseñar un circuito que realice la multiplicación propuesta utilizando el menor número de sumadores (completos o semisumadores, indicando el número de bits de cada sumador) y puertas lógicas básicas.

En el primer tema se han hecho problemas sobre métodos de multiplicación para números con signo en complemento-2 (Booth, Baugh-Wooley). En este problema se desarrolla un método "ad-hoc" para un multiplicador de 3 bits. Lo primero es obtener el número de bits de la salida P. Los valores de A y B están en [-4, 3]. El menor resultado se consigue en (-4) * 3 = -12, y el mayor en (-4) * (-4) = 16. (-12) en complemento-2 requiere 5 bits (-16 8 4 2 1), pero para +16 se necesitan 6 bits, ya que el bit de más peso debe ser negativo (-32 16 8 4 2 1). Luego P debe tener 6 bits (P5 P4 P3 P2 P1 P0). La fórmula de P se obtiene de la multiplicación de los operandos, donde Xij = Bi * Aj, son los productos parciales de 1 bit.

P = B • A = (-4 B2 + 2 B1 + B0) * (-4 A2 + 2 A1 + A0) =
= 16 X22 – 8 X21 – 4 X20 – 8 X12 + 4 X11 + 2 X10 – 4 X02 + 2 X01 + X00 =
= 16 X22 + 4 X11 + 2 (X10 + X01) + X00 – [8 (X21 + X12) + 4 (X20 + X02)]



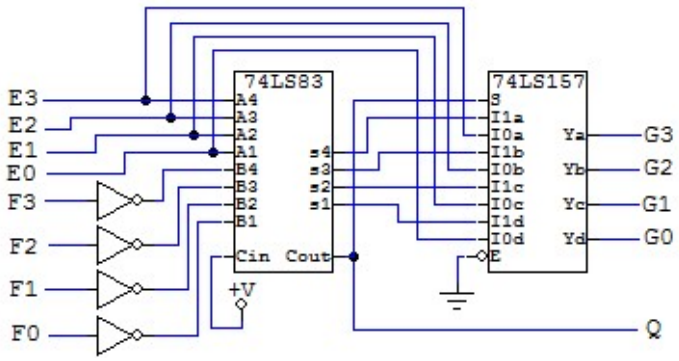
Los productos parciales Xij se generan con puertas AND: 9 puertas en total. Divido los productos parciales en positivos y negativos, y los sumo generando Y (positivos) y Z (negativos), y luego realizo P = Y – Z usando un circuito restador. Para generar Y uso un único "half-adder" de 2 bits, igual que para generar Z. Por último, para generar P utilizo un restador de operandos de 3 bits hecho con un sumador y 3 puertas NOT. En el circuito utilizo por comodidad un "full-adder" de 4 bits, extendiendo los operandos a 4 bits añadiendo un 0 por la izquierda, pero podría hacerse con un "full-adder" de 3 bits y una puerta NOT más, para generar $P5 = 0 \oplus \bar{0} \oplus Cout = \overline{Cout}$. El circuito queda así:



Página 16. La <u>división</u> A/B de dos números A (a3a2a1a0) y B (b3b2b1b0) de 4 bits, calculando el cociente Q (q3q2q1q0) y el resto R (r3r2r1r0), puede realizarse según el siguiente método:
- Tomar el dividendo como (000a3a2a1a0) y el divisor como (b3b2b1b0).
- Sea D1 los 4 bits de la izquierda (000a3) del dividendo, comparar D1 con el divisor B, si D1 ≥ B el bit del cociente q3 es 1 y se genera X (x3x2x1x0) = D1 – B; si no el cociente Q3 es 0 y X (x3x2x1x0) = D1.
- Tomar D2 como (x2x1x0a2), si D2 ≥ B el bit del cociente q2 es 1 y se genera Y (y3y2y1y0) = D2 – B; si no el cociente q2 es 0 e Y (y3y2y1y0) = D2.
- Tomar D3 como (y2y1y0a1), si D3 ≥ B el bit del cociente q1 es 1 y se genera Z (z3z2z1z0) = D3 – B; si no el cociente q1 es 0 y Z (z3z2z1z0) = D3.
- Tomar D4 como (z2z1z0a0), si D4 ≥ B el bit del cociente q0 es 1 y se genera el resto R (r3r2r1r0) = D4 – B; si no el cociente Q0 es 0 y el resto R (r3r2r1r0) = D4.
Diseñar un circuito digital que realice la división usando sumadores 74LS83 (4 chips), multiplexores 74LS157 (4 chips), y cuatro inversores 74LS04 (1 chip), utilizando el algoritmo anterior basado en operaciones de comparaciones y restas.
Calcular el tiempo de propagación máximo del circuito.

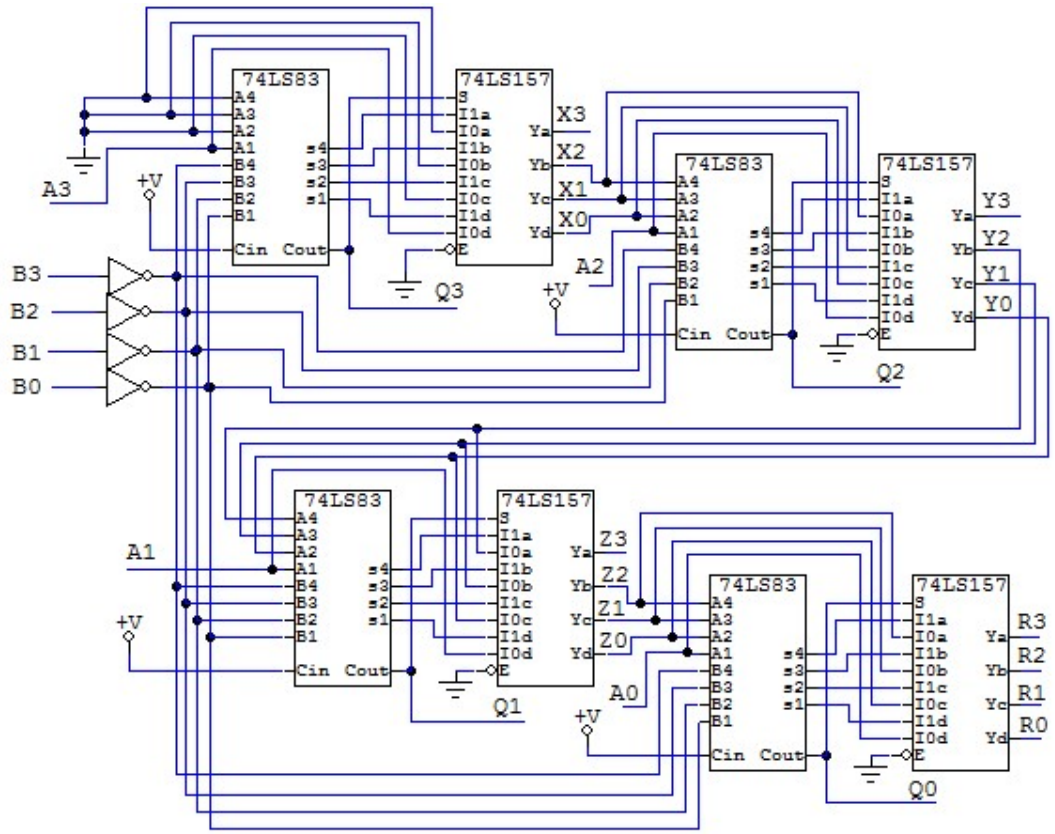
El divisor se puede realizar utilizando un circuito que se repite en cada una de las cuatro etapas de la división. El circuito tiene dos operandos de entrada de 4 bits E (E3E2E1E0) y F (F3F2F1F0), y una salida Q de un bit y una salida G (G3G2G10) de 4 bits. Si E ≥ F, Q es 1 y G = E – F; si no la salida Q es 0 y G = E. Este circuito en principio requiere un comparador, un restador y un multiplexor de dos entradas.

El comparador y el restador es el mismo circuito y está hecho en el problema 10_1. Si uso un restador en complemento-2, hecho con un sumador y puertas NOT, y aplico la fórmula del complemento-2 sobre un restador $G = E - F = 2^N + (E - F)$. Si $E \geq F$ entonces $G \geq 2^N$, por lo que Cout es 1, si $E < G$ entonces $G < 2^N$, por lo que Cout es 0. Según esto Q = Cout. Además, la salida G puede obtenerse con cuatro multiplexores de 2 entradas: Cout es la entrada de selección S y si Cout es 1, la salida G (Y) es E – F (I1), y si Cout es 0 la salida G (Y) es E (I0), siendo. Y es la salida del multiplexor en la figura. El circuito queda así:



Para hacer el circuito divisor con entradas de 4 bits el dividendo A (A3 A2 A1 A0) y divisor B (B3 B2 B1 B0), y salidas de 4 bits cociente Q (Q3 Q2 Q1 Q0) y resto R (R3 R2 R1 R0), hay cuatro etapas de este circuito, definidas como en el enunciado. Para hacer el restador solo se necesita un grupo de puertas NOT, ya que B es la misma entrada en cada etapa.

Etapa	E	F	Q	G
1	0 0 0 A3	B3 B2 B1 B0	Q3	X3 X2 X1 X0
2	X2 X1 X0 A2	B3 B2 B1 B0	Q2	Y3 Y2 Y1 Y0
3	Y2 Y1 Y0 A1	B3 B2 B1 B0	Q1	Z3 Z2 Z1 Z0
4	Z2 Z1 Z0 A0	B3 B2 B1 B0	Q0	R3 R2 R1 R0



Para calcular el tiempo de propagación máximo, el camino crítico es desde B hasta R, a través de las puertas NOT y cuatro etapas de sumador y multiplexor. De la estructura de la etapa sumador-multiplexor se observan dos caminos, cuyos tiempos son (tomados de las hojas de características: 74LS83 diapositiva 49 del tema IIIa, 74LS157 diapositiva 4 del tema IIIb):

- Tp(A-S) + Tp(I-Y) = 24 ns + 14 ns = 38 ns
- Tp(A-Cout) + Tp(S-Y) = 17 ns + 27 ns = 44 ns

Tomando el tiempo más largo de una etapa 44 ns, el tiempo de propagación máximo es:

Tp = Tp(NOT) + 4 Tp(etapa) = 15 ns + 4 * 44 ns = 191 ns