TEMA 5: SISTEMAS ARITMÉTICOS Y LÓGICOS.

5.1. Sumadores binarios.

Casi todo se hace con sumas: sumas, restas, productos, ... Concepto de acarreo.

5.1.1. Semisumador. Half Adder (HA).

Entradas de 1 bit y salida SUMA y ACARREO.

A_i	B_i	C_{i+1}	S_i
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

$$C_{i+1} = A_i \cdot B_i$$
$$S_i = A_i \oplus B_i$$

5.1.2. Sumador completo de 1 bit. Full Adder (FA).

Como el semisumador, pero además con entrada de acarreo.

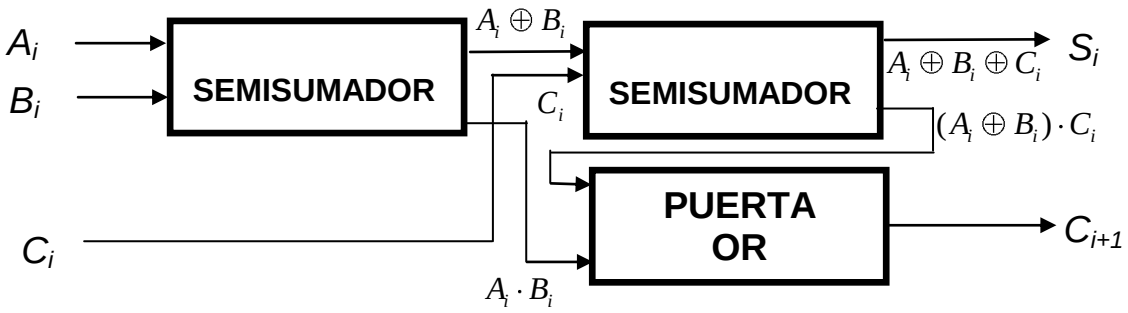
A_i	B_i	C_i	C_{i+1}	S_i
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

$C_{i+1} = (A_i + B_i) \cdot C_i + A_i \cdot B_i$ o bien $C_{i+1} = (A_i \oplus B_i) \cdot C_i + A_i \cdot B_i$

$S_i = A_i \oplus B_i \oplus C_i$

Se puede observar que el número binario compuesto por $C_{i+1}S_i$ indica, codificado en binario, el resultado de la operación y también el número de unos que hay en la terna de bits A_i, B_i y C_i.

Considerando que el semisumador es un bloque que realiza la operación XOR de dos variables y la operación AND de las mismas variables, se puede construir un sumador completo de 1 bit a partir de dos semisumadores y una puerta OR según la figura adjunta:



Restador de un bit. Se puede definir directamente la resta binario según la siguiente tabla:

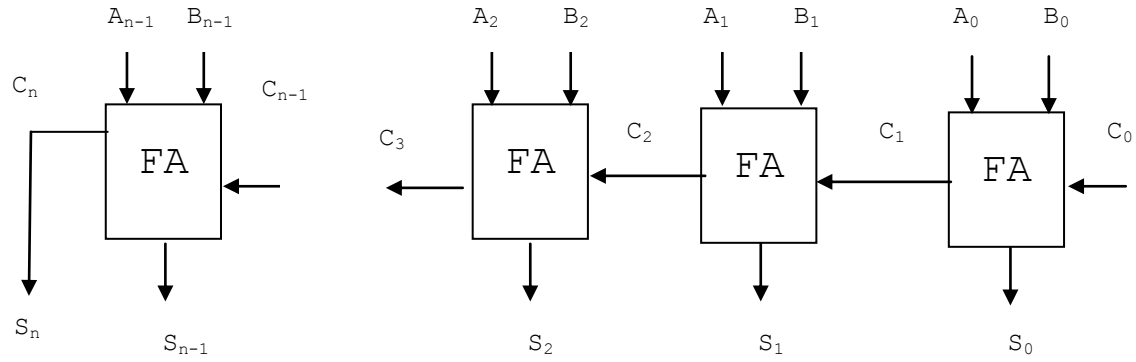
M_i	S_i	C_i	C_{i+1}	R_i	Resultado
0	0	0	0	0	0
0	0	1	1	1	−1
0	1	0	1	1	−1
0	1	1	1	0	−2
1	0	0	0	1	+1
1	0	1	0	0	0
1	1	0	0	0	0
1	1	1	1	1	−1

Se puede observar que el resultado de la operación está expresado en complemento a dos con los bits C_{i+1} R_i

De igual forma se podría encontrar un sencillo circuito realizado con puertas para llevar a cabo esta función elemental de resta.

5.1.3. Sumador paralelo con acarreo serie.

Usando n sumadores completos de 1 bis se construye un sumador completo de n bits.



Si t_s es el tiempo para realizar una suma y t_c el tiempo para realizar un acarreo, resulta:

Dato en	S_0	C_1	S_1	C_2	……	S_{n-1}	$S_n = C_n$
Tiempo	t_s	t_c	$t_s + t_c$	$2\ t_c$	……	$t_s + (n-1)\ t_c$	$n\ t_s$

Inconveniente: necesita que actúen los anteriores sumadores para que pueda actuar uno en particular. Se acumulan los retardos de propagación.

5.1.4. Sumador de arrastre anticipado.

Los acarreos se generan en paralelo (simultáneamente) con los resultados, evitando el problema de propagación serie de los sumadores vistos anteriormente. Por tanto, estos sumadores son más rápidos que los de acarreo serie.

Teniendo en cuenta que el sumador total cumple las siguientes ecuaciones:

$$S_i = A_i \oplus B_i \oplus C_i$$

$$C_{i+1} = (A_i \oplus B_i) \cdot C_i + A_i \cdot B_i = P_i \cdot C_i + G_i$$

Llamamos *Pi* al llamado término *Propagador* y *Gi* el llamado término *Generador*, definidos por:

$P_i = A_i \oplus B_i$ y $G_i = A_i \cdot B_i$

Particularizando la expresión $C_{i+1} = P_i \cdot C_i + G_i$ para *i = 0, 1, 2*, y *3* queda:

$$C_1 = P_0 \cdot C_0 + G_0$$

$$C_2 = P_1 \cdot C_1 + G_1 = P_1 \cdot (P_0 \cdot C_0 + G_0) + G_1 = P_1 \cdot P_0 \cdot C_0 + P_1 \cdot G_0 + G_1$$

$$C_3 = P_2 \cdot P_1 \cdot P_0 \cdot C_0 + P_2 \cdot P_1 \cdot G_0 + \cdot P_2 \cdot G_1 + G_2$$

$$C_4 = P_3 \cdot P_2 \cdot P_1 \cdot P_0 \cdot C_0 + P_3 \cdot P_2 \cdot P_1 \cdot G_0 + P_3 \cdot P_2 \cdot G_1 + P_3 \cdot G_2 + G_3$$

Las salidas dependen de los datos de entradas y acarreos. Los acarreos dependen de los términos propagadores y generadores. Los términos propagadores y generadores dependen sólo de los datos de entrada.

Por tanto, las salidas dependen sólo de los datos de entrada y otros que se obtienen directamente de éstos, y todos se conocen desde el primer momento.

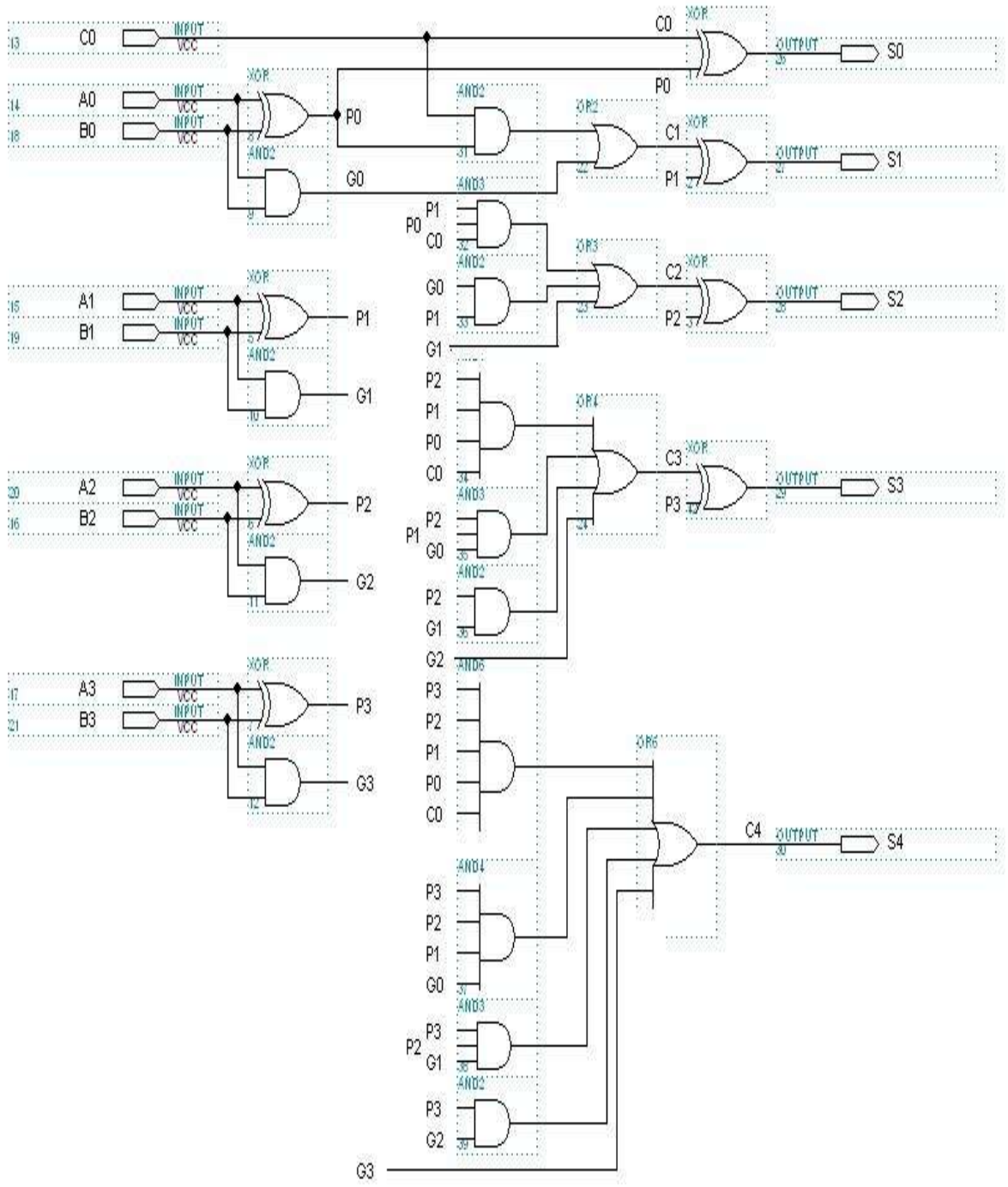
ECUACIONES:

$S_i = P_i \oplus C_i$ $\quad$ $C_1 = P_0 \cdot C_0 + G_0$

$P_i = A_i \oplus B_i$ $\quad$ $C_2 = P_1 \cdot P_0 \cdot C_0 + P_1 \cdot G_0 + G_1$

$G_i = A_i \cdot B_i$ $\quad$ $C_3 = P_2 \cdot P_1 \cdot P_0 \cdot C_0 + P_2 \cdot P_1 \cdot G_0 + P_2 \cdot G_1 + G_2$

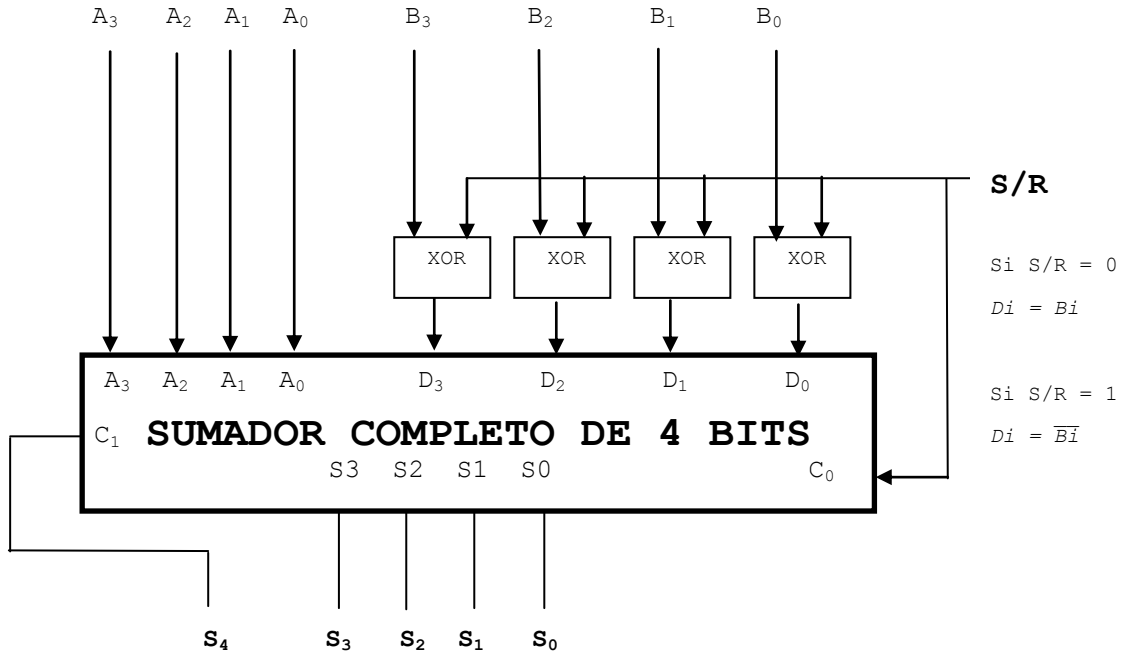
$C_{i+1} = P_i \cdot C_i + G_i$ $\quad$ $C_4 = P_3 \cdot P_2 \cdot P_1 \cdot P_0 \cdot C_0 + P_3 \cdot P_2 \cdot P_1 \cdot G_0 + P_3 \cdot P_2 \cdot G_1 + P_3 \cdot G_2 + G_3$



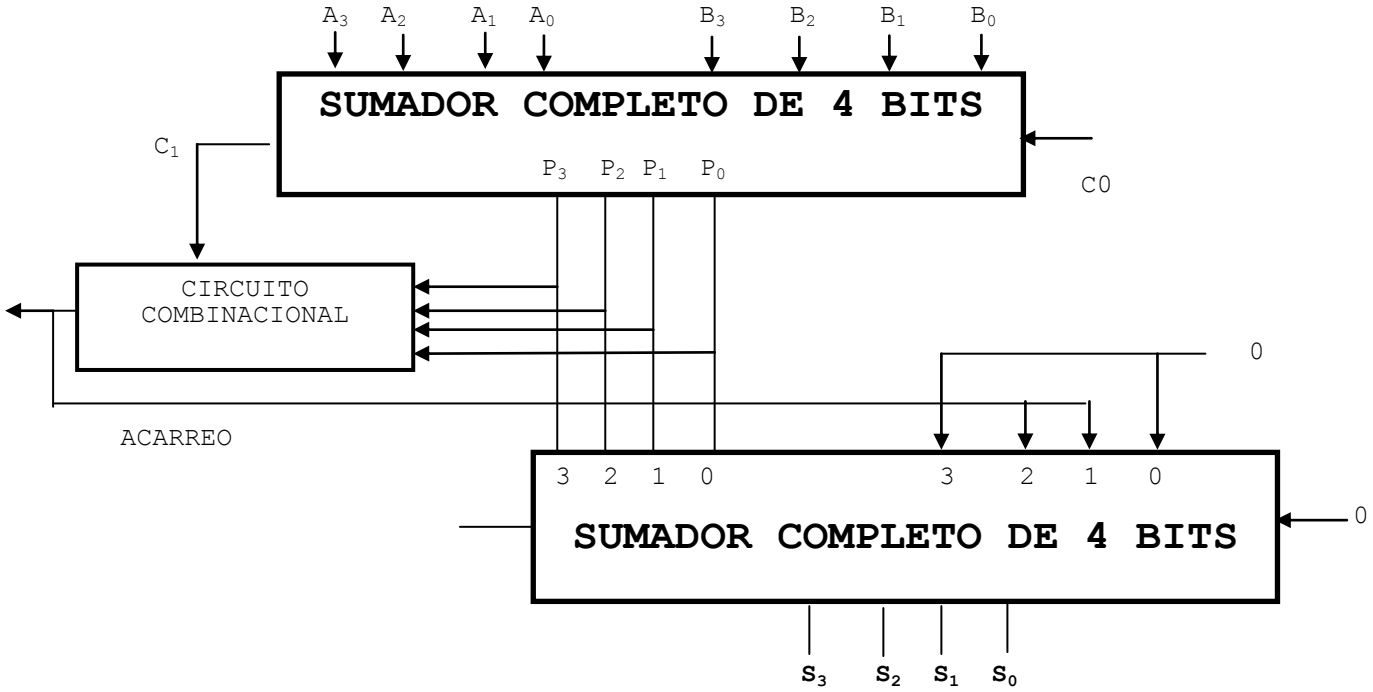
Sumador de arrastre anticipado

5.1.5. Sumador / Restador.

Si S/R=0 la operación es una suma. Las puertas XOR dejan pasar el dato sin más. Si S/R=1, las puertas XOR invierten el dato de B, con lo que obtiene el complemento a 1, y, además, introduce un 1 en el acarreo C_0 para tener el complemento a 2.



5.1.6. Sumador BCD.



El circuito combinacional detecta que la suma es superior a 9 y da una señal que sirve a la vez para:

1) Activar el acarreo posterior.

2) Restar 10 (o sumar 6, ya que los 4 bits menos significativos son los mismos) al número que dé como salida el sumador de 4 bits si la suma está entre 10 y 15, ambos incluidos, o sumar el número 6 a los 4 bits que salen del primer sumador si la suma está entre 16 y 19.

Puede parecer que es igual sumar 6 que restar 10. No, hay una diferencia de 16, pero si a un número en binario se le suma 16 (10000), los 4 bits menos significativos del resultado son los mismos, igual que si en decimal le sumamos a un número 10.000, las 4 últimas cifras no cambian.

La salida de acarreo del segundo sumador completo de 4 bits es irrelevante.

El diseño de dicho circuito responde a la siguiente tabla de verdad:

	C_1	P_3	P_2	P_1	P_0	ACARREO
0	0	0	0	0	0	0
1	0	0	0	0	1	0
2	0	0	0	1	0	0
3	0	0	0	1	1	0
4	0	0	1	0	0	0
5	0	0	1	0	1	0
6	0	0	1	1	0	0
7	0	0	1	1	1	0
8	0	1	0	0	0	0
9	0	1	0	0	1	0
10	0	1	0	1	0	1
11	0	1	0	1	1	1
12	0	1	1	0	0	1
13	0	1	1	0	1	1
14	0	1	1	1	0	1
15	0	1	1	1	1	1
16	1	0	0	0	0	1
17	1	0	0	0	1	1
18	1	0	0	1	0	1
19	1	0	0	1	1	1
20	1	0	1	0	0	X
…	…	…	…	…	…	X
…	…	…	…	…	…	X
31	1	1	1	1	1	X

cuya solución es:

$$ACARREO = C_1 + P_3 \cdot P_1 + P_3 \cdot P_2$$

Los casos comprendidos entre 20 y 31, ambos incluidos, no se darán nunca, ya que el mayor número posible resulta de sumar 9 + 9 + 1 = 19, en el caso de sumar los mayores dígitos BCD y suponiendo entrada de acarreo de una etapa anterior.

5.2. Multiplicador combinacional.

La multiplicación aritmética binaria coincide con el producto lógico. El resultado es 1 sólo cuando ambos operando son 1. El algoritmo de multiplicación utilizado en base decimal es también aplicable a base binaria. Ejemplo:

```
        1 1 0 0 1
          1 1 0 1 x
        ----------
        1 1 0 0 1
      0 0 0 0 0
    1 1 0 0 1
  1 1 0 0 1
-----------------
1 0 1 0 0 0 1 0 1
```

```
        1 1 0 0 1
          1 1 0 1 x
        ----------
        1 1 0 0 1
      0 0 0 0 0
    -------------
      0 1 1 0 0 1
    1 1 0 0 1
    -------------
      1 1 1 1 1 0 1
  1 1 0 0 1
  ---------------
1 0 1 0 0 0 1 0 1
```

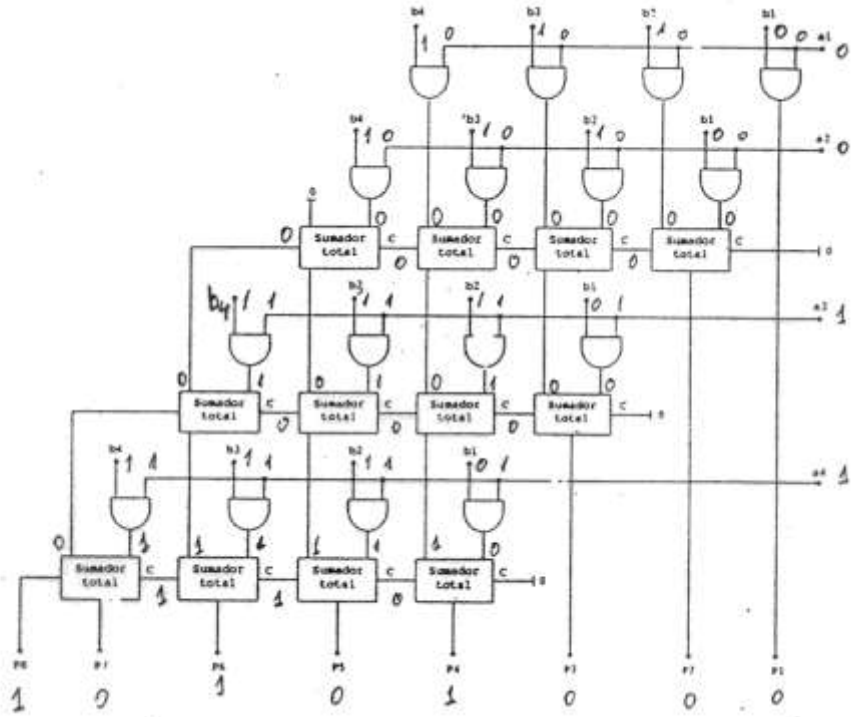
```
  25
  13 x
 ----
  75
 25
 ----
 325
```

Sumas totales de 1 bit **Sumas parciales de 4 bits**

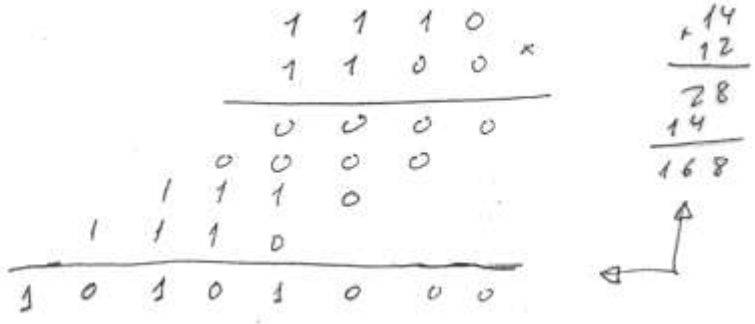
Según que las sumas parciales se realicen de golpe o por partes, hay dos circuitos que implementan esta operación: uno con sumadores completos de 1 bit y otro con sumadores completos de varios bits, respectivamente.

MULTIPLICADOR COMBINACIONAL

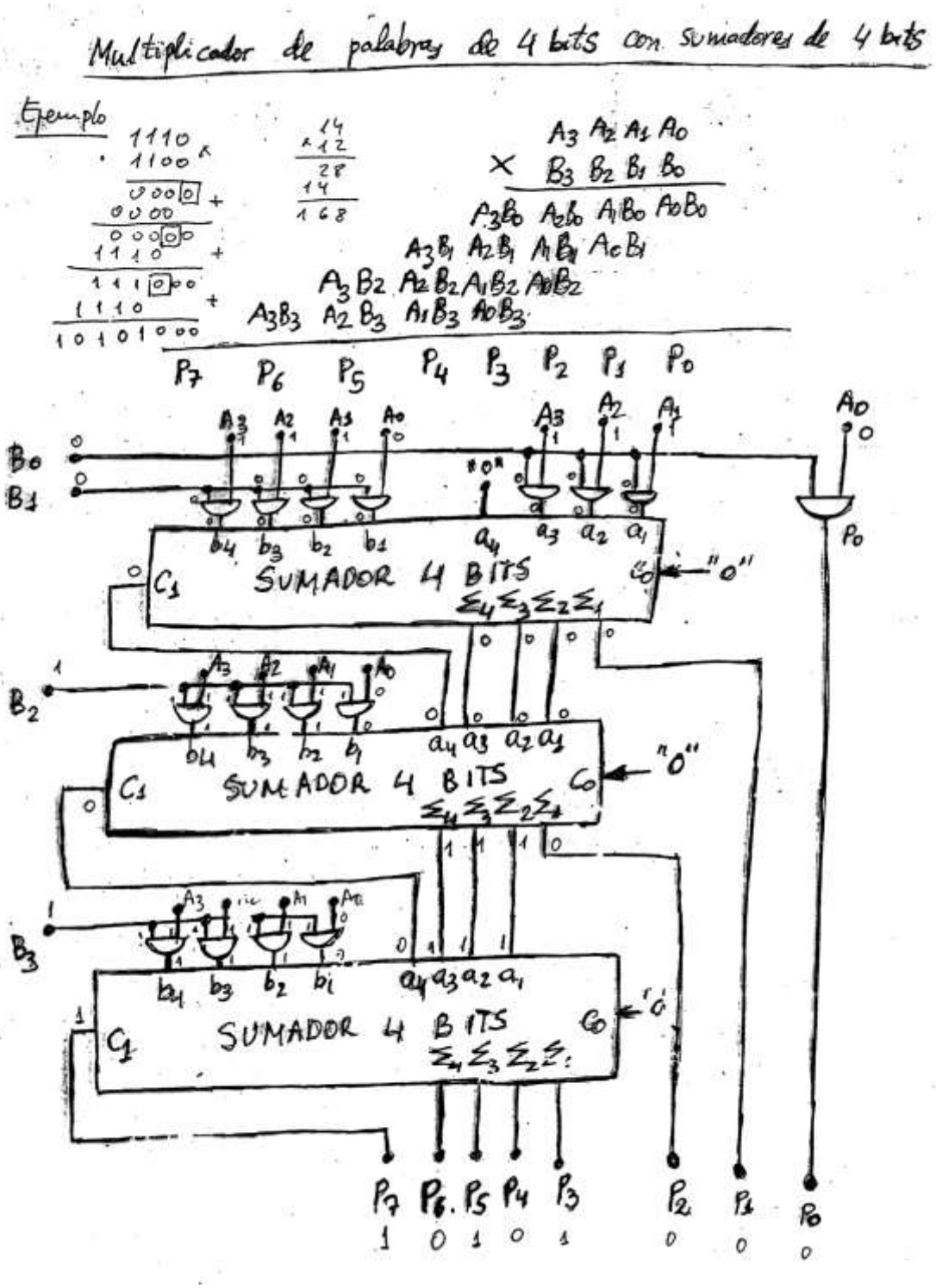
					B4	B3	B2	B1
					x A4	A3	A2	A1
P. parcial 1	=				A1B4	A1B3	A1B2	A1B1
P. parcial 2	=			A2B4	A2B3	A2B2	A2B1	
P. parcial 3	=		A3B4	A3B3	A3B2	A3B1		
P. parcial 4	=	A4B4	A4B3	A4B2	A4B1			
Producto final	=	P8	P7	P6	P5	P4	P3	P2	P1



Ejemplo



Multiplicador haciendo sumas totales con 1 bit



Multiplicador haciendo sumas parciales de 4 bits

5.3. Módulos Lógicos.

5.3.1. Comparadores.

a1	a0	b1	b0	Z1	Z2	Z3
0	0	0	0	0	1	0
0	0	0	1	0	0	1
0	0	1	0	0	0	1
0	0	1	1	0	0	1
0	1	0	0	1	0	0
0	1	0	1	0	1	0
0	1	1	0	0	0	1
0	1	1	1	0	0	1
1	0	0	0	1	0	0
1	0	0	1	1	0	0
1	0	1	0	0	1	0
1	0	1	1	0	0	1
1	1	0	0	1	0	0
1	1	0	1	1	0	0
1	1	1	0	1	0	0
1	1	1	1	0	1	0

Compara dos números y según cuál sea mayor, o si son iguales, activa una salida, dejando desactivadas las otras dos

Se pueden hacer comparadores de números binarios de 4 bits usando comparadores de 2 bits. Para ello, hay que tener en cuenta que al comparar dos números de 4 bits, los dos bits más significativos determinan cuál de los dos números es mayor, independientemente de los otros dos. Esto también ocurre en base 10.

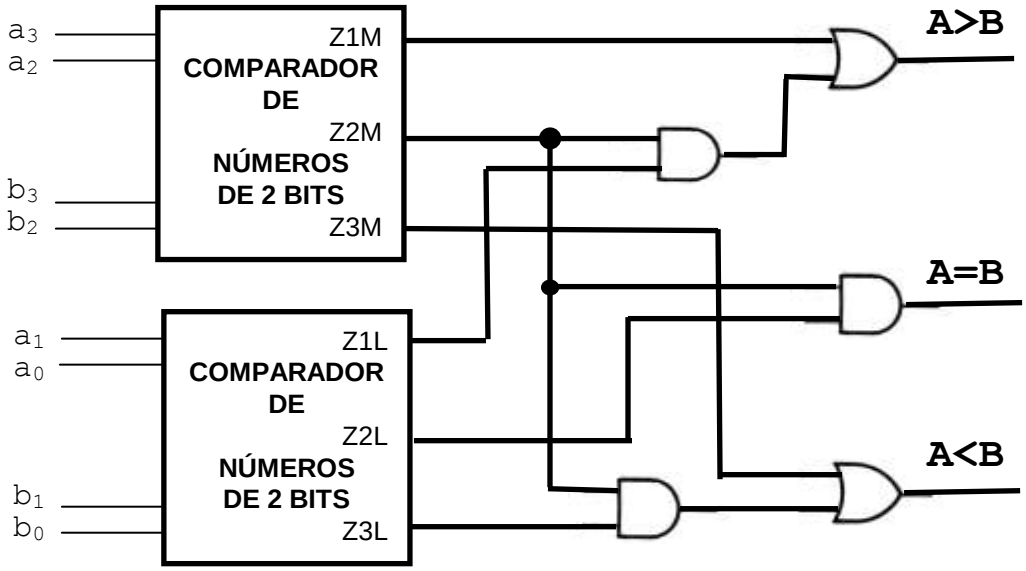
De la comparación de dos números A y B de 4 bits se pueden obtener los siguientes casos:

- A es mayor que B si los dos bits más significativos de A son mayores que los dos bits más significativos de B, independientemente de los dos bits menos significativos, o, si siendo iguales los dos bits más significativos, los dos bits menos significativos de A son mayores que los dos bits menos significativos de B.
- A es igual que B si los dos bits más significativos de A son iguales que los dos bits más significativos de B, y, además, los dos bits menos significativos de A son iguales que los dos bits menos significativos de B.
- A es menor que B si los dos bits más significativos de A son menores que los dos bits más significativos de B, independientemente de los dos bits menos significativos, o, si siendo iguales los dos bits más significativos, los dos bits menos significativos de A son menores que los dos bits menos significativos de B.

Si llamamos Z1M, Z2M y Z3M a las salidas A>B, A=B y A<B, respectivamente, del comparador de los bits más significativos y Z1L, Z2L y Z3L a las salidas A>B, A=B y A<B, respectivamente, del comparador de los bits menos significativos, las salidas Z1 (A>B), Z2 (A=B) y Z3 (A<B), se pueden expresar como:

$$Z1 = Z1M + Z2M \cdot Z1L$$
$$Z2 = Z2M \cdot Z2L$$
$$Z3 = Z3M + Z2M \cdot Z3L$$

Un comparador de 4 bits con comparadores de 2 bits se puede construir de la siguiente forma:



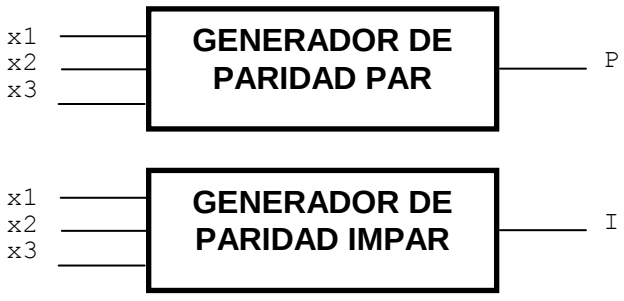
Comparador de 4 bits con dos comparadores de 2 bits

Los circuitos integrados comparadores pueden tener entradas >, =, < que transmiten el resultado de la entrada a la salida cuando las palabras de bits a comparar son iguales. Esto se utiliza para realizar comparaciones de un gran número de bits a partir de comparadores más pequeños, colocándolos en cascada.

5.3.2. Detectores y generadores de paridad.

Generador de paridad.

x1	x2	X3	P	I
0	0	0	0	1
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	0



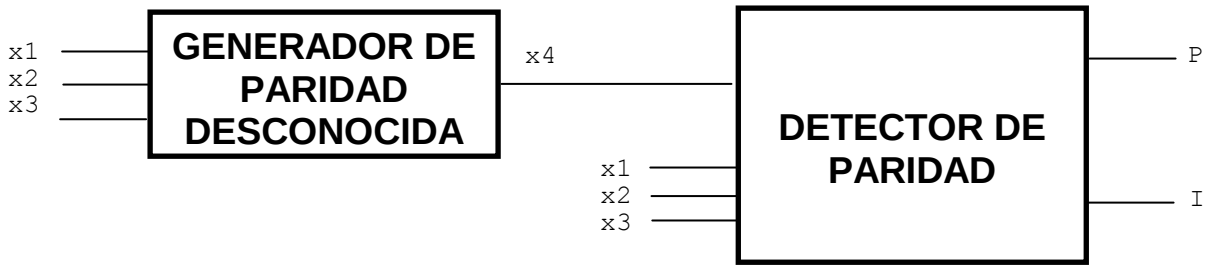
$$P = x1 \oplus x2 \oplus x3 \qquad I = \overline{x1 \oplus x2 \oplus x3}$$

Los bits de paridad constituyen una información redundante que sirve para detectar errores en 1 bit. Si el bit de paridad es el correcto, no ha habido error. Si algún bit se cambia de valor, el bit de paridad da resultado incorrecto, ha habido un error, aunque no se sabe en qué bit.

Sin embargo, la comprobación de paridad no detecta error si se cambian dos bits. Si el bit erróneo es el propio bit de paridad, sin que haya error en el resto de los bits, detecta error cuando en realidad no lo ha habido.

Detector de paridad.

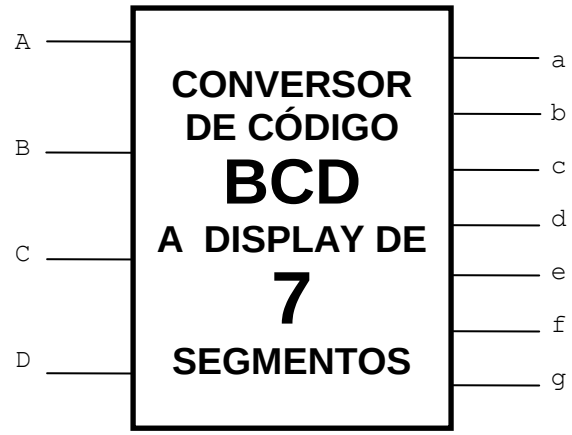
x1	x2	x3	x4	P	I
0	0	0	0	1	0
0	0	0	1	0	1
0	0	1	0	0	1
0	0	1	1	1	0
0	1	0	0	0	1
0	1	0	1	1	0
0	1	1	0	1	0
0	1	1	1	0	1
1	0	0	0	0	1
1	0	0	1	1	0
1	0	1	0	1	0
1	0	1	1	0	1
1	1	0	0	1	0
1	1	0	1	0	1
1	1	1	0	0	1
1	1	1	1	1	0

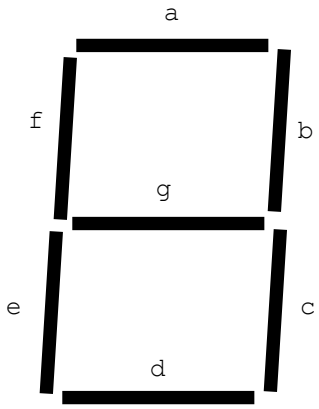


$$P = \overline{x1 \oplus x2 \oplus x3 \oplus x4} \qquad I = x1 \oplus x2 \oplus x3 \oplus x4$$

5.3.3. Conversores de códigos.

Ejemplo: veamos un conversor de BCD a display de 7 segmentos



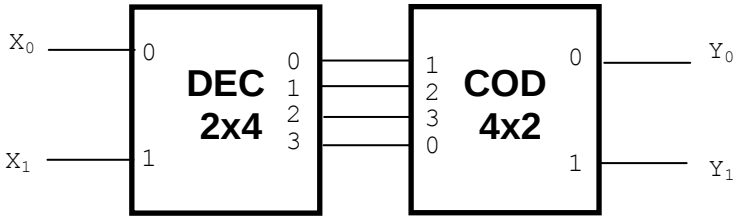


D	C	B	A	a	b	c	d	e	f	g
0	**0**	**0**	**0**	**1**	**1**	**1**	**1**	**1**	**1**	**0**
0	**0**	**0**	**1**	**0**	**1**	**1**	**0**	**0**	**0**	**0**
0	**0**	**1**	**0**	**1**	**1**	**0**	**1**	**1**	**0**	**1**
0	**0**	**1**	**1**	**1**	**1**	**1**	**1**	**0**	**0**	**1**
0	**1**	**0**	**0**	**0**	**1**	**1**	**0**	**0**	**1**	**1**
0	**1**	**0**	**1**	**1**	**0**	**1**	**1**	**0**	**1**	**1**
0	**1**	**1**	**0**	**1**	**0**	**1**	**1**	**1**	**1**	**1**
0	**1**	**1**	**1**	**1**	**1**	**1**	**0**	**0**	**0**	**0**
1	**0**	**0**	**0**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
1	**0**	**0**	**1**	**1**	**1**	**1**	**0**	**0**	**1**	**1**
1	0	1	0	0	0	0	0	0	0	0
1	0	1	1	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0
1	1	0	1	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0

Se puede hacer un conversor de código usando una pareja de módulos: Decodificador y Codificador. Los códigos pueden tener distinto número de bits.

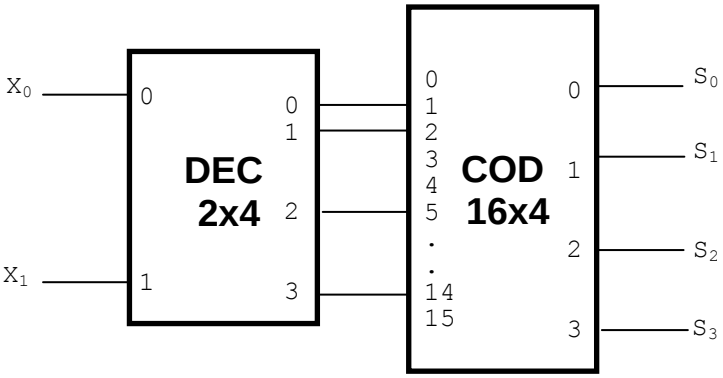
Ejemplo: convertir el código A en el código B según la tabla adjunta:

Código A		Código B	
X_1	X_0	Y_1	Y_0
0	0	0	1
0	1	1	0
1	0	1	1
1	1	0	0



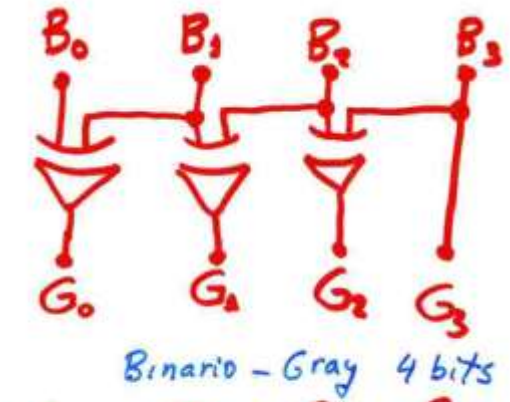
Puede ocurrir que los dos códigos no tengan el mismo número de bits. Lo vemos en este otro ejemplo: convertir el código A en el código B según la tabla adjunta:

Código A		Código B			
X_1	X_0	S_3	S_2	S_1	S_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	1
1	1	1	1	1	0

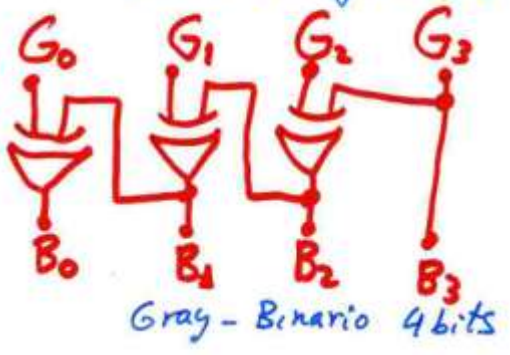


Hay circuitos integrados que realizan conversiones de códigos BCD – binario y viceversa. Ejemplo: el circuito integrado 74184 convierte 6 bits BCD (4 de menos peso y 2 de más peso, es decir, el mayor número que se puede representar es el 39) en binario de 6 bits. Por ejemplo, 110011, que representaría al 33, lo convierte a 33 en binario, que es 100001. Por el contrario, el circuito 74185 convierte un número binario de 6 bits en dos BCD (con 8 bits); por ejemplo, el número 50, que en binario sería 110010, lo expresa como 0101 0000.

Otro ejemplo de conversor de código es un conversor de 4 bits (siendo G_3 y B_3 los bits más significativos de los códigos Gray y binario, respectivamente) que cambia un código binario puro en Gray y viceversa. Se rige por las siguientes ecuaciones:



$$G_3 = B_3$$
$$G_2 = B_3 \oplus B_2$$
$$G_1 = B_2 \oplus B_1$$
$$G_0 = B_1 \oplus B_0$$



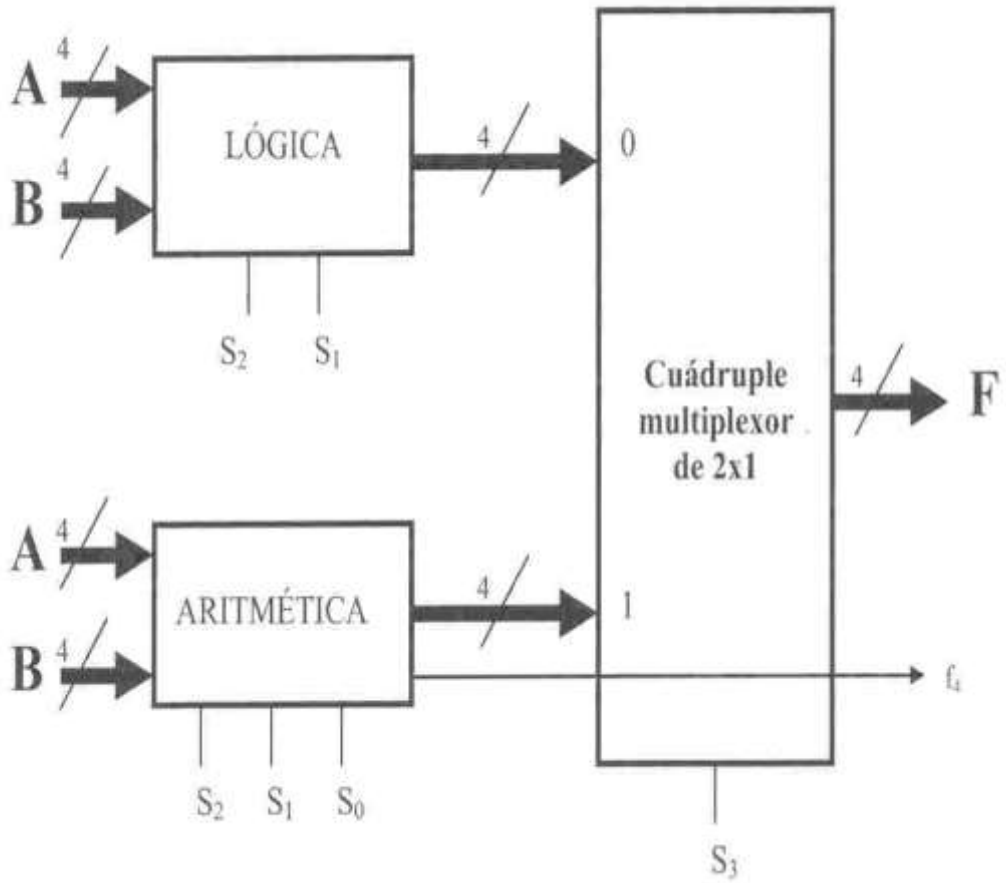
$$B_3 = G_3$$
$$B_2 = G_3 \oplus G_2 = B_3 \oplus G_2$$
$$B_1 = B_2 \oplus G_1 = G_3 \oplus G_2 \oplus G_1$$
$$B_0 = B_1 \oplus G_0 = G_3 \oplus G_2 \oplus G_1 \oplus G_0$$

5.4. Unidad Aritmética-Lógica combinacional elemental.

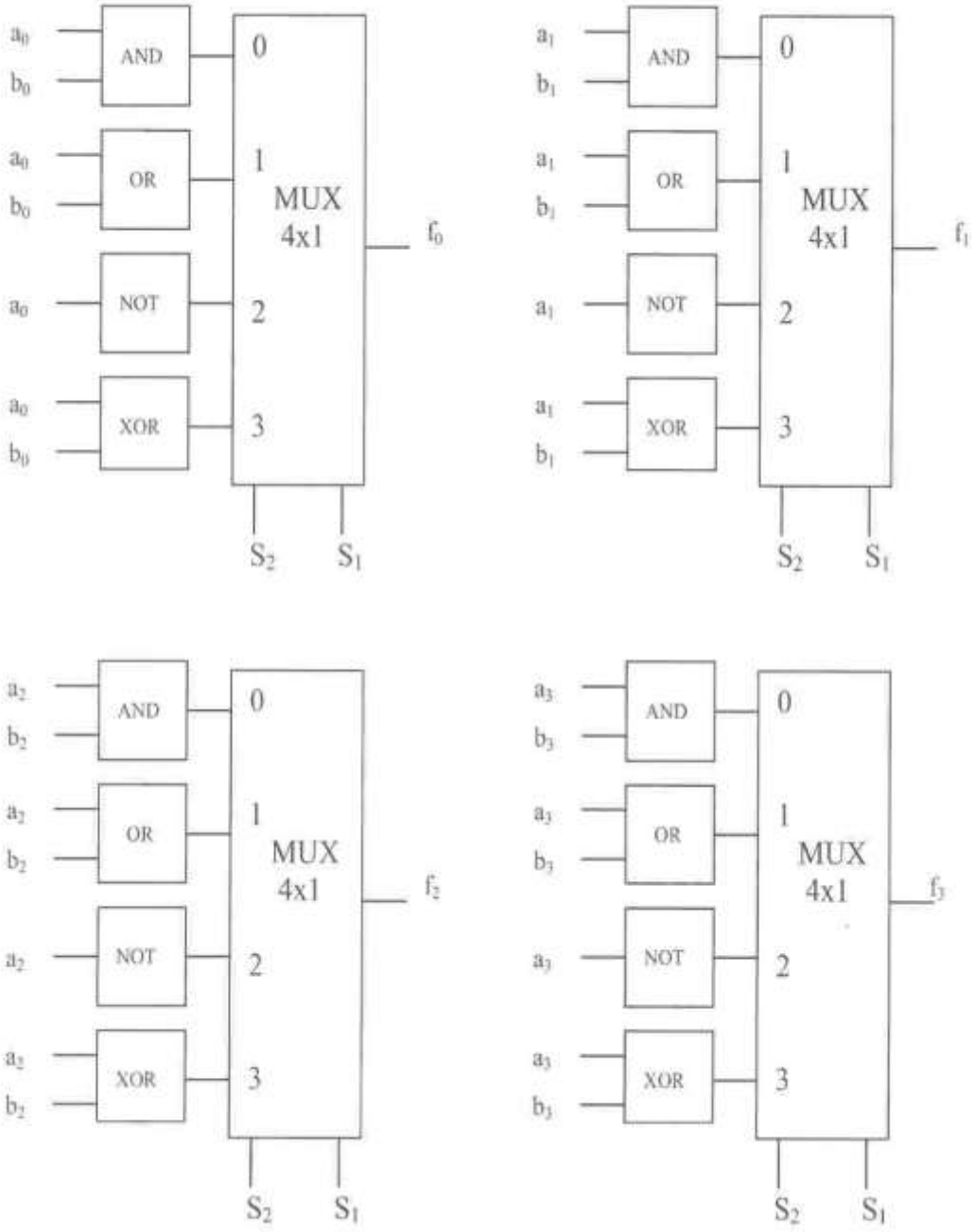
Se trata de diseñar una ALU con las siguientes características: Datos de 4 bits, A y B, 4 líneas de selección S3 hasta S0 para controlar la operación de la siguiente forma

S3=0 (LÓGICA)			S3=1 (ARITMÉTICA)			
S2	**S1**	**Operación**	**S2**	**S1**	**S0**	**Operación**
0	0	A and B	0	0	0	A + B
0	1	A or B	0	0	1	A + B + 1
1	0	NOT A	0	1	0	A - 1
1	1	A xor B	0	1	1	A (sumando 1111+1)
			1	0	0	A (sumando 0000+0)
			1	0	1	A + 1
			1	1	0	A - B - 1
			1	1	1	A – B

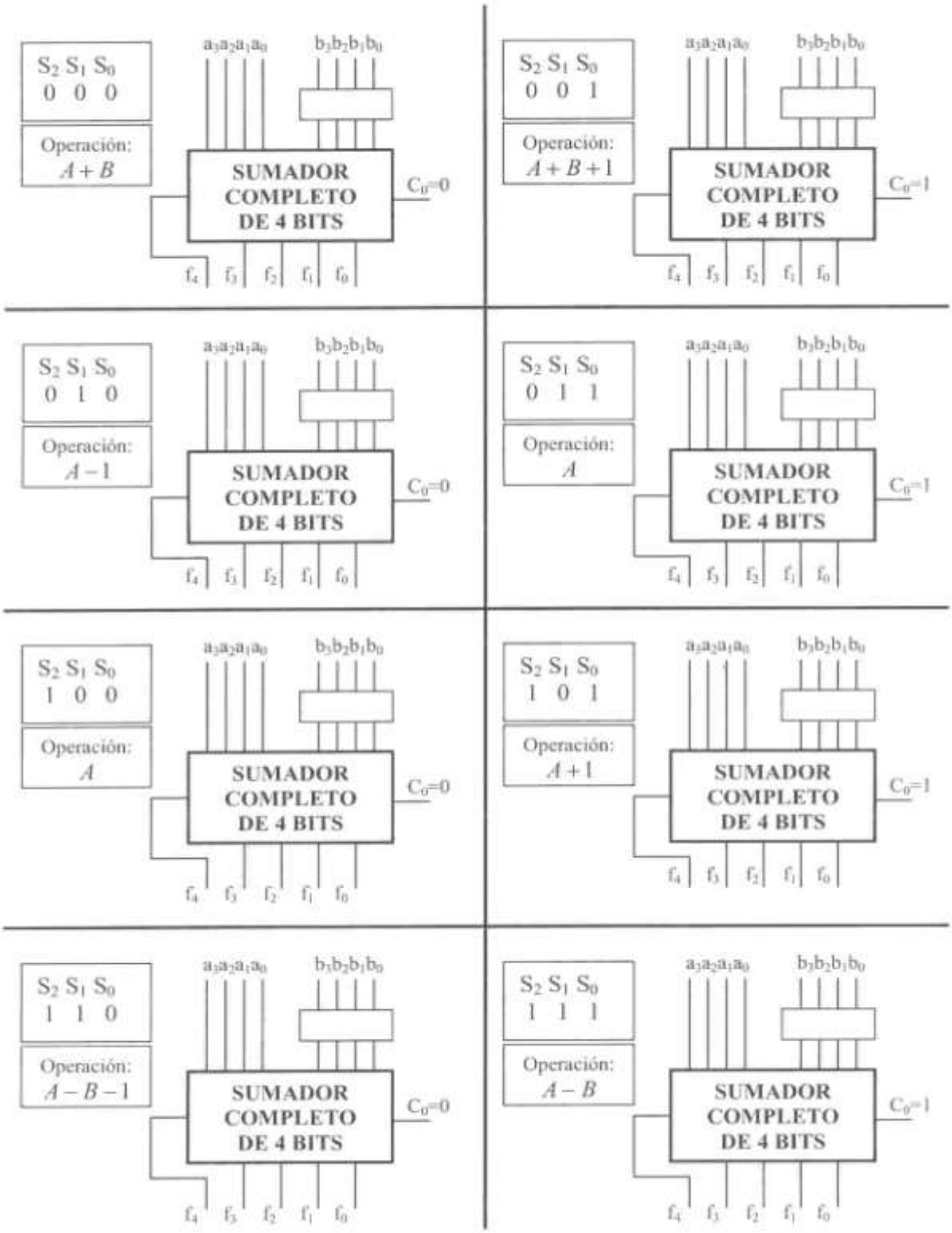
Diagrama de bloques



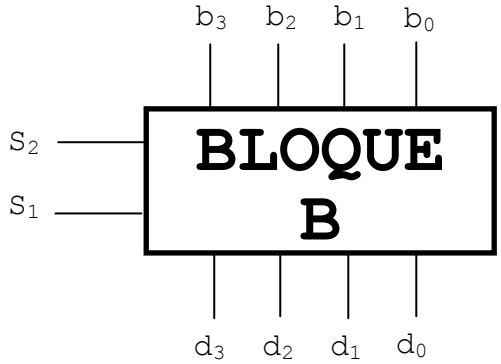
PARTE LÓGICA



PARTE ARITMÉTICA



Se puede observar que $C_0 = S_0$. ¿Qué hay que sumarle a A en función de S_2 y S_1?



La operación a realizar por el Bloque B según las señales de control S_2 y S_1 es:

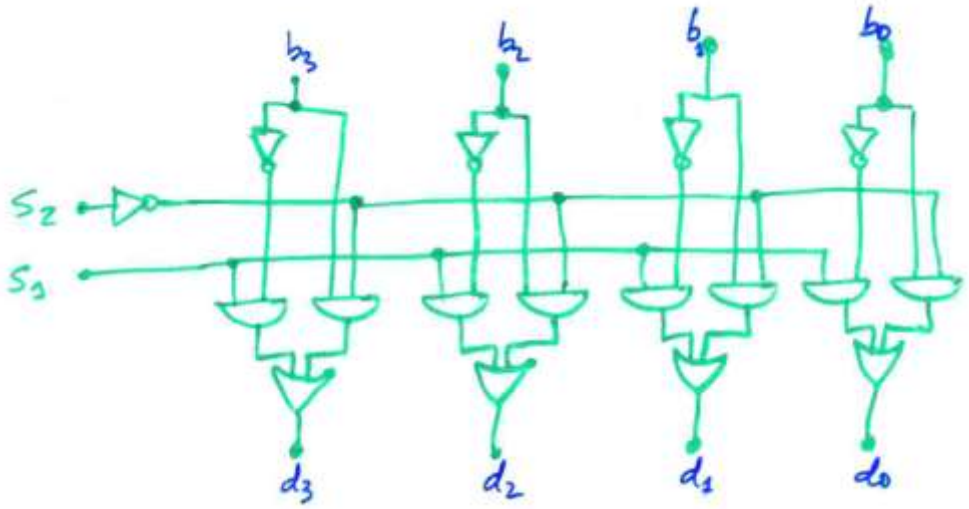
S_2	S_1	SUMAR
0	0	B
0	1	*1111*
1	0	*0000*
1	1	$\overline{B}$

que, desarrolada, se puede expresar como:

S_2	S_1	b_i	d_i
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

en donde la función booleana d_i puede expresarse en función de las variables S_2, S_1 y b_i de la forma

$$d_i = \overline{S_2} \cdot b_i + S_1 \cdot \overline{b_i}$$



EJERCICIOS PROPUESTOS.

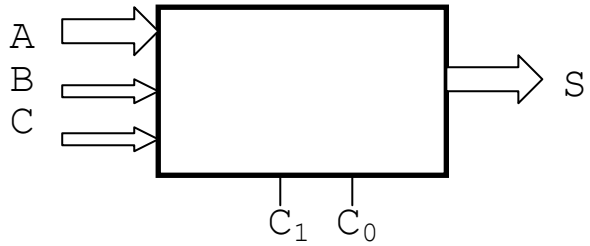
1) Dados tres números binarios de 4 bits, A, B y C, codificados en binario natural, diseñar un circuito que realice la suma de A con el mayor de B y C. Si B = C, el resultado debe ser A. Para ello usar sumadores binarios y comparadores y las puertas lógicas que sean necesarias.

2) Un sistema digital con dos señales de control C_1 y C_0 tiene en su entrada 3 buses **(*)** A, B y C de 4, 3 y 3 bits, respectivamente. A representa un número expresado en BCD exceso 3, B representa un número en binario natural y C representa un número en binario natural.

La salida del sistema es un bus de 4 bits según la tabla:

C_1 C_0	SALIDA
0 0	Todos ceros
0 1	A codificado en binario natural
1 0	La mitad de B (si es impar, la parte entera de la mitad)
1 1	El doble de C

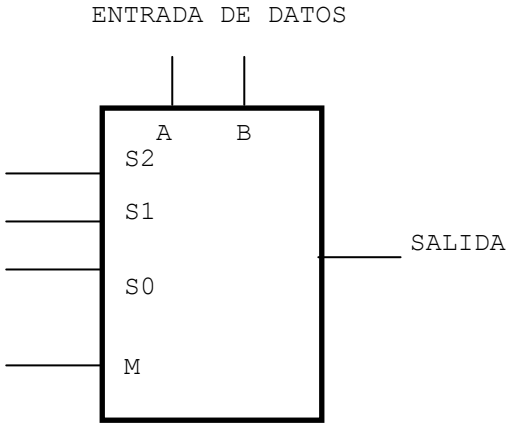
Diseñar el circuito usando sólo multiplexores y un sumador completo de 4 bits, y explicando la solución del diseño.



(*) NOTA: Un bus es un conjunto de líneas cada una de las cuales lleva información de 1 bit.

3) Se dispone de una ALU descrita en la tabla adjunta con tres entradas S_2,S_1,S_0 que seleccionan la función a realizar y una cuarta entrada M que distingue si la función es lógica o aritmética. Diseñar un circuito que realice las cuatro funciones siguientes de forma secuencial y cíclica: $F1 = A + B$; $F2 = A + \overline{B}$; $F3 = (A + B) MAS (A \cdot \overline{B})$; $F4 = A \oplus B$ con las siguientes herramientas. Se dispone siempre de una ALU y de módulos contadores necesarios).

NOTA: *"MAS"* significa SUMA ARITMÉTICA, mientras que *"+"* significa SUMA LÓGICA. El signo *"-"* es la resta aritmética.



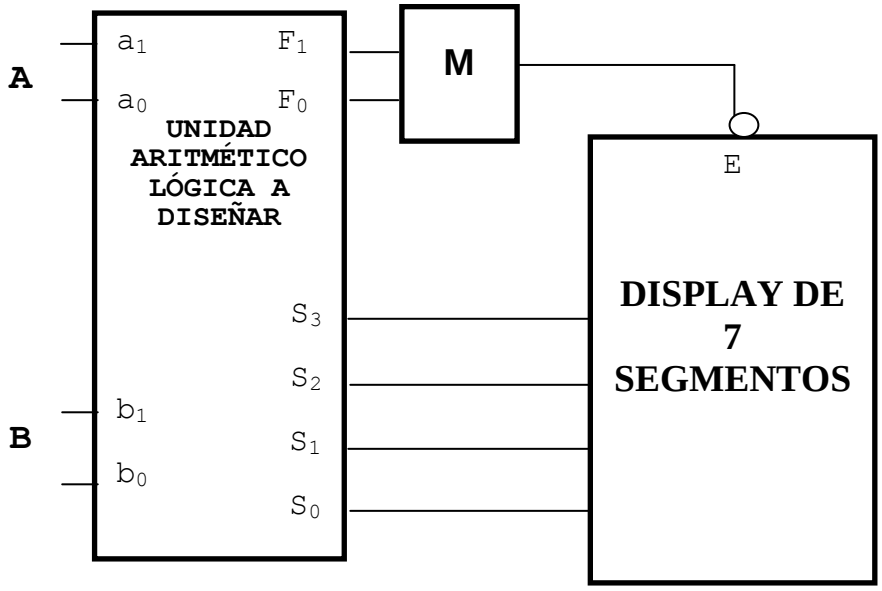
1) Con puertas lógicas.
2) Con módulos combinacionales: decodificadores y codificadores.

Descripción del funcionamiento de la ALU

$S_2\ S_1\ S_0$	*M=1 Funciones Lógicas*	*M=0 Funciones Aritméticas*
0 0 0	$F=\overline{A}$	$F=A$
0 0...1	$F=\overline{(A+B)}$	$F=A+B$
0 1 0	$F=\overline{A}\cdot B$	$F=A+\overline{B}$
0 1 1	$F=0$	$F=A-1$
1 0 0	$F=\overline{(A\cdot B)}$	$F=(A)MAS(A\cdot\overline{B})$
1 0 1	$F=\overline{B}$	$F=(A+B)MAS(A\cdot\overline{B})$
1 1 0	$F=A\oplus B$	$F=A-B-1$
1 1 1	$F=A\cdot\overline{B}$	$F=A\cdot\overline{B}-1$

4) Se trata de diseñar una Unidad Aritmético Lógica con las siguientes características: las entradas serán dos números A (a_1,a_0) y B (b_1,b_0) de 2 bits cada uno. Las salidas serán 4 líneas S_3, S_2, S_1, S_0 más dos líneas adicionales F_1 y F_0. El funcionamiento es: Si $A > B$, en las líneas S_3, S_2, S_1, S_0 estará el producto aritmético de A y B, siendo $F_1 = 1$ y $F_0 = 0$. Si $A = B$ la salida será igual que en el caso anterior salvo que $F_1 = 0$. Si $A < B$, $S_3 = a_1 + b_1$, $S_2 = a_0 \cdot b_0$, $S_1 = \overline{(a_1 + b_0)}$, y $S_0 = \overline{a_0 \cdot b_1}$, siendo $F_1 = 0$ y $F_0 = 1$.

NOTA: ("+" es la suma lógica, "·" es el producto lógico).



1. Encontrar la tabla de verdad que resuelve el problema (0,4 puntos).

2. Implementar las funciones con los elementos solicitados:
   - F_1 usando sólo puertas NAND. (0,3 puntos).
   - F_0 como suma de productos. (0,2 puntos).
   - S_3 usando sólo puertas NOR. (0,3 puntos).
   - S_2 como producto de sumas. (0,2 puntos).
   - S_1 con un DEC 4x16 con salidas activas a nivel alto. (0,2 puntos).
   - S_0 con un MUX 8x1. (0,2 puntos).

3. Contenido del bloque M sabiendo que el número se debe ver cuando la operación realizada haya sido una multiplicación aritmética, siendo E la señal de habilitación, activa a nivel bajo.