7. SISTEMAS ARITMÉTICOS

I TEORÍA

7.1 ARITMÉTICA BINARIA

7.1.1 OPERACIONES BÁSICAS

El fundamento de este tipo de aritmética es similar al de la decimal, aunque más simple debido a que sólo se trabaja con dos números el 0 y el 1. Las reglas principales se resumen en la siguiente tabla.

SUMA	RESTA	MULTIPLICACIÓN	DIVISIÓN
0 + 0 = 0	0 - 0 = 0	0 x 0 = 0	0 : 0 = Sin definir
0 + 1 = 1	0 - 1 = 1 con acarreo	0 x 1 = 0	0 : 1 = 0
1 + 0 = 1	1 - 0 = 1	1 x 0 = 0	1 : 0 = ∞
1 + 1 = 1 con acarreo	1 - 1 = 0	1 x 1 = 0	1 : 1 = 1

Ejemplos:

Suma:	Resta:	Multiplicación:	División:
1 0 1 1 0 1 + 1 1 0 0 1 1 0 0 0 1 1 0	1 0 0 1 0 - 1 0 0 1 0 1 0 0 1	1 1 0 1 x 1 0 1 1 1 0 1 0 0 0 0 1 1 0 1 0 1 0 0 0 0 1	1 0 1 1 0 1 1 | 1 1 1 1 0 0 0 1 1 0 1 0 0 1 1 1 0 0 0

7.1.2 NÚMEROS CON SIGNO

En electrónica digital para representar el signo de un número se añade un bit a la izquierda para indicar el signo, 0 si es positivo y 1 si es negativo.

Ejemplo:

$13_{10} = 0\ 1101_2$ $\qquad$ $-13_{10} = 1\ 1101_2$

En la práctica, los números negativos no se ponen como en el ejemplo anterior ya que dificulta el diseño de circuitos aritméticos en los que deban aparecer este tipo de números.

Para solucionar este problema, los números negativos se codifican empleando los conceptos de "complemento a la base" y "complemento a la base menos uno", que a continuación se detallan.

a) Complemento a la base: Dado un número N de n dígitos expresado en un sistema de numeración de base b, se define el complemento a la base como el número $C_b(N)$ a:

$$C_b(N) = b^n - 1$$

Ejemplos:

$N = 280_{10} \Rightarrow C_{10}(280) = 10^3 - 280 = 720$
$N = 10110_2 \Rightarrow C_2(10110) = 10^5 - 10110 = 01010$

b) Complemento a la base menos uno: Como su propio nombre indica:

$$C_{b-1}(N) = C_b(N) - 1 = b^n - N - 1$$

Ejemplos:

$N = 280_{10} \Rightarrow C_9(280) = 10^3 - 280 - 1 = 719$
$N = 10110_2 \Rightarrow C_1(10110) = 10^5 - 10110 - 1 = 01001$

En binario, para los complementos a la base y a la base menos uno hablaremos de complemento a doses y a unos respectivamente. Podemos comprobar que el complemento a unos de un número binario se obtiene cambiando los unos por ceros y los ceros por unos. Para obtener el complemento a doses bastará con sumar uno al complemento a unos.

Ejemplo:

$13_{10} = 0\ 1101_2$ $\quad -13_{10} = 1\ 0010_2$ (en complemento a unos)
$\quad -13_{10} = 1\ 0011_2$ (en complemento a doses)

El interés en codificar los números binarios negativos en complemento a doses está en la facilidad de diseñar circuitos que lo realicen y en la siguiente propiedad matemática:

$$Ab - Bb = Ab + C_b(B)$$

Convertir una resta en suma facilita el diseño de circuitos aritméticos digitales como veremos más adelante.

7.1.3 NÚMEROS EN COMA FLOTANTE

Matemáticamente un número **N** expresado en coma flotante se indica como:

$$N = Sm\ M \cdot b^{SeE}$$

Donde **M** es el valor absoluto de la mantisa, **E** es el valor absoluto del exponente, **Sm** es el signo de la mantisa, **Se** el signo del exponente, y **b** la base. La forma normalizada de expresar un número en forma de coma flotante, es situando la coma en la mantisa antes de la primera cifra significativa distinta de cero.

Ejemplos:

$-\ 2.234.500 = -\ 0{,}2345 \cdot 10^7$ $\qquad 0{,}000431 = 0{,}431 \cdot 10^{-3}$

Para los números en coma flotante binarios, el formato estándar puede tomar tres formas en función del número de bits que se necesiten para representarlos: simple precisión (32 bits), doble precisión (64 bits) y precisión ampliada (80 bits). Veremos sólo el primero de los formatos.

Los números binarios en coma flotante de simple precisión tienen la siguiente estructura:

- Bit de signo: 0 si es positivo y 1 si es negativo
- Exponente: 8 bits que representan un exponente desplazado (se suma 127 al exponente real)
- Mantisa: Parte fraccionaria expresada con 23 bits.

1 bit	8 bits	23 bits
Signo(S)	Exponente (E)	Mantisa (M)

El método para obtener el valor de un número N en coma flotante se expresa en la siguiente fórmula:

$$N = (-1)^S(1 + M)(2^{E-127})$$

Dado que el exponente puede ser cualquier número comprendido entre -126 y +128, pueden expresarse números extremadamente grandes y pequeños.

En el CD_Alumnos (\Miscelanea\Varios\floatlab) se encuentra un programa, el Floating Point Lab, que pasa valores decimales a binario en coma flotante y viceversa. Como ejemplo vamos a obtener el valor decimal del siguiente número

1	1 0 0 0 1 1 0 1	1 0 0 0 0 1 1 1 0 0 0 1 0 0 1 0 0 0 0 0 0 0

$$N = (-1)^1(1+ 0{,}100001110001001)(2^{141-127}) =$$
$$-(1 + 2^{-1} + 2^{-6} + 2^{-7} + 2^{-8} + 2^{-12} + 2^{-15})(2^{14)} = - 25.028{,}5$$

Se deja a cargo del alumno el proceso inverso, esto es, dado un número decimal, expresarlo en binario en coma flotante.

7.1.4 SUMA Y RESTA DE NÚMEROS CON SIGNO

La representación en complemento a doses es la forma más utilizada a la hora de operar en circuitos internos dedicados a realizar operaciones aritméticas.

Para la realización de estas operaciones hay que tener en cuenta el concepto de desbordamiento (*overflow*), que puede aparecer cuando se opera con números de n bits y el resultado es necesario representarlo con n+1 bits. En la suma de números con signo, pueden presentarse los cuatro casos siguientes:

a) Ambos números son positivos ⇒ Puede haber rebosamiento.
b) Ambos números son negativos ⇒ Puede haber rebosamiento.
c) El sumando positivo es mayor que el valor absoluto que el negativo ⇒ No puede haber rebosamiento.

d) El sumando positivo es mayor que el valor absoluto que el negativo ⇒ No puede haber rebosamiento.

Ejemplos:

```
  1 0 0 1 0 1 0      (+74)          0 0 0 1 0 0 1      (+9)
+ 1 0 1 0 0 1 1    +(+83)         1 0 0 0 0 1 0    +(66)
1 0 0 1 0 1 0 1    (+157)         1 0 0 1 0 1 1    (+75)
```

Para realizar la operación resta (R), realizaremos la siguiente operación matemática para convertirla en suma:

$$R = A - B = A + (-B)$$

La resta, por tanto, se realizará sumando al minuendo el complemento a doses del sustraendo (que deberá tener el mismo número de bits que el minuendo).

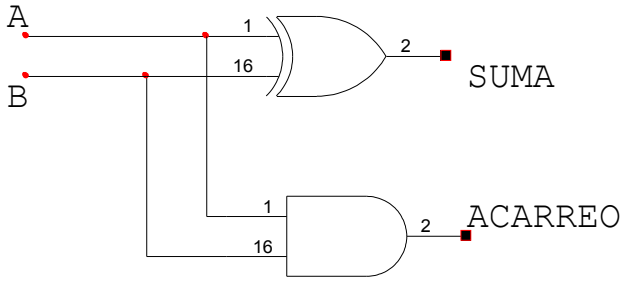
```
0  1 0 1 0 0 0 1     (+81)        0  0 1 1 0 0 0 0     (+48)
1  1 1 1 0 0 0 0    +(-16)        1  0 1 0 1 1 1 1    +(-81)
-------------------  ------       -------------------  ------
0  1 0 0 0 0 0 1     +  65        1  1 0 1 1 1 1 1      -33
```

7.2 CIRCUITOS ARITMÉTICOS

A partir de los circuitos aritméticos básicos se diseña IC's capaces de realizar cualquier operación matemática, incluso a elementos que realizan diversas tareas lógicas y aritméticas. En el presente punto daremos una visión general de todos ellos.

7.2.1 SEMISUMADOR Y SUMADOR COMPLETO

Es el circuito aritmético más sencillo de todos y realiza la tabla de verdad de la suma aritmética. Su esquema es el siguiente:

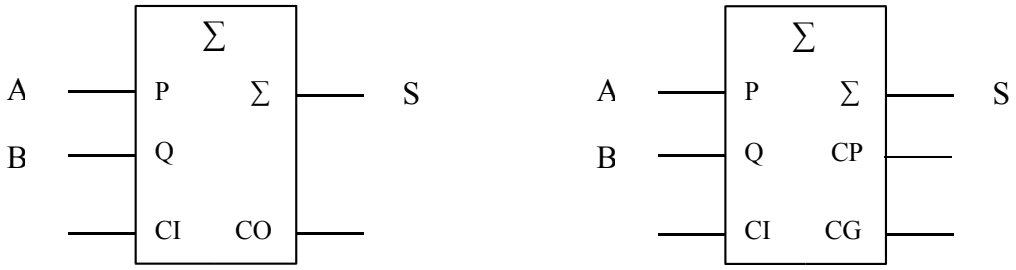


Para sumas superiores a un bit, el anterior circuito no nos valdría ya que es incapaz de tomar los acarreos que se producen en la suma. El sumador completo o sumador total puede sumar dos bits, A y B, más el acarreo de entrada CI. Si formamos su tabla de verdad y simplificamos obtendremos, identificando S como la salida de suma y CO con el acarreo de salida:

$$S = A \oplus B \oplus CI \qquad CO = AB + (A + B)CI = G + PCI$$

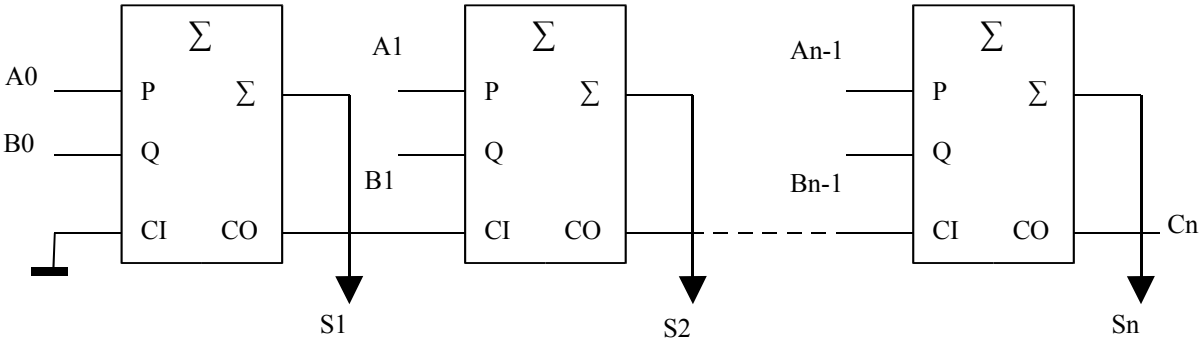
Al producto de los bits se le denomina generador de acarreo, G, y a su suma lógica propagador de acarreo, P.

A continuación se representa el esquema de un sumador total con salida de acarreo CO y otro con salidas de generación y propagación de acarreo, CG y CP respectivamente:



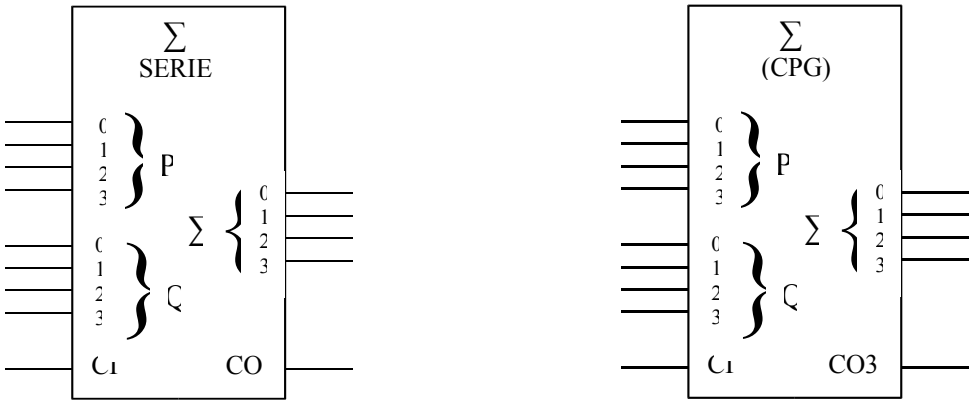
7.2.2 SUMADORES PARALELO

Si conectamos sumadores totales tal y como se muestra en el circuito de la siguiente figura, obtendremos un sumador paralelo de n bits con generación del acarreo en serie.

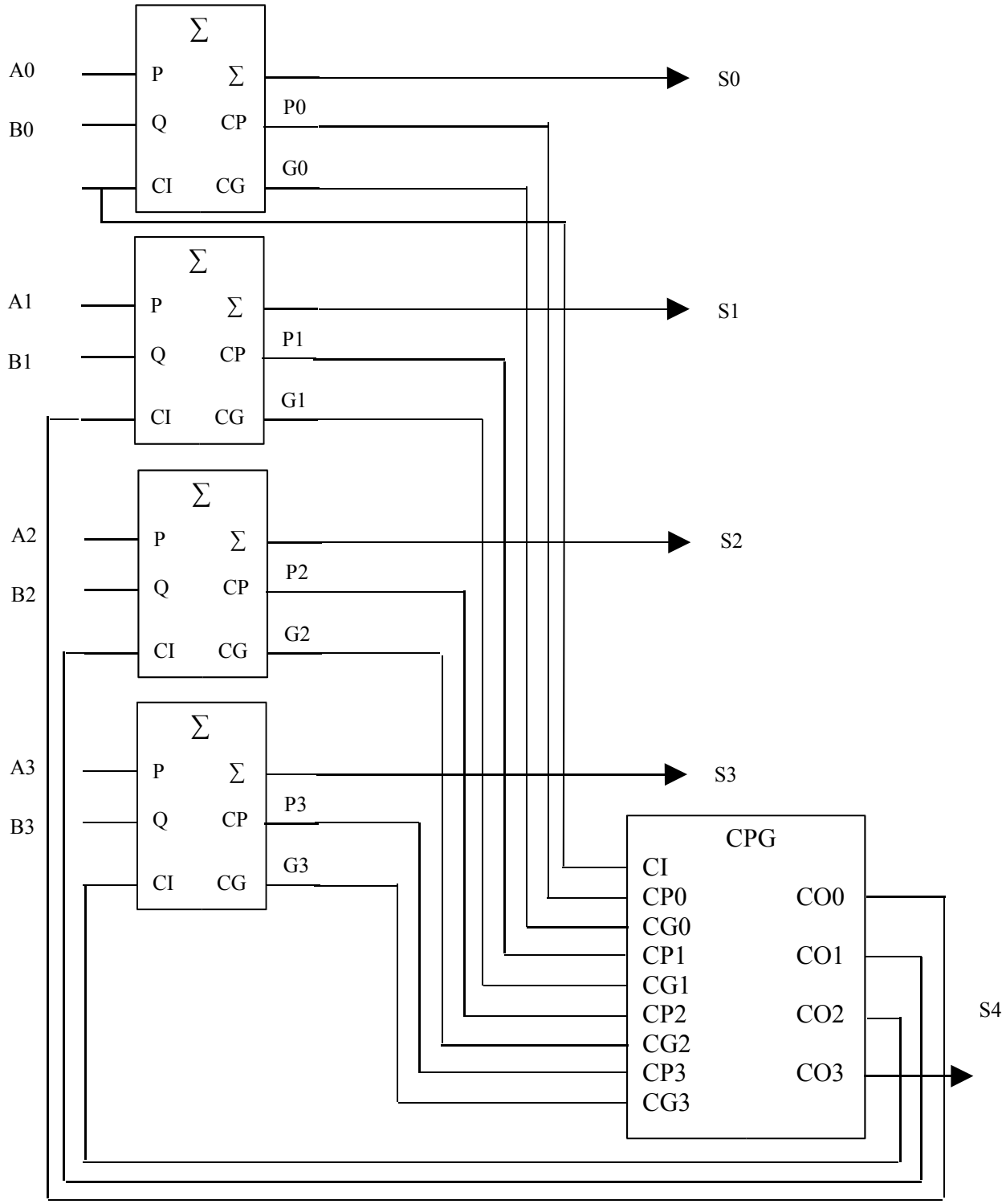


La generación del acarreo en serie implica que el tiempo necesario para que se realice la suma sea igual a n veces el tiempo medio que tarda en generarse el acarreo en un sumador. Cuando se necesite reducir al mínimo el tiempo de operación, este montaje no será el adecuado y en su lugar se utilizará el sumador con generación de acarreo en paralelo.

A continuación se muestran los esquemas normalizados para un sumador de 4 bits con generación de acarreo en serie y en paralelo:



El sumador con generación de acarreo en paralelo integra sumadores totales con salidas de acarreo CP y CG conectados a un CPG o generador-propagador de acarreo (*look-ahead carry block*) tal y como podemos ver en el siguiente esquema.



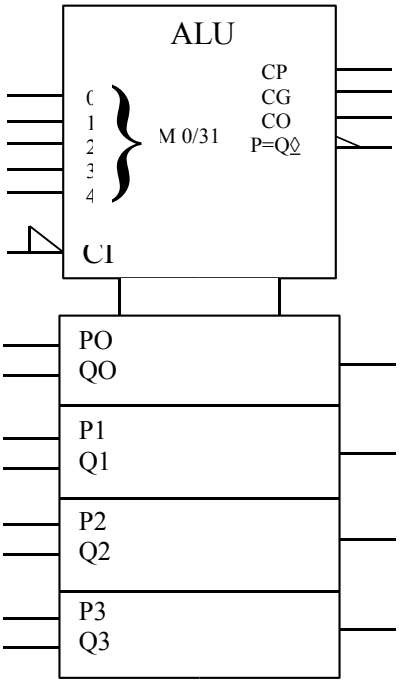
Como puede observarse, el CPG toma las salidas de acarreo de cada sumador para generar todos los acarreos. Los distintos acarreos de salida vienen dados por la siguiente ecuación:

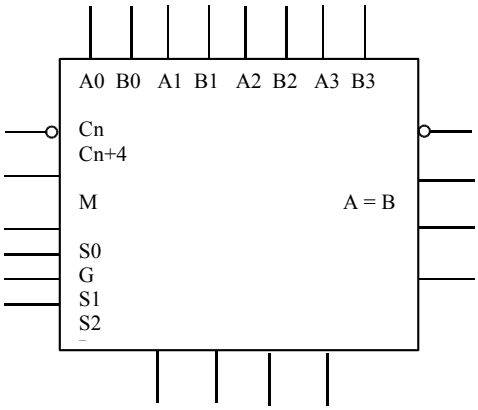
$$COi = G_i + P_iC_{i-1}$$

7.2.3 UNIDADES ARITMÉTICAS Y LÓGICAS

La necesidad de realizar operaciones lógicas y aritméticas en ciertos sistemas digitales llevó a la fabricación de circuitos integrados capaces de realizar ambos tipos de operaciones. Estos circuitos reciben el nombre de unidades aritméticas y lógicas, más conocidas por sus siglas en inglés ALU (Arithmetic Logic Unit).

A continuación se muestran los símbolos normalizado y no normalizado de una ALU de 4 bits:





Los terminales podemos agruparlos del siguiente modo:

a) **Operandos**: Se denomina de esta manera las entradas de datos de n bits (Pi, Qi y Ai, Bi).
b) **Resultado**: Las salidas (Fi), también de n bits, dará el resultado de la operación realizada.
c) **Selección de Operación**: Es el conjunto de bits mediante el cual se selecciona la operación a realizar (M 0/31 y Si).
d) **Acarreo de entrada**: Es una línea que está ligada a los bits de menor peso de los operandos (CI y Cn) que tendrán efecto en las operaciones aritméticas.
e) **Salidas auxiliares**: Son salidas que proporcionan datos auxiliares a las operaciones realizadas como pueden ser: acarreos de salida, desbordamientos, etc…

Las ALU´s comerciales trabajan habitualmente con cuatro bits pero pueden expandirse utilizando integrados semejantes y colocando las señales de control en paralelo según indique el fabricante.

7.3 GUÍA DE DISPOSITIVOS ARITMÉTICOS

A continuación se muestra una tabla con la mayoría de los dispositivos aritméticos que podemos encontrar en la serie 74 de TTL y en la 40 y 45 de CMOS. En el CD_Alumnos se encuentran las características de los números marcados en negrita.

TYPE	DESCRIPTION	TYPE	DESCRIPTION
7483	4-bits parallel adder	**74181**	4-bits ALU
74283		74381	
4008		74382	
4560	BCD Adder	74881	
4561	9's complementer	4581	
4527	BCD multiplier	74182	Look-ahead carry generators
4554	2x2-bit parallel binary multiplier	4582	

II IMPROVE YOUR TECHNICAL ENGLISH

1 Combinational Multiplier

In this section we look at the design of multiplier circuitry. The methods we introduce are combinational, although alternative methods based on circuits with state are also possible. However, the fastest circuits for multiplication use just the techniques we will be discussing here.

Basic Concept Throughout this section, we will look only at multiplication techniques for unsigned numbers. Alternatively, the hardware we present is suitable[1] for sign and magnitude multiplication, but we concentrate on the manipulation of the magnitude part. Recall that the two numbers involved in a multiplication are called the *multiplicand* and the *multiplier*.

The process of binary multiplication is best illustrated with an example. In this case, the multiplicand is 1101_2 (13) and the multiplier is 1011_2 (11):

```
multiplicand  1101 (13)
multiplier    1011 (11)
              1101
             1101
            0000
           1101
          1 000 1111 (143)
```

Each bit of the multiplier is multiplied against[2] the multiplicand, the product is aligned according to the position of the bit within the multiplier, and the resulting products are then summed to form the final result. One attraction of binary multiplication is how easy it is to form these intermediate products: if the multiplier bit is a 1, the product is an appropriately shifted copy of the multiplicand; if the multiplier bit is a 0, the product is simply 0.

For an n-bit multiplicand and multiplier, the resulting product will be $2n$ bits. Stated Thus, the product of two 4-bit numbers requires 8 bits, of two 8-bit numbers requires 16 bits, and so on.

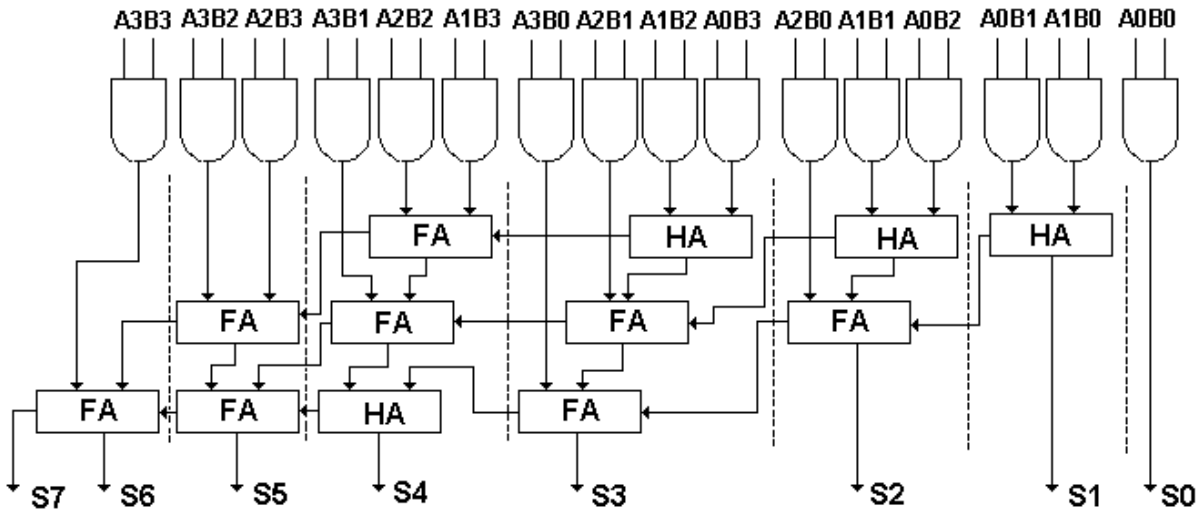
```
                    A3    A2    A1    A0
                    B3    B2    B1    B0
              ------------------------------
                    A3B0  A2B0  A1B0  A0B0
              A3B0  A2B0  A1B0  A0B0
        A3B0  A2B0  A1B0  A0B0
  A3B0  A2B0  A1B0  A0B0
-------------------------------------------
  S6    S5    S4    S3    S2    S1    S0
```

Partial Product Accumulation We can construct a combinational circuit that directly implements the process described by the preceding example. The method is called *partial product accumulation*.

First, we rewrite the multiplicand bits as $A3$, $A2$, $A1$, $A0$ and the multiplier bits as $B3$, $B2$, $B1$, $B0$. The multiplication of A and B becomes

Each of the ANDed terms is called a *partial product*. The resulting product is formed by accumulating down the columns of partial products, propagating the carries from the rightmost columns to the left.

A combinational circuit for implementing the 4-bit multiplier is shown in following figure.



The first level of 16 AND gates computes the individual partial products. The second- and third-level logic blocks form the accumulation of the products on a column-by-column basis. The column sums are formed by a mixture of cascaded half adders (HA) and full adders (FA). In the figure, inputs from the top are the bits to be added and the input from the right is the carry-in. The output from the bottom is the sum and to the left is the carry-out.

To see how the partial products are accumulated, let's look at the circuit of previous figure in a little more detail. $S1$ is the straightforward[3] sum of just two partial products, $A1 \cdot B0$ and $A0 \cdot B1$. $S2$ is the sum of three products. We implement this with two cascaded adders, one of which takes the carry-out from $S1$'s column.

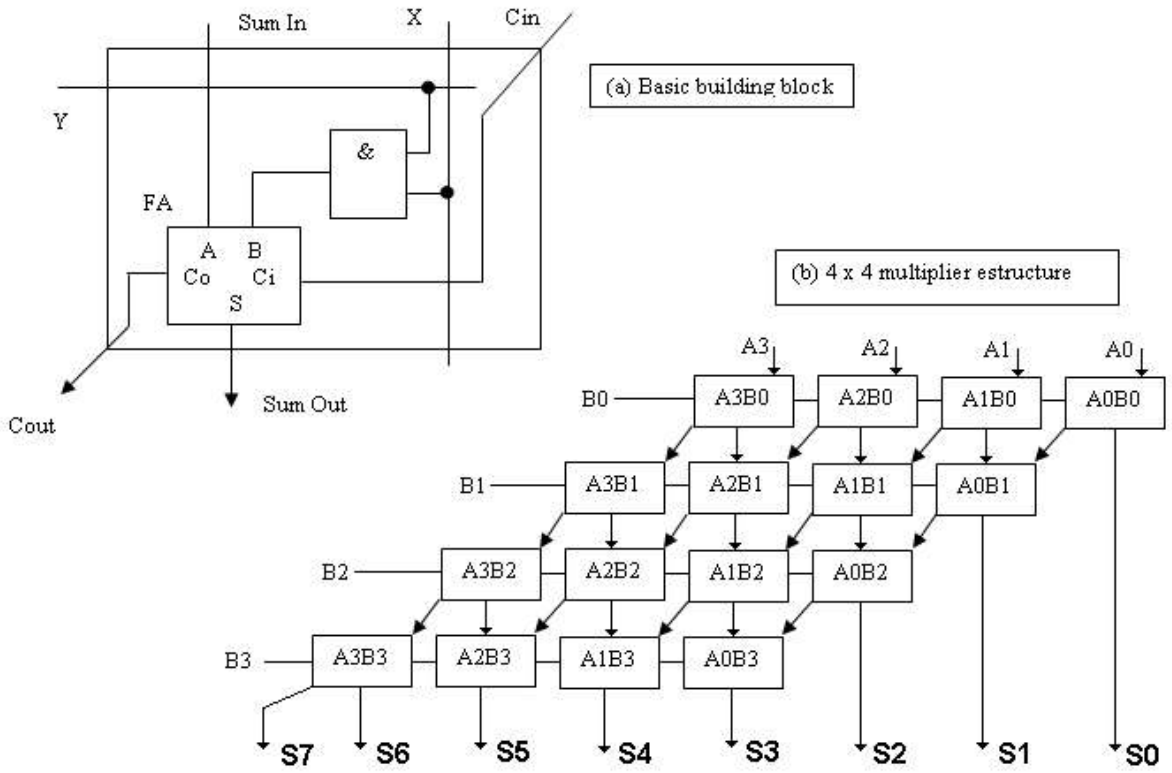
$S3$ is a little more complicated, because there are two different carry-outs from the previous column. We use three cascaded adders, two full adders and one half adder, to implement the sum. The two carry-outs from $S2$ are accumulated through the carry-in inputs of the two full adders.
$S4$ is the sum of three products and three possible carry-outs from $S3$. The carries make this case more complicated than the $S2$ sum. The solution is to implement the sum with three adders-two full adders and one half adder. The two full adders sum the three products and two of the carry-outs. The half adder adds to this result the third possible carry-in.

The logic for $S5$ is similar. Here we must sum two products and three carries. Two full adders do the job. Note that the second full adder sums two of the three carries from the previous column with the result of the first full adder. A similar analysis applies to $S7$ and $S6$.
The delay through the multiplier is determined by the ripple[4] carries between the adders. We can use a carry look-ahead[5] scheme to reduce these delays.

Clearly, the full combinational multiplier uses a lot of hardware. The dominating costs are the adders-four half adders and eight full adders. To simplify the implementation slightly[6], a designer may choose to use full adders for all of the adder blocks, setting the carry input to 0

where the half adder function is required. From the full adder schematic, this is 12 adders of six gates each (two OR, two AND and two X-OR), for a total of 72 gates. When we add to this the 16 gates forming the partial products, the total for the whole circuit is 88 gates. It is easy to see that combinational multipliers can be justified only for the most high performance of applications.

A slightly different implementation of the 4-by-4 combinational multiplier is shown in following figurure.

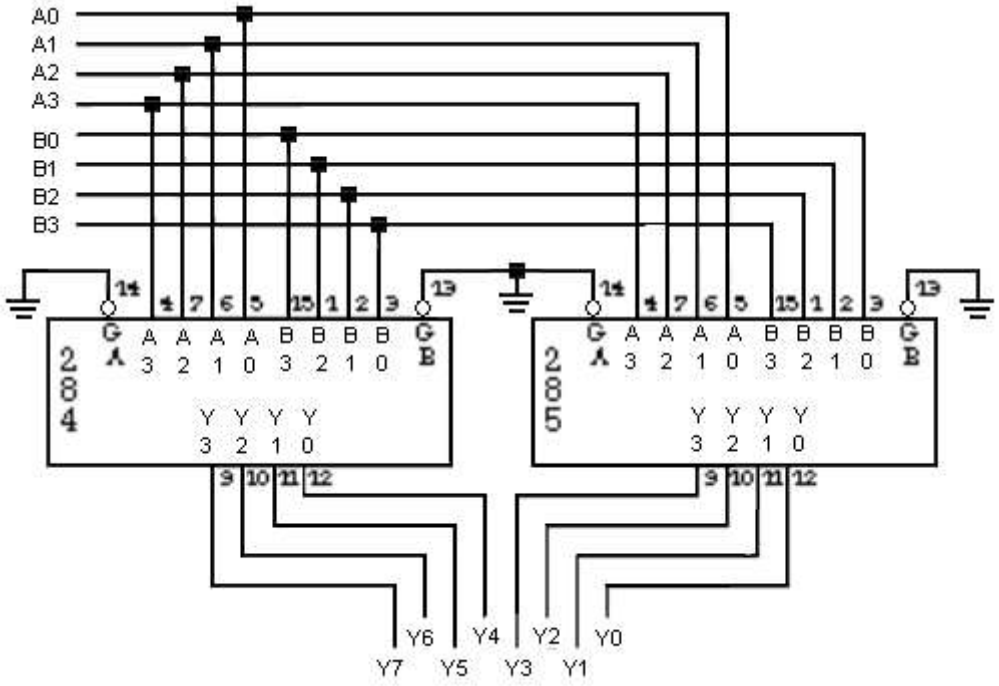


Part (a) gives the basic building block, a full adder circuit that sums a locally computed partial product ($X \cdot Y$), an input passed into the block from above (*Sum In*), and a carry passed from a block diagonally above. It generates a carry-out (*Cout*) and a new sum (*Sum Out*). Part (b) shows the interconnection of 16 of these blocks to implement the full multiplier function. The *A*i values are distributed along block diagonals and the *B*i values are passed along rows. This implementation uses the same gate count as the previous one: 16 AND gates and 12 adders (the top row does not need adders).

TTL Multiplier Components The TTL components 74284 and 74285 provide a two-chip implementation of a 4-by-4 parallel binary multiplier.

Following figure illustrates their use. The 74284 component implements the high-order 4 bits of the product, while the 74285 implements the low-order bits. Both chips have dual active low output enable signals, GA' and GB'. When both enables are 0, the outputs are valid. Otherwise, they are in the high-impedance state.

7. SISTEMAS ARITMÉTICOS



1 conveniente 2 con 3 directa 4 propagación

5 generador de acarreo en paralelo 6 ligeramente

III PRÁCTICA Y EJERCICIOS

1. APARTADOS PRÁCTICOS

1. Diseñe con puertas lógicas NAND un sumador completo, compruebe su funcionamiento.

2. Obtenga mediante el simulador digital un sumador total con salidas CP y CG y haga con él un circuito macro. A continuación monte la CPG de modo que puedan sumarse palabras de 3 bits con propagación del *carry* en paralelo.

3. Compruebe el funcionamiento del CI 7483. Obtenga a partir de él un sumador-restador de 4 bits.

4. Compruebe el funcionamiento de la ALU 74181 para los datos activos por alto que se dan y las operaciones que se indican.

DATOS: A = ____ B = ____

OPERACIÓN	A'+B	AB+A'B'	A PLUS AB	AB MINUS 1	A PLUS AB' PLUS 1
RESULTADO					

2. EJERCICIOS

1. Razone cómo a partir del 7483 podemos obtener un comparador de 4 bits.

2. Pase a binario en coma flotante de simple precisión los siguientes valores decimales:

a) 147.354,25 b) 1,3457E-6 c) 3,14159E9

3. Calcule el rango de valores que pueden codificarse en el formato de simple precisión.

4. Diseñe un circuito que sea capaz de sumar el menor con el mayor de tres números de cuatro bits.