CAPÍTULO

6

Elementos aritméticos

CAPÍTULO

6.1. Introducción
6.2. Semisumador y sumador completo
6.3. Sumador en paralelo con acarreo en serie
6.4. Sumador paralelo con acarreo anticipado
6.5. Restadores en binario con signo
  6.5.1. Sumador y restador en binario puro con C-1
  6.5.2. Sumador y restador en binario puro con C-2
6.6. Sumador BCD
  6.6.1. Sumador y restador en BCD con signo en complemento
6.7. ALU's y circuitos MSI
6.8. Resumen

6.1. INTRODUCCIÓN

Los dispositivos aritméticos son capaces de completar distintas operaciones en distintos códigos.

Las principales operaciones son la suma y la resta, así como el producto y la división. En cuanto a los códigos, los más utilizados sin signo son el binario puro y el BCD puro, y algo menos el BCD XS3. Los códigos con signo más utilizados son el binario puro con C-1 y con C-2, y el BCD con C-9 y C-10.

En el libro sólo se abordará el diseño de circuitos aritméticos para coma fija, obviándose los de coma flotante por su complejidad y porque suelen ser resueltos mediante diseño VLSI.

En cuanto a la implementación, ésta puede ser de dos tipos: combinacional o secuencial. La primera estrategia se aplica fundamentalmente al diseño de sumadores y restadores, mientras que la segunda se utiliza principalmente para multiplicadores y divisores.

En los operadores aritméticos, al igual que en los combinacionales MSI, es de gran importancia el concepto de palabra. Así, los diseños aritméticos tienen que ser fácilmente ampliables para cualquier número de bits. Cada operador aritmético será descrito teóricamente mediante el algoritmo, implementado con puertas lógicas y descrita la manera de extender la capacidad del circuito diseñado.

También serán presentadas en este capítulo las Unidades Aritmético Lógicas (ALU), circuitos MSI capaces de realizar varias operaciones en un solo circuito integrado.

Se aconseja al lector que antes de leer el capítulo relea el correspondiente a códigos, ya que los algoritmos y planteamientos de cada operación proceden de él.

6.2. SEMISUMADOR Y SUMADOR COMPLETO

El primer elemento aritmético a desarrollar es aquel que suma dos bits para ofrecer su resultado en forma de acarreo y suma. La tabla de verdad correspondiente a la operación suma es la Tabla 6.1, cuyo circuito lógico y bloque están en la Figura 6.1.

A este dispositivo se le denomina semisumador —o HA, acrónimo de half adder— y tiene el inconveniente de que no permite la suma de varios bits, ya que no contempla como entrada el acarreo.

El sumador completo sí contempla el acarreo de entrada, lo que permite su extensión.

Un sumador completo —FA, full adder— se comporta como indica su Tabla de verdad 6.2.

Tabla 6.1. *Tabla de verdad y ecuaciones del semisumador, HA.*

A	B	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

$$C = A \cdot B$$
$$S = \overline{A} \cdot B + A \cdot \overline{B} = A \oplus B$$

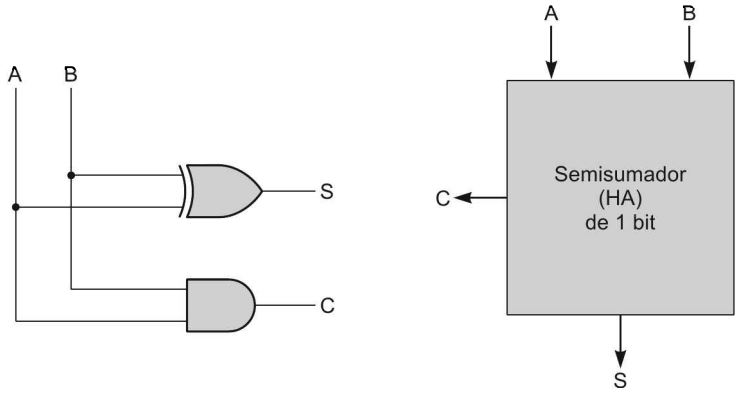


Figura 6.1. *HA: circuito lógico y bloque.*

Tabla 6.2. *T-V del sumador completo FA.*

Ai	Bi	Ci-1	Ci	Si
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

Su implementación puede tomar 3 caminos:

- Implementación como combinacional.
- Implementación utilizando la regla de la suma.
- Implementación basada en semisumadores.

En el primer camino resolvemos e implementamos los correspondientes diagramas de V-K de la Tabla 6.2 obteniendo las ecuaciones (6.1) y el circuito de la Figura 6.2.

$$C_i = A_i \cdot C_{i-1} + B_i \cdot C_{i-1} + A_i \cdot B_i \qquad (6.1)$$
$$S_i = A_i \cdot B_i \cdot C_{i-1} + A_i \cdot \overline{B_i} \cdot \overline{C_{i-1}} + \overline{A_i} \cdot B_i \cdot \overline{C_{i-1}} + \overline{A_i} \cdot \overline{B_i} \cdot C_{i-1}$$

En la segunda implementación hay que tener en cuenta que sumar tres bits es como sumar dos bits dos veces, y que el acarreo existe si al menos dos de los bits de entrada son 1. Si reescribimos lo anterior en forma booleana tenemos la ecuación (6.2) y el circuito de la Figura 6.3.

$$S_i = (A_i \oplus B_i) \oplus C_{i-1} \qquad (6.2)$$
$$C_i = A_i \cdot B_i + (A_i \oplus B_i) \cdot C_{i-1}$$

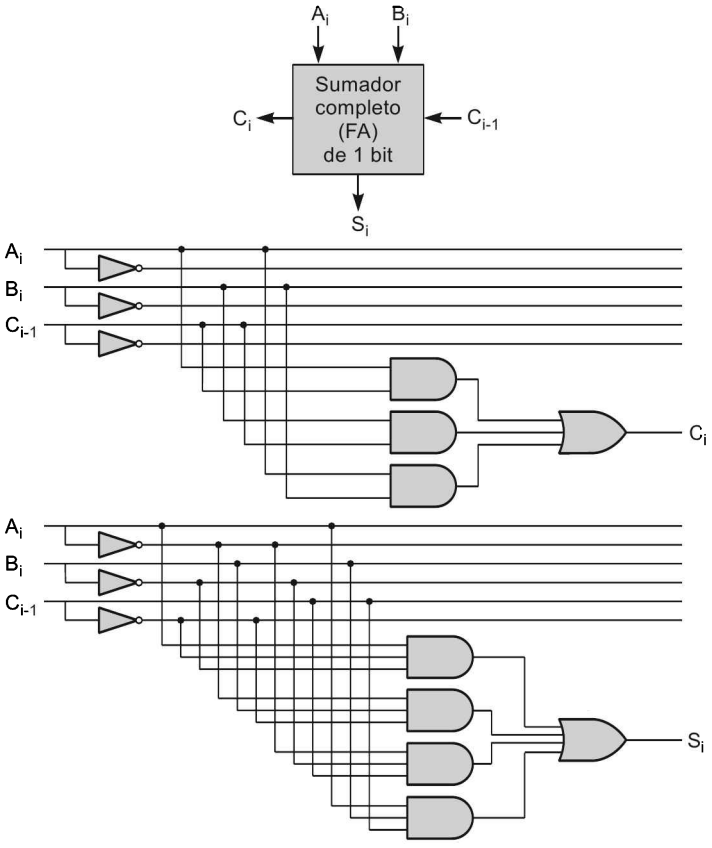


Figura 6.2. *FA: bloque y circuito lógico.*

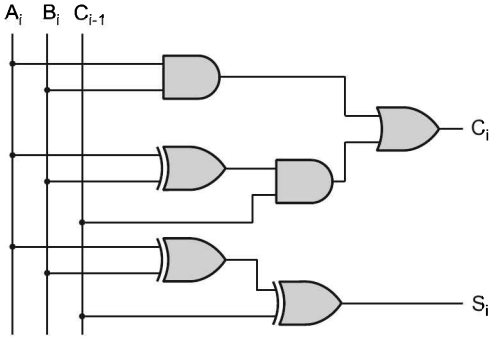


Figura 6.3. *Otro circuito lógico de un FA.*

En la tercera simplemente reordenaremos las anteriores ecuaciones respecto del semisumador, resultando la implementación de la Figura 6.4, que, como se puede observar, respeta las ecuaciones 6.1 y 6.2.

$$S'_i = A_i \oplus B_i \quad \text{y} \quad C'_i = A_i \cdot B_i$$
$$S''_i = S'_i \oplus C_{i-1} \quad \text{y} \quad C''_i = S'_i \cdot C_{i-1} = (A_i \oplus B_i) \cdot C_{i-1}$$
$$S = S''_i \qquad C_i = C'_i + C''_i = A_i \cdot B_{i-1} + (A_i \oplus B_i) \cdot C_{i-1}$$

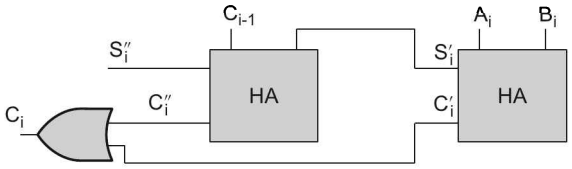


Figura 6.4. *FA basado en HA.*

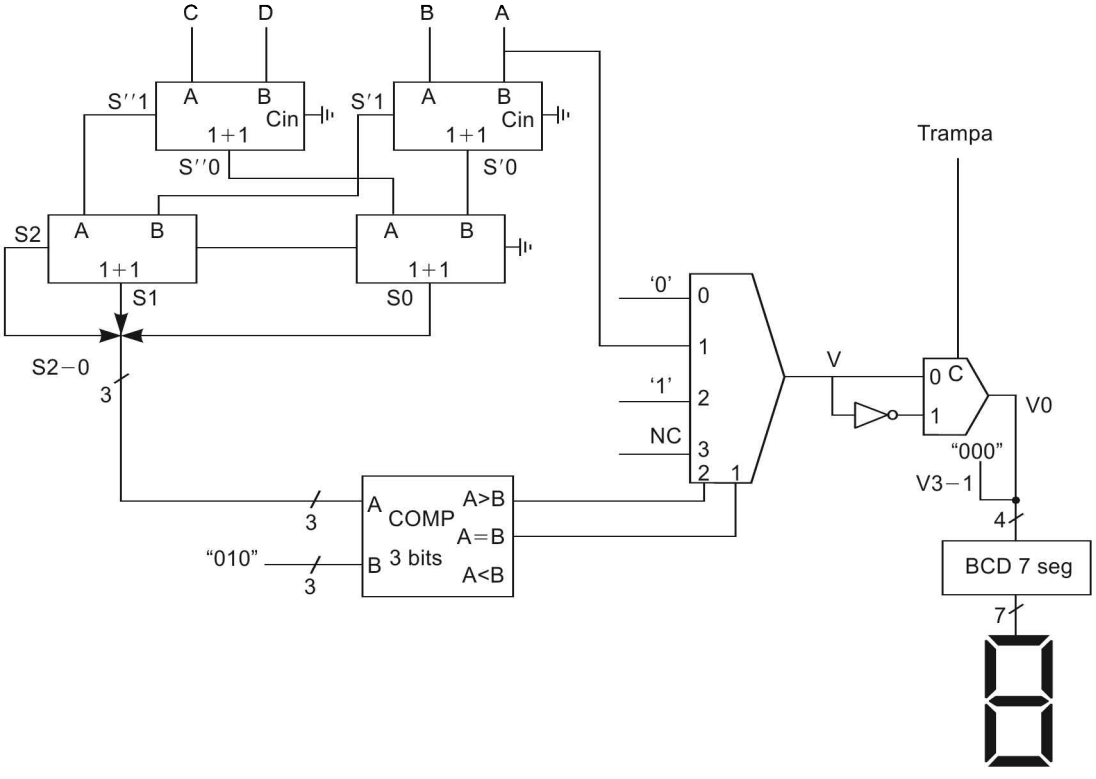
Las tres implementaciones son válidas, aunque con distinto coste en tiempo y puertas y distinto comportamiento frente a los riesgos lógicos. Su planteamiento nos permite observar cómo de distintos enfoques se obtienen soluciones correctas; situación ésta, que aunque es puramente teórica, es muy típica en los circuitos aritméticos.

Toda vez que tenemos un sumador completo de 1 bit podemos abordar un sumador de cualquier número de bits.

Ejemplo 6.1. Cuatro países, A, B, C y D, votan en un consejo. Visualizar en un 7-segmentos el resultado de la votación sabiendo que en caso de empate decide el voto de calidad de A. Además, el país anfitrión tiene un interruptor de trampa para dar la vuelta al resultado. Diseñar el circuito lógico correspondiente.

Rediseñar el circuito para seis países donde los votos de A y B valen doble. Añádase el derecho de veto de un país.

Solución



Explicación:

- Sumar por separado S′ = A + B y S′′ = C + D. 2 sumadores 1 + 1.
- Sumar S′ + S′′ = S2S1S0. 2 sumadores 1 + 1.
- Comparar S con 2 (010), utilizando A > B y A = B. Comparador de 4 bits.
- Elegir entre 0, A y 1 según el valor de A > B (C1) y A = B (C0):

  — C1C0 = 00, S es menor que 2, menos de 2 votos, VOTO = 0.
  — C1C0 = 01, S es igual a 2, 2 votos, manda A, VOTO = A.
  — C1C0 = 10, S es mayor que 2, más de 2 votos, VOTO = 1.
  — C1C0 = 11, es imposible que sean 2 y mayor que 2.

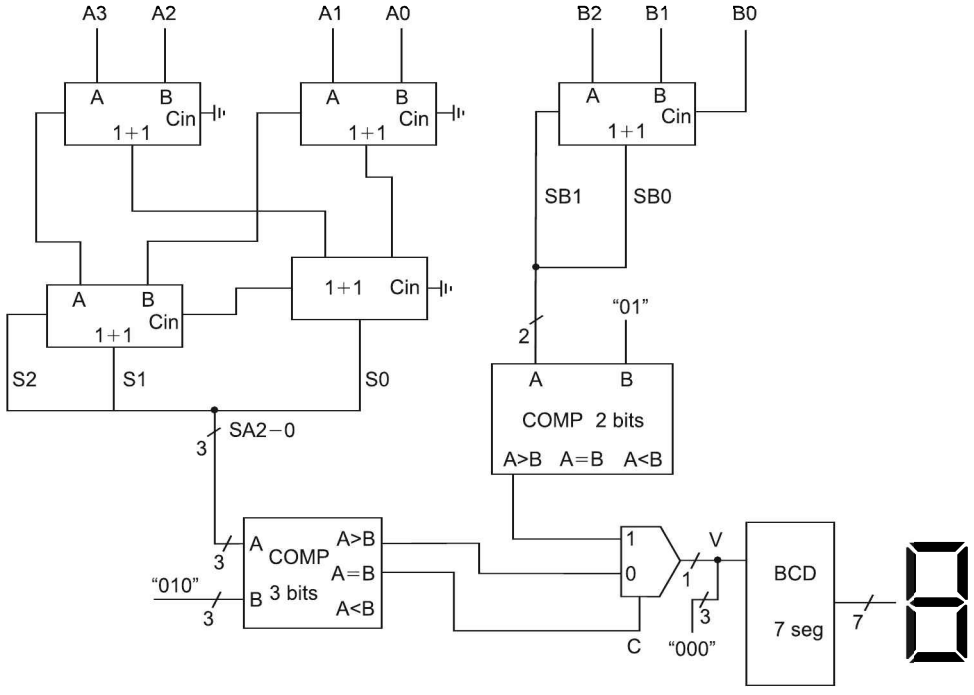
  Multiplexor 4:1.

- Elegir entre V y su contrario según el valor de la entrada TRAMPA, si es 0 pasa V, y si es 1, pasa V negada. Multiplexor 2:1.
- Unir "000" a lo elegido, y que así quede "0000" o "0001". Visualizar este resultado en un 7 segmentos. Decodificador BCD-7 segmentos.
- Los sumadores deben agruparse en dos niveles para no mezclar los pesos de las sumas. No valdría usar un sumador 2 + 2, ya que entonces dos países valdrían doble, los asignados a A1 y B1 del sumador.

Ejemplo 6.2.

En una votación los países están separados por bloques: cuatro países en A y tres en B. En un 7-segmentos se visualizará el resultado de la votación: si hay mayoría en A ése es el resultado, pero si hay empate en A el resultado lo marca B. Diseñar el circuito lógico correspondiente.

Primera solución:



Explicación:

- Sumar en dos capas los votos del grupo A. 4 sumadores 1 + 1.
- Sumar los votos del grupo B. Sumador 1 + 1.
- Comparar los votos de A con el valor 2 (010), utilizando A > B (A > 2) y A = B (A = 2). Comparador de 3 bits.
- Comparar los votos de B con el valor 1 (01), utilizando A > B (B > 1). Comparador de 2 bits.
- Elegir entre A > 2 y B > 1 según el valor de A = 2 (C) :

  — C = 0, no hay 2 votos en A, el voto se obtiene de A, VOTO = (A > 2):

    - (A > 2) = 1, más de 2 votos en A, VOTO = 1.
    - (A > 2) = 0, menos de 2 votos en A, VOTO = 0.

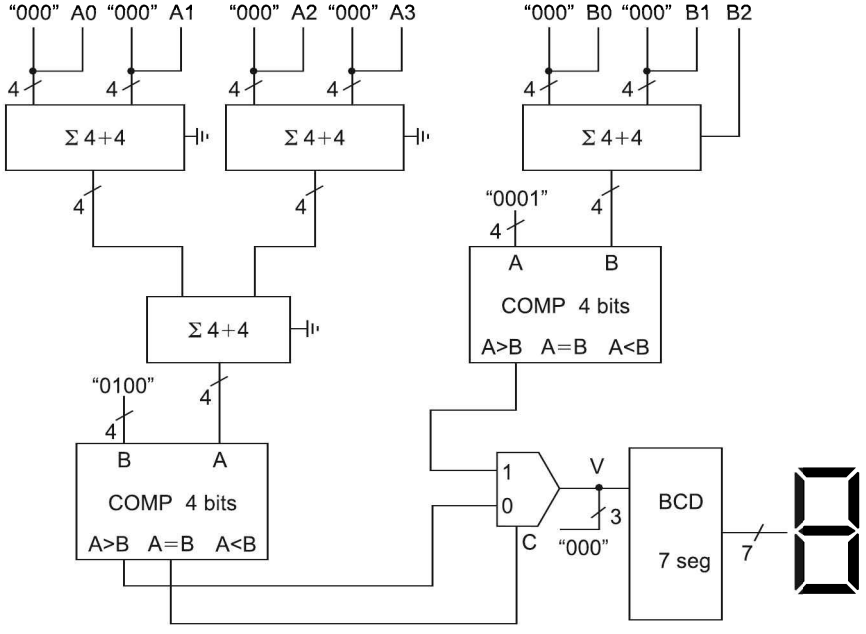
  — C = 1, hay 2 votos en A, el voto se obtiene de B, VOTO = (B > 1):

    - (B > 1) = 1, más de 1 voto en B, VOTO = 1.
    - (B > 1) = 0, no más de 1 voto en B, VOTO = 0.

  Multiplexor 2:1.

Segunda solución

Esta solución es muy parecida a la anterior, simplemente utiliza sumadores 4 + 4.



6.3. SUMADOR EN PARALELO CON ACARREO EN SERIE

Esta implementación concuerda con nuestra forma de sumar: primero sumo dos dígitos, luego los dos siguientes más el acarreo anterior, etc. Este mecanismo es fácilmente implementable con sumadores completos; la Figura 6.5 muestra el esquema y el bloque de un sumador de 4 bits.

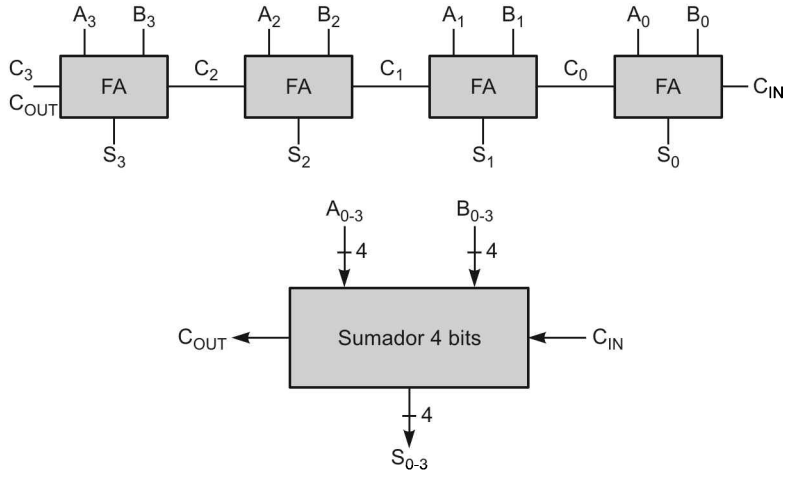


Figura 6.5. *Sumador paralelo-serie de 4 bits.*

Repitiendo el planteamiento anterior, un sumador de 8 bits se obtiene conectando en serie dos sumadores de 4 bits (Figura 6.6), y del mismo modo se procederá para mayor número de bits.

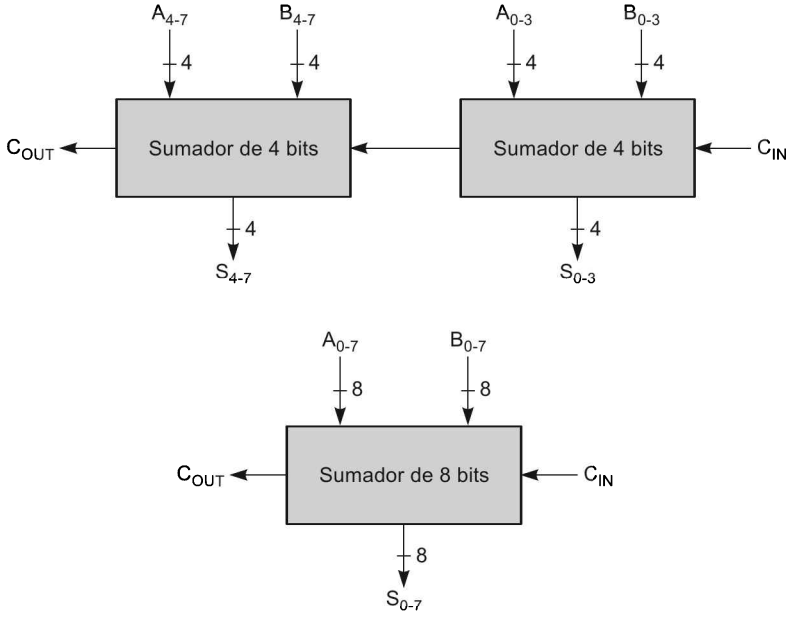
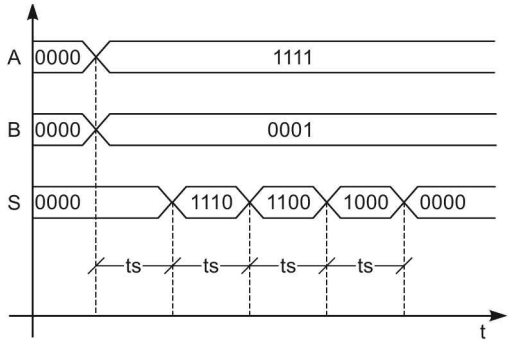


Figura 6.6. *Sumador paralelo-serie de 8 bits.*

Al circuito de la Figura 6.6, cualquiera que sea el número de bits, se le denomina sumador paralelo con acarreo serie porque las entradas se presentan simultáneamente, en paralelo, mientras que el acarreo se produce en serie, etapa a etapa. Por ejemplo, el acarreo C_2 no puede generarse hasta que no lo haya sido el C_1, y así sucesivamente. Lo anterior supone que si asociamos un tiempo ts para sumar 1 bit, pasado este tiempo tendremos un resultado para los cuatro bits, pero éste no tiene por qué ser correcto, ya que el acarreo se genera en serie. Veamos en el ejemplo el peor caso que se puede dar: A = 1111 y B = 0001.

Tabla 6.3. *Evolución temporal de la suma en el peor caso posible.*

0	1	1	1	0	$t = ts$
0	1	1	0	0	$t = 2 \times ts$
0	1	0	0	0	$t = 3 \times ts$
1	0	0	0	0	$t = 4 \times ts$



El resultado no es correcto hasta pasados $4 \times ts$. Por ejemplo, para $t \leq ts$ la etapa 2 suma $A_1 = 1$, $B_1 = 0$ y $C_0 = 0$, pues todavía no se ha generado correctamente el C_0, resultando que la suma es 1110. Así pues, la suma es correcta sólo si $t > 4 \times ts$.

Por tanto, para obtener el resultado correcto de sumar n bits hay que esperar a que transcurran los $n \times ts$ segundos correspondientes al caso más desfavorable (otros casos necesitarían menos tiempo, pero no son generalizables).

$$\text{Tiempo de suma de n bits} = n \times ts \tag{6.3}$$

Si el número de bits es elevado —a partir de 16 bits— este tiempo de retardo no es despreciable, convirtiéndose en muchos casos en inaceptable. Para resolver esta situación se plantean circuitos más rápidos, aunque lo sean a costa de una mayor complejidad y coste circuital.

6.4. SUMADOR PARALELO CON ACARREO ANTICIPADO

En este circuito los acarreos se generan en paralelo al igual que las entradas, por tanto se reduce el tiempo total de suma.

Para plantear el circuito comencemos por recordar que en un sumador completo la ecuación correspondiente al acarreo es $C_i = A_i \cdot B_i + (A_i \oplus B_i) \cdot C_{i-1}$. Al observar esta ecuación podemos decir que $A_i \cdot B_i$ es el producto que genera el acarreo —y así lo denominamos G_i—, mientras que de $(A_i \oplus B_i)$ podemos decir que propaga el acarreo de la anterior etapa —y lo denominamos P_i—, ya que si $P_1 = 1$ se propaga el valor de C_{i-1}. Reescribamos en 6.4 la ecuación correspondiente al acarreo.

$$\begin{aligned} G_i &= A_i \cdot B_i \\ P_i &= A_i \oplus B_i \\ C_i &= G_i + P_i \cdot C_{i-1} \end{aligned} \tag{6.4}$$

Teniendo en cuenta lo anterior planteemos de nuevo las ecuaciones correspondientes para un sumador de 4 bits, donde podemos representar los acarreos de salida de las cuatro etapas como:

$$C_0 = G_0 + P_0 \cdot C_{IN}$$
$$C_1 = G_1 + P_1 \cdot C_0$$
$$C_2 = G_2 + P_2 \cdot C_1$$
$$C_3 = G_3 + P_3 \cdot C_2$$

Ahora bien, estas ecuaciones están serializadas, pues C_i se genera a partir de C_{i-1}. Para paralelizarlas simplemente reeescribimos las ecuaciones como indica 6.5.

$$\begin{aligned} C_0 &= G_0 + P_0C_{IN} \\ C_1 &= G_1 + P_1\,(G_0 + P_0C_{IN}) = G_1 + P_1G_0 + P_1P_0C_{IN} \\ C_2 &= G_2 + P_2\,(G_1 + P_1G_0 + P_1P_0C_{IN}) = G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0Ca_{IN} \\ C_3 &= G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0 + P_3P_2P_1P_0C_{IN} \end{aligned} \qquad (6.5)$$

La implementación del sumador correspondiente a las ecuaciones 6.5 es el de la Figura 6.7. A primera vista puede parecer complejo de obtener, pero fijándose no es así. En la Figura 6.7 vemos dos líneas no contempladas en las ecuaciones, G y P, que serán explicadas más adelante.

Una cuestión importante a dilucidar es: ¿cuánto tiempo necesita un sumador con acarreo anticipado para completar una suma? Para determinarlo, fijémonos en tres aspectos:

- La obtención en paralelo de Pi y Gi.
- La obtención en paralelo de Ci.
- La suma en cada etapa toda vez que se ha obtenido Ci.

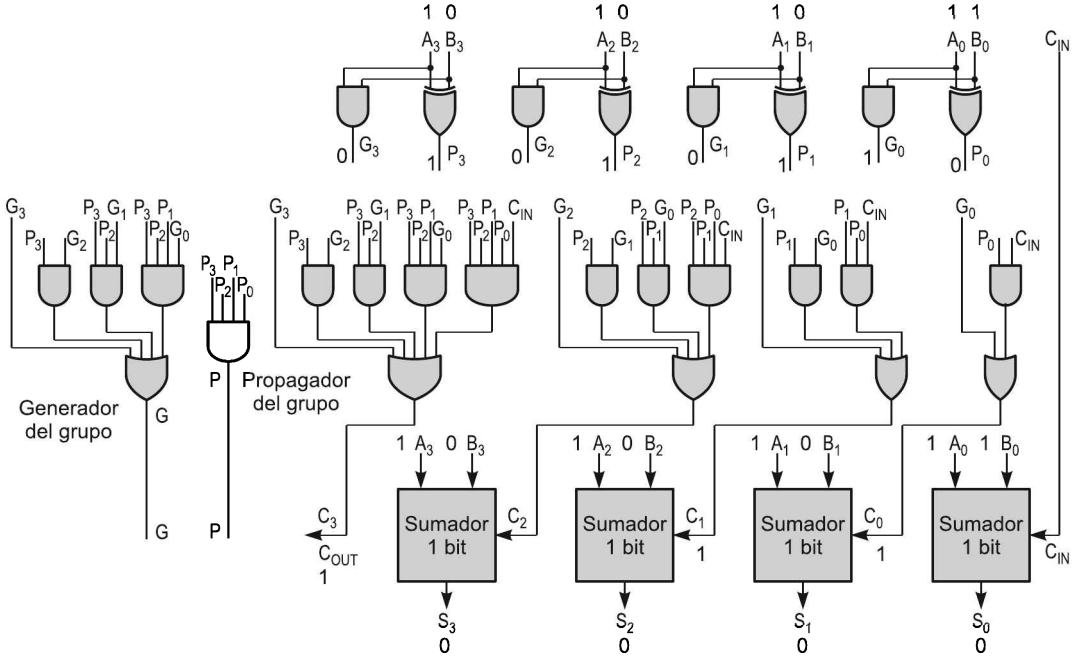


Figura 6.7. *Circuito lógico de un sumador 4 + 4 con acarreo anticipado.*

La obtención de P_i y G_i consume $2 \times$ tp (tp = tiempo puerta), a esto hay que sumar otros $2 \times$ tp por la obtención de C_i; así cualquier acarreo está disponible en $4 \times$ tp. Para obtener S_i hay que sumar $2 \times$ tp —toda vez que consideramos que una XOR viene a tardar el doble que una puerta normal—. La suma por tanto se obtiene de $6 \times$ tp segundos, sea cual sea el número de bits; de hecho en la expresión desaparece el término n. La Tabla 6.4 compara ambos sumadores.

Tabla 6.4. *Comparación entre sumador serie y paralelo.*

Sumador con acarreo serie	Sumador con acarreo paralelo
Tsuma = n × 2tp Tacarreo = (n + 1) × 2tp	Tsuma = 6 × tp Tacarreo = 4 × tp = TGP de grupo

Volvamos a la Figura 6.7, donde las líneas G y P de grupo permiten extender la capacidad de los sumadores como veremos más adelante. La línea G se pone a 1 cuando dicha etapa o grupo ha generado un acarreo, y P se pone a 1 cuando dicha etapa ha permitido la propagación del acarreo de entrada en la etapa. La obtención de estas líneas retarda el proceso hasta $5 \times$ tp.

Como ya hemos dicho, en los tiempos de retardo de este circuito ha desaparecido el término n, a cambio el circuito se ha complicado hasta el punto de que es muy extraño que un sumador con acarreo anticipado supere los 8 bits. Cuando haya que implementar sumadores de mayor número de bits y no se pueda aceptar el retardo del sumador con acarreo serie, se optará por implementaciones híbridas basadas en un nuevo circuito, el LAC (LookAhead Carry).

6.5. RESTADORES EN BINARIO CON SIGNO

Si para diseñar un sumador planteamos el circuito correspondiente a un sumador de 1 bit, podemos repetir el mismo procedimiento para un restador. En un restador de 1 bit se restan dos bits y el correspondiente préstamo para obtener la resta y el préstamo. La tabla de verdad, las ecuaciones booleanas y el circuito correspondiente son la Tabla 6.5, las ecuaciones 6.6 y la Figura 6.8.

Tabla 6.5. *T-V de un restador de un bit.*

A	B	PIN	POUT	R
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	1	0
1	0	0	0	1
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Resolviendo los diagramas de V-K obtenemos las ecuaciones correspondientes a un restador de 1 bit.

$$\begin{aligned} P_{OUT} &= \overline{A} \cdot P_{IN} + B \cdot P_{IN} + \overline{A} \cdot B \\ R &= A \cdot \overline{B} \cdot \overline{P}_{IN} + \overline{A} \cdot B \cdot \overline{P}_{IN} + \overline{A} \cdot \overline{B} \cdot P_{IN} + A \cdot B \cdot P_{IN} = \\ &= P_{IN} (A \cdot \overline{B} + \overline{A} \cdot B) + P_{IN} \cdot (\overline{A} \cdot \overline{B} + A \cdot B) = A \oplus B \oplus P_{IN} \end{aligned} \qquad (6.6)$$

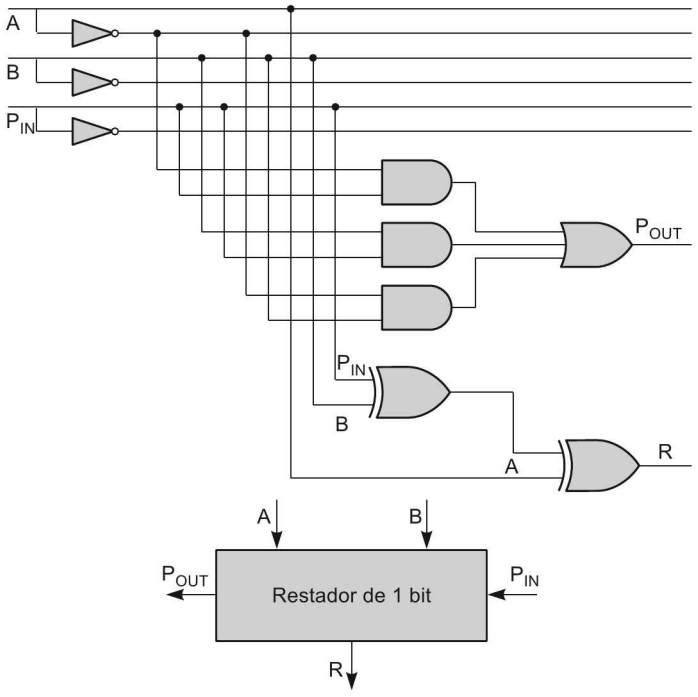


Figura 6.8. *Circuito y bloque de un restador.*

Para obtener un restador de 4 bits sólo queda unir en serie cuatro restadores de 1 bit, como muestra la Figura 6.9.

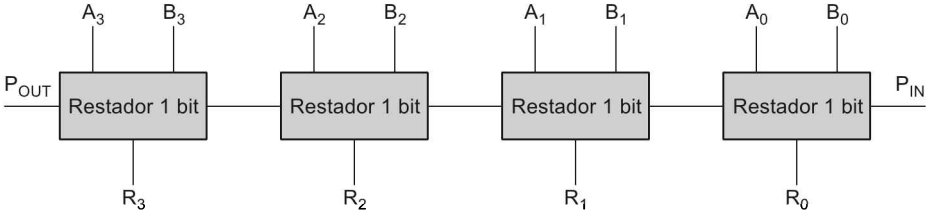


Figura 6.9. *Restador serie de cuatro bits.*

Como podemos observar, podríamos seguir una estrategia idéntica a la utilizada en la suma. Sin embargo, en vez de seguir este camino paralelo, resulta más cómodo reeescribir la resta como una suma con signo en 6.7.

$$A - B = A + (-B) \tag{6.7}$$

Según la anterior expresión la resta se convierte en el siguiente algoritmo:

- Tomar A y B.
- Cambiar de signo el sustraendo B' = (−B).
- Sumar A y B'.

Así pues, en los siguientes apartados el diseño correspondiente a la resta se reescribe como el diseño de una suma donde los operandos tienen signo. Además aprovecharemos para plantear aquellos

circuitos versátiles, capaces de realizar tanto restas como sumas en el código determinado, dando lugar a los Sumadores/Restadores. También abordaremos en estos códigos la detección del desborde en la operación, es decir, determinar en qué situación el resultado obtenido no puede escribirse con el número de bits original, y por tanto el resultado obtenido es erróneo y no utilizable. En los sumadores anteriormente vistos el desborde coincidía con el valor del C_{OUT}, pero en los siguientes circuitos su determinación no es tan espontánea.

Para todos los diseños siguientes seguiremos un esquema parecido:

- Manejo del signo en ese código.
- Determinación del algoritmo correspondiente.
- Obtención del desborde u overflow.
- Implementación del circuito restador para 4 bits.
- Tiempo consumido en la operación.
- Estudio de la extensibilidad del circuito.
- Ejemplo aclaratorio si fuera necesario.
- Implementación del circuito sumador/restador.

6.5.1. Sumador y restador en binario puro con C-1

En este caso un número negativo se escribe como el complemento a 1 del correspondiente positivo, así el algoritmo de la resta es:

- Tomar A (minuendo) y B (sustraendo).
- Complementar a 1 (negar) el sustraendo B'= C-1(B).
- Obtener la resta como R = A + B', recirculando el acarreo.

Estudio del desborde

La suma A + B' puede desbordarse sólo si A y B' son positivos o negativos.

Concretemos el estudio para cuatro bits, donde el rango es [−7,+7]:

- Si A y B' son positivos, en ambos su bit de signo (A_3 y B_3) es cero, resultando que la suma ha de ser positiva, y su bit de signo (S_3) cero. Por tanto, habrá overflow si al sumar $S_3 = 0 + 0 + + C_2 = 1$, lo que sólo puede ser cierto si $C_2 = 1$. Reescribiendo lo anterior booleanamente:

$$\text{Des. positivo} = \overline{A}_3 \cdot \overline{B}_3 \cdot C_2 = \overline{C}_3 \cdot C_2$$

- Si A y B' son negativos en ambos su bit de signo (A_3 y B_3) es uno, resultando que la suma ha de ser negativa, y su bit de signo (S_3) uno. Por tanto, habrá overflow si al sumar $S_3 = 1 + 1 + C_2 = 0$, esto sólo puede ser cierto si $C_2 = 0$. Reescribiendo lo anterior booleanamente:

$$\text{Des. negativo} = A_3 \cdot B_3 \cdot \overline{C}_2 = C_3 \cdot \overline{C}_2$$

- Si A es positivo y B' es negativo, o viceversa, no puede darse en ningún caso desborde.

- Reuniendo el desbordamiento negativo y positivo resulta 6.8.

$$\text{Desborde} = \overline{C}_3\, C_2 + C_3\, \overline{C}_2 = C_3 \oplus C_2$$
$$\text{DE} = C_{n-1} \oplus C_{n-2} \qquad (6.8)$$

donde n = n.º de bits.

La implementación del restador en BP-C_1 aparece en la Figura 6.10.

En la Figura 6.10 observamos que el circuito complementador a 1 simplemente consta de 4 inversores, y que es necesario recircular el acarreo final para que el resultado sea correcto. En el bloque final podemos observar cómo éste no tiene C_{IN} (en nuestro caso P_{IN}), lo que quiere decir que no es directamente extensible al modo del sumador con acarreo serie, aunque por supuesto sería fácil implementar un restador de 8 bits, como el de la Figura 6.11.

Respecto del tiempo consumido en la resta no es, como en la suma, (n + 1) × ts; sino que el hecho de recircular el acarreo hace que el retardo asociado a la resta sea 2 × (n + 1) × ts, es decir, el doble de una suma en binario puro.

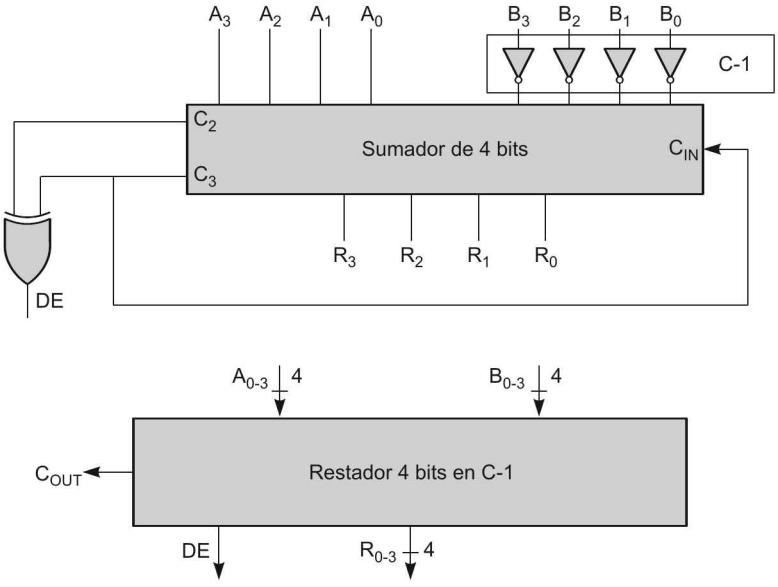


Figura 6.10. *Restador 4-4 en binario puro con C-1.*

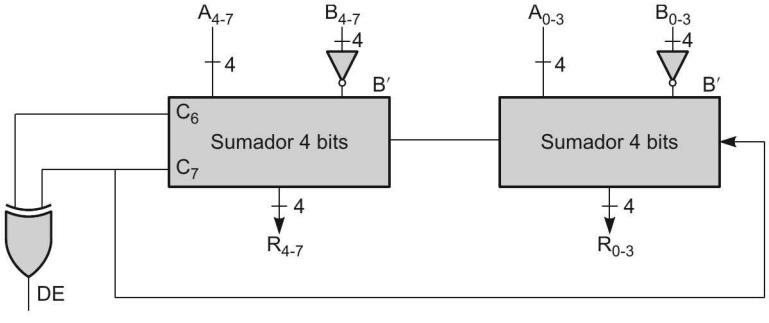


Figura 6.11. *Restador 8-8 con acarreo en serie.*

6.5.2. Sumador y restador en binario puro con C-2

Si el código elegido es el binario puro con C-2, el enfoque es idéntico al correspondiente al binario puro con C-1. Así, el algoritmo queda:

- Tomar A (minuendo) y B (sustraendo).
- Complementar a 2 (negar) el sustraendo B' = C-2 (B).
- Obtener la resta como R = A + B', sin recircular el acarreo.

Estudio del desborde

Se puede repetir el planteamiento correspondiente al complemento a 1 para obtener la expresión de cuándo hay desbordamiento en el código.

$$\text{Desborde} = C_3 \oplus C_2 = C_{n-1} \oplus C_{n-2} \tag{6.9}$$

donde n = n.º de bits

Para implementar el restador recordemos que el complemento a 2 de un número es su complemento a 1 más 1, C-2 = C-1 + 1, así el circuito resultante es el de la Figura 6.12.

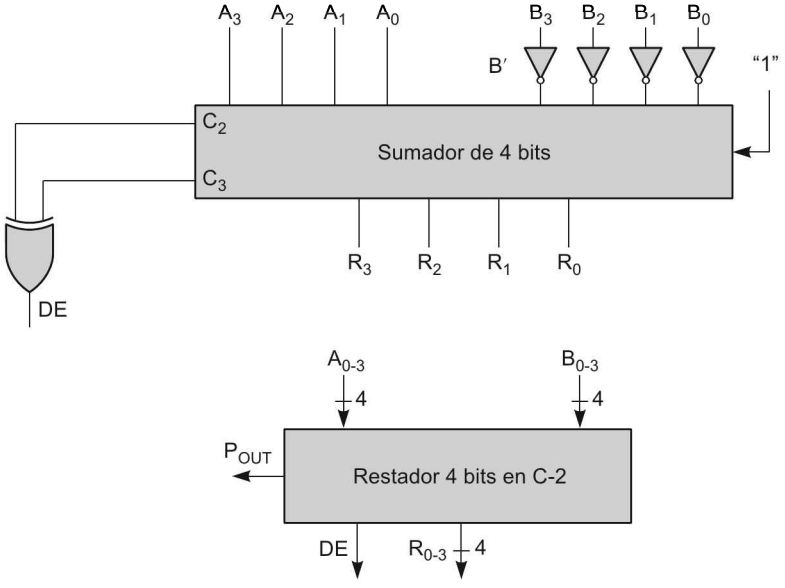


Figura 6.12. *Restador de cuatro bits en binario puro con C-2.*

En este caso la extensión de la capacidad de un restador es directa, sin más que duplicar el circuito correspondiente al restador de 4 bits y unirlo con el original.

El diseño del sumador/restador en este código es muy sencillo, sólo hay que tener en cuenta que si MODO = 1 (resta) hay que C-2 el operando B, y que si MODO = 0 (suma) no hay que complementar. Así, un sumador/restador en binario puro con C-2 para 8 bits queda como muestra la Figura 6.13.

Si MODO = 0: R = A + B
Si MODO = 1: R = A + C-1(B) + 1 = A − B.

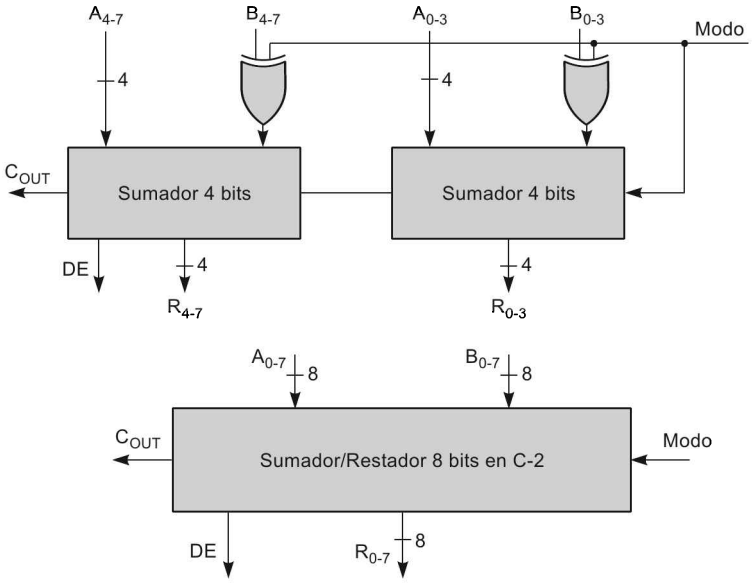


Figura 6.13. *Sumador/Restador de ocho bits en binario puro con C-2.*

6.6. SUMADOR BCD

El código BCD asocia a cada dígito decimal cuatro bits. Al sumar dos dígitos A y B en BCD el resultado puede ser mayor o menor que diez. Si el resultado fuera menor que diez sería correcto, no así si fuera mayor que diez. Por ejemplo, si sumamos 7 + 5 resulta 0111 + 0101 = 1100, que no es 12 en BCD.

Así pues, si sumáramos A + B obtendríamos un resultado S′ y un C'_{OUT} que no tendrían por qué ser correctos, y que por tanto habría que modificar para que fueran los adecuados.

A continuación se clasifican los posibles resultados de la suma, indicando si la suma es correcta o no, cómo determinar su incorrección y cómo corregirla. También se contempla el comportamiento del C_{OUT}.

- $A + B \leq 9$. El resultado de la suma y el acarreo son correctos.
- $10 \leq A + B \leq 15$. El resultado de la suma y el acarreo no son correctos.
- $A + B \geq 16$. El resultado de la suma no es correcto, el del acarreo sí.

Generalizando, cuando la suma es mayor o igual que 10 hay que transformar el resultado, restándole 10, o lo que es lo mismo, sumándole 6.

Ya sabemos qué hacer y cuándo hacerlo, pero nos falta expresarlo en términos booleanos. El resultado de sumar A + B es mayor que diez si:

$$S' > 10 = C'_{OUT} + S'_3 S'_2 + S'_3 S'_1$$

Si reescribimos lo anterior en forma de expresión booleana, donde (A + B) BP significa sumar A y B en binario puro y (A + B) BCD lo correspondiente en BCD, el resultado es el siguiente:

$$\begin{aligned} S' &= (A + B)\ BP \\ S' > 10 &= C'_{OUT} + S'_3 S'_2 + S'_3 S'_1 \\ S &= S' + (0110) \cdot (S' > 10) = (A + B)\ BCD \\ C_{OUT} &= (S' > 10) \end{aligned} \qquad (6.10)$$