Capítulo 4 Algoritmos de multiplicación rápida

En este capítulo se habla sobre las distintas arquitecturas y algoritmos de multiplicación rápida que se encontraron en la bibliografía.

Los sistemas en un solo chip ofrecen grandes ventajas para desarrollar nuevos productos electrónicos. El uso de procesadores embebidos es cada vez mayor y uno de los principales bloques funcionales del procesador es el multiplicador.

Las operaciones de multiplicación y división son más complejas que las de adición y sustracción. Originalmente las multiplicaciones y divisiones se hacían por medio de *software,* haciendo programas por medio de sumas o restas sucesivas. Hoy en día, debido al avance tecnológico, se opta por realizar diseños en *hardware* ya que pueden conseguir velocidades más rápidas en dispositivos de bajo costo.

Las reglas de multiplicación binaria son sencillas:

1. Si el dígito multiplicador es 1, el multiplicando simplemente se copia y representa un producto parcial.
2. Si el dígito multiplicador es 0 entonces el producto es cero.

Para diseñar un multiplicador se debe tener la circuitería necesaria para lograr los siguientes propósitos:

1. debe ser capaz de identificar si el multiplicador es 0 ó 1.
2. debe ser capaz de desplazar los productos parciales.
3. debe ser capaz de sumar los productos parciales
4. como propósito adicional, debe examinar el signo de los números a multiplicar para determinar el signo del producto.

La mayoría de los algoritmos de multiplicación se basan en el método de suma y desplazamiento, por ello se analizarán los desplazamientos previamente.

Un desplazamiento aritmético es una operación en la que se mueve un número binario, ya sea a la derecha o a la izquierda. Un desplazamiento a la izquierda equivale a multiplicar el número por dos, y un desplazamiento a la derecha equivale a dividir dicho número por dos. Los desplazamientos no deben afectar el signo del número ya que las multiplicaciones o divisiones por 2 no afectan el signo.

Hablando de números binarios se trata de una operación que desplaza todos los bits del operando; cada bit del operando simplemente se mueve un número determinado de posiciones y las posiciones vacías se van rellenando. En los desplazamientos a la izquierda se introduce al bit menos significativo un cero, mientras que en el desplazamiento a la derecha, el bit más significativo, que representa el signo del número, se replica para llenar las posiciones vacías.

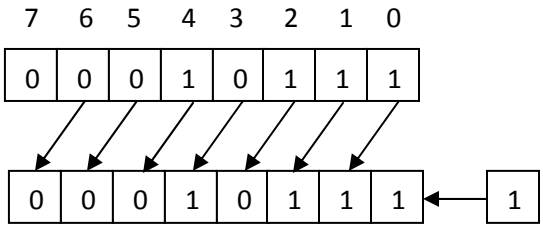


Figura 4.1 desplazamiento a la izquierda de un número de 8 bits.

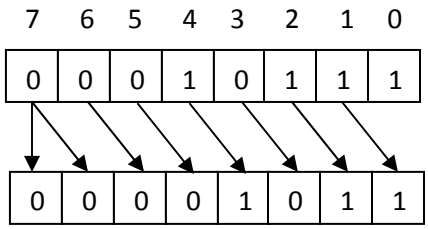


Figura 4.2 Desplazamiento a la derecha de un número de 8 bits.

4.1 Algoritmo clásico de multiplicación

El algoritmo clásico de multiplicación funciona bajo el conocido método de multiplicación decimal. Consiste de un multiplicador a, un multiplicando b y un producto p = a x b. la figura x muestra la multiplicación completa de dos palabras de cuatro bits.

$$
\begin{array}{cccccccc}
 & & & & a_3 & a_2 & a_1 & a_0 \\
 & & & \times & b_3 & b_2 & b_1 & b_0 \\
\hline
 & & & & b_0a_3 & b_0a_2 & b_0a_1 & b_0a_0 \\
 & & & b_1a_3 & b_1a_2 & b_1a_1 & b_1a_0 & \\
 & & b_2a_3 & b_2a_2 & b_2a_1 & b_2a_0 & & \\
 & b_3a_3 & b_3a_2 & b_3a_1 & b_3a_0 & & & \\
\hline
p_7 & p_6 & p_5 & p_4 & p_3 & p_2 & p_1 & p_0
\end{array}
$$

Figura 4.3 multiplicación de dos palabras de 4 bits.

Al igual que en la multiplicación decimal, el primer bit del multiplicando b_0 se combina con todos los bits de a para formar el primer producto parcial. Los demás productos parciales se generan de la misma forma y el producto final es la suma de los productos parciales. Dentro de la multiplicación binaria, cuando se requiere hacer productos con números negativos (en formato de complemento a dos) se debe de considerar que los bits que se encuentran después del bit de signo son importantes y deben tomarse en cuenta. De esta manera el tamaño de palabra aumenta al doble de su tamaño original. Tomando en cuenta esto, el proceso de multiplicación con signo o sin signo es idéntico pero el tamaño de palabra se duplica.

Si x e y son números naturales representados por cadenas de n dígitos X e Y en un sistema base r, la operación de multiplicar produce p = x*y, representándose P

mediante una cadena de dígitos P. La multiplicación convencional se realizaría de la siguiente forma:

$$
\begin{array}{cccccccc}
 & & & X_3 & X_2 & X_1 & X_0 & \\
 & & & Y_3 & Y_2 & Y_1 & Y_0 & \\
\hline
 & & & X_3Y_0 & X_2Y_0 & X_1Y_0 & X_0Y_0 & \times Y_0 \\
 & & X_3Y_1 & X_2Y_1 & X_1Y_1 & X_0Y_1 & & \times Y_1\, r \\
 & X_3Y_2 & X_2Y_2 & X_1Y_2 & X_0Y_2 & & & \times Y_2\, r^2 \\
X_3Y_3 & X_2Y_3 & X_1Y_3 & X_0Y_3 & & & & \times Y_3\, r^3 \\
\hline
P7 & P6 & P5 & P4 & P3 & P2 & P1 & P0
\end{array}
$$

Del lado derecho se muestran los renglones de la multiplicación en una expresión más simplificada. El producto p se puede describir también de la siguiente forma:

$$x * y = x\sum_{i=0}^{n-1} Y_i r^i = \sum_{i=0}^{n-1} xY_i r^i \qquad \text{Ecuación 4.1}$$

Esta operación indica que primero se calculan los n términos Xr^iY_i y finalmente se calcula la suma. La mecanización de este método no es adecuada ya que requiere excesiva memoria y la capacidad de realizar sumas con operandos múltiples.

4.2 Multiplicador secuencial basado en sumas y desplazamientos

El multiplicador secuencial de números binarios sin signo de N bits basado en sumas y desplazamientos sucesivos utiliza un sumador completo y registros de almacenamiento y/o de desplazamiento, para acumular los productos parciales.

Debido a que se trabaja con números signados, se requiere también incluir dentro del algoritmo un método por el cual se discrimine el signo de los multiplicandos de tal manera que se determine el signo del producto final.

El algoritmo de sumas y desplazamientos genera, empleando la notación de la sección pasada, los términos xY_j y se van sumando uno a uno, teniendo en cuenta que antes de sumar los productos parciales, hay que desplazarlos.

Este algoritmo puede definirse mediante la siguiente recurrencia:

$p^{(0)} = 0$

$p^{(j+1)} = (p^{(j)} + r^n xY_j)1/r$ para j=0,1,…,n-1 Ecuación 4.2

En el caso de trabajar con 4 bits, tenemos que:

$p^{(0)} = 0$

$p^{(1)} = (r^4 xY_0)1/r = r^3 xY_0$

$p^{(2)} = (r^3 xY_0 + r^4 xY_1)1/r = r^2 xY_0 + r^3 xY_1$

$p^{(3)} = (r^2 xY_0 + r^3 xY_1 + r^4 xY_2)1/r = r^1 xY_0 + r^2 xY_1 + r^3 xY_2$

$p^{(4)} = (r^1 xY_0 + r^2 xY_1 + r^3 xY_2 + r^4 xY_3)1/r = xY_0 + r^1 xY_1 + r^2 xY_2 + r^3 xY_3$

Por lo tanto, el producto de dos números de n bits se obtiene en n pasos mediante este método. Los productos parciales se suman en los bits más significativos, y el resultado se desplaza un bit a la derecha (1/r). Para esto se necesita un sumador de n bits.

La siguiente figura representa un paso del algoritmo:

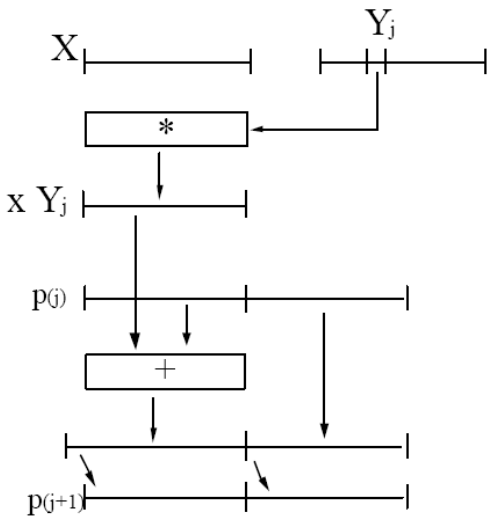


Figura 4.4 Un paso del algoritmo secuencial baso en sumas y desplazamientos a la derecha.

Se toma el bit j de Y y se multiplica por X, obteniendo XY_j, a continuación se suma con la parte alta de p(j) y se le aplica un desplazamiento.

La implementación de este algoritmo resulta atractiva con números de base r=2, es decir, binarios, ya que xY_j resulta 0 o bien x, dependiendo de que Y_j sea 0 ó 1.

A continuación se presenta una prueba de escritorio del algoritmo en cuestión con dos números de 4 bits.

n =5	r =2	x=23 X = 1 0 1 1 1	y =26 Y = 1 1 0 1 0
xY_0	0 0 0 0 0		
$p^{(0)}$	0 0 0 0 0		
	0 0 0 0 0		
xY_1	1 0 1 1 1		
$p^{(1)}$	0 0 0 0 0	0	
	1 0 1 1 1		
xY_2	0 0 0 0 0		
$p^{(2)}$	0 1 0 1 1	1 0	
	0 1 0 1 1		
xY_3	1 0 1 1 1		
$p^{(3)}$	0 0 1 0 1	1 1 0	
	1 1 1 0 0		
xY4	1 0 1 1 1		
p(4)	0 1 1 1 0	0 1 1 0	
	1 0 0 1 0 1		
p(5)	**1 0 0 1 0**	**1 0 1 1 0**	**= 598**

Figura 4.5 Prueba de escritorio del algoritmo

El diagrama de bloques del algoritmo para el multiplicador secuencial es el siguiente:

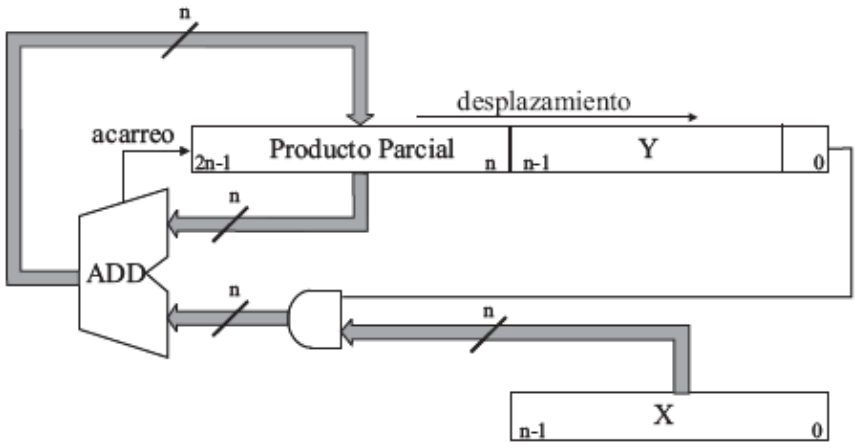


Figura 4.6 Diagrama de bloques del multiplicador secuencial basado en sumas y desplazamientos.

Los multiplicandos son X e Y, dos números de n bits, y el producto total de 2n bits. Dentro del registro del producto total se guarda en la parte baja el multiplicando Y, mientras que la parte alta se llenará de ceros. El diagrama nos muestra que se hacen las multiplicaciones parciales del bit menos significativo de Y con todos los bits de X. En seguida se realiza una suma del producto parcial con la parte alta del producto total y el resultado se guarda en la misma parte alta del producto total. Finalmente se hace un desplazamiento del registro del producto total, obteniendo así un nuevo bit menos significativo de Y con el cual se obtiene un nuevo producto parcial. Este proceso se repite n veces y al final se obtiene el resultado en el registro del producto final.

4.3 Multiplicador Ripple Carry

El multiplicador basado en un arreglo de sumadores de acarreo propagado es una aproximación para implementar el algoritmo de sumas sucesivas y desplazamientos; este tipo de multiplicador tiene la característica de transferir la propagación del acarreo a la siguiente suma parcial hasta que terminen los productos parciales correspondientes a esta fila, en este instante el acarreo generado se propaga al último producto de la siguiente fila de productos parciales y así sucesivamente hasta culminar la multiplicación.

El bloque principal es un sumador completo de 1 bit, y el diagrama de bloques de un multiplicador ripple carry de 4 bits se muestra en la Figura 4.7.

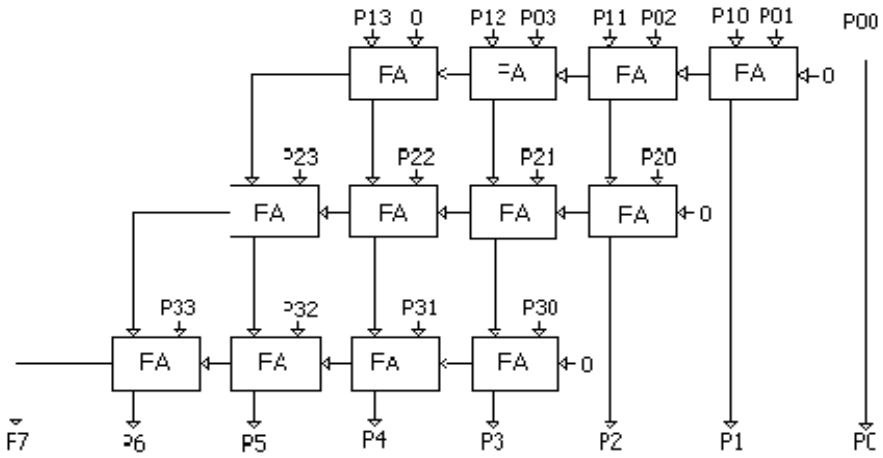


Figura 4.7 Multiplicador Ripple Carry.

4.4 Multiplicador Carry Save

El multiplicador basado en el arreglo de sumadores con acarreo salvado es otra aproximación para implementar el algoritmo de sumas sucesivas y desplazamientos. La idea es romper la cadena de acarreo propagado del sumador ripple para disminuir el retardo de cada suma, lo cual permite acelerar la multiplicación. El multiplicador tiene la característica de permitir salvar el acarreo generado en las sumas parciales y transferirlo como acarreo de entrada a la siguiente suma parcial de la fila de productos parciales siguiente. Este multiplicador es también conocido con el nombre de Multiplicador de Braun. El diagrama de un multiplicador carry save de 4 bits, se muestra en la Figura 4.8.

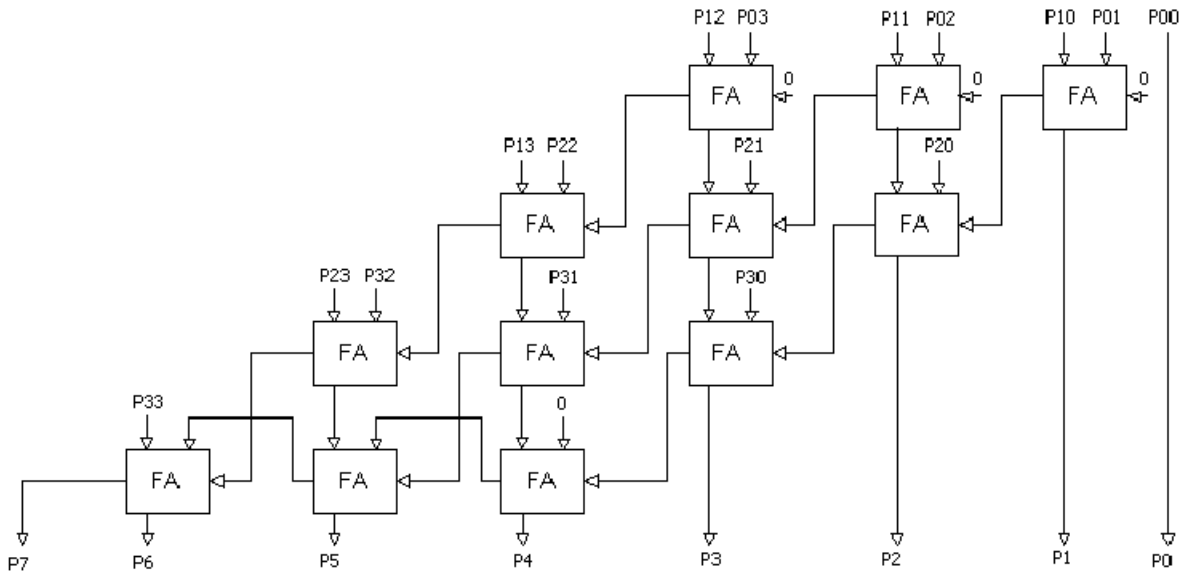


Figura 4.8 Multiplicador Carry Save.

4.5 Multiplicador de Wallace

El multiplicador de Wallace es una variante del algoritmo de sumas sucesivas y desplazamientos, donde se utilizan los bloques de sumadores completos con sus tres entradas recibiendo términos productos y generando un término suma que se adiciona con otro termino suma. El diagrama de bloques de un multiplicador de Wallace de 4 bits se muestra en la Figura 4.9.

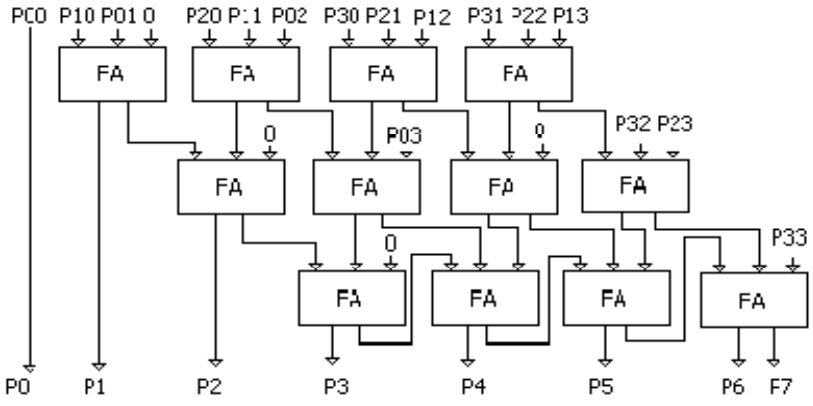


Figura 4.9 Multiplicador de Wallace.

4.6 Multiplicador de Baugh-Wooley

El multiplicador de Baugh-Wooley permite realizar la multiplicación de números con signo usando la representación en complemento a dos. Este multiplicador es una adecuación del algoritmo de sumas sucesivas y desplazamientos que permite realizar la multiplicación de números con signo. Este tipo de multiplicador está basado en arreglos de sumadores de acarreo salvado. El diagrama de un multiplicador de Baugh–Wooley de 4 bits se muestra en la Figura 4.10.

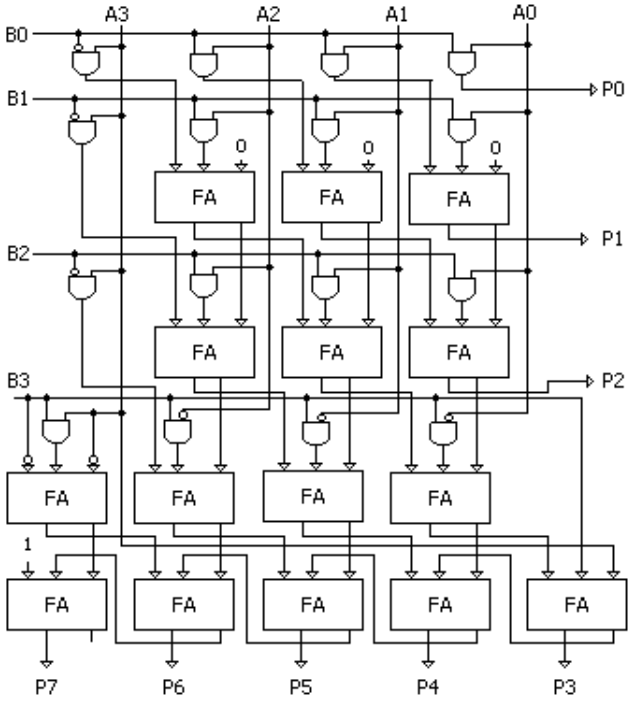


Figura 4.10 Multiplicador Baugh-Wooley.

4.7 Algoritmo de Booth

Como se ha visto, la multiplicación se basa en dos operaciones básicas: la suma y el desplazamiento. De estas dos, la que consume más tiempo es la suma. El algoritmo de Booth pretende minimizar el número de sumas que se deben realizar durante el proceso de multiplicación para reducir el tiempo en el que se realiza.

El algoritmo de Booth realiza multiplicaciones de números binarios con signo. De esta forma, los números negativos deben tener el formato de complemento a dos.

El algoritmo de Booth funciona haciendo una recodificiación del multiplicador, analizando los últimos dos bits (B_i y B_{i-1}) de acuerdo a la tabla que se muestra a continuación.

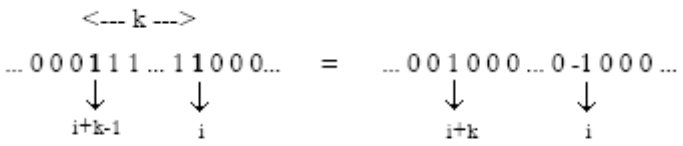
Bits del multiplicador B_i	B_{i-1}	Dígito recodificado	Operación
0	0	0	0 x multiplicando
0	1	1	1 x multiplicando
1	0	-1	-1 x multiplicando
1	1	0	0 x multiplicando

Tabla 4.1 Recodificación del algoritmo de Booth

Como se puede observar de la tabla, el algoritmo de Booth agiliza la multiplicación al encontrar cadenas de unos consecutivos. Esto se basa en que una cadena de unos se puede reducir usando la siguiente equivalencia:

$$2^{i+k-1} + \ldots + 2^{i+1} + 2^{i} = 2^{i+k} - 2^{i} \qquad \text{Ecuación 4.3}$$

Si se encuentra una cadena de k bits a 1, desaparece y se sustituye por una más simple que tiene un 1 al inicio (dígito más significativo en la cadena sustituida) y un -1 al final, dejando los dígitos intermedios en 0.



En el número recodificado, el peso del dígito D_i sigue siendo 2^i para cualquier dígito, sólo que ahora se aceptan números negativos.

Ejemplos:

109	→	0110 1101	→	10-11 0-11-1 (128-32+16-4+2-1 = 109)
-109	→	1001 0011	→	-101-1 010-1 (-128+32-16+4-1= -109)

Al realizar la multiplicación se usa la nueva codificación de modo que si se trata de un cero sólo se desplaza, si es un 1, se realiza una suma y el desplazamiento, y si se encuentra un -1, lo que se hace es una resta y el desplazamiento. Todas las iteraciones son iguales. El número de iteraciones que se realizan es igual al número de bits de los multiplicandos.

A continuación se muestra una prueba de escritorio del algoritmo de Booth.

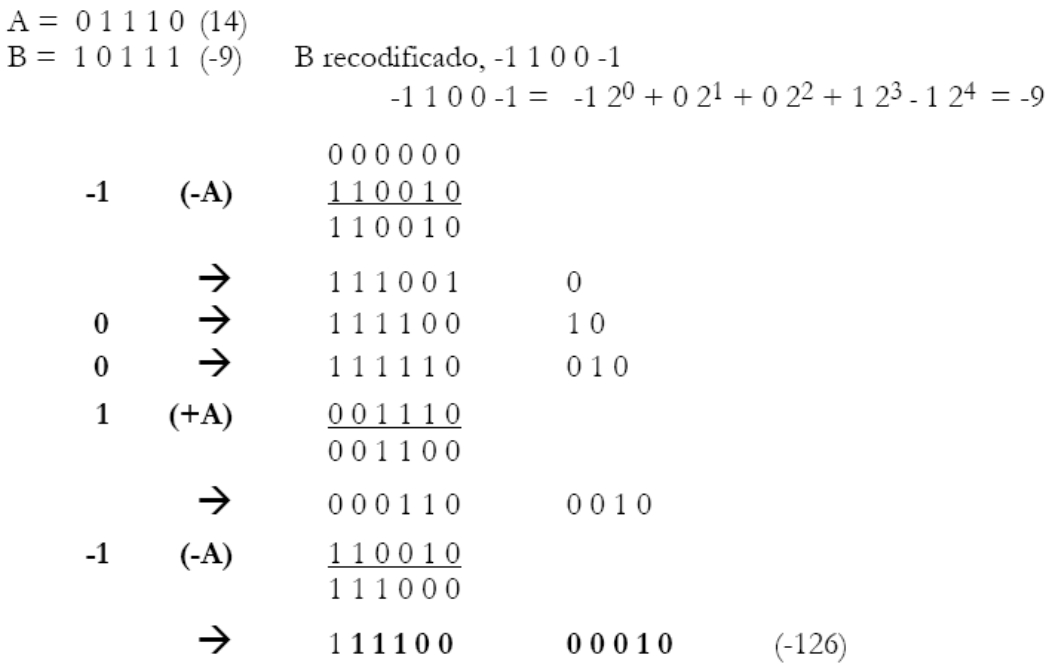


Figura 4.11 Prueba de escritorio del algoritmo de Booth

En la figura anterior, del lado izquierdo se muestra el dígito recodificado en cada paso, con lo que se decide si se hace una suma, una resta o simplemente un desplazamiento. Los desplazamientos se pueden observar en la figura, cada vez que se encuentra una flecha, lo cual produce un corrimiento de bits a la derecha.

Se puede observar que al encontrarse varios unos o ceros seguidos, se omiten sumas o restas con lo que se ahorra el tiempo que tomaría realizarlas, reduciendo el tiempo total de ejecución. El peor caso en este algoritmo resultaría ser claramente el que no incluya cadenas de ceros seguidos o de unos seguidos, ya que se realizaría la suma, o resta, en cada paso del proceso. Es por esto que el peor caso del algoritmo de Booth se asemeja mucho al tiempo que tarda el algoritmo de sumas y desplazamientos, sin embargo este tiempo se puede ver reducido significativamente en otros casos.

La siguiente figura muestra el diagrama de flujo del algoritmo de Booth.

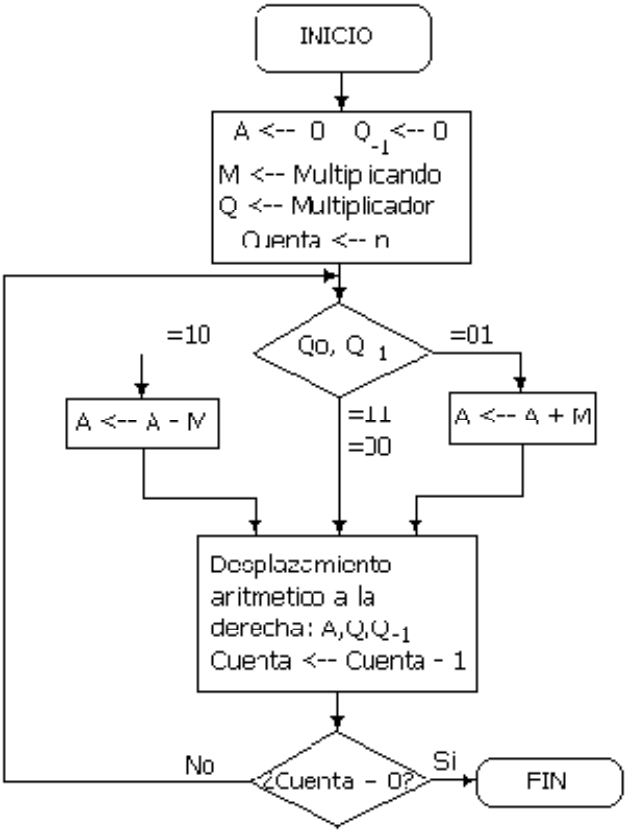


Figura 4.12 Algoritmo de Booth.

En el estado inicial se mandan a cero los bits del vector A, y al multiplicador Q se le agrega un 0 como bit menos significativo para realizar la primera comparación. Se tiene también un contador de ciclos inicializado en n= número de bits de los multiplicandos. En cada ciclo se leen los últimos dos bits de Q y se decide si se hace la suma o resta de A y M, o simplemente se omite esa operación. Posteriormente se hace un desplazamiento a la derecha del vector que contiene A y Q, el cual contendrá el resultado final, y se disminuye el contador. Finalmente se checa si ya se realizaron todos los ciclos, en caso de que no, se vuelve a realizar

otro, y en el caso de que el contador haya llegado a cero, ya se ha terminado la multiplicación.

Mega función de Altera

Altera provee una librería de mega funciones parametrizadas, dentro de la cual se encuentra la mega función *lpm_mult*, la cual es un bloque parametrizado descrito en lenguaje de alto nivel. Sin embargo no se recomienda usarla, puesto que utiliza demasiada área (elementos lógicos) dentro del dispositivo lógico programable.

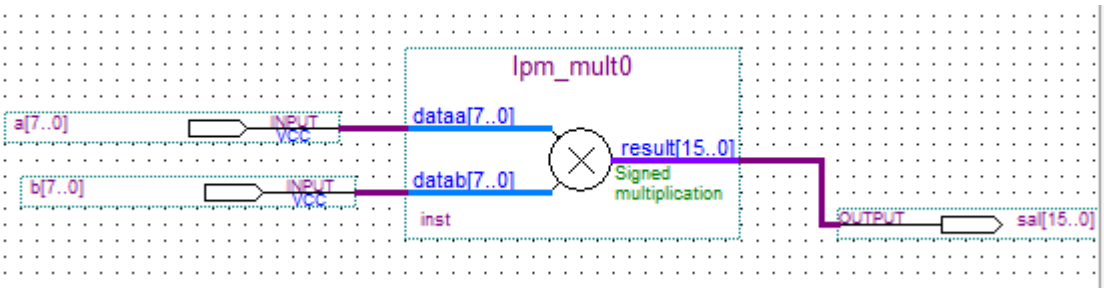


Figura 4.13 Bloque de la mega función LPM_MULT.