Capítulo III

Metodología de Operación

1. Estructura de los operadores
2. Operaciones aritméticas de números enteros

III - 1

Estructura de los operadores

La expresión sin error de valores racionales junto con la concepción y el diseño de operadores que procesan números expresados en este formato de representación componen los pilares sobre los que se fundamenta la computación exacta sobre los números racionales. Las funciones de interés serán aquellas que constituyen una ley de composición interna entre los elementos del conjunto $\mathbb{Q}$.

Con esos principios, se establece un modelo de computación formado por el esquema de representación numérica formalizado mediante la función identidad y las funciones de suma y producto sobre números racionales. El modelo debe disponer de un funcionamiento flexible para procesar operandos de distinta longitud y considerar, asimismo, elementos en el problema que limiten la precisión del resultado. En relación con este último aspecto, la instrumentación en precisión variable de las funciones incorpora un parámetro $\overline{d}$ que refleja estos requerimientos de acuerdo con la formulación del problema [1.6]:

$$\forall\ \overline{x} \in \text{dominio}(\Gamma_f),\ |\Gamma_f^{VP}(\overline{x},\ \overline{d}) - f(\overline{x})| \leq \varepsilon$$

11,0010010000111111010101000100010000101101000110000100011010011000100111000101000101110000000110111000001110011010001001010010000001001001110000010001
0001010011001111110011000111010 0

La metodología de operación que se presenta en este apartado presta especial atención a la estructura de los operadores así como a su implementación. El resultado de este examen determinará la conveniencia de aplicar métodos de cálculo basados en el aumento de tamaño de operación elemental. La operatoria del procesador debe sacar partido de la estructura interna de los operandos y aprovechar sus características, además, su carácter variable favorece el empleo de técnicas iterativas que, por repetición, sean capaces de procesar todos las cifras del número.

Granularidad de los operadores

La estructura de los datos que establece el formato propuesto resulta difícil de encajar en los métodos rígidos que procesan operandos con una cantidad fija de cifras. Esta circunstancia sugiere la necesidad de buscar arquitecturas que coordinen la estructura de los operandos y la metodología de operación y sean capaces de adaptarse a números de distinto tamaño.

Tradicionalmente, la mayoría de operadores elementales consideran al bit como la unidad mínima de información procesable. Un nuevo paso en su evolución consiste en aumentar la granularidad de los operadores y considerar como unidad mínima de cálculo un conjunto de bits.

Como aspecto importante para la construcción de esos operadores se destaca el *retardo temporal* de la operación. También la *complejidad de la unidad aritmética* y el *área ocupada* por los operadores debe considerarse sobretodo en el diseño e implementación del circuito. En esto último influye de manera esencial el *grado de paralelismo* y *reutilización* de los módulos que constituyen la propia unidad aritmética. Finalmente, también son deseables características que aportan calidad al procesamiento, como son la *robustez* de los resultados y la *tolerancia a fallos* de los operadores. Otros aspectos, como el *consumo de potencia*, no se contemplan en este trabajo.

Se denomina *k-operadores*, a los operadores que toman como unidad mínima de proceso una cantidad de k bits. La idea fundamental consiste en obtener ventajas en la instrumentación de las funciones

11,00100100001111110110101001000100001011010001100001000110100110001001100011001100101000010111000000011011100000111001101000100101001000000100100111000001000100010011001111110011000111010 0

aritméticas utilizando en su construcción elementos k-operadores. Este aumento de granularidad contiene una serie de mejoras inherentes en aquellos casos en los que es posible su aplicación frente a la operatoria a nivel de bit. La figura 3-1 muestra esquemáticamente la funcionalidad de un k-operador genérico y de un operador a nivel de bit.

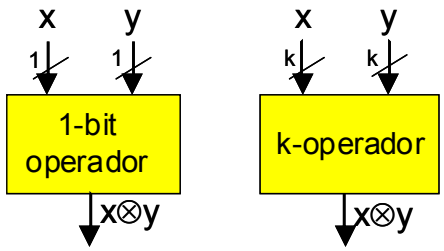

Figura 3-1: Operadores de 1 y k bits

Se simplifica la estructura de la unidad aritmética al utilizar menos unidades de procesamiento individual para operar grupos de bits. Por ejemplo el algoritmo de suma con propagación de acarreo (CPA —*Carry Propagate Adder*) [Zimmermann, 1987] utiliza tantas unidades sumadoras de un bit (FA—*Full-Adder*) como cifras tenga el número. En este caso, como se observa en la figura 3-2, la cantidad de lógica de interconexión debe ser igual a la cantidad de bits de los operandos.

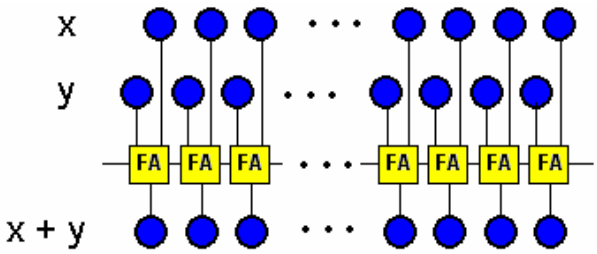

Figura 3-2: Suma CPA bit a bit

La figura 3-3 muestra cómo el diseño mediante k-operadores reduce la complejidad de interconexión al rebajar el número de enlaces entre los módulos de cálculo en un factor k. Alternativamente, con la misma cantidad de lógica de interconexión se opera con datos de mayor

11,0010010000111111010101000100001011010001100001000110101100010011000110010100010111000000011011100000111001101000100101001000000100100111000001000100010011001111100110001110100

tamaño. Esta cualidad puede mejorar algunas de las características de interés en el diseño de los operadores del procesador al reducir la complejidad de la unidad aritmética y favorece la construcción de unidades aritméticas flexibles que procesen distintos tamaños de operandos. En este aspecto la aplicación de métodos iterativos permitirá procesar progresivamente todas las cifras de los operandos, por ejemplo en una unidad de suma para números de distinta longitud [García et al, 2003a], [García et al, 2003b], [Mora, 2001].

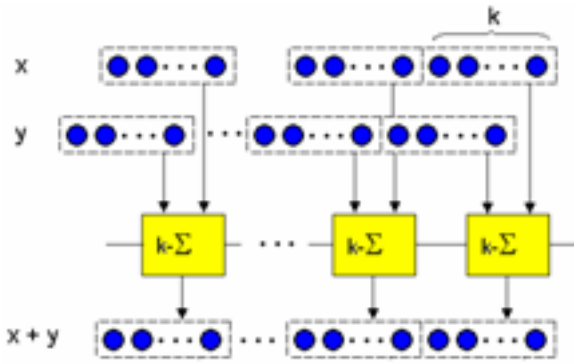

Figura 3-3: Suma CPA en bloques de k bits

Otra ventaja consiste en la mejor predisposición de la unidad para construir operadores paralelos que calculen simultáneamente varias operaciones autónomas similares. La disposición de cauces de entrada independientes y la lógica de conexión adecuada proporcionará un elevado nivel de paralelismo sobre operandos de longitud k o múltiplo de k. Esta idea es la base en el diseño de las unidades aritméticas de los procesadores multimedia [Conte et al, 1987], [Peleg y Weiser, 1996]. Estos diseños integran varias unidades procesadoras similares e independientes que operan en paralelo sobre datos de un tamaño fijo reducido, por ejemplo, muestras de sonido (16 bit), componentes de color de un pixel (8 bit), etc.

Como se observa en la figura 3-4, un operador formado por la misma cantidad de k-operadores que en la figura 3-3 se puede configurar para realizar varias operaciones en paralelo sobre datos independientes.

11,0010010000111111011010100010001000010110100011000010001101001100010011000110001010001011100000001101110000011100110100010010100100000010010011100000100010100110011111001100011101000

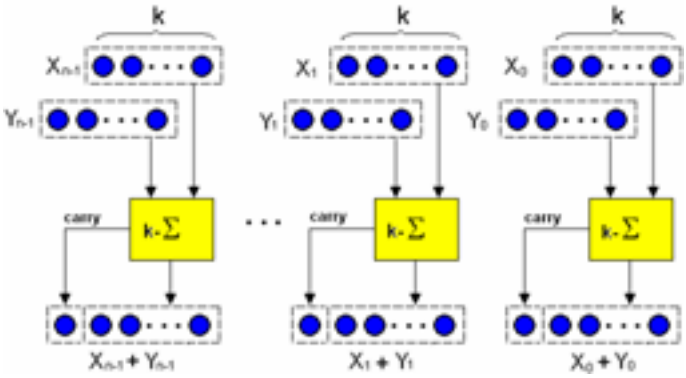

Figura 3-4: Unidad de suma con n módulos sumadores

El estudio de las operaciones de la unidad aritmética determinará aquellas que admiten su construcción mediante k-operadores que actúen sobre grupos de bits independientemente, así como su posterior combinación de los resultados correspondientes.

Para k=1 los k-operadores son idénticos a las unidades que trabajan bit a bit. Por tanto el aumento de la granularidad de un operador supone una generalización de éstos.

Diseño de los k-operadores

El objetivo de diseño de los k-operadores consiste en proporcionar una arquitectura que mejore las prestaciones del microprocesador para el mayor número de funciones. Se plantean las siguientes alternativas:

Diseño basado en lógica combinacional

Esta opción consiste en construir un k-operador mediante un circuito combinacional que calcule el resultado de la operación para k bits. Éste circuito puede estar compuesto a su vez por la conjunción de k operadores elementales bit a bit. La figura 3-5 muestra el sumador CPA con unidades k-procesadoras formadas por sumadores completos de un bit. En este caso, la interconexión entre ellos será la misma que la que se aplica entre los propios k-operadores.

11,0010010000111111010101000100010000101101000110000100011010011000100111000110011001010001011100000001101110000011100110100010010100100000010010011100000100010001010011001111110011000111010 0

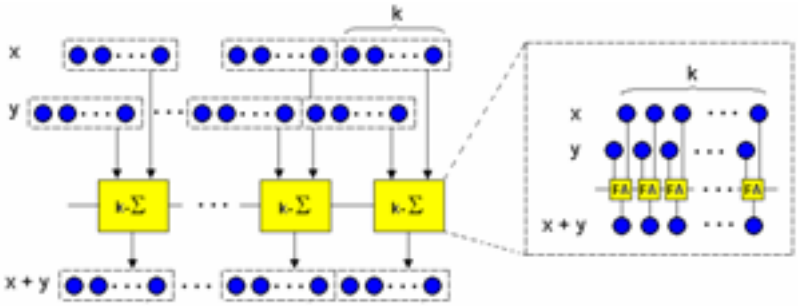

Figura 3-5: Diseño combinacional de los k-operadores

Aunque este diseño siempre será posible de realizar, no garantiza ventajas en las prestaciones de la unidad aritmética frente a las operaciones bit a bit.

Alternativamente, se plantean métodos de cálculo basados igualmente en circuitos combinacionales que obtengan el resultado completo de la función a computar. Por ejemplo para la función suma, los algoritmos de suma de anticipación de acarreo (CLA —*Carry Look Ahead Adder*) y suma condicional (COSA —*Conditional Sum Adder*) [Zimmermann, 1987]. Cada diseño posee unas características propias que pueden mejorar algunos de los aspectos de la operación que aconsejan su utilización en determinadas situaciones.

Diseño basado en lógica almacenada

Los avances en la tecnología de fabricación de circuitos y dispositivos permiten concebir nuevos métodos de operación que hubieran sido prohibitivos tiempos atrás. El procedimiento que se propone para diseñar los nuevos operadores consiste en la utilización de memorias de acceso rápido, también llamadas tablas look-up (LUT —*Look-Up Table*) como medio para realizar el cálculo efectivo.

La estructura de memoria contiene, para cualquier par de bloques de k bits, el resultado precalculado de la operación. Estas tablas look-up almacenan todos los resultados para operandos de tamaño k, de manera que tan sólo hay que seleccionar la celda en la que se encuentra el resultado en función del propio valor de los operandos sin realizar

11,00100100001111110110101001000100001011010001100001000110100110001001100011001101000101000101110000000110111000001110011010001001010010000001001001110000010001
000101001100111111001100011110100

ningún otro procesamiento más [García et al, 2003b], [Mora, 2001]. La figura 3-6 muestra esquemáticamente un k-operador suma basado en esta arquitectura.

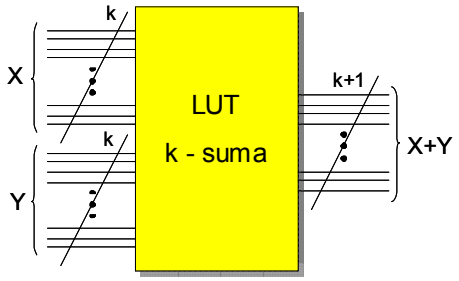

Figura 3-6: Unidad procesadora para la operación suma basada en lógica almacenada

Este procedimiento no es generalizable para todas las operaciones ni produce ventajas en todos los casos, por lo que se debe estudiar cada operación para comprobar sus beneficios. En el siguiente apartado se presenta un modelo de operación basado en este método de cálculo.

Modelo de operación basado en lógica almacenada

La idea principal consiste en implementar las operaciones del procesador mediante k-operadores construidos según un diseño de lógica almacenada.

La realización de los k-operadores mediante tablas look-up permite una mayor densidad de integración en las implementaciones VLSI que los otros métodos combinacionales de cálculo. Su utilización rebaja los costes en el desarrollo del hardware, aporta flexibilidad y reduce la cantidad de módulos requeridos en el diseño de microprocesadores. Además, estos dispositivos pueden incorporar elementos de detección de errores y corrección de los datos y, por tanto, de los resultados que se producen [Parhami, 2000].

Las posibilidades de construcción de las memorias pueden ser aprovechadas para proporcionar solidez y flexibilidad al sistema. El uso

11,0010010000111111011010100010000100010110100011000010001101001100011001100110010100010111000000011011100000111001101000100101001000000100100111000001000100010100110011111001100011101100

de memorias de sólo lectura es una opción más robusta que los circuitos combinacionales y, alternativamente, el uso de memorias de lectura y escritura permite configurar varias funciones distintas en el mismo circuito lógico facilitando su mantenimiento y reparación [Wong y Goto, 1995], [Tang, 1991].

La naturaleza de los dispositivos de memoria facilita tanto su reutilización como un alto grado de paralelismo. Memorias multipuerto con varios canales de acceso obtienen cauces de resultados paralelos en la misma pastilla de almacenamiento. En este sentido se pueden tomar decisiones de diseño y escoger elementos de acceso múltiple o estructuras compuestas por varios elementos similares con acceso concurrente.

En relación con el coste temporal, el retardo del circuito está formado por el tiempo necesario en acudir a la tabla y tomar el dato y depende de las vías de comunicación con la memoria, su estructura interna y la tecnología de fabricación [Nambu et al, 1998], [Wilton y Jouppi, 1994], [Wada et al, 1992]; además es constante e independiente del valor de los operandos. Si éstos tienen una longitud mayor que k se deberá tener en cuenta también el tiempo de combinación de los resultados parciales para conformar el resultado final.

11,0010010000111111011010100010001000010110100011000010001101001100010011000110011000101000101110000000110111000001110011010001001010010000001001001110000010001
0001010011001111110011000111101 00

Sea:

$T(\Gamma_f^n)$: Tiempo en ejecutar la función f para operandos de tamaño n según la implementación Γ.

$T(\Delta M_f^k)$: Tiempo de respuesta de una memoria según un diseño Δ que contiene todos los resultados parciales de la operación f para operandos de tamaño k.

$T_{\Delta M}$: Tiempo de acceso a la memoria de diseño Δ.

$T(C_f^{n/k})$: Tiempo de composición de los resultados parciales de la función f para operandos de longitud n calculados en bloques de tamaño k.

Los métodos basados en resultados almacenados representan una mejora en rendimiento cuando el retardo conjunto empleado en el cálculo de la función sea inferior al de otras implementaciones de f, tal como formula la expresión siguiente.

$$T(\Delta M_f^k) + T_{\Delta M} + T(C_f^{n/k}) < T(\Gamma_f^n) \qquad [3.1]$$

Las distintas arquitecturas y tecnologías de memorias poseen expresiones de $T(\Delta M_f^k)$ diferentes. En este aspecto, el desarrollo de la tecnología juega un papel determinante en la mejora de las prestaciones y la reducción del coste. Los sucesivos avances tecnológicos permiten aumentar el tamaño de k para el cálculo de operandos cada vez mayores y disminuir el retardo de combinación $T(C_f^{n/k})$. Asimismo, la ubicación de las tablas de resultados en la propia unidad aritmética junto al resto de la lógica de la operación minimiza el coste de acceso a los datos $T_{\Delta M}$ [Carr, 1993].

La complejidad espacial de los k-operadores basados en tablas look-up crece exponencialmente con la longitud de los operandos. Esta circunstancia limita su uso para el caso general y afecta al valor deseable de k que maximiza el rendimiento, lo que da lugar a una situación de compromiso entre el valor de k y el tamaño de la memoria necesaria [García et al, 2003b], [Mora, 2001]. Una solución para reducir

11,0010010000111111011010100010000100010110100011000010001101001100011001100110001010001011100000001101110000011100110100010010100100000010010011100000100010100100111110011000111010

el coste espacial consiste en aplicar etapas de preprocesamiento y postprocesamiento que reduzcan la cantidad de entradas necesarias, con el consiguiente aumento de la complejidad temporal.

III - 2

Operaciones aritméticas de números enteros

En los objetivos de este trabajo se encuentra la concepción de un procesador flexible que contenga una instrumentación de funciones con características de precisión variable. Estas operaciones tienen la consideración de primitivas del procesador, lo que impone severas restricciones de eficiencia y de complejidad.

La traducción de los números al esquema de representación propuesto condiciona fuertemente el diseño de las funciones. Los operandos se componen de una serie de campos de distinta longitud de tipo entero. La primera consecuencia de ello será la naturaleza de la relación entre la implementación realizada y las funciones matemáticas que definen.

El desarrollo de los operadores recoge la reflexión sobre esos aspectos y se sustenta tanto en la metodología de cálculo que se ha descrito como en la operatoria con números enteros. Por esta razón, se presenta en primera instancia los métodos de suma y producto para valores de esta naturaleza como base de las operaciones individuales entre las partes de los operandos.

11,0010010000111111011010100010001000101101000110000100011010011000100111000110011000101000101110000000110111000001110011010001001010010000001001001110000010001000101001100111111001100011101001

Suma de números enteros

El método de cálculo de la función suma para operandos de distinta longitud consiste en un esquema iterativo que va construyendo el resultado de forma incremental en cada paso. A grandes rasgos la operación consiste en dividir los datos en partes manejables del mismo tamaño, sumar por separado cada parte y componer finalmente los resultados parciales por combinación. En este método se obtienen ventajas del aumento de granularidad de las operaciones y de la operatoria basada en LUT.

En la siguiente figura se observa el esquema de fragmentación y suma parcial de los operandos. Los números se alinean desde la derecha para hacer corresponder las cifras significativas del mismo orden de magnitud. Las posiciones que queden vacías por la izquierda se completan con una extensión del signo del número más corto.

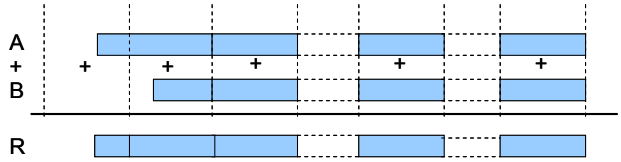

Figura 3-7: Fragmentación de los operandos y suma parcial

Las operaciones de suma de las partes se ejecutan secuencialmente comenzando desde las posiciones menos significativas. De este modo, el proceso de combinación y de formación del resultado se simplifica al considerar el acarreo serie y propagarlo de una parte hacia la siguiente siguiendo un modelo de sumador CPA. La cantidad de iteraciones necesarias para completar el procesamiento del número dependerá del tamaño de los sumandos parciales y de la longitud total de los números. Este aspecto debe buscar un equilibrio entre la complejidad espacial y temporal del operador completo intercambiando número de iteraciones por complejidad del sumador de cada parte.

El coste temporal de la operación completa depende del tiempo empleado en cada suma parcial así como del número de iteraciones que se realizan. El retardo de la operación parcial es constante debido a que

se suma una cantidad de cifras fija en cada iteración mientras que la cantidad de iteraciones es directamente proporcional a la longitud total de los sumandos. Por estos motivos, la expresión asintótica de la complejidad temporal de la operación completa es lineal con la cantidad de cifras del número mayor e independiente del método utilizado para la concatenación de las sumas parciales de bloques. Su retardo sólo influirá en la constante multiplicativa de la complejidad de la suma completa. La siguiente expresión ilustra el coste temporal,

$$T_{suma\text{-}\mathbb{Z}} \in O(n_A, n_B) \qquad [3.2]$$

donde n_A y n_B representan la cantidad de cifras de cada número respectivamente.

Suma de operandos de longitud fija

La suma de números enteros de longitud fija se puede instrumentar mediante cualquier método presente en la literatura especializada [Cheng et al, 2000], [Takagi y Horiyama, 1999], [Parhi, 1997], [Srinivas y Parhi, 1992], [Quach y Flynn, 1990], [Wei y Thompson, 1990], [Zimmermann, 1987], [Brent y Kung, 1982]. En esta memoria se propone realizar esta operación siguiendo la metodología de cálculo descrita en el apartado anterior. Esta técnica contempla un aumento de la granularidad de la operación agrupando las cifras a sumar en bloques de k dígitos y el uso de lógica almacenada.

El método que se utiliza para ejecutar la suma parcial se compone de tres etapas [García et al, 2003c], [Mora, 2001]: fragmentación de los sumandos en bloques de tamaño k, suma de todos los pares de bloques mediante acceso a una memoria con resultados almacenados y la posterior combinación ordenada de los resultados obtenidos considerando la lógica de los acarreos. La figura siguiente muestra la etapa de obtención de los resultados precalculados.

11,0010010000111111011010100010000101101000110000100011010011000100111000101000101110000000110111000001110011010001001010010000001001001110000010001
0001010011001111110011000111010 0

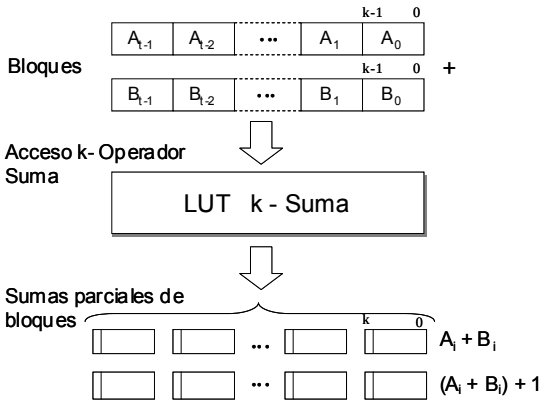

Figura 3-8: Obtención de los resultados precalculados de una suma parcial

Como se observa en la figura anterior, para cada par de bloques se extrae la suma y su sucesor de una LUT k-suma. La combinación de estas sumas parciales se realiza según una estructura que puede ser secuencial o en árbol [García et al, 2003c], [Mora, 2001]. Será decisión de diseño decidir el procedimiento a utilizar. Las figuras 3-9 y 3-10 muestran esquemáticamente el proceso de combinación en cada caso.

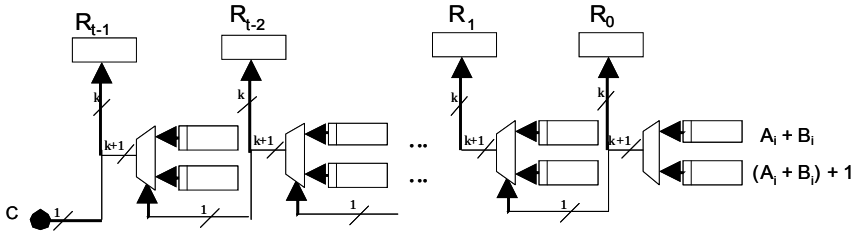

Figura 3-9: Combinación secuencial de los resultados precalculados de la suma parcial

11,00100100001111110110101010001000100001011010001100001000110101001100011001100110001010001011100000001101110000011100110100010010100100000010010011100000100010100110011111001100011110100

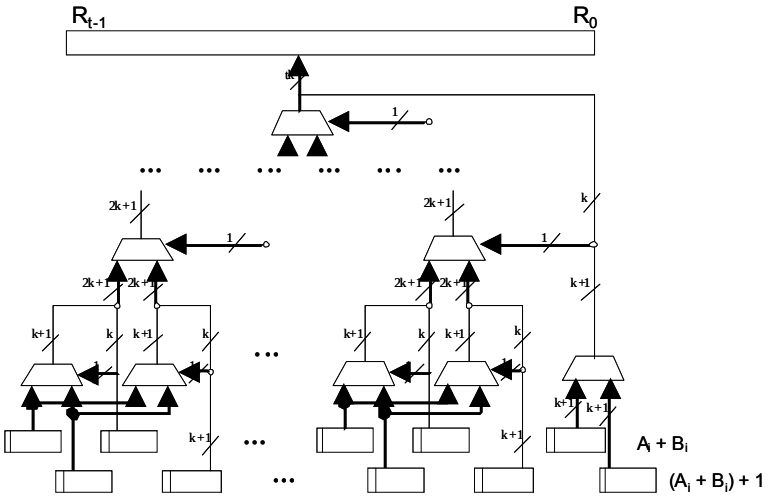

Figura 3-10: Combinación en árbol de los resultados precalculados de la suma parcial

La complejidad espacial de la operación depende en gran medida del tamaño del k-operador basado en lógica almacenada. Éste se calcula considerando las líneas de dirección de la memoria y la longitud de cada celda. Para almacenar la suma de todos los números de k cifras se necesita una memoria de $2^k \cdot 2^k \cdot (k+1)$ bits. Sin embargo, como la LUT es simétrica sólo es necesario almacenar la mitad más la diagonal principal. La siguiente expresión muestra la cantidad total de memoria necesaria.

$$M_{k\text{-suma}} = 2^{k-1} \cdot (2^k + 1) \cdot (k+1) \text{ bits} \qquad [3.3]$$

La siguiente tabla muestra la complejidad espacial del k-operador suma para distintas instancias de k.

k	$M_{k\text{ - suma}}$
1	6 bits
2	30 bits
4	680 bits
6	1,77 KB
8	36,14 KB

11,0010010000111111011010100010000100010110100011000010001101001100011000100111000101000101110000000110111000001110011010001001010010000001001001110000010001
0001010011001111110011000111010 0

16	≈ 4,4MB
32	>1 GB

Tabla 3-1: Tamaño del k-operador suma basado en lógica almacenada

Como se observa, la memoria aumenta de forma exponencial con el valor de k. Debido a esto, se tiene que buscar un equilibrio entre la memoria necesaria y la complicación del circuito que se requiere.

Multiplicación de números enteros

La importancia del operador de multiplicación se constata por su frecuente utilización en la mayoría de aplicaciones [Ing-Jer y Tzu-Chin, 1998], [Oberman, 1996]. A lo largo de la historia se han propuesto numerosos métodos y algoritmos de multiplicar con la finalidad de mejorar su rendimiento. Entre las propuestas más significativas se encuentran aquellas que se centran en reducir la complejidad asintótica del procedimiento de cálculo mediante la reducción sucesiva del tamaño de los operandos y el uso de la transformada rápida de Fourier (FFT – *Fast Fourier Tramsform*).

11,00100100001111110110101001000100001011010001100001000110101100010011000110010100010111000000011011100000111001101000100101001000000100100111000001000100010011001111110011000111010 0

La siguiente tabla muestra algunas propuestas representativas:

Método	Complejidad
[Karatsuba y Ofman, 1963]	$O(n^{1,585})$
[Toom, 1963]	$O(n^{1,465})$
[Bailey, 1993] [Schönhage y Strassen, 1971]	$O(n \lg(n) \lg(\lg(n)))$

Tabla 3-2: Complejidad temporal de diversos algoritmos del producto

Todos esos métodos de multiplicar obtienen una mejor expresión analítica del coste que el algoritmo clásico, $O(n^2)$ [Parhami, 2000], [Bewick y Flynn, 1992]; sin embargo, poseen una excesiva complejidad de su metodología de cálculo que los convierten en difíciles de implementar en un computador. El estudio realizado por Zuras [Zuras, 1994] demuestra que, tras su implementación hardware, los métodos rápidos anteriores no proporcionan ventajas en el rendimiento, ya que la complejidad del circuito resultante provoca unos retardos que no compensan la mejora de la expresión asintótica [Zuras, 1994].

Por esas consideraciones se toma como punto de partida el método clásico de multiplicar adaptándolo a operandos de longitud variable. La operación clásica se compone de las conocidas etapas de generación y reducción de productos parciales y suma final [Wen-Chang y Chein-Wei, 2000], [Oberman, 1996], [Bewick, 1994]. El algoritmo que se utiliza aplica un esquema iterativo para construir el resultado progresivamente mediante una multiplicación por columnas [Chen-Ying, 1996]. De forma resumida, el procedimiento de cálculo consiste en fragmentar los operandos en partes manejables, multiplicar cada una de esas porciones y combinar sucesivamente los resultados parciales para obtener el resultado final de la operación. La cantidad de partes en las que se fragmentan los operandos determina el número de multiplicaciones y sumas parciales según una relación cuadrática.

La figura 3-11 muestra de forma esquemática el proceso de multiplicación, donde se observa la formación del resultado desde las

11,0010010000111111010101000100010000101101000110000100011010011000100011000110001010001011100000001101110000011100110100010010100100000010010011100000100010100110011111001100011101000

posiciones menos significativas. En el ejemplo se considera, sin pérdida de generalidad, que ambos operandos tienen el mismo número de cifras.

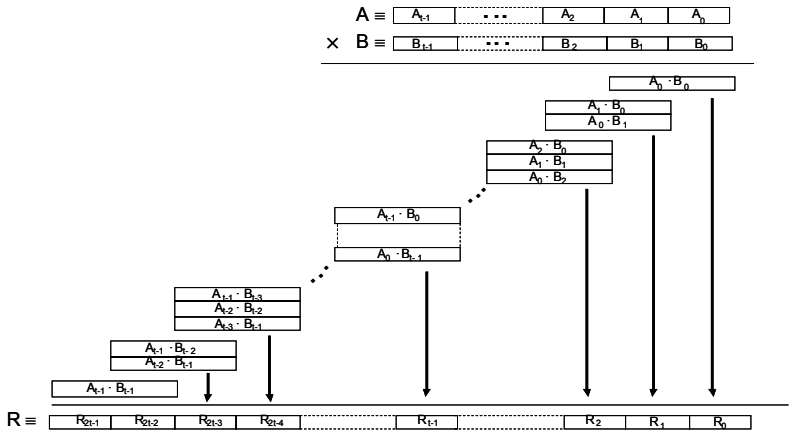

Figura 3-11: Multiplicación por columnas

Cada uno de estos productos parciales se suma al resultado parcial hasta el momento de la operación completa teniendo en cuenta el desplazamiento a la izquierda relativo a su posición. Tras cada operación, un conjunto de cifras en la zona menos significativa del resultado final son correctas. La figura 3-12 muestra la formación progresiva del resultado final en cada producto parcial.

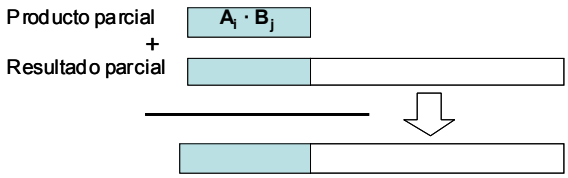

Figura 3-12: Formación del resultado final con la suma de productos parciales

La implementación de este proceso de formación del resultado debe considerar algunos aspectos de diseño con implicaciones en el rendimiento del circuito resultante en relación con el esquema iterativo y al grado de paralelismo de su implementación.

11,00100100001111110110101000100010000101101000110000100011010011000100111000110011000101000101110000000110111000001110011010001001010010000001001001110000010001
0001010011001111110011000111010100

La longitud de los operandos iniciales y la longitud de los factores de las operaciones parciales son determinantes para establecer la cantidad de iteraciones necesarias. La siguiente figura muestra dos ejemplos de operandos del mismo tamaño fragmentados en distintas partes donde se observa las diferencias en la cantidad de operaciones necesarias.

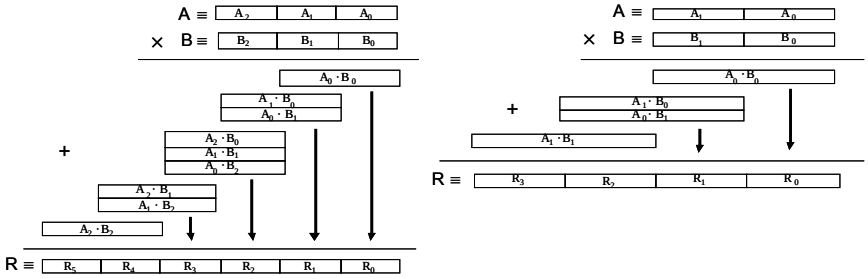

Figura 3-13: Esquema de cálculo de la multiplicación con operandos fragmentados en 2 y 3 partes

Con respecto a su implementación, se favorece la construcción de cauces segmentados entre las operaciones. Se presentan múltiples posibilidades de diseño, por ejemplo, la siguiente figura muestra estructuras segmentadas entre las operaciones de suma y producto que intervienen en la formación del resultado.

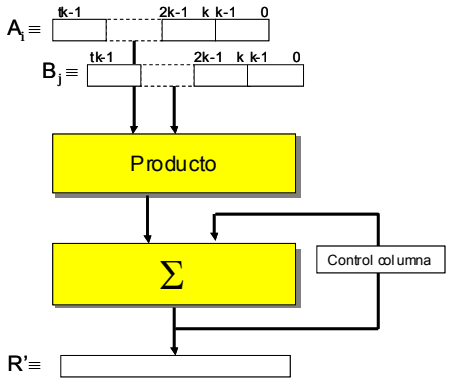

Figura 3-14: Cadena segmentada entra las operaciones de producto y suma

11,00100100001111110110101000100010000101101000110000100011010011000100110001100110100101000100001101110000011100110100010010100100000010010011100000100010100110011111100110001110100

Con respecto a la complejidad temporal de la operación, las decisiones de diseño se reflejan en las constantes multiplicativas de la expresión del coste pero no reducen el orden de complejidad de la operación. La expresión asintótica del coste mantiene una relación cuadrática con la cantidad de cifras de los operandos.

$$T_{\text{Multiplicación-}\mathbb{Z}} \in O(n_A \cdot n_B) \qquad [3.4]$$

Siendo n_A y n_B la cantidad de cifras de cada uno de los números enteros.

Multiplicación de operandos de longitud fija

Se conocen multitud de algoritmos para realizar la multiplicación de números de longitud fija [Parhami, 2000], [Oberman, 1996], [Bewick, 1994], [Omondi, 1994]. El procedimiento que se utiliza en esta memoria pone en práctica las ideas relativas a la metodología de operación propuesta y consta de las mismas tres etapas que la operación completa. Como se observa en la figura siguiente, en cada multiplicación se multiplican dos conjuntos de t bloques de k cifras.

11,0010010000111111011010100010001000010110100011000010001101001100010011001100110001010001011100000001101110000011100110100010010010000001001001110000010001
0001010011001111110011000111010 0

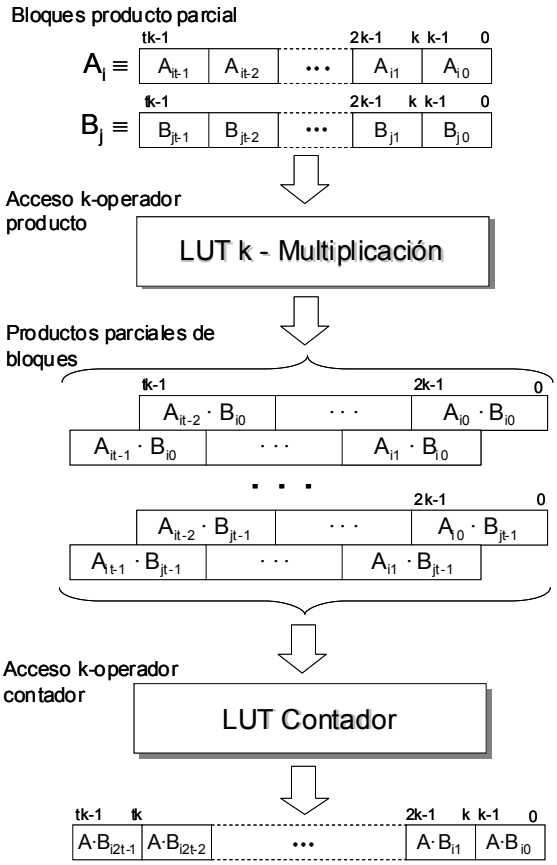

Figura 3-15: Cálculo de los productos parciales

La generación de productos parciales se realiza tomando el resultado de la operación directamente de una LUT para bloques de cifras de tamaño k, la combinación de los productos parciales reduce su número hasta un total de dos mediante el empleo de etapas de *operadores contadores* [García et al, 2003a], [Mora, 2001], [Zimmermann, 1997], [Dadda, 1965], [Wallace, 1964] y la suma final se puede realizar con cualquier método de suma de números enteros.

Al igual que en la operación de suma, la complejidad espacial de la operación depende del tamaño de los operadores basados en memorias con resultados almacenados.

11,00100100001111110110101001000100001011010001100001000110100110001001100011001100010100010111000000011011100000111001101000100101001000000100100111000001000100010010011001111100110001110100

La memoria que implementa el operador producto contiene 2k líneas de dirección para celdas de 2k bits de longitud. Además, el tamaño total necesario tan sólo debe considerar la mitad de la tabla y la diagonal principal, por tanto, la expresión resultante de su tamaño corresponde con la siguiente:

$$M_{k\text{-multiplicación}} = (2^k + 1)\cdot 2^k \cdot k \text{ bits} \qquad [3.5]$$

En la tabla siguiente se muestra el tamaño del k-operador producto para diferentes valores de k.

k	$M_{k\text{ - multiplicación}}$
2	40 bits
4	136 B
6	3,04 KB
8	64,25 KB
12	≈ 24,5MB

Tabla 3-3: Tamaño k-operador producto basado en lógica almacenada

Para el operador contador se dispone de una memoria que contiene la cuenta del número de unos que hay entre los bits de la entrada. Para una cantidad de k líneas de entrada el número de unos se puede representar con *s* bits, siendo,

$$s = \lg_2(k+1) \qquad [3.6]$$

El tamaño total del k-operador contador es:

$$M_{k\text{-contador}} = 2^k \cdot \lg_2(k+1) \text{ bits} \qquad [3.7]$$

La tabla siguiente refleja el tamaño del operador contador para varios valores de k.

k	$M_{k\text{ - contador}}$
3	2 B

7	48 B
15	16 KB
31	>1 GB

Tabla 3-4: Tamaño k-operador contador basado en lógica almacenada

Conclusiones

En este capítulo se ha desarrollado la metodología de cálculo de las operaciones y presentado los algoritmos para la suma y el producto de operandos de tipo entero. Entre los aspectos destacables se mencionan los siguientes:

- El aumento de la granularidad de los operadores abre nuevos caminos en la concepción de procedimientos de operación y el diseño de procesadores.
- La construcción de los operadores mediante lógica almacenada proporciona ventajas inherentes a la propia estructura de la memoria relativas a flexibilidad, robustez, paralelismo y reutilización.
- La algoritmia basada en esquemas iterativos permite abordar el procesamiento de operandos de una longitud variable así como aplicar técnicas que mejoren el rendimiento y la reutilización del hardware.

Una vez presentados estos principios se está en condiciones de desarrollar algoritmia para el procesamiento de números racionales expresados en el formato propuesto. En los siguientes capítulos se proponen los métodos de cálculo de la suma y el producto racional exacto.