Diseño y Optimización de un Multiplicador Modular usando Hardware Reconfigurable

Freddy Bolaños Martínez y Alvaro Bernal Noreña

UNIVERSIDAD DEL VALLE.
Grupo de Arquitecturas Digitales y Microelectrónica
{fremar; alvaro} @univalle.edu.co

Recibido para revisión May-2006, aceptado Jun-2006, versión final recibida Jun-2006

Resumen: Este artículo sugiere diversas alternativas para la implementación en hardware del operador Multiplicación Modular. Se propone una función de costo para evaluar las alternativas de diseño y se les compara con el objeto de optimizar su comportamiento respecto a parámetros como el área ocupada y el tiempo de ejecución. Éste último parámetro es de vital importancia en entornos criptográficos en donde el operador Multiplicación Modular se usa ampliamente.

Palabras Clave: Aritmética Modular, FPGAs, Lenguajes de Descripción de Hardware.

Abstract: Several choices for the hardware implementation of the modular multiplication operator are presented. It proposes a cost function to evaluate these design choices and compares them for the optimization of parameters such as area and execution time. This last parameter is very important in cryptographic applications, where the modular multiplication operator is widely used.

Keywords: Modular Arithmetic, Field Programmable Gate Arrays, Hardware Description languages.

1 INTRODUCCIÓN

1.1 Criptografía y Aritmética Modular

La criptografía asimétrica se fundamenta en que cada usuario del sistema posee un par de claves distintas [Menezes. Oorschot y Vanstone (1996)]. Una de las claves es llamada pública y la otra clave privada. En cualquier tipo de aplicación y para que el sistema funcione correctamente, debe cumplirse que luego de someter cierto mensaje sucesivamente los procesos de cifrado y descifrado usando las claves pública y privada de un usuario particular, se debe obtener el mensaje original.

Para cumplir con el requerimiento anterior, los esquemas criptográficos modernos se valen de operaciones aritméticas de complejidad apreciable. Tradicionalmente estas operaciones se han implementado en software. Sin embargo, dado el crecimiento continuo de las capacidades de los sistemas de cómputo en los últimos años, ha sido necesario reajustar gradualmente la complejidad de las operaciones para cumplir con los requerimientos de seguridad específicos de cada aplicación.

Normalmente estos ajustes hechos en las operaciones criptográficas implican el manejo de operandos de mayor longitud, lo que obviamente desmejora los tiempos de ejecución. Como consecuencia de esto, existe el interés de implementar estos operadores en hardware, de modo que los ajustes a los operadores se puedan hacer cumpliendo simultáneamente con tiempos de ejecución razonables.

La Multiplicación Modular es un operador usado en técnicas criptográficas como RSA [R.L., Shamir y Adleman (1978)], Rabin [Rabin (1979)], MCEliece [Sendrier (1996)], ElGamal [Elgamal (1985)] e incluso en criptografía de Curvas Elípticas [Koblitz (1987)]. El operador queda definido por (1), donde se muestra que la Multiplicación Modular tiene tres operandos de entrada: A, B y M. Cada uno de estos operandos corresponde a un entero sin signo representado como un número binario de una longitud dada.

Para tres números enteros A, B y M, se define la multiplicación modular como: $P \equiv A \cdot B \bmod M$, tal que: $A \cdot B = M \cdot k + P$, siendo k cualquier número entero. (1)

La primera aproximación algorítmica al problema de la Multiplicación Modular consiste en calcular el producto binario natural entre los números A y B, para luego calcular el módulo (residuo) del resultado parcial respecto al número M. A pesar de su sencillez, este algoritmo supone la disponibilidad de recursos de almace-

namiento y tiempos de ejecución que pueden llegar a ser restrictivos en aplicaciones en donde se requiere operar con números de gran cantidad de bits [Bernal y Guyot (1998)].

Como consecuencia de lo anterior, se han propuesto varios algoritmos tendientes a la optimización del operador Multiplicación Modular, que tienen como objetivo la reducción del tiempo de ejecución del algoritmo y la minimización de los recursos de cómputo necesarios para su ejecución. Entre los algoritmos propuestos de más amplio uso se encuentra la multiplicación de Montgomery [Montgomery (1985)], que usa una representación especial de los operandos de entrada para evitar el cálculo repetitivo de operaciones módulo. El algoritmo es bosquejado en la Figura 1.

Como puede verse, la multiplicación de Montgomery está basada en operadores sencillos tales como la suma y el desplazamiento. Entre las ventajas asociadas al algoritmo están que se puede calcular un producto en un total de N iteraciones, donde N es el tamaño en bits de los operandos y que para almacenar el resultado intermedio solo se requiere un total de N + 1 bits. La principal desventaja es que su resultado no es exactamente el valor de P definido en (1), sino que aparece una constante (R-1 mod M). Esto exige el ajuste de los resultados del multiplicador, la complejidad que ofrece este ajuste del resultado es equivalente a la complejidad del mismo producto de Montgomery. Se ha demostrado empíricamente que el desempeño del operador de Montgomery es superior a medida que el tamaño en bits de los operandos de entrada aumenta.

Entradas: $A = \{A_{n-1}, A_{n-2}, ..., A_1, A_0\}$, B, M
Salida: $S = A \times B \times R^{-1} \bmod M$

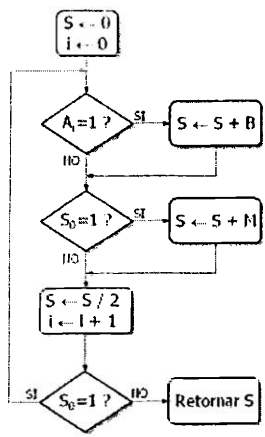


Figura 1: Algoritmo de Multiplicación de Montgomery

1.2 Lenguajes de Descripción de Hardware y los dispositivos lógicos reconfigurables

Una de las razones por las cuales la implementación de operaciones complejas en hardware no había sido muy atractiva hasta hace poco más de una década era el alto costo que implicaba la etapa de diseño y la construcción de los prototipos. Por esta razón, el uso de circuitos integrados de aplicación específica (ASIC por sus siglas en inglés) estaba limitado a un conjunto reducido de utilidades.

La aparición de dispositivos electrónicos que podían comportarse como un chip ASIC y que podían configurarse de acuerdo a las necesidades de cada aplicación particular, permitió la incursión del diseño hardware en un sinnúmero de aplicaciones hasta ahora exclusivas de entornos de desarrollo software. Cada fabricante tiene una nombre diferente para estos dispositivos: PLD's (programmable logic devices), CPLD's (PLD's complejos), FPGA's (field programmable gate arrays). Sin embargo, a pesar de las diferencias semánticas, todos estos elementos conservan una característica común: Son circuitos integrados con una cantidad de recursos lógicos disponibles que el usuario puede manipular e interconectar de manera autónoma.

Para clasificar este tipo de elementos, tradicionalmente se ha usado la cantidad de recursos lógicos disponibles en cada dispositivo particular. Este valor regularmente se da en términos de la cantidad de compuertas equivalentes de la lógica que el usuario tiene disponible para su uso. Hoy en día es común encontrar dispositivos de altas prestaciones con más de un millón de compuertas. Otra forma de medir la capacidad de un dispositivo particular es expresarla de acuerdo con la cantidad de bloques lógicos disponibles. A pesar de la diversidad de elementos que existen hoy en día en el mercado, casi todos los bloques lógicos de los dispositivos reconfigurables son similares entre sí. La arquitectura de un bloque lógico es un compromiso entre granularidad y funcionalidad. Es decir, un bloque lógico no puede ser tan complejo como para no poder integrar una cantidad razonable de bloques en un dispositivo, pero también debe proveer una cierta funcionalidad al usuario, de modo que éste pueda construir arquitecturas más complejas a partir de él. La Figura 2 muestra la arquitectura del bloque lógico del dispositivo XC3S200 de Xilinx, que es el usado para la síntesis de las arquitecturas a lo largo de este documento.

Celdas o bloques lógicos como el mostrado en la Figura 2 se organizan en alguna forma de arreglo, en el interior de un dispositivo reconfigurable. Cada bloque es configurable por el usuario, así como lo es la forma en que se ese bloque se comunica con todos los demás dentro del arreglo. La matriz de conmutación o interconexión es precisamente un conjunto de recursos de co-

municación que le permite al usuario construir arquitecturas de mayor complejidad a partir de los bloques lógicos simples.

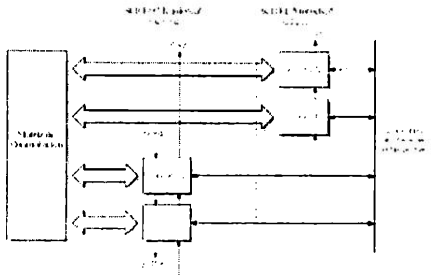


Figura 2: Arquitectura de una celda o bloque lógico del dispositivo XC3S200

El XC3S200 tiene un total de 1920 celdas lógicas organizadas en un arreglo de dos dimensiones, tal y como lo muestra la Figura 3. En esta, los bloques etiquetados como CLB representan celdas lógicas, mientras que los bloques IOB representan la interfaz a pines de entrada/salida. Nótese además que el dispositivo cuenta con una cantidad de elementos especiales como los bloques DCM (manipulador de señales de reloj), bloques de memoria RAM y multiplicadores embebidos.

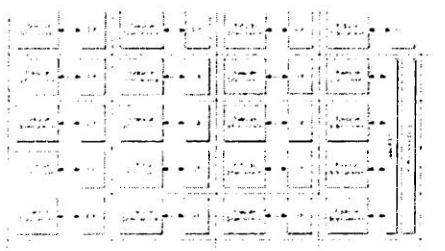


Figura 3: Matriz de celdas en un dispositivo XC3S200

Inicialmente, un diseñador debía manipular directamente los recursos disponibles en un dispositivo reconfigurable. A medida que la tecnología de integración permitió incluir más y más recursos en estos elementos, fue necesario agregar algún nivel de abstracción al proceso de diseño. Aparecen entonces lenguajes de descripción de hardware como el Verilog y el VHDL. Se trata básicamente de lenguajes de alto nivel con prestaciones como funciones de biblioteca, programación orientada a objetos y estructuras de decisión y control, que le permiten al diseñador dejar a un lado los detalles de la implementación y concentrarse en la funcionalidad de su diseño.

2 METODOLOGÍA

A pesar de su flexibilidad y de su bajo costo, el diseño de hardware usando dispositivos lógicos reconfigurables y lenguajes de descripción de hardware supone un problema importante. Dado que la filosofía de estas herramientas es permitirle al diseñador abstraerse de los detalles de la implementación, es difícil saber que tipo de descripciones pueden llegar a ser óptimas para una función de costo particular. Aplicaciones en donde se busca optimizar el área ocupada por un diseño, o donde el tiempo de ejecución es una condición restrictiva, o donde se busca ahorrar potencia eléctrica, suponen un conocimiento profundo de la arquitectura del dispositivo reconfigurable y de los algoritmos por medio de los cuales se sintetiza una descripción de alto nivel en hardware.

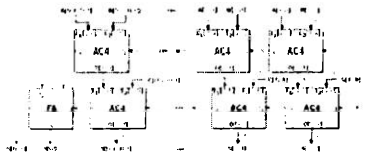
La metodología propuesta para el diseño del multiplicador modular, supone la descomposición del mismo en una serie de operadores más simples. Cada uno de estos operadores a su vez puede estar definido en términos de operaciones de menor complejidad. Una inspección a la Figura 1, que bosqueja uno de los algoritmos disponibles para el cálculo de la multiplicación modular mostrará que la descomposición propuesta no es difícil de conseguir. Para cada uno de los operadores mencionados, se proponen descripciones alternativas en lenguaje VHDL. Seguidamente, las descripciones son evaluadas en términos de una función de costo y se escogen aquellas con el mejor desempeño para la implementación de los operadores más complejos. De esta forma finalmente se obtiene una implementación óptima de la multiplicación modular.

La función de costo escogida para la optimización de los operadores es el producto entre el área ocupada, expresada como un porcentaje (el valor de 1 significaría que el diseño consume la totalidad de los recursos regulares del dispositivo), y el tiempo de ejecución asociado al operador particular.

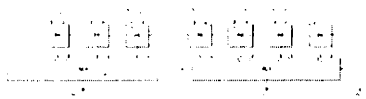
La función de costo debe involucrar el área ocupada, ya que como se dijo, en entornos criptográficos la tendencia es manejar operadores de gran tamaño, lo que condiciona la cantidad de recursos disponibles para la implementación de los operadores. De manera similar, se ha mencionado la inferencia que tiene el tiempo de ejecución en operaciones criptográficas y que la implementación en hardware de estas operaciones supone la optimización de este parámetro.

Como ejemplo, considérese el proceso de optimización del operador suma, presente en algoritmo de multiplicación de Montgomery. Para el algoritmo, que supone la suma condicionada de tres operandos, se propusieron tres descripciones alternativas. La primera está

basada en la suma con acarreo anticipado cada cuatro bits, mientras que la segunda hace uso de lógica de acarreo guardado (CSA, por sus siglas en inglés). Las arquitecturas de estas dos opciones se bosquejan en la Figuras 4a y 4b, respectivamente. La tercera opción fue aprovechar los operadores de alto nivel que ofrece el lenguaje VHDL. La figura 4c muestra la descripción en VHDL de un operador de suma para operandos de 32 bits. Obsérvese cómo la suma (línea 43 de la Figura 4c) se describe usando el operador '+' del lenguaje VHDL.



(a)



(b)

```
25  library IEEE;
41  use IEEE.STD_LOGIC_1164.ALL;
42  use IEEE.STD_LOGIC_ARITH.ALL;
43  use IEEE.STD_LOGIC_UNSIGNED.ALL;
44
45  entity sum_adder is
46      Port ( A : in  STD_LOGIC_VECTOR (31 downto 0);
47             B : in  STD_LOGIC_VECTOR (31 downto 0);
48             Cin : in  STD_LOGIC;
49             Sum : out STD_LOGIC_VECTOR (31 downto 0);
50             Cout : out STD_LOGIC);
51  end sum_adder;
52
53  architecture Behavioral of sum_adder is
54
55  signal temp: STD_LOGIC_VECTOR (32 downto 0);
56  signal Aimage,Bimage, Simage: STD_LOGIC_VECTOR(32 downto 0);
57  begin
58
59      Aimage <= '0' & A;
60      Bimage <= '0' & B;
61      Simage <= '0' & Sum;
62
63
64      temp <= Aimage + Bimage + Cin;
65      Simage <= temp;
66  end Behavioral;
```

(c)

Figura 4: Descripciones alternativas para el operador suma

La Figura 4 muestra una comparación de desempeño entre las tres alternativas para diferentes tamaños de los operandos de entrada (N). Las gráficas están etiquetadas como (a), (b) y (c) y corresponden a las funciones de costo de las descripciones bosquejadas en la Figura 4a, 4b y 4c, respectivamente. Los valores de la función de costo se presentan en un eje logarítmico para lograr una comparación más adecuada. Puede verse cómo a pesar de tener la descripción menos elaborada y pese a no suponer ninguna arquitectura hardware, el sumador de la Figura 4c es el que tiene los desempeños óptimos. Lo anterior se explica a partir del hecho de que una descripción arquitectural como aquellas de la Figuras 4a o 4b, obliga a la herramienta de síntesis a usar los recursos regulares del dispositivo reconfigurable. La descripción de alto nivel de la Figura 4c permite que la herramienta de síntesis haga uso de los recursos especiales de los que dispone el dispositivo para la implementación de sumadores (revísense en la Figura 2, las líneas etiquetadas como Cin y Cout), lo que repercute simultáneamente y de manera positiva en la cantidad de recursos regulares usados (área) y en la velocidad de respuesta.

3 RESULTADOS OBTENIDOS

Para la implementación del multiplicador modular igualmente se plantearon tres descripciones alternativas. En primer lugar está el multiplicador de Montgomery bosquejado ya en la Figura 1.

En segundo lugar se propone un algoritmo y una arquitectura basados en las apreciaciones hechas en Chiou (1993). Ya se ha dicho que la multiplicación modular es el equivalente de calcular un producto y luego un módulo. Esta forma de cálculo es intuitiva, además se conocen muchas arquitecturas hardware para los operadores producto y módulo [Flynn (2001)]. Sin embargo, el hecho de calcular estas operaciones separadamente genera altos requerimientos de recursos de almacenamiento intermedio y una cantidad importante de iteraciones. El algoritmo de Montgomery soluciona el problema integrando la reducción modular al producto, pero para hacer eso debe usar una representación no tan intuitiva de los operandos de entrada, además que se debe hacer una ajuste final al resultado, para llevarlo a su representación modular natural. El algoritmo propuesto se bosqueja en la Figura 6 y consigue hacer la reducción modular del producto en cada iteración, eliminando así la necesidad de una representación como la propuesta por Montgomery.

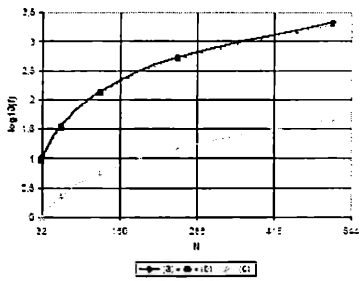


Figura 5: Función de costo (f) del operador suma

Finalmente, la tercera estrategia para la implementación de la multiplicación modular es el cálculo separado de la multiplicación y el módulo. Esta estrategia se adoptó debido a la experiencia con la síntesis de los sumadores, cuyos desempeños se presentaron en la Figura 5. Tal como puede verse de la arquitectura del dispositivo lógico reconfigurable, se cuenta con recursos especiales para calcular productos de números representados en binario. La comparación entre las diversas alternativas para la implementación de los sumadores sugiere que es conveniente usar estos recursos de procesamiento especiales, de modo que se quiere verificar si el hecho de usar multiplicadores embebidos (que son más rápidos y no consumen recursos regulares del dispositivo) puede contrarrestar las desventajas inherentes al algoritmo de multiplicación y módulo independientes.

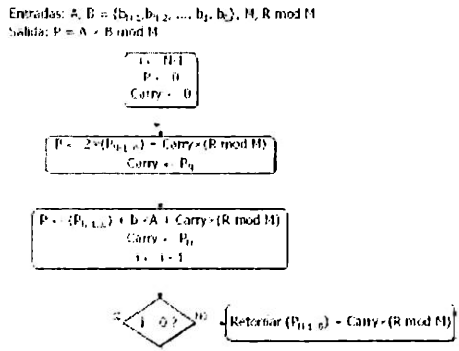


Figura 6: Algoritmo propuesto para el cálculo del producto modular

La Figura 7 muestra un multiplicador de 32 bits implementado usando cuatro de los multiplicadores embebidos con los que cuenta el dispositivo XC3S200. Debido a que tanto las multiplicaciones como las sumas son operaciones optimizadas y con recursos especiales dentro del dispositivo, se puede esperar que el multiplicador de la figura ofrezca desempeños superiores a cualquier multiplicador sintetizado sobre lógica regular.

Dado que el XC3S200 solo cuenta con ocho multiplicadores embebidos, fue necesaria la adopción de una estrategia secuencial (iterativa) para la implementación de multiplicadores modulares con operandos de entrada de más de 32 bits de longitud. Todos estos multiplicadores están basados en el operador de 32 bits mostrado en la Figura 7. La estrategia es el equivalente a trabajar en base 232 y su uso es muy común en aplicaciones criptográficas, en donde se conoce como aritmética de bases altas.

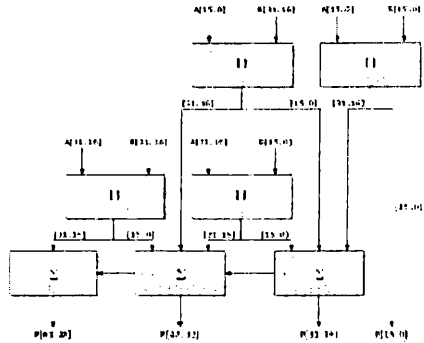


Figura 7: Multiplicador de 32 bits construido con 4 multiplicadores de 16 bits

La Figura 8 muestra un gráfico de las funciones de costo obtenidas para diferentes tamaños de los operandos de entrada (N). Las descripciones de los multiplicadores de Montgomery, del algoritmo propuesto y del que usa multiplicadores embebidos, están etiquetadas como MULT1, MULT2 y MULT3 en la figura, respectivamente.

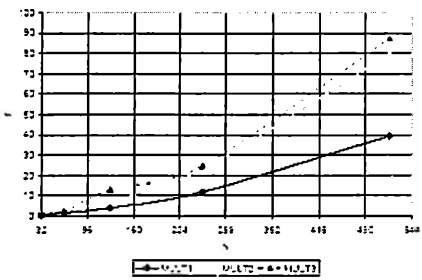


Figura 8: Comparación entre diferentes alternativas para el multiplicador modular

La Figura 8 señala cómo el multiplicador de Montgomery muestra un desempeño muy superior al de las otras dos estrategias cuando el valor de N aumenta. Sin embargo, debe tenerse en cuenta el costo asociado al ajuste de los resultados, debido al factor R-1 que aparece multiplicado en el resultado. Si se desease calcular una sola multiplicación modular los oeradores MULT2 y MULT3 entregarían el resultado correcto en una sola ejecución. El multiplicador de Montgomery por su parte requerirá dos ejecuciones sucesivas. Lo anterior implica que para una sola multiplicación los tres algoritmos tendrán desempeños similares.

Para una aplicación en dor de se requiere el cálculo

de muchas multiplicaciones modulares, se podría dejar el ajuste (el ajuste se entiende como el paso de la representación de Montgomery a la representación modular tradicional) para el final de los cálculos, es decir trabajar todo el tiempo con representaciones de Montgomery. Tal es la estrategia de la llamada Exponenciación de Montgomery [Koç y Acar (1997)]. Si la cantidad de productos crece lo suficiente, el multiplicador de Montgomery llega a duplicar el desempeño de las otras dos estrategias.

4 CONCLUSIONES

Se ha corroborado experimentalmente la utilidad de la estrategia de diseño propuesta. Dado que rara vez los detalles de la arquitectura del dispositivo y de los algoritmos de síntesis están disponibles, las labores de optimización deben hacerse a partir de la comparación de varias alternativas de diseño. La comparación propuesta no debiera significar un esfuerzo de diseño excesivo, debido a la enorme flexibilidad que ofrecen las herramientas como el lenguaje VHDL y los dispositivos lógicos reconfigurables.

De manera general, los dispositivos lógicos reconfigurables de última generación tienen incluidos recursos lógicos de naturaleza especial, dedicados a la optimización de ciertas operaciones. Aunque su uso es recomendable y puede mejorar el desempeño de una aplicación particular, la experiencia con los multiplicadores muestra que no siempre es así y que en el proceso de optimización deben considerarse otras alternativas.

El multiplicador de Montgomery parece ser la mejor estrategia para el cálculo de la multiplicación modular en hardware. Su desempeño se destaca particularmente en aplicaciones en donde los operandos de entrada son de gran tamaño y en donde la cantidad de multiplicaciones modulares que deben calcularse es importante. Por tal razón se concluye que el multiplicador de Montgomery es muy apropiado para implementaciones hardware de operaciones criptográficas.

REFERENCIAS

Bernal, A. y Guyot, A. (1998), Hardware for computing modular multiplication algorithm, *in* '13th Conference on Design of Circuits and Integrated Systems (DCIS'98), Spain'.

Chiou, C. (1993), 'A fast logic for modular multiplication', *Int. J. Electronics* **74**(6), 851–855.

Elgamal, T. (1985), 'A public key cryptosystem and a signature scheme based on discrete logarithms', *IEEE Transactions of Information Theory* **IT-31**(4).

Flynn, M. (2001), *Advanced Computer Arithmetic Design*, John Wiley & Sons Publishers.

Koç, K. y Acar, T. (1997), Fast software exponentiation in GF(2^k), *in* 'Proceedings, 13th Symposium on Computer Arithmetic', IEEE Computer Society Press.

Koblitz, N. (1987), 'Elliptic curve cryptosystem', *Mathematics of Computation* **48**, 203–209.

Menezes, A., Oorschot, P. y Vanstone, S. (1996), *Handbook of Applied Cryptography*, CRC Press.

Montgomery, P. (1985), 'Modular multiplication without trial division', *Math. of Computation* **44**(70), 519–521.

Rabin, M. (1979), Digitalized signatures and public key functions as intractable as factorization, Technical report, Massachusetts Institute of Technology. Laboratory for Computer Science.

R.L., R., Shamir, A. y Adleman, L. (1978), A method for obtaining digital signatures and public-key cryptosystems, Technical report, Communications of the ACM.

Sendrier, N. (1996), Mceliece public key cryptosystems. project codes, Technical report, Inria Rocquencourt.