IMPLEMENTACION DE UN MULTIPLICADOR PARALELO A NIVEL DE DIGITO SOBRE $GF(2^{163})$ USANDO BASES NORMALES GAUSSIANAS

Paulo Realpe Muñoz, Vladimir Trujillo Olaya, Jaime Velasco Medina
Grupo de Bionanoelectrónica, Escuela EIEE Universidad del Valle
A.A. 25360, Cali Colombia
E-mail: prealpe@unicauca.edu.co, vlatruo@univalle.edu.co, jvelasco@univalle.edu.co,

RESUMEN

En este artículo se presenta la implementación de un multiplicador con salida paralela a nivel de digito en el cuerpo finito binario $GF(2^{163})$ usando bases normales Gaussianas presentada por Reyhani-Masoleh en [1]. La arquitectura del hardware es implementada usando VHDL y sintetizada con la herramienta Quartus II de Altera® sobre la FPGA (Field Programmable Gate Array) EP2S60F1020C3. Además se muestran los resultados de la simulación para diferentes diseños fácilmente extendidos donde varía el tiempo de operación del multiplicador.

1. INTRODUCCION

La seguridad de los sistemas de clave pública actuales se basan en la complejidad del cálculo de un problema matemático como lo es la factorización de números grandes o el cálculo de logaritmos discretos para enteros grandes. Para implementaciones en hardware es generalmente mas conveniente usar campos finitos binarios de característica dos, $GF(2^m)$ que un campo de enteros modulo un número primo grande. Esto se debe en gran medida a que su aritmética esta libre de acarreos, lo cual implica reducción del área en la implementación y un mejor desempeño [2].

Diferentes algoritmos para la multiplicación en el cuerpo finito $GF(2^m)$ tanto a nivel de software como a nivel de hardware han sido presentados en la literatura [3], [4] y [5]. Adicionalmente se han utilizado diferentes bases para la representación de los elementos del cuerpo finito $GF(2^m)$ en los que opera tales multiplicadores [2].

Con el propósito de realizar la multiplicación en el cuerpo finito $GF(2^{163})$ usando bases normales Gaussianas, este articulo presenta la implementación en hardware de la arquitectura del multiplicador con salida paralela propuesto por [1]. Para el diseño de los bloques de la arquitectura del multiplicador se utilizo descripción VHDL, y para la síntesis se utilizó la herramienta Quartus II de Altera® sobre la FPGA EP2S60F1020C3.

La organización de este artículo es la siguiente: En la sección 2 se presenta una descripción de algoritmos de multiplicación para bases normales Gaussianas en el cuerpo finito $GF(2^m)$. En la sección 3 se describe el procedimiento para realizar la multiplicación con salida paralela a nivel de digito en el cuerpo finito $GF(2^m)$. En la sección 4 se presenta la implementación del multiplicador en $GF(2^{163})$. En la sección 5 se muestra los resultados de simulación. Finalmente, en la sección 6 se presentan las conclusiones.

2. ALGORITMOS DE MULTIPLICACION

En esta sección algunos algoritmos de multiplicación presentados en [1] son descritos.

2.1. Multiplicación usando una base normal arbitraria.

Una base normal de $GF(2^m)$ sobre $GF(2)$ es un conjunto $N = \{b, b^2, ..., b^{2^{m-1}}\}$ cuyos elementos son linealmente independientes. Sea $A = (a_0, a_1, ..., a_{m-1})$ y $B = (b_0, b_1, ..., b_{m-1})$, respectivamente, y $C \in GF(2^m)$, entonces

$$C = (c_0, c_1, ..., c_{m-1}) = AB = \sum_{i=0}^{m-1} \sum_{j=0}^{m-1} a_i b_j m_{i,j} \qquad (1)$$

Donde de C_i puede ser expresado como

$$c_l = \underline{a} M^{(l)} \underline{b}^{tr},\ 0 \le l \le m-1 \qquad (2)$$

donde $M^{(l)} = \left[m_{i,j}^{(l)}\right]_{i,j=0}^{m-1},\ m_{i,j}^{(l)} \in GF(2),\ 0 \le i, j \le m-1$

Algoritmo 1. (Algoritmo usando una BN arbitraria)
Entrada :
$A = (a_0, a_1, ..., a_{m-1}), B = (b_0, b_1, ..., b_{m-1}) \in GF(2^m)$
y M
Salida : $C = (c_0, c_1, ..., c_{m-1}) = AB$
1. Inicializar X := A y Y := B
2. Para k = 0 a m −1{
3. Computar : $c_k = \underline{x} M \underline{y}^{tr}$
4. X := X <<1 y Y := Y <<1}
5. Salida $C = (c_0, c_1, ..., c_{m-1})$

donde $X <<1 = (x_1, ..x_{m-1}, x_0)$.

2.2. Algoritmo de multiplicación Ning-Ying

En [6], [7], Ning y Yin propusieron algoritmos eficientes en software que precalculan las diferentes rotaciones a la izquierda de los vectores A y B y almacenarlos en una memoria.

Algoritmo 2. (Algoritmo Ning-Ying)
Entrada :
$A = (a_0, a_1, ..., a_{m-1}), B = (b_0, b_1, ..., b_{m-1}) \in GF(2^m)$
y M
Salida : $C = (c_0, c_1, ..., c_{m-1}) = AB$
1. Precomputar arreglo para A_i y B_i
2. Inicializa r C := 0
3. Para i = 0 a m − 1 {
4. Inicializa r S := 0
5. Para j = 0 a m − 1 {
6. Si M(i, j) = 1 entonces S := S + B_j }
7. C := C + A_i • S}
8. Salida $C = (c_0, c_1, ..., c_{m-1})$

2.3. Multiplicación convencional en GNB

Sea $C_0 = F(\underline{a}, \underline{b})$, el cual puede ser obtenido por *m* impar como sigue:

$$c_0 = F(\underline{a}, \underline{b}) = \sum_{k=1}^{p-2} a_{F(k+1)} b_{F(p-k)} \qquad (4)$$

En (4), la secuencia *F(1), F(2), …, F(p - 1)* necesita ser calculado usando (5):

$$F(2^i u^j \bmod p) = i, 0 \le i \le m-1, 0 \le j \le T, \qquad (5)$$

donde *u* es un entero de orden *T mod p*. Es claro que cada elemento de (4) corresponde a las entradas no nulas de la matriz de multiplicación **M**.

Algoritmo 3. (Multiplicación convencional)
Input:
$A = (a_0, a_1, ..., a_{m-1}), B = (b_0, b_1, ..., b_{m-1}) \in GF(2^m)$
and F(x, y)
Output: $C = (c_0, c_1, ..., c_{m-1}) = AB$
1. Initialize X := A and Y := B
2. For k = 0 to m − 1 {
3. Compute: c_k = F(x, y)
4. X := X <<1 and Y := Y <<1}
5. Ouput $C = (c_0, c_1, ..., c_{m-1})$

2.4. Algoritmo de Multiplicación Ning-Ying modificado

En un intento para modificar el algoritmo 2, la matriz **R** (m - 1) x T cuyos elementos *R(i, j);* $0 \le R(i, j) \le m - 1$, $1 \le i \le m - 1$, $1 \le j \le T$, contiene los índices de los 1s en la fila *i* de **M**. Si el número de 1s en la fila *i* de **M** es *T*, entonces todos los *R(i, j)*, $1 \le j \le T$, contienen un entero en [0, *m* - 1]. De otro modo, se inicializan los elementos restantes de **R**, cuyo número es par, con un valor constante, por ejemplo con 0.

Entonces se puede escribir c_0 como

$$c_0 = a_0 b_1 + \sum_{i=1}^{m-1} a_i \left(\sum_{j=1}^{T} b_{R(i,j)} \right) \qquad (6)$$

Usando (6), se puede obtener c_l por adición *l* modulo *m* en todos los índices en (6),

$$c_l = a_l b_{l+1 \bmod m} + \sum_{i=1}^{m-1} a_{l+i \bmod m} \left(\sum_{j=1}^{T} b_l + R(i, j) \bmod m \right),$$

$0 \le l \le m - 1$

Algoritmo 4. (Algoritmo Ning-Ying modificado)
Entrada :
$A = (a_0, a_1, ..., a_{m-1}), B = (b_0, b_1, ..., b_{m-1}) \in GF(2^m)$
y R
Salida : $C = (c_0, c_1, ..., c_{m-1}) = AB$
1. Precomputar arreglo para A_i y B_i
2. Inicializa r C : $A \bullet B_1$
3. Para i = 0 a m − 1 {
7. $C := C \oplus \left(A_i \bullet \left(B_{R(i,1)} \oplus B_{R(i,2)} \oplus ... \oplus B_{R(i,T)} \right) \right)$}
8. Salida $C = (c_0, c_1, ..., c_{m-1})$

3. MULTIPLICADOR A NIVEL DE DIGITO CON SALIDA PARALELA

En esta sección se presenta los principios teóricos acerca del multiplicador presentado por Reyhani-Masoleh en [1]. El número total de ciclos de reloj que se necesitan para la multiplicación es $q = \lceil m/d \rceil$, donde *d*, $1 \le d \le m$, es expresado como el número de bits de cada digito. Sea:

$$S'(k, B) = \left(B_{R(2k,1)} \oplus B_{R(2k,2)} \oplus ... \oplus B_{R(2k,h_{2k})} \right) >> k, \quad 1 \le k \le \frac{m-1}{2}, \qquad (7)$$

donde *R(i, j)*, $1 \le i \le m - 1$, es el (*i*, j)-esimo elemento de la matriz **R,** *m* – 1 x *T*. Sea C = AB con A, B y C ∈ GF(2^m). Entonces

$$c = (A \bullet B_1) \oplus \sum_{k=1}^{(m-1)/2} \left(\left((A_k \bullet S'(k, B)) << k \right) \oplus \left((A_{m-k} \bullet S'(k, B)) >> k \right) \right) \qquad (8)$$

donde S'(k, B) ∈ GF(2^m) definido en (7).
Sea *S'(m – k, B) = S'(k, B), 1 ≤ k ≤ (m-1)/2*. Entonces, (8) puede ser expresado como

$$(A_k \bullet S'(k, B)) << k = (A_k \bullet S'(k, B))^{2^{m-k}} \text{ y}$$

$$(A_{m-k} \bullet S'(k, B)) >> k = (A_{m-k} \bullet S'(k, B))^{2^k}$$

siendo A, B ∈ GF(2^m),

$$C = AB = \sum_{k=0}^{m-1} (A_{m-k} \bullet S'(k, B))^{2^k}, \qquad (9)$$

donde S'(0, B)=B_1.

Sea *S'(k, B)* ∈ *GF*(2^m) representado con respecto a GNB como $S'(k, B) = \sum_{l=0}^{m-1} s_l'(k, B) b^{2^l}$, donde $s'(k, B) \in \{0,1\}$. Luego un elemento de campo,

$$J(X, Y) = \sum_{k=0}^{m-1} x_{m-k} s_0'(k, Y) b^{2^k} \qquad (10)$$

donde *J(X, Y)* es una función de dos elementos de campo de *X = (x₀, x₁,…,x_{m-1})* y *Y = (y₀, y₁,…,y_{m-1})*, donde $s_0'(0, Y) = y_1$ y $s'(k, Y) = s_0'(m-k, Y)$ para *1 ≤ k ≤ (m-1)/2*. Entonces se puede escribir (9) como

$$C = \sum_{k=0}^{m-1}\left(\sum_{l=0}^{m-1} a_{m-k+l}\, s_l'(k,B)\, b^{2^l}\right)^{2^k} = \sum_{l=0}^{m-1}\left(\sum_{k=0}^{m-1} a_{m-k+l}\, s_l'(k,B)\, b^{2^k}\right)^{2^l} = \sum_{l=0}^{m-1} J^{2^l}(A^{2^{m-1}}, B^{2^{m-1}}). \quad (11)$$

donde $m=qd - r$, donde $0 \le r \le d-1$ Entonces,

$$Z(X,Y) = (\ldots((L^{2^d}(X,Y) \oplus L(X^{2^d}, Y^{2^d}))^{2^d} \oplus \ldots \oplus L(X^{2^{(q-2)d}}, Y^{2^{(q-2)d}}))^{2^d} \oplus L_0(X^{2^{(q-1)d}}, Y^{2^{(q-1)d}}), \quad (12)$$

Donde $L(X,Y) = \sum_{i=0}^{d-1} J^{d-1-i}(X^{2^i}, Y^{2^i})$ (13)

$$L_0(X,Y) = \sum_{i=0}^{d-r-1} J^{d-1-i}(X^{2^i}, Y^{2^i}) \quad (14)$$

Comparando (12) con (13) para valores de $X = A^2$ y $Y = B^2$, se puede verificar que $Z(A^2, B^2) = C^{2^r}$. Por lo tanto, $C = AB$ puede ser encontrado usando (12) si $X = A^{2^{1-r}}$ y $Y = B^{2^{1-r}}$, entonces, $Z(A^{2^{1-r}}, B^{2^{1-r}}) = C$.

4. IMPLEMENTACION DEL MULTIPLICADOR CON SALIDA PARALELA SOBRE $GF(2^{163})$

En esta sección se presenta la implementación en hardware del multiplicador propuesto por Reyhani-Masoleh en el cuerpo finito $GF(2^{163})$. La arquitectura presenta diversos bloques funcionales que permiten realizar diferentes funciones para llevar a cabo la multiplicación. En la figura 1 se muestra el diagrama de bloques del multiplicador presentado en [1].

Sea $Z_{(j)}$ como el contenido de la salida del registro Z en la j-ésima operación $0 = j = q - 1$ del ciclo de reloj. Al iniciar la operación el registro de salida Z debe ser borrado, $Z_{(j)} = 0$ y los registros de entrada X y Y deben ser cargados con las coordenadas iniciales $X_{(0)} = A^{2^{1-r}}$ y $Y_{(0)} = B^{2^{1-r}}$. Para realizar la multiplicación de una manera más eficiente se puede escribir la ecuación (12) de la siguiente forma recursiva: $Z_{(j+1)} = Z_{(j)}^{2^d} \oplus L(X_j, Y_j)$ (15)

Donde $X_{(j+1)} = X_{(j)}^{2^d}, Y_{(j+1)} = Y_{(j)}^{2^d}$ (16)

Después de q ciclos de reloj el registro de salida Z contiene las coordenadas de $C=AB$.

4.1. Bloques de Rotación

En la Figura 1 los bloques de rotación ***CS*** rotan a la derecha d posiciones el valor de la salida de los multiplexores 1 y 2. Los bloques $\boldsymbol{CS^x}$ rotan a la derecha x posiciones los datos de salida del bloque **J**.

4.2. Bloque Sumador

El término $L(X_{(j)}, Y_{(j)})$ en (15) es implementado por un bloque sumador en $GF(2^{163})$ con d entradas mediante compuertas XOR.

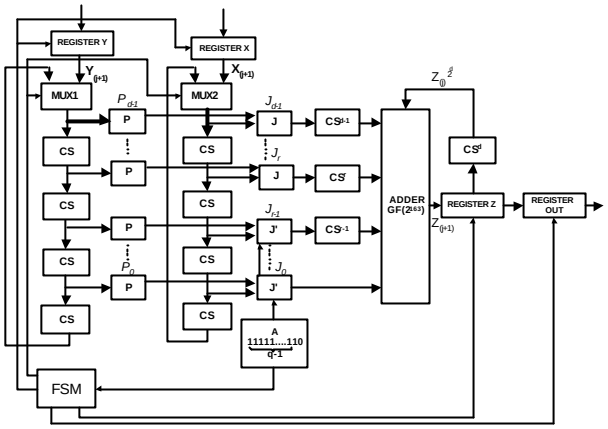

Figura 1. Diagrama en bloques del multiplicador paralelo a nivel de digito.

4.3. Bloque *P*

La entrada de este bloque corresponde a $Y_{(j)}^{2^i}$, $0 = i = d-1$ donde la salida $s_0'(k,Y)$ puede ser calculada usando la ecuación (7). En este caso, para determinar la matriz **R** se realizó un programa en Matlab según los parámetros de F(U,V) presentados en [2]. El hardware del bloque *J* es un circuito combinatorio basado en compuertas XOR.

4.4. Bloque *J*

El bloque *J(X,Y)* según (10) se puede escribir mediante la siguiente ecuación $J(X,Y) = X'(X) \bullet P(Y)$ donde

$$X'(X) = (x_0, x_{m-1}, x_{m-2}, \ldots, x_2, x_1) \quad (17)$$ y

$$P(Y) = (y_1, s_0'(1,Y), s_0'(2,Y), \ldots, s_0'(2,Y), s_0'(1,Y)) \quad (18)$$

Entonces la salida $J_{d-1-i}, 0 \le i \le d-1$ implementa $J(X_{(j)}^{2^i}, Y_{(j)}^{2^i})$ usando (17). El hardware del bloque *J* es un circuito combinatorio basado en compuertas AND de dos entradas.

4.5. Bloque *J'*

Durante el último ciclo de reloj $j = d-1$ todas las entradas r generadas desde los bloques $J_0,\ldots,J_{r-1}$ son 0 y las otras entradas corresponden a los términos que aparecen en (14). Estas entradas son controladas por el bloque ***A*** que genera una señal la cual es 0 únicamente en el último ciclo de reloj conectado a los bloques $J_0,\ldots,J_{r-1}$. El hardware del bloque *J* es un circuito combinatorio basado en compuertas AND de tres entradas.

4.6. Unidad de control

La unidad de control es implementada usando máquina de estados finito (FSM), la función es controlar la secuencia de entrada *X* y *Y* de datos del multiplicador y habilitar los registros *Z* y *Out*.

4.7. Registros I/O y Acumulador

Los operandos del multiplicador se almacenan en los registros *X* y *Y*. Al finalizar la operación la unidad de control habilita el registro *Out* para almacenar el resultado del multiplicador. El registro acumulador *Z* guarda los resultados que se calculan usando la ecuación (15).

Para este multiplicador con salida paralela de tipo $T = 4$ sobre $GF(2^{163})$ requiere $163d$ compuertas AND, $=404d$ compuertas XOR y aproximadamente 652 registros.

5. RESULTADOS DE SIMULACIÓN

Con el propósito de verificar el funcionamiento del multiplicador se realizaron diferentes simulaciones en el cuerpo finito $GF(2^{163})$. Los resultados de simulación se muestran en las tablas 1, y 2. En la Figura 2 se muestra el resultado de simulación para A^2.

$A*A=A^2=X$

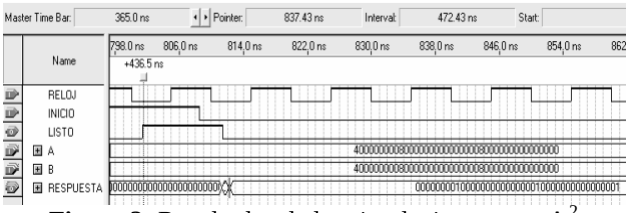

Figura 2. Resultados de las simulaciones para A^2

En la Tabla 1 se presentan los resultados para diferentes implementaciones hardware considerando varios valores para los parámetros *d* y *r* donde $r = qd - 163$. En este caso el periodo de reloj es de 10 ns.

	d	*r*	*q*	ALUTs	% ALUTs	Tiempo de operación en ns
1	2	1	82	903	1%	842.4
2	4	1	41	1888	2%	436.5
3	8	5	21	2083	4%	238.8
4	16	13	11	3460	7%	137.5
5	32	29	6	6390	13%	87.74

Tabla 1 Tiempo de operación y número de ALUTs.

En la Tabla 2 se muestra la comparación entre el tiempo de retardo generado presentado en [1] y los resultados de tiempo de retardo presentados en este trabajo para cada *d* y *r*.

El tiempo de retardo del multiplicador paralelo presentado en [1] esta dado por la siguiente relación

$$\leq T_A + \left(\lceil \log_2 T \rceil + \lceil \log_2 (d - r + 1) \rceil\right) T_x \qquad (19)$$

donde T_A y T_X corresponden al tiempo de retardo de la compuerta AND y la compuerta XOR, respectivamente. En este caso $T = 4$, $T_A \approx 1.25$ ns y $T_X \approx 1.25$ ns.

	d	*r*	Camino critico de retardo (Teórico) in ns	Camino critico de retardo (Practico) in ns
1	2	1	5.0	4.32
2	4	1	6.25	4.15
3	8	5	6.25	4.26
4	16	13	6.25	4.29
5	32	29	6.25	5.61

Tabla 2. Comparación de los retardos de tiempo.

6. CONCLUSIONES

En este articulo se realizó el diseño del multiplicador paralelo a nivel de digito presentada en [1] usando bases normales Gaussianas de tipo T=4 sobre $GF(2^{163})$. En este trabajo se presentan cinco diferentes implementaciones para la arquitectura del multiplicador paralelo. Cada implementación se ha realizado (variando el parámetro *d*) con el propósito de alcanzar un mayor desempeño a nivel de velocidad y área.

Como trabajo futuro se pretende implementar el inverso multiplicativo mediante el algoritmo de T.Ithoh, S. Tsujii presentado [8] sobre $GF(2^{163})$ usando este multiplicador paralelo para implementar un procesador basado en curvas elípticas sobre $GF(2^{163})$.

7. AGRADECIMIENTOS

Este trabajo ha sido patrocinado por Altera Corporation a través del programa universitario. Los autores dan especial agradecimientos a Mrs. Ralene Marcoccia de Altera Corporation.

8. REFERENCIAS

[1] A. Reyhani-Masoleh, "Efficient Algorithms and Architectures for Field Multiplication Using Gaussian Normal Basis". IEEE Transaccions on computers, Vol 55, No 1 January 2006.

[2] V. Trujillo, J. Velasco, J. López, "Multiplicador en el Cuerpo Finito $GF(2^{163})$ usando Bases Normales Gaussianas", Grupo de Bioelectrónica y Nanotecnologia. EIEE. Universidad del Valle. http://www.iberchip.org/IX/Articles/PAP074.pdf

[3] L. Song and K.K. Parhi, "Low-Energy Digit-Serial/Parallel Finite Field Multipliers," J. VLSI Signal Processing, vol. 19, pp. 149-166, 1998.

[4] L. Gao, S. Shirivastva, and G. Sovelman. "Elliptic Curve scala multipler desing using FPGAs," Workshop on cryptographic hardware and embedded systems, 1999. http://www.springerlink.com/index/FTYED84TNM1UFW8R.pdf

[5] G. Orlando and C.Paar, "A super-serial Galois field multiplier for FPGAs and its applications to public key algorithms", IEEE symposium on field programmable custom computing machine, 1999. http://ieeexplorer.ieee.org/iel5/6529/17422/00803685.pdf

[6] P. Ning and Y.L. Yin, "Efficient Software Implementation for Finite Field Multiplication in Normal Basis," Proc. Int'l Conf. Information and Comm. Security (ICICS 2001), pp. 177-181, Nov. 2000. http://springerlink.com/index/U7466/KEYP4U68jaT.pdf

[7] Y.L. Yin and P. Ning, "Efficient Finite Field Multiplication in Normal Basis," US Patent No. 6,389,442, May 2002. Assignee: RSA Security Inc. http://springerlink.com/index/U7466/KEYP4U68jaT.pdf

[8] T. Ithoh, S. Tsujii "A fast algorithm for computing multiplicative inverses in $GF(2^m)$ using bases normales" Information and Computation,1998. http://doi.ieeecomputersociety.org/10.1109/12.926155.html