RIELAC, Vol.XXXIV 3/2013 p.32-40 Septiembre - Diciembre ISSN: 1815-5928

RIELAC
Revista de Ingeniería Electrónica, Automática y Comunicaciones

Implementación eficiente de la multiplicación modular de Montgomery sobre hardware reconfigurable

Ander Torres López[1]; Yosbel Martínez García[2]; Raudel Cuiman Márquez[1]; Humberto Díaz Pando[3] y Alejandro J. Cabrera Sarmiento[1]

[1]Dpto. de Automática y Computación, Instituto Superior Politécnico "José Antonio Echeverría", La Habana, Cuba.

[2]Complejo de Investigaciones Tecnológicas Integradas, La Habana, Cuba.

[3]Centro de Estudios de Ingeniería y Sistemas, Instituto Superior Politécnico "José Antonio Echeverría", La Habana, Cuba.

RESUMEN / *ABSTRACT*

Este documento propone implementaciones tanto, software como hardware, sobre un FPGA de Xilinx, del algoritmo de multiplicación de Montgomery utilizando el método de Escaneo de Operandos por Separado (SOS, de sus siglas en inglés), que es una forma eficiente para calcular multiplicaciones modulares. La solución software ha sido desarrollada sobre la plataforma de procesamiento MicroBlaze y la de hardware se realizó utilizando directamente los recursos lógicos y dedicados de la FPGA. Los resultados se presentan con una comparación entre ambos casos en función de su rendimiento y el empleo de recursos.

Palabras claves: multiplicación modular de Montgomery, SOS, hardware reconfigurable, FPGA.

High-performance implementation of Montgomery's modular multiplication on reconfigurable hardware

This paper proposes implementations for both, software and hardware on Xilinx FPGAs of the Montgomery multiplication algorithm using the Separated Operand Scanning (SOS) method, which is an efficient way to compute modular multiplications. The software solution has been developed on the MicroBlaze processing platform and the hardware was carried out directly in logic and dedicated resources of the FPGA. The results are presented with a comparison between the two cases based on their performance and use of resources.

Key words: Montgomery's modular multiplication, SOS, reconfigurable hardware, FPGA.

INTRODUCCIÓN

Los algoritmos criptográficos asimétricos generalmente utilizan operaciones modulares de suma, multiplicación y exponenciación con operandos típicamente en el orden de las centenas o millares de bits. Tales son los casos de criptosistemas altamente difundidos como RSA, Rabin, ELGamal, e incluso algunos basados en la Criptografía de Curvas Elípticas (ECC, de sus siglas en inglés).[1] La exponenciación modular, expresada como $\boldsymbol{c = a^b \bmod n}$, donde $\boldsymbol{a}$ y $\boldsymbol{b}$ son la base y el exponente respectivamente y $\boldsymbol{n}$ el módulo, constituye una de las operaciones principales y de mayor complejidad; lo cual determina el rendimiento de los mecanismos de cifrado-descifrado de dichos criptosistemas. El cálculo de la exponenciación suele realizarse mediante el empleo de sucesivas operaciones de multiplicación modular. Seleccionando entonces algoritmos adecuados para la implementación de esta última, es posible mejorar significativamente la eficiencia total del sistema.

Uno de los métodos diseñado para tales fines es el propuesto por Peter L. Montgomery.[2-3] El método de Montgomery es especialmente útil en escenarios que requieran grandes volúmenes de operaciones de multiplicación modular como sucede en el caso de la exponenciación modular.[4] Montgomery propuso representar los operandos en el Sistema de Números Residuales (RNS, de sus siglas en inglés) lo que permite simplificar la reducción del producto en el módulo ***n*** a una simple división por una potencia de ***2***.[5] La división por potencias de ***2*** es una operación fácilmente implementada en entornos que utilicen los números expresados en forma binaria pues se traduce a desplazamientos de bits. Por esta razón la multiplicación modular de Montgomery constituye una alternativa atractiva en implementaciones basadas en microprocesadores de propósito general y particularmente en implementaciones basadas en hardware; siendo motivo, por sus buenos resultados, de numerosos estudios que proponen, en muchos casos, mejoras en busca de su optimización.

Si bien las soluciones software ofrecen un conjunto de ventajas deseables en la implementación de algoritmos criptográficos, los resultados de rendimiento en cuanto a tiempo de respuesta son discretos con respecto a implementaciones hardware.[6] La causa radica en que la arquitectura de los microprocesadores de propósito general no se encuentra optimizada para realizar operaciones y manejar datos con tamaños en el orden de los utilizados en los algoritmos criptográficos, además de que la ejecución de las instrucciones es eminentemente secuencial. Añádasele a lo anterior el hecho de que las implementaciones hardware proporcionan un mayor nivel de seguridad física en cuanto al almacenamiento de la claves ya que los ataques a dispositivos hardware son más complejos y costosos de realizar.[6] De ahí que una alternativa de gran aceptación para elevar el rendimiento y los niveles de seguridad es utilizar dispositivos externos a la computadora para almacenar las claves y ejecutar los algoritmos criptográficos.

Considerando además que la tendencia actual se dirige al empleo de dispositivos portátiles con posibilidad de ser acoplados a diferentes tipos de redes, mediante las que intercambian información en muchas ocasiones confidencial, existen nuevos retos abiertos a la investigación sobre la seguridad, orientados a la incorporación de aplicaciones criptográficas en dichos dispositivos para garantizar niveles de seguridad adecuados.[7] Una de las principales limitantes radica en la capacidad de procesamiento y cantidad de recursos hardware limitados de estos dispositivos, que conducen a la necesidad de encontrar métodos eficientes que permitan realizar implementaciones software ajustadas a los recursos disponibles, o bien diseñar arquitecturas que puedan ser implementadas mediante hardware y proporcionen una relación adecuada entre rendimiento y consumo de recursos.

Uno de los criterios a considerar en el diseño e implementación de algoritmos criptográficos sobre hardware es la selección de la tecnología que mejor se ajuste a los requisitos de la aplicación. En este sentido los dispositivos de hardware reconfigurable cuentan con la capacidad de proporcionar una flexibilidad similar a las implementaciones software además de las ventajas con respecto al rendimiento y a la seguridad que se derivan de las soluciones basadas en hardware. Otro aspecto importante a tener en cuenta radica en la selección de una arquitectura eficiente que proporcione una relación adecuada entre rendimiento y recursos de hardware utilizados.

En este artículo se propone una implementación software y una implementación hardware del algoritmo de multiplicación modular de Montgomery utilizando el método SOS, ambas realizadas sobre la FPGA XC3S700A de la familia Spartan-3A de Xilinx. En el caso de la implementación software fue empleado el procesador empotrado MicroBlaze. En las siguientes secciones del artículo se realiza una breve descripción del estado del arte, se exponen las características principales de la tecnología empleada, se abordan los detalles del método SOS y de las implementaciones realizadas, se muestran los resultados obtenidos y finalmente se enumeran las conclusiones.

TRABAJOS RELACIONADOS

Varios autores aceptan, de manera general, que el método de Montgomery es altamente eficiente.[8-15] Montgomery[4] propone el siguiente algoritmo para efectuar la multiplicación modular:

function *MonPro(a,b)*

*Step 1. t := a * b*

*Step 2. u:= (t+(t*n' mod r)*n)/r*

Step 3. ***if*** *u ≥ n* ***then return*** *u-n* ***else return*** *u*

Además de la eficiencia que logra por sí solo el algoritmo de Montgomery, es posible encontrar trabajos dirigidos a mejorarlo aún más. Así, existen estudios que demuestran la posibilidad de eliminar el paso de la sustracción (tercer paso).[11, 16-17] Rivas[18] propone el desarrollo de un multiplicador hardware en un PLD de Altera, para números de 64 y 128 bits. El núcleo está basado en el método de Montgomery para base ***2***, donde en cada iteración se lee un bit y en dependencia del resultado se realizan determinadas operaciones. Este método se basa en operaciones aritméticas muy simples (sumas y desplazamientos) con tamaños de variables

grandes lo que permite que puedan realizarse en un solo ciclo de reloj influyendo positivamente en la velocidad del algoritmo. Una de las desventajas de este método consiste en que son necesarias demasiadas iteraciones por cada multiplicación; además de que al aumentar el tamaño de las variables aumenta el consumo de recursos y el retardo entre las interconexiones del dispositivo programable, por lo que disminuye la frecuencia máxima de trabajo. Esta característica hace difícil desarrollar un multiplicador modular que cumpla con los tamaños de clave de los estándares actuales de seguridad.

Koc[12, 19] propone varios métodos para implementar la multiplicación de Montgomery dando solución a tales problemas. A diferencia del método para base ***2***, en el cual se escanean las variables bit a bit, en estos se trabaja con los operandos fragmentados en palabras de mayor tamaño, lo cual reduce notablemente el número de iteraciones de la multiplicación modular a expensas de aumentar la complejidad de las operaciones básicas que conforman el algoritmo. El ancho de estas palabras generalmente está relacionado con los tamaños de palabras de la computadora o del entorno donde será ejecutada la multiplicación. Šimka[20] realiza una implementación software del método SOS sobre el procesador empotrado NIOS II de Altera, planteándose que es el algoritmo más apropiado para arquitecturas RISC. Además se hace referencia a la implementación de un coprocesador hardware que utiliza el método MWR2MM (de sus siglas en inglés, *Multiple Word Radix-2 Montgomery Multiplication*). Este método resulta altamente eficiente pero su implementación presenta la desventaja de que hay que resintetizar el diseño si se desea cambiar alguno de los parámetros. Además no quedan claros algunos elementos referentes a componentes hardware utilizados, la conexión del coprocesador al procesador, etc., lo cual implica que resulte difícil la asimilación de dicha implementación en el contexto de este trabajo y bajo otra plataforma. Por otra parte en los experimentos realizados se utiliza como exponente público el cuarto número de Fermat y para el descifrado el Teorema del Resto Chino (*CRT, de sus siglas en inglés*), cuestiones que disminuyen el tiempo de ejecución del algoritmo RSA.

A partir del análisis de las propuestas anteriores se decide realizar un diseño basado en el método SOS puesto que el procesador Microblaze, sobre el que se implementa la solución software, también presenta arquitectura RISC.

MÉTODO DE ESCANEO DE OPERANDOS POR SEPARADO (SOS)

En las implementaciones típicas se separan los números en palabras. Si ***w*** es el ancho de una palabra entonces los números pueden ser vistos como una secuencia de enteros cada uno en base ***W=2w***. Si estos números de múltiple precisión requieren ***s*** palabras para su representación, entonces se toma ***r=2s*w***.[19] En el método SOS primero se calcula el producto ***t=a*b*** según muestra el siguiente pseudocódigo:[19]

```
for i=0 to s-1
        C:=0
        for j=0 to s-1
                (C,S) := t[i+j] + a[j]* b[i] + C
                t[i+j] := S
        t[i+s]:=C
```

Para esto, inicialmente se debe inicializar ***t=0***. El resultado final será el valor ***t***, el cual debe ser almacenado en un número de ***2*s*** palabras: ***t[0]***, ***t[1]***, **…**, ***t[2*s-1]***. Luego se calcula el valor de ***u*** mediante la forma ***u= (t+m*n) /r*** donde ***m= t*n' mod r***. Para ello primero se debe asignar ***u=t*** y luego actualizar este valor adicionándole ***m*n***. Para la división por ***r*** basta con ignorar las ***s*** palabras menos significativas de ***u***. Como el proceso de reducción se realiza palabra por palabra, se puede sustituir el parámetro ***n'*** por un parámetro ***n'0***, el cual se calcula de la forma $n_0' = -n_0^{-1} \pmod W$.[21] La actualización de ***t=m*n*** puede realizarse según el siguiente código:[19]

```
for i=0 to s-1
        C:=0
        M:= t[i]* n'[0] mod W
        for j=0 to s-1
                (C,S) := t[i+j] + m*n[j] + C
                T[i+j] := S
        ADD ( t[i+s], C )
```

La función ADD es necesaria para la propagación del acarreo hasta la última palabra de ***t***, lo cual incrementa el tamaño de ***t*** a ***2*s*** palabras y un bit. Como el acarreo es salvado en una palabra, entonces el tamaño de ***t*** debe ser de ***2*s+1*** palabras. Para realizar la división, según se comentó anteriormente, se puede utilizar el siguiente código:[19]

```
for j=0 to s
        u[j] := t[j+s]
```

La resta de múltiple precisión es implementada para restar ***n*** en caso de que sea necesario. Para esto puede ejecutarse la siguiente subrutina:[19]

```
B:=0
for i=0 to s-1
        (B,D) := u[i] – n[i] – B
        t[i] := D
(B,D) := u[s] – B
t[s] := D
if B=0 then return t[0],t[1],….,t[s-1]
        else return u[0],u[1],…..,u[s-1]
```

Este método disminuye considerablemente la cantidad de ciclos de ejecución del algoritmo de Montgomery. Debe notarse que para su implementación son necesarias ***2*s+2*** palabras para almacenar resultados intermedios: ***2*s+1*** para almacenar ***t*** y una palabra para almacenar el valor de ***m***.

IMPLEMENTACIÓN DE SOS EN SOFTWARE

La implementación software de la solución se desarrolló sobre la plataforma de procesamiento Microblaze, módulo IP desarrollado por Xilinx. MicroBlaze es un microprocesador RISC de 32 bits con arquitectura Harvard, donde los buses de datos e instrucciones han sido a su vez divididos en buses locales y de expansión. La configuración de Microblaze permite ajustar determinados parámetros en pos de obtener los mejores resultados en dependencia de la naturaleza y la demanda de las aplicaciones. Existe la posibilidad de incluirle controladores de memoria CACHE, modificarle el número de etapas del pipeline, ser optimizado en área o velocidad e implementar mediante hardware las siguientes operaciones:[22]

- Multiplicadores de 32 o 64 bits.
- Divisores de 32 bits.
- Barrel Shifters, útiles para lograr desplazamientos de bits en un solo ciclo de reloj.
- Instrucciones de comparación de patrones.
- Unidad de aritmética flotante de simple precisión (FPU, de sus siglas en inglés).
- Unidad de manejo de memoria (MMU, de sus siglas en inglés).

Las primeras tres características son particularmente interesantes en la implementación de algoritmos criptográficos para acelerar las operaciones aritméticas básicas con el objetivo final de incrementar el rendimiento.

En este caso se ha configurado Microblaze con cinco etapas de pipeline, se le han habilitado además el barrel shifter y la operación de multiplicación capaz de generar productos de 64 bits. Las funciones para la aritmética de múltiple precisión, que son invocadas constantemente en el algoritmo, son descritas en ensamblador, lo cual contribuye notablemente al incremento de la velocidad de ejecución, puesto que se hace un uso más eficiente de los recursos del procesador. La implementación se desarrolla a una frecuencia de 90 MHz, la cual está condicionada por limitaciones de la propia FPGA Spartan-3A XC3S700A; utilizando un ancho de palabra de 32 bits debido a que este es el ancho máximo que permiten los registros de Microblaze y, por ende, la mejor manera de aprovechar su arquitectura.

IMPLEMENTACIÓN DE SOS EN HARDWARE

La operación de multiplicación modular por sí sola no cumple ningún objetivo, sino que generalmente se encuentra inmersa en sistemas criptográficos más complejos. Una aceleración de esta operación condiciona un incremento en la velocidad de ejecución. Por tal razón se decide implementar un coprocesador hardware que se conecte a Microblaze de manera tal que el procesador se encargaría de ejecutar otras tareas del criptosistema, aprovechando la funcionalidad del módulo para aumentar el rendimiento. Como se muestra en la figura 1, la comunicación se realiza a través de un bus FSL (Fast Simple Link), pues esta conexión punto a punto permite el flujo de una gran cantidad de datos a velocidades superiores a las del bus PLB (Processor Local Bus).

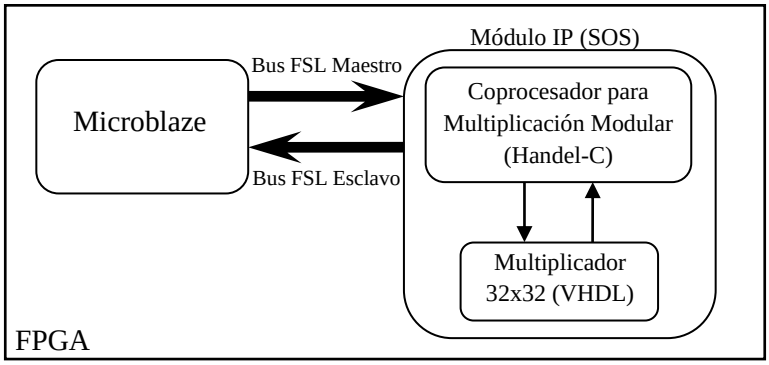

Figura 1

Esquema básico del sistema empotrado

El coprocesador diseñado tiene un ancho de palabra de 32 bits. Esta solución utiliza 4 multiplicadores dedicados de la FPGA. Para implementar el módulo IP con un ancho de palabra de 64 bits se requieren 16 multiplicadores, y dado que la configuración seleccionada para Microblaze consume 5, sería necesario contar con un total de 21. La Spartan-3A utilizada cuenta con sólo 20 multiplicadores por lo que no es posible implementar el diseño con este ancho de palabra. Por otra parte, no es recomendable utilizar una base que no sea múltiplo de 2^{32} pues se complejiza la lógica de recepción de datos por FSL.

Debido a la naturaleza secuencial del algoritmo, el módulo IP fue descrito en Handel-C, lenguaje de descripción de hardware muy parecido al ANSI C. Este lenguaje implementa una máquina de estados, transparente al programador, que brinda una buena abstracción y posibilita ejecutar tareas secuencialmente. Cada instrucción del lenguaje representa un estado y se ejecuta en un ciclo de reloj. Estas características disminuyen considerablemente el tiempo de diseño pues facilitan la programación del algoritmo y el trabajo con el almacenamiento temporal de variables en bloques de memoria.

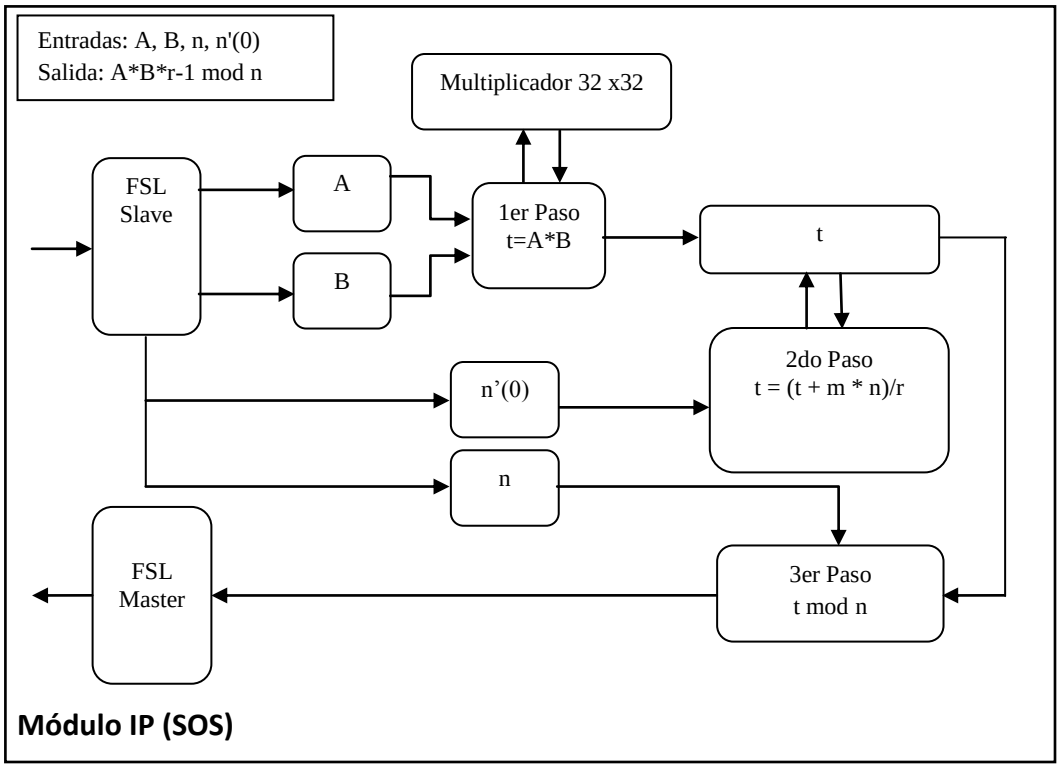

Figura 2

Esquema básico del Módulo IP que implementa el método SOS

La figura 2 muestra un esquema básico del componente desarrollado, donde se implementan los pasos del método SOS descritos anteriormente. Los factores A y B son obtenidos a través de la interfaz FSL esclava y almacenados cada uno en un bloque de memoria RAM independiente (elemento ***A*** y ***B***). Luego se multiplican utilizando el algoritmo clásico de multiplicación de números de múltiple precisión. Este algoritmo se comunica con el módulo de multiplicación, desarrollado en VHDL y valiéndose de la herramienta CORE Generator de Xilinx, a través de una interfaz personalizada que le permite obtener productos de 64 bits de ancho a partir de factores de 32 bits. El resultado del producto es almacenado en otra memoria (elemento ***t***). Posteriormente se ejecuta el algoritmo correspondiente al segundo paso del método SOS. Para ello se utilizan el producto almacenado en ***t***, el parámetro ***n`(0)*** mencionado en secciones anteriores y el módulo ***n***; estos dos últimos recibidos por FSL. Por cuestiones de aprovechamiento de los recursos, el módulo es almacenado en la memoria que antes ocupaba el factor ***A***. El resultado de la operación se almacena nuevamente en ***t***. Por último se resta de ***t*** el módulo ***n*** tantas veces como sea necesario hasta que el resultado quede en los límites que define dicho módulo. Este resultado se envía a través de la interfaz FSL maestra hacia el microprocesador.

RESULTADOS

En la tabla 1 se presentan los resultados obtenidos por la implementación del método SOS sobre Microblaze. Los slices[*], multiplicadores y bloques de RAM son referidos al costo en área del procesador. El consumo de recursos que se reporta es el que se obtiene por la utilización de Microblaze con las características que se discutieron en la sección anterior. Como puede apreciarse el diseño que se concibió explota la flexibilidad de la implementación en software, ya que es posible aumentar los tamaños de clave sin necesidad de incurrir en un gasto extra de recursos de hardware; lo cual representa una opción eficiente

[*] Término introducido por Xilinx. Es la unidad de procesamiento básico de la FPGA.

cuando se trata de ahorrar espacio en la FPGA. El primer apartado muestra los tiempos de ejecución para una multiplicación modular simple, mientras que el segundo expone los de una operación de cifrado RSA (exponenciación modular) para diferentes tamaños de clave.

Tabla 1
Resultados de la implementación software del método SOS sobre Microblaze.

Multiplicación modular					
Tamaño de clave (bits)	Ciclos de reloj	Tiempo (90MHz)	Slices	MULT18X18	BRAM
4096	1,550,685	17.22 ms	2197	4	16
2048	390,429	4.3 ms	2197	4	16
1024	99,069	1.1 ms	2197	4	16
Cifrado RSA con clave privada					
Tamaño de operandos (bits)	Ciclos de reloj	Tiempo (90MHz)	Slices	MULT18X18	BRAM
4096	8,070,933,511	89.7 s	2197	4	16
2048	1,044,779,184	11.6 s	2197	4	16
1024	151,093,982	1.67 s	2197	4	16

En la tabla 2 se muestran los resultados de velocidad y área consumida por el módulo IP hardware cuando se realiza una multiplicación modular simple. Se puede observar una disminución notable en cuanto al tiempo de ejecución, lo cual se traduce en una mejora de la velocidad de ejecución del algoritmo, con respecto a la solución software. Por otra parte, el consumo de recursos no experimenta cambios considerables cuando aumenta el tamaño de los datos, lo que permite utilizar dicho componente con operandos grandes para cumplir con los estándares de seguridad actuales.

Tabla 2
Resultados de la implementación hardware del método SOS con tamaño de palabra de 32 bits.

Multiplicación modular					
Tamaño de clave (bits)	Ciclos de reloj	Tiempo(90MHz)	Slices	MULT18X18	BRAM
4096	137,538	1528 us	828	4	3
2048	36,032	400 us	819	4	3
1024	9,856	101 us	810	4	3

En la tabla 3 se muestran los resultados de velocidad y área consumida por el módulo IP hardware de multiplicación modular y el procesador Microblaze, cuando se ejecuta una operación de cifrado RSA (exponenciación modular) para diferentes tamaños de clave. Se puede observar un aumento notable de la velocidad de ejecución en esta solución híbrida con respecto a la realizada en software (Tabla 1).

Tabla 3
Resultados de la implementación híbrida para el cifrado RSA utilizando el módulo hardware.

Cifrado RSA con clave privada					
Tamaño de operandos (bits)	Ciclos de reloj	Tiempo(90MHz)	Slices	MULT18X18	BRAM
4096	845,208,604	9.39 s	828+2197	4	3+16
2048	110,957,586	1.23 s	819+2197	4	3+16
1024	15,079,006	167.5 ms	810+2197	4	3+16

En aras de analizar la validez de la implementación realizada y de trazar estrategias de mejora en el futuro, se han utilizado los resultados obtenidos por Šimka[20] como referente de tiempo. Es importante aclarar que la intención no es hacer una comparación fiel, pues los métodos y tecnologías utilizadas por Šimka[20] difieren de los empleados en este trabajo.

La variante software del método SOS, implementada sobre Microblaze y trabajando con operandos de 2048 bits, invierte 11.6 s a una frecuencia de reloj de 90 MHz para una operación de cifrado RSA. Por su parte, la implementación de SOS propuesta por Šimka[20] y desarrollada sobre Nios a 50MHz consume 6.4 s para una operación de cifrado con RSA con datos de igual tamaño. En el caso de la implementación del coprocesador hardware para la multiplicación modular, Šimka[20] utiliza el método MWR2MM que, con un pipeline de 8 etapas, datos de 2048 bits y una frecuencia de 100 MHz, invierte 354 ms para una operación de descifrado RSA; mientras que el coprocesador desarrollado en este trabajo mediante método SOS consume, para igual tamaño de datos, 1.23 s a una frecuencia de 90 MHz. Algunas de estas diferencias se deben a que en el contexto en el cual se validan los algoritmos en el trabajo de Šimka[20] (Criptosistema RSA) se realizan algunas mejoras, que no constituyen optimizaciones de la multiplicación modular, como el uso de la clave pública $E=F_4$ (cuarto número de Fermat) y del Teorema del Resto Chino (CRT); cuestiones que influyen notablemente en las velocidades de cifrado y descifrado respectivamente.

Los resultados alcanzados con la implementación híbrida descrita ofrecen tiempos acordes con las expectativas de los autores. El hecho de que algunos parámetros del módulo puedan ser modificados sin que esto implique resintetizar el diseño, resulta una característica muy ventajosa, con la cual no cuenta la solución de Simka[20] y que dota al mismo de una gran flexibilidad. El modulo está concebido de manera tal que no es necesario almacenar los factores p y q, dado que no se utiliza el CRT para descifrar. Considerando que se cuenta con poca memoria, es de vital importancia que se almacenen los datos mínimos necesarios. La solución que se presenta en este trabajo está orientada a ser utilizada en un criptosistema real capaz de manejar varias llaves.

CONCLUSIONES Y TRABAJO FUTURO

En este artículo se realiza la implementación del algoritmo de multiplicación modular de Montgomery, variante SOS. La implementación llevada a cabo permite contar con un coprocesador que realiza esta operación con un costo en área relativamente bajo, 14.06% con respecto a la cantidad total de la FPGA utilizada. El diseño de este componente posibilita que el consumo de recursos no aumente significativamente, aunque se incremente el tamaño de los operandos. Los resultados obtenidos con la implementación del multiplicador modular en hardware permiten disminuir en 11 veces los ciclos de reloj con respecto a la implementación software. Por último, se logra la realización híbrida del algoritmo de cifrado RSA, la cual utiliza el 51.37% de los recursos de la FPGA, incluyendo el procesador principal y el coprocesador SOS. Esto brinda la posibilidad de utilizar dicha implementación en otras aplicaciones y sistemas embebidos donde se necesite un nivel elevado de seguridad.

A partir de los resultados obtenidos, el trabajo futuro estará dirigido a la aceleración de algoritmos de criptografía asimétrica incluidos la multiplicación modular de Montgomery, en conjunto con la aceleración de los algoritmos de generación de llaves criptográficas. Por otra parte se hará énfasis en buscar algoritmos y modelos para el particionado de algoritmos criptográficos con el objetivo de lograr implementaciones híbridas que logren un equilibrio adecuado entre las métricas de diseño (principalmente coste y rendimiento).

REFERENCIAS

1. A. J. Menezes, P. C. V. Oorschot, and S. A. Vanstone, *Handbook of applied cryptography*. New York: CRC Press, 1997.
2. E. Celebi, M. Gozutok, and L. Ertaul, "Implementations of Montgomery Multiplication Algorithms in Machine Languages," in *International Conference on Security & Management*, SAM 2008, pp. 491-497.
3. F. B. Martínez and A. B. Noreña, "Diseño y Optimización de un Multiplicador Modular usando Hardware Reconfigurable," *Avances en Sistemas e Informática*, vol. 3, No. 2, pp. 77-82, Diciembre 2006.
4. P. L. Montgomery, "Modular Multiplication Without Trial Division," *Mathematics of Computation*, vol. 4, No. 170, pp. 519-521, April 1985.
5. Y. Martínez, "Diseño de un esquema de firma digital RSA usando co-diseño hardware-software," Automática y Computación, ISPJAE, La Habana, 2011.
6. T. Wollinger and C. Paar, "How Secure Are FPGAs in Cryptographic Applications?," in *Field Programmable Logic and Application*. vol. 2778, P. Y. K. Cheung and G. Constantinides, Eds., ed: Springer Berlin / Heidelberg, 2003, pp. 91-100.
7. P. Kocher, R. Lee, G. McGraw, and A. Raghunathan, "Security as a new dimension in embedded system design," presented at the Proceedings of the 41st annual Design Automation Conference, San Diego, CA, USA, 2004.
8. D. Alan and M. William, "Efficient architectures for implementing montgomery modular multiplication and RSA modular exponentiation on reconfigurable logic," in *ACM/SIGDA tenth international symposium on Field-programmable gate arrays* Monterrey, California, USA, 2002.

9. T. Blum and C. Paar, "High-Radix Montgomery Modular Exponentiation on Reconfigurable Hardware," *IEEE Transactions on Computers*, vol. 50, No. pp. 759-764, 2001.
10. R. Garg and V. Renu, "An Efficient Montgomery Multiplication Algorithm and RSA Cryptographic Processor," in *Conference on Computational Intelligence and Multimedia Applications*, 2007, pp. 188-195.
11. G. Hachez and J.-J. Quisquater, "Montgomery Exponentiation with no Final Subtractions: Improved Results," 2000, pp. 293-301.
12. C. K. Koc, "High-Speed RSA Implementation," ed: RSA Laboratories, Redwood City, CA, 1994.
13. N. Nedjah and L. d. M. Mourelle, "Two Hardware Implementations for the Montgomery Modular Multiplication: Sequential versus Parallel," presented at the Proceedings of the 15th symposium on Integrated circuits and systems design, 2002.
14. N. Nedjah and L. d. M. Mourelle, "Three hardware architectures for the binary modular exponentiation: sequential, parallel, and systolic," *Circuits and Systems I: Regular Papers, IEEE Transactions on*, vol. 53, No. pp. 627-633, 2006.
15. A. F. Tenca and C. K. Koc, "A Scalable Architecture for Montgomery Multiplication," in *Cryptographic Hardware and Embedded Systems*, 1999, pp. 94-108.
16. C. D. Walter, "Montgomery exponentiation needs no final subtractions," *Electronics Letters*, vol. 35, No. pp. 1831-1832, 1999.
17. C. D. Walter, "Montgomery's Multiplication Technique: How to Make It Smaller and Faster " in *Cryptographic Hardware and Embedded Systems*, 1999, pp. 80–93.
18. R. D. P. Rivas, R. N. Londoño, and A. B. Noreña, "Implementación de la multiplicación modular de Montgomery en hardware reprogramable. ," *Ingeniería Electrónica, Automática y Comunicaciones*, vol. 25, No. 1, pp. 18-24, 2004.
19. C. K. Koc, T. Acar, and B. S. K. Jr, "Analyzing and Comparing Montgomery Multiplication Algorithms," No. 1996.
20. M. Šimka, "RSA implementation on reconfigurable hardware," in *Proceedings of the III. PhD student conference*, Technical University of Košice, Slovakia, April 2003, pp. 81–82.
21. T. Acar, "High-Speed Algorithms & Architectures For Number-Theoretic Cryptosystems," Oregon State University, 1997.
22. "MicroBlaze Processor Reference Guide v11.4," ed: Xilinx, Inc., 2010

AUTORES

Ander Torres López, Ingeniero en Automática, Profesor Instructor, Departamento de Automática y Computación del Instituto Superior Politécnico "José Antonio Echeverría", calle 114 Nº 11901, CUJAE, Marianao, La Habana, Cuba.

e-mail: ander@electrica.cujae.edu.cu

Yosbel Martínez García, Ingeniero en Automática, Especialista Principal en Automática, Complejo de Investigaciones Tecnológicas Integradas, calle 114 Nº 11901, CUJAE, Marianao, La Habana, Cuba. **e-mail**: ymartinez@udio.cujae.edu.cu

Raudel Cuiman Márquez, Ingeniero en Automática, Profesor Instructor, Departamento de Automática y Computación del Instituto Superior Politécnico "José Antonio Echeverría", calle 114 Nº 11901, CUJAE, Marianao, La Habana, Cuba. **e-mail**: raudel@electrica.cujae.edu.cu

Humberto Díaz Pando, Centro de Estudios de Ingeniería y Sistemas del Instituto Superior Politécnico "José Antonio Echeverría", calle 114 Nº 11901, CUJAE, Marianao, La Habana, Cuba.**e-mail**: hdiazp@ceis.cujae.edu.cu

Alejandro José Cabrera Sarmiento, Ingeniero Electricista, Doctor en Ciencias Técnicas, Profesor Titular, Departamento de Automática y Computación del Instituto Superior Politécnico "José Antonio Echeverría", calle 114 Nº 11901, CUJAE, Marianao, La Habana, Cuba. **e-mail:** alex@electrica.cujae.edu.cu