Implementación del algoritmo Threefish-256 en hardware reconfigurable

Threefish-256 algorithm implementation on reconfigurable hardware

Nathaly Nieto-Ramírez
Ing. Electrónica, Investigador Grupo de Arquitecturas Digitales y Microelectrónica
Universidad del Valle
Cali, Colombia
nathaly.nieto@correounivalle.edu.co

Rubén Darío Nieto-Londoño
Ph.D. Investigador Grupo Arquitecturas Digitales y Microelectrónica
Universidad del Valle
Cali, Colombia
ruben.nieto@correounivalle.edu.co

Resumen— En este artículo se presenta la descripción y los resultados de la implementación en hardware del algoritmo criptográfico Threefish en su proceso de cifrado. La implementación se realizó usando la arquitectura de ronda iterativa sobre la Field Programmable Gate Array (FPGA) Virtex-5 presente en el sistema de desarrollo XUPV5-LX110T. Los resultados posteriores al place and route muestran que el diseño Threefish-256 de ronda iterativa tiene un throughput de 551Mbps.

Palabras claves— Criptografía, diseño síncrono, FPGA, Threefish, VHDL.

Abstract— This article presents both the description and results of the Threefish cryptographic algorithm hardware implementation for encryption process. The implementation of the algorithm was performed by using the iterative round architecture on the FPGA (Field Programmable Gate Array) Virtex-5 present in the development system XUPV5-LX110T. Place and route results show that the design Threefish-256 iterative round has a throughput of 551Mbps.

Keywords— Cryptographic, FPGA, synchronous design, Threefish, VHDL.

1. INTRODUCCIÓN

Las funciones *hash* son primitivas criptográficas que toman un mensaje y producen una representación sintetizada o mensaje resumido [1], [2]. Estas funciones fueron creadas inicialmente para generar firmas digitales de manera eficiente y segura, sin embargo, actualmente son utilizadas en muchas aplicaciones que, por ejemplo, buscan proteger la información al inicio de sesión con contraseña o para asegurar la conexión a páginas web y para administración de claves de cifrado, entre otras.

En años recientes, la seguridad de las funciones *hash* más utilizadas, SHA-0, SHA-1, SHA-256 y SHA-512, han presentado deficiencias. Los ataques conocidos contra SHA-1 aún no son prácticos, pero se ha demostrado que se puede romper la seguridad en al menos 269 operaciones, mientras que un ataque de fuerza bruta requiere unas 280 operaciones [3]. Para enfrentar una situación indeseable, el National Institute of Standard an Statistics (NIST) de Estados Unidos convocó un concurso de diseño en el año 2008 para la siguiente generación de funciones *hash*, SHA-3 [4],[5] cuyos resultados fueron entregados en octubre del año 2012, presentando a *Keccak* como el ganador.

Skein fue seleccionado como uno de los cinco finalistas de la competencia del *NIST*, su diseño se basa en un cifrador de bloque conocido como *Threefish*, el cual está definido para tres diferentes tamaños de bloque, 256, 512 o 1024 bits, y permite modos de operación tanto secuencial como paralelo. Este trabajo se centra en el diseño y análisis del desempeño de la implementación iterativa de una sola ronda de *Threefish*. En la sección dos se realiza una breve descripción del mismo cifrador de bloque. Posterior a esto, se describe la implementación de *Threefish* en la FPGA *Virtex-5 XUPV5-LX110TFF136* de Xilinx [6], [7] para finalmente, presentar el análisis de los resultados en la sección cuatro.

Recibido: 18/03/2014/ Aceptado: 12/05/2014/ ITECKNE Vol. 11 Número 2 • ISSN 1692 - 1798 • ISSN Digital 2339-3483 • Diciembre 2014 • 149 - 156

2. CIFRADOR DE BLOQUE THREEFISH

2.1. Especificación del algoritmo

Threefish es un cifrador de bloque ajustable [8] que actúa como núcleo de la función *hash Skein*. La función *hash Skein* [9] opera solamente con palabras sin signo de 64 bits y está definido para ser implementado en tres diferentes tamaños de bloque: 256 bits, 512 bits o 1024 bits. El algoritmo toma como argumentos la clave de cifrado, *K*, el Tweak, *T*, y el texto plano, *P*. La clave, es del mismo tamaño que el texto plano, y el tamaño de la entrada adicional (conocida como *tweak*) es de 128 bits para todos los tamaños de bloque. La función de la entrada *tweak* es proporcionar variabilidad y permitir nuevos modos de funcionamiento [8]. El valor de *tweak* tiene algunas restricciones dependiendo del tipo de bloque que se esté procesando, ver Figura 9 y tabla 5 de [2]. Todas las entradas se convierten a palabras de 64 bits. El número de palabras, N_w, y el número de rondas, N_r, están definidas en función del tamaño de bloque tal como se relaciona en la Tabla I.

TABLA I

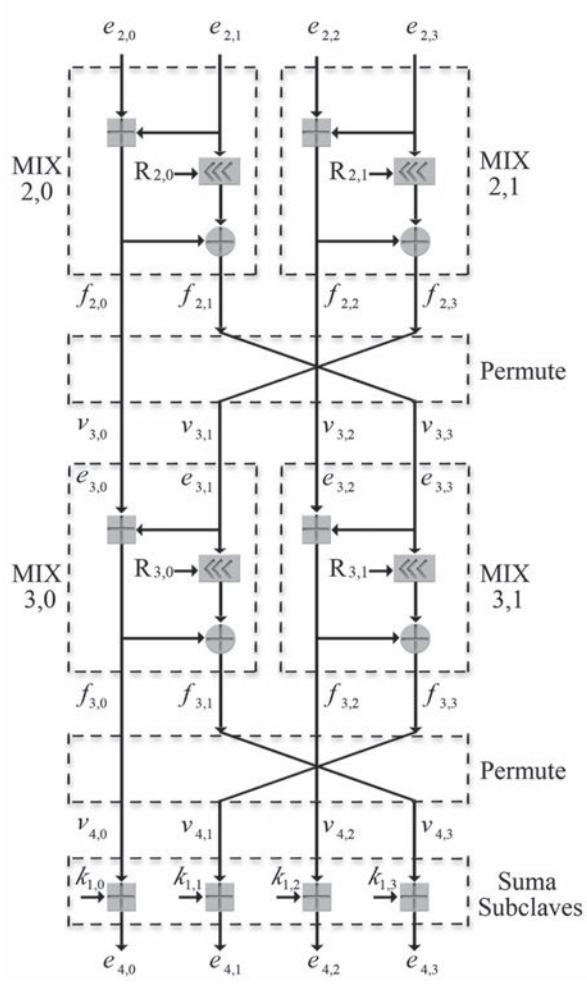
NÚMERO DE RONDAS EN FUNCIÓN DE LOS TAMAÑOS DE BLOQUES

Tamaño de Bloque/ Clave	256 bits	512 bits	1024 bits
No. de Palabras N_w	4	8	16
No. de Rondas N_r	72	72	80

Fuente: the skein hash function family [2].

El núcleo de *Threefish* consta de una función simple de mezcla no-lineal llamada *MIX*, que opera con dos palabras de 64 bits. Cada función *MIX* está compuesta por una suma ($\boxplus$), una operación *or* exclusiva ($\oplus$) y una rotación ($\ll$) que contiene como parámetro una constante ($R_{r,i}$) especificada en la Tabla II. La Fig. 1 ilustra la manera en que las funciones *MIX* permuten, y la suma de subclaves se utilizan para construir *Threefish*-256, el cual se ejecuta en 72 rondas como se especifica en la Tabla I. Cada ronda consta de dos funciones *MIX* seguidas de una permutación de cuatro palabras de 64 bits.

Fig. 1. RESUMEN DE DOS RONDAS DEL ALGORITMO THREEFISH-256



Fuente: adaptado de [10].

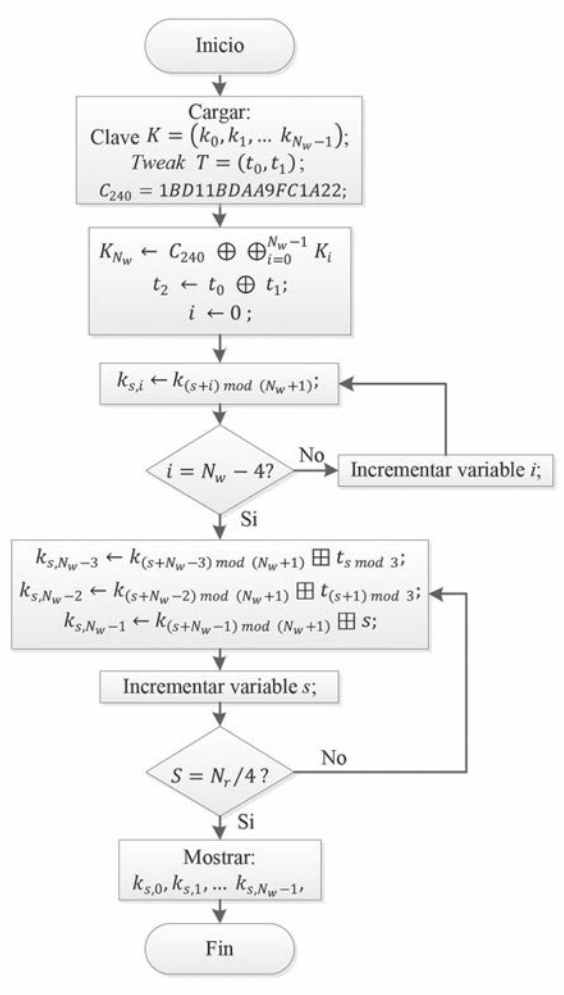
Threefish maneja una lista de subclaves que son adicionadas cada 4 rondas, como se especifica en [2]. Para crear esta lista se requieren dos palabras adicionales de 64 bits, las cuales garantizan que ninguna de las subclaves tenga todos sus bits en cero [2]. La palabra de clave adicional se obtiene después aplicar la operación *XOR* entre todas las palabras de clave, $K = \left(k_0, k_1, \cdots k_{N_w-1}\right)$ con la constante C_{240} y la palabra de *tweak* adicional se calcula mediante la operación *XOR* entre las dos palabras de *tweak,* $T=(t_0,t_1)$ (Fig. 2, proceso 2). Cada subclave está formada por una combinación de una de las palabras de clave extendida, $K_{ext} = \left(k_0, k_1, \cdots k_{N_w}\right)$, dos de las tres palabras de *tweak* extendidas, $T_{ext}=(t_0,t_1,t_2)$ y el número de subclave (Fig. 2, proceso 3, 4 y 5) según las relaciones definidas en [2].

TABLA II
CONSTANTES DE ROTACIÓN $R_{d,j}$ POR CADA N_w

N_w	4		8				16							
d \ j	0	1	0	1	2	3	0	1	2	3	4	5	6	7
0	14	16	46	36	19	37	24	13	8	47	8	17	22	37
1	52	57	33	27	14	42	38	19	10	55	49	18	23	52
2	23	40	17	49	36	39	33	4	51	13	34	41	59	17
3	5	37	44	9	54	56	5	20	48	41	47	28	16	25
4	25	33	39	30	34	24	41	9	37	31	12	47	44	30
5	46	12	13	50	10	17	16	34	56	51	4	53	42	41
6	58	22	25	29	39	43	31	44	47	46	19	42	44	25
7	32	32	8	35	56	22	9	48	35	52	23	31	37	20

Fuente: the skein hash function family [2].

Fig. 2. DIAGRAMA DE FLUJO KEYSCHEDULE



Fuente: autores del proyecto.

2.2. Proceso de cifrado

En la Figura 3 se muestra el diagrama de flujo del proceso de cifrado de un mensaje con el algoritmo *Threefish.* La primera fase consiste en aplicar la función *BytesToWords* a cada una de las entradas para convertir los bytes en palabras de 64 bits, después se realizan una serie de rondas, N_r, en las que se calcula la función de mezcla no lineal simple $MIX_{d;j}$ (Fig. 3, proceso 6) y la permutación de palabras (Tabla III y Fig. 3, proceso 7). Una subclave se agrega cada cuatro rondas (Fig. 3, proceso 5) y finalmente, se adiciona la última subclave (Fig. 3, proceso 8) para producir las palabras de 64 bits del texto cifrado para luego, mediante la aplicación de la función *WordsToBytes,* obtener los bytes del mismo texto.

3. DESCRIPCIÓN DEL MÓDULO HARDWARE

La arquitectura implementada en este trabajo del cifrador de bloque *Threefish* se basa en la arquitectura de ronda iterativa que se muestra en la Fig. 4. En este diseño existe una relación directa entre el circuito de ronda en hardware y las operaciones de ronda. La entrada se modifica mediante la función *BytesToWords* que a su vez alimenta el multiplexor que selecciona la entrada del bloque que implementa las operaciones de una ronda del *Threefish*. El bloque de ronda almacena el resultado en un registro. El registro realimenta el circuito a través del multiplexor, o una vez finalizadas

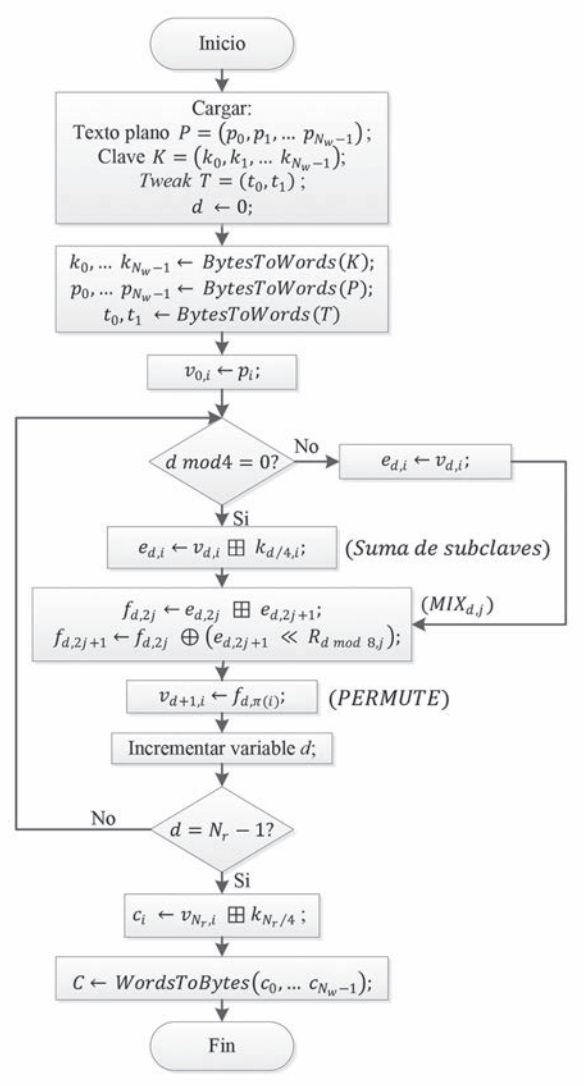
TABLA III

VALORES PARA LAS PERMUTACIONES DE PALABRA $\pi(i)$

N_w \ i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
4	0	3	2	1												
8	2	1	4	7	6	5	0	3								
16	0	9	2	13	6	11	4	15	10	7	12	3	14	5	8	1

Fuente: the skein hash function family [2].

Fig. 3. PROCESO DE CIFRADO DE THREEFISH

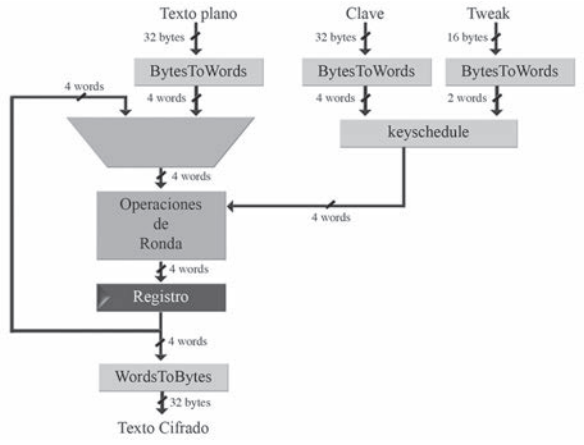


Fuente: autores del proyecto.

las 72 rondas, este resultado ingresa a la función *WordsToBytes* para así obtener el texto cifrado. Esta estructura utiliza un circuito de programación de subclaves que calcula las mismas durante la sistematización del texto cifrado para evitar claves precalculadas [11].

Con el fin de verificar que los modelos creados eran funcionalmente correctos y de acuerdo con la especificación del algoritmo *Threefish*, la unidad de control se configura para que los datos de entrada, texto plano, clave y *tweak*, sean los valores de los vectores de prueba proporcionados en la presentación de *Skein* ante el NIST en el archivo skein_golden_kat_internals.txt [2].

Fig. 4. ARQUITECTURA DE RONDA ITERATIVA



Fuente: adaptado de [11].

BytesToWords (Fig. 4) es la primera función que recibe los datos y efectúa la conversión de la cadena de 32 bytes de entrada en una cadena de cuatro palabras de 64 bits. Esta función, al igual que *WordsToBytes,* tiene a cargo la adecuación de los datos para ser manipulados a lo largo de proceso de cifrado.

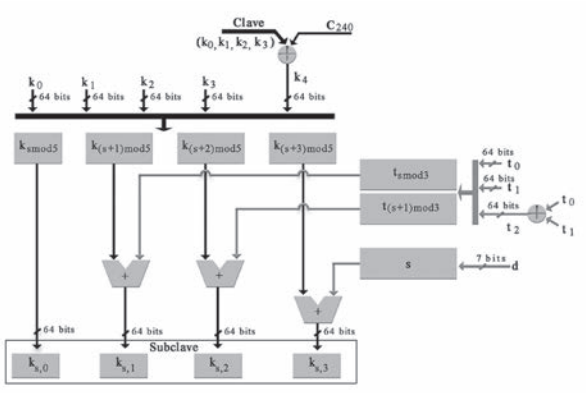
3.1. Bloque de generación de subclaves

Las entradas de este bloque son: el número de ronda, *d*, las palabras de 64 bits de la clave y el *tweak*. La salida del bloque la constituye la sub-

clave que se genera cada cuatro rondas y luego se suma con la permutación anterior.

En la Fig. 5 se muestra la arquitectura de la función *KeySchedule*. Inicialmente se realiza la operación lógica *XOR* con cada una de las palabras de la *clave* y con la variable inicializada C_{240}, formando la "clave extendida". De la misma manera, para formar el *tweak* extendido, se realiza la operación lógica *XOR* con cada una de las palabras del *tweak*. Seguido a esto, se seleccionan cuatro palabras de la *clave extendida* y se suman con dos palabras del *tweak extendido* y con la variable *s,* que depende del número de ronda (s=*d mod* 4). Las palabras de subclave se van almacenando en los registros nombrados $k_{s,0}$, $k_{s,1}$, $k_{s,2}$, $k_{s,3}$.

Fig. 5. DIAGRAMA DEL BLOQUE DE GENERACIÓN DE SUBCLAVES

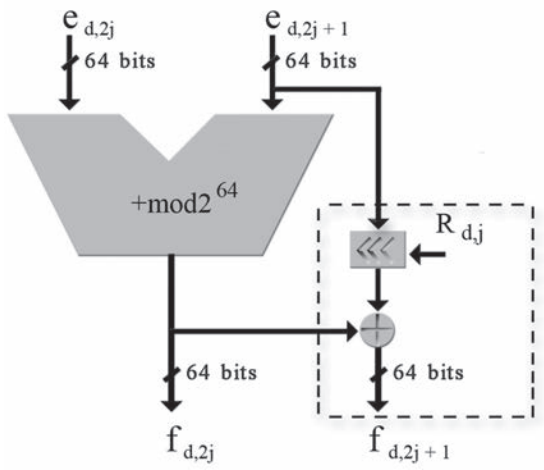


Fuente: adaptado a partir del *"Hardware Implementation of the Compression Function for Selected SHA-3 Candidates"* [12].

3.2. Bloque de cifrado

El diseño propuesto tiene un tamaño de bloque de 256 bits, el bloque de cifrado está conformado por cuatro sumadores de 64 bits, dos funciones de trasformación *MIX*, una permutación y un contador de ronda, *d*. El bloque *MIX* toma como entradas dos palabras extraídas de la función suma, $e_{d,2j}$ y $e_{d,2j+1}$, las cuales dependen del número entero, j, que varía entre 0 y 1. Este número entero, junto con el contador de ronda, se encargan de seleccionar el valor de la constante de rotación utilizada por la función *ROL* para operar la entrada $e_{d,2j+1}$. Las salidas $f_{d,2j}$ y $f_{d,2j+1}$ se detallan en el proceso de cifrado de la Fig. 3. La Fig. 6 muestra la arquitectura implementada de la función *MIX*.

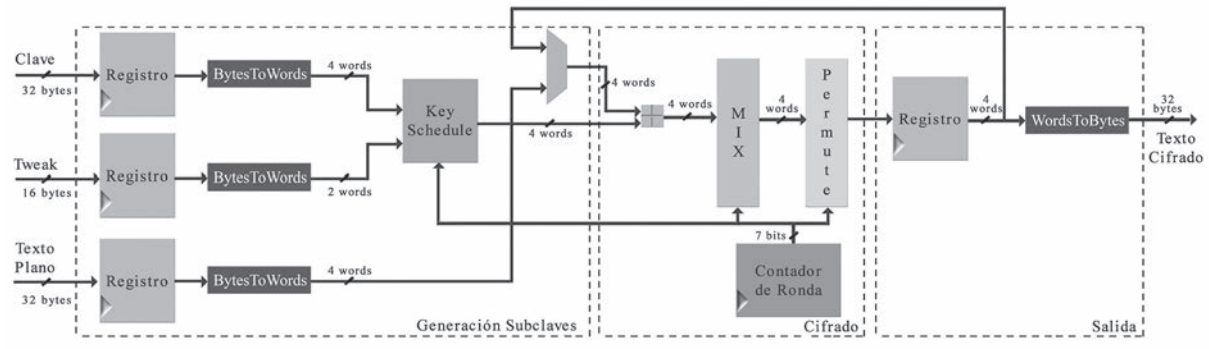
Fig. 6. ARQUITECTURA FUNCIÓN MIX



Fuente: adaptado de *A Skein-512 Hardware Implementation* [13].

La función de permutación es la misma para cada ronda y cambia la posición de las palabras en todo el vector. Es necesario adicionar un MUX que tenga como entradas el texto plano y la permutación de la ronda anterior, y la salida se conecta al sumador, como se muestra en la Fig. 7, esto se debe a que en la primera ronda no se realiza la operación *Permute* sino que se suma la primera subclave generada con el texto plano extendido.

Fig. 7. DIAGRAMA DE BLOQUES DE THREEFISH



Fuente: autores del proyecto.

3.3. Bloque de salida

Está compuesto por un registro encargado de guardar el texto cifrado y por la función *WordsToBytes* que se encarga de la visualización del resultado final. El tamaño del registro de salida puede variar dependiendo del tamaño del bloque.

En la Fig. 7 se muestran los tres bloques principales de *Threefish*: el bloque de generación de subclaves, el bloque de cifrado y el bloque de salida.

3.4. Unidad de control

La unidad de control diseñada se encarga de activar las señales que habilitan los registros de entrada y también las que habilitan las señales para el contador. Adicionalmente, se encarga de asignar los valores de los vectores de prueba propuestos a las entradas del sistema completo para garantizar así el correcto funcionamiento de todo el sistema.

4. RESULTADOS Y ANÁLISIS

4.1. Plataforma de pruebas

Las funciones descritas en la sección 3.2 fueron implementadas en la tarjeta de desarrollo *"Xilinx XUPV5-LX110T Development System",* la cual cuenta con un FPGA de familia *Virtex 5* (LX110T–FF1136) y varios módulos de hardware interconectados a la misma FPGA. La herramienta usada para la síntesis, el mapeo y la implementación en la FPGA fue XST de Xilinx ISE Design Suite. Para la verificación del correcto funcionamiento de la implementación se compararon los resultados con los vectores de prueba oficiales dados en [9] y, adicional a esto, se utilizó la herramienta ChipScopePro de Xilinx que permite verificar en tiempo real los resultados de las implementaciones en la FPGA, esto se logra insertando núcleos para depurar software en el diseño.

La Tabla IV muestra resultados de retardos de las funciones de transformación de *Threefish* obtenidos con la herramienta de diseño Xilinx ISE. *Keyschedule* es la función de transformación que presenta el mayor valor latencia, esto debido a la cantidad de lógica que utiliza para calcular cada subclave.

En la Tabla V se reportan los resultados de la implementación del algoritmo Threefish completamente autónomo, usando la arquitectura de ronda iterativa y se comparan con los resultados de otros autores que usan la FPGA Virtex-5. El producto de otras implementaciones que usan dispositivos diferentes no se incluyeron.

TABLA IV
RESULTADO DE LA SÍNTESIS FUNCIONES DE TRANSFORMACIÓN

Función	Retardo máximo [ns]	Retardo de lógica [ns]	Retardo de rutas [ns]
KeySchedule	8.862	4,985	3,876
MIX	6,136	5,025	1,111
Permute	3,827	2,924	0,903

TABLA V
RESULTADOS DE IMPLEMENTACIÓN DE RONDA ITERATIVA PARA THREEFISH-256

Diseño	Dispositivo	Área [Slices]	Frecuencia Máxima [MHz]	Rendimiento [Mbps]	TPA [Mbps/Slice]
[11]	XC5VLX50	1001	114,942	408,7	0,408
[14]	XC5VLX110	1074	175.865	616,73	0,574
[16]	Virtex-5	519	299.0	262.0	0.5
Este trabajo	XC5VLX110T	815	154,984	551,08	0,676

El rendimiento de los diseños realizados se puede calcular por:

$$TP = \frac{Tamaño\ de\ Bloque}{Latencia} \quad (1)$$

Los valores obtenidos en este trabajo permiten concluir que para procesar un mensaje de 256 bits con *Threefish-256* se obtiene una latencia de $6{,}452\text{ns} \times 72(\text{rondas}) = 464{,}544\text{ns}$. Por lo tanto, el rendimiento calculado con (1) para *Threefish-256* es $(256/464{,}544) \times 10^9 = 0{,}551\text{Gbps}$.

Tanto los resultados de [14] y los de este trabajo incluyen el hardware de control para un funcionamiento totalmente autónomo, mientras que el trabajo de [11] presenta sólo el núcleo de la función. Los resultados muestran que el diseño presentado en este trabajo tiene un área menor y presenta la mejor relación entre el área y la velocidad (TPA). Esto se puede atribuir principalmente al diseño del generador de subclaves, *KeySchedule*, que usa un enfoque basado en contadores para determinar el número de la subclave a calcular. Sin embargo, si este diseño desea escalarse para ser usado en una arquitectura *8-unrolled round*, el área aumenta considerablemente. Una solución a esta situación es implementar el generador de subclaves usando registros de desplazamiento como en el propuesto en [15] en el que no deben agregarse más *flip-flops*, solo se necesitan dos circuitos sumadores más.

5. CONCLUSIONES

Este trabajo reporta resultados de la implementación del algoritmo *Threefish-256* sobre la tarjeta de desarrollo *Xilinx XUPV5-LX110T.* Los resultados se obtuvieron mediante la síntesis de la descripción del circuito en lenguaje VHDL y fueron verificados mediante *ChipScopePro*. El objetivo de este trabajo era explicar el enfoque de ronda iterativa para la construcción del *núcleo de* Skein. En este caso, el algoritmo implementa una sola ronda que se itera 72 veces para obtener finalmente el texto cifrado. Lo anterior permite optimizar el uso de los recursos de la FPGA pues no es necesario implementar las 72 rondas por separado.

El enfoque de ronda iterativa es un método común para implementar una función criptográfica basada rondas, su propósito es equilibrar el área consumida con el rendimiento del circuito. Se obtuvo un rendimiento de $551{,}08\ \text{Mbps}$ y una relación rendimiento con respecto al área de $0{,}676\ \text{Mbps/Slice}$, superando a las implementaciones de [11] y [14] en un 65.69% y 17.77% respectivamente.

AGRADECIMIENTOS

El presente trabajo de investigación fue realizado gracias a COLCIENCIAS y a la Universidad del Valle con la financiación del proyecto “Diseño síncrono y especificación asíncrona de la función Skein-256 para firmas digitales”. Código: P-2011-0778, cuyo desarrollo ha posibilitado la apropiación del conocimiento en el tema afín al área de la criptografía y la implementación a nivel de hardware reconfigurable de algoritmos criptográficos.

REFERENCIAS

[1] D. Stinson, *Cryptography: Theory and Practice*, Third Edit. CRC Press, Inc, 2005.

[2] B. Schneier, N. Ferguson, and S. Lucks, “The skein hash function family,” *Submission to NIST (Round 3)*, 2010. [Online]. Available: http://www.skein-hash.info/sites/default/files/skein1.3.pdf.

[3] NIST, “Cryptographic hash algorithm competition,” *National Institute of Standards and Technology,* 2005. [Online]. Available: http://csrc.nist.gov/groups/ST/hash/sha-3/index.html.

[4] NIST, “Tentative Timeline of the Development of New Hash Functions,” *National Institute of Standards and Technology,* 12-Jul-2006. [Online]. Available: http://csrc.nist.gov/ groups/ST/hash/timeline.html.

[5] NIST, “Announcing request for candidate algorithm nominations for a new cryptographic hash algorithm (SHA-3) Family,” *Federal Register*, 29-Oct-2007. [Online]. Available: https://federalregister.gov/a/E7-21581.

[6] Xilinx, “Virtex-5 FPGA Data Sheet,” *Xilinx Inc.*, 2009. [Online]. Available: http://www.xilinx.com/support/documentation/data_sheets/ds202.pdf.

[7] Xilinx, “Xilinx UG190 Virtex-5 FPGA User Guide,” *Xilinx Inc.*, 16-Mar-2012. [Online]. Available: http://www.xilinx.com/support/documentation/user_guides/ug190.pdf.

[8] M. Liskov, R. L. Rivest, and D. Wagner, “Tweakable block ciphers,” *J. Cryptol.*, vol. 24, no. 3, pp. 588–613, Sep. 2011.

[9] N. Ferguson, S. Lucks, B. Schneier, D. Whiting, M. Bellare, T. Kohno, J. Callas, and J. Walker, “The Skein

Hash Function Family," *Schneier on Security*, 01-Nov-2008. [Online]. Available: https://www.schneier.com/skein.html.

[10] N. At, J. L. Beuchat, and I. San, "Compact implementation of threefish and Skein on FPGA," in *2012 5th International Conference on New Technologies, Mobility and Security - Proceedings of NTMS 2012 Conference and Workshops*, 2012, pp. 1–5.

[11] M. Long, "Implementing skein hash function on xilinx virtex-5 fpga platform," 02-Feb-2009. [Online]. Available: http://www.skein-hash.info/sites/default/files/skein_fpga.pdf.

[12] A. H. Namin and M. A. Hasan, "Hardware Implementation of the Compression Function for Selected SHA-3 Candidates," *CACR 2009-28*, 2009. [Online]. Available: http://cacr.uwaterloo.ca/techreports/2009/cacr2009-28.pdf.

[13] J. Walker, F. Sheikh, S. Mathew, and R. Krishnamurthy, "A Skein-512 Hardware Implementation," Aug-2010. [Online]. Available: http://csrc.nist.gov/groups/ST/hash/sha-3/Round2/Aug2010/documents/papers/WALKER_skein-intel-hwd.pdf.

[14] A. Schorr, "Performance analysis of a scalable hardware fpga skein implementation," Kate Gleason College of Engineering, Rochester Institute of Technology, Rochester, NY, 2010.

[15] S. Tillich, "Hardware Implementation of the SHA-3 Candidate Skein.," *IACR Cryptology ePrint Archive*, Apr-2009. [Online]. Available: http://eprint.iacr.org/2009/159.pdf?origin=publication_detail.

[16] B. Jungk and J. Apfelbeck, "Area-Efficient FPGA Implementations of the SHA-3 Finalists," in *2011 International Conference on Reconfigurable Computing and FPGAs*, 2011, pp. 235–241.