DISEÑO ASÍNCRONO DE LAS FUNCIONES DE TRANSFORMACIÓN DEL ALGORITMO *THREEFISH-256.*

ASYNCHRONOUS DESIGN OF TRANSFORMATION FUNCTIONS OF *THREEFISH-256* ALGORITHM

Nathaly Nieto Ramírez[1]
Rubén Darío Nieto Londoño[2]

[1.] Investigadora Grupo GADyM, nathaly.nieto@correounivalle.edu.co
[2.] Docente Tiempo Completo Universidad del Valle, ruben.nieto@correounivalle.edu.co

Recibido: 07 de febrero de 2014
Aceptado: 08 de marzo de 2014
Correspondencia del autor:nathaly.nieto@correounivalle.edu.co
Universidad del Valle Calle 13 No.100-00 Edificio 354. Cali, Colombia

RESUMEN

Los sistemas digitales han crecido en complejidad y la velocidad del reloj aumenta continuamente, incrementando de la misma manera algunos problemas como el retraso de la señal de reloj, el rendimiento total del sistema y el consumo de potencia. Debido a esto se está experimentando un interés en el diseño de circuitos asíncronos, los cuales no manifiestan este tipo de problemas. Este trabajo presenta los resultados de la implementación asíncrona en hardware de las funciones de transformación del algoritmo criptográfico *Threefish* en su proceso de cifrado, con el fin de utilizar las bondades de los diseños asíncronos en criptografía. Para la especificación, síntesis y simulación asíncrona de los módulos de *Threefish* se usó una herramienta EDA (*Electronic Design Automation*) de distribución libre conocida como Balsa. Esta plataforma define su propio lenguaje de descripción de hardware, mientras la implementación se realizó sobre hardware reconfigurable a través de la plataforma ISE Design Suite de Xilinx Inc.

Palabras Clave: *Balsa, Criptografía, Diseño asíncrono, FPGA, Threefish.*

ABSTRACT

Digital systems have grown in complexity and the clock speed increases progressively and subsequently some problems such as the latency, the throughput system and power consumption. Because of this, there is a current interest in the design of asynchronous circuits, which do not exhibit such problems. This paper show the results of asynchronous hardware implementation of the transform functions of cryptographic algorithm *Threefish* in its encryption process, in order to utilize the benefits of asynchronous designs in cryptography . For the specification, synthesis and simulation of asynchronous Threefish modules, an EDA (*Electronic Design Automation*) freeware tool called Balsa was used. Balsa defines its own hardware description language, while the implementation was using reconfigurable hardware via the ISE Design Suite Xilinx platform Inc.

Keywords: *Asynchronous Design, Balsa, Cryptography, FPGA, Threefish*

INTRODUCCIÓN

La evolución de las técnicas criptográficas sugiere el desarrollo de métodos de cifrado más sofisticados y eficientes, así como técnicas de criptoanálisis. Mediante los ataques a la seguridad de la información se pretenden obtener claves y/o mensajes cifrados, lo cual se puede lograr a nivel de hardware, usando ataques activos o pasivos (1). Con los ataques pasivos se logra la filtración o monitoreo de la transmisión aprovechando las fugas físicas presentes en el propio dispositivo criptográfico durante el procesamiento de un algoritmo.

Una forma de prevenir los ataques pasivos consiste en diseñar sistemas criptográficos usando técnicas asíncronas. Los circuitos asíncronos son diferentes a los circuitos síncronos pues no asumen la premisa de un tiempo discreto; en lugar de ello, estos circuitos se sincronizan a través de señales *handshake* entre sus componentes.

En particular, las funciones criptográficas *hash* han sido diseñadas para generar firmas digitales de manera eficiente y segura; sin embargo, hoy se utilizan para proteger la información al inicio de sesión con contraseña, para asegurar las conexiones a páginas web y administrar claves de cifrado entre otras aplicaciones.

Threefish es un cifrador de bloque ajustable que actúa como núcleo de la función *hash Skein*, opera con palabras sin signo de 64 bits y se puede implementar en tres diferentes tamaños de bloque: 256 bits, 512 bits o 1024 bits (2). La función *Skein* participó como candidato en el proceso de selección para convertirse en el próximo estándar SHA-3 para firmas digitales del Instituto Nacional de Estándares y Tecnología de Estados Unidos (*National Institute of Standards and Technology: NIST*) que inició en el año 2008 y finalizó a finales del año 2012 (3).

Este artículo muestra una metodología de diseño asíncrono enfocada en el área de la criptografía. En particular, se mostrarán los resultados de implementación en términos de tiempo de retardo y utilización de área para las funciones de transformación presentes en el cifrador *Threefish-256* especificadas mediante el estilo de diseño asíncrono.

MATERIALES Y MÉTODOS

En los circuitos asíncronos la información se propaga dónde y cuándo se necesita. Sin embargo, las partes del circuito conforman un sistema total con un propósito y el flujo de la información entre las partes del circuito debe producirse con una secuencia determinada; los circuitos asíncronos al no contar con un reloj global que realice esta tarea, utilizan protocolos de *handshake* como medio de sincronización (4).

En el caso más simple de comunicación asíncrona se tienen dos bloques, un receptor y un transmisor, y un canal de datos. El bloque activo es el que inicia la comunicación y el bloque pasivo es el que responde a la solicitud del bloque activo. Cuando el transmisor inicia la comunicación transmitiendo datos a través del canal, el transmisor es el bloque activo y el canal es tipo *push* pero si por el contrario el que inicia la comunicación solicitando datos es el receptor, el bloque activo es el receptor y el canal es tipo *pull* (5).

Para la negociación de la transferencia de información entre los elementos de comunicación de los circuitos asíncronos es necesario seguir uno de los protocolos de comunicación establecidos. En este caso se usa el protocolo *single-rail* o de dato acotado de 4 fases (Figura 1) en el que se usan niveles lógicos para codificar la información y se utiliza una sola línea por bit para la transferencia de datos. El término '4 fases' se refiere al número de acciones de comunicación (6).

Para un canal tipo push se tienen las siguientes eventos (para el canal tipo pull las señales, la dirección de los eventos de *request y acknowledge* son inversos):

1. el transmisor envía al receptor los datos junto con la señal de *request* en alto.
2. el receptor captura los datos y envía la señal *acknowledge* en alto.
3. el emisor responde poniendo la señal request en bajo (no se garantizan la validez de los datos).
4. el receptor reconoce esto poniendo la señal *acknowledge* en bajo.

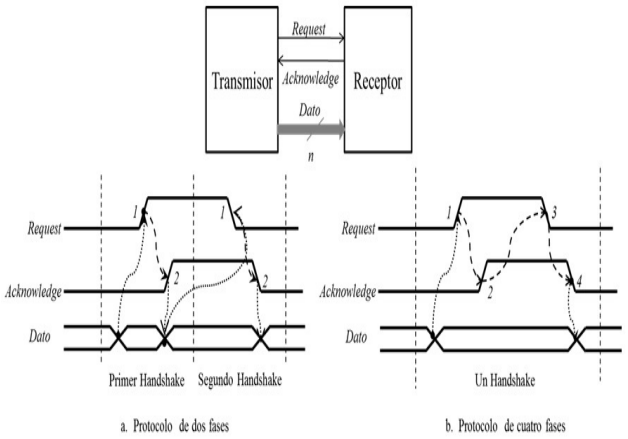

Figura 1. Protocolo de dato acotado de dos y cuatro fases con canal tipo push (6).

Los circuitos asíncronos diseñados se realizaron utilizando la plataforma de diseño conocida como Balsa. Este es el nombre del conjunto de herramientas para la síntesis de sistemas de hardware asíncrono y también el nombre del lenguaje para describir estos sistemas. La metodología de compilación Balsa está dirigida a la síntesis de los componentes de *handshake*. La ventaja de usar este lenguaje de descripción es que en la compilación de las descripciones circuitales hay una correspondencia uno a uno entre la especificación y los circuitos intermedios de *handshake* que se producen (7).

Balsa, al ser un sistema orientado a la síntesis, no soporta la etapa de implementación hardware pero si puede generar un archivo de tipo *netlist* (lista de nodos) que se pueden importar desde una herramienta CAD (*Computer Aided Design*) comercial para producir la implementación en hardware (8). La herramienta CAD usada en la implementacion de los diseños es Xilinx ISE y la plataforma reconfigurable es la FPGA de Xilinx Virtex5. La metodología de diseño que propone Balsa está basada en el intercambio de señales entre componentes de *handshake* (9). Un elemento funcional asíncrono se representa por un componente de *handshake* (Figura 2).

Los Circuitos de *handshake* combinan diseño macromodular con las comunicaciones insensibles a retardos para producir una metodología de diseño en la que la totalidad de los diseños se describen utilizando macromódulos conectados entre sí por canales de comunicación asíncrona. Cada componente de handshake en un diseño es una instancia de una biblioteca de celdas que se pueden parametrizar con ciertas limitaciones (10).

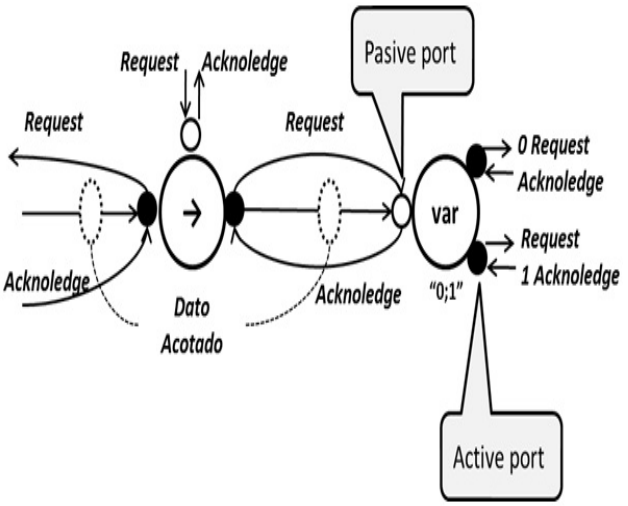

Figura 2. Componentes de handshake (7), (9).

RESULTADOS Y DISCUSIÓN

El algoritmo de *Threefish* está compuesto por cuatro funciones de transformación: *BytesTowords*, que prepara el mensaje a procesar; *MIX*, mezcla no-lineal; *Permute*, que conmuta la posición de palabras y *WordsToBytes*, que ordena la salida. Además, consta de una unidad de generación de subclaves, *KeySchedule*. En la Figura 3 se presenta el diagrama de bloques general del algoritmo *Threefish*. En (2) se encuentra la descripción detallada de cada una de estas funciones y del algoritmo en general.

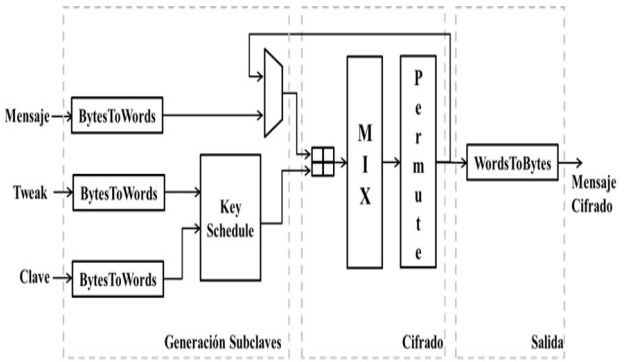

Figura 3. Diagrama de bloques del algoritmo Threefish.

Funciones *BytesToWords*

El objetivo principal de la función *BytesToWords* es adecuar la entrada para que las funciones intermedias realicen el proceso de cifrado. *BytesToWords* divide el texto en palabras de 64 bits y a su vez realiza un cambio de posición a nivel de bytes. Esta función tiene como entradas un texto plano, el *Tweak* o la clave, y

genera como salida un número de palabras de 64 bits que, dependiendo del tamaño del bloque (16, 32, 64 o 128), serán 2, 4, 8 o 16 respectivamente.

BytesToWords realiza llamados a los bloques acond, *funor y shiftl*. El procedimiento acond, adapta los 32 Bytes de entrada del algoritmo en 32 palabras de 64 bits para que puedan ser desplazadas en el módulo siguiente, *ShiftL*, y finalmente realiza la función lógica *OR* entre grupos de cuatro palabras para conformar las palabras expandidas. Para efectos de una visualización más clara, el diagrama de *handshake* de la Figura 4 muestra la transformación de ocho bytes a una palabra de 64 bits.

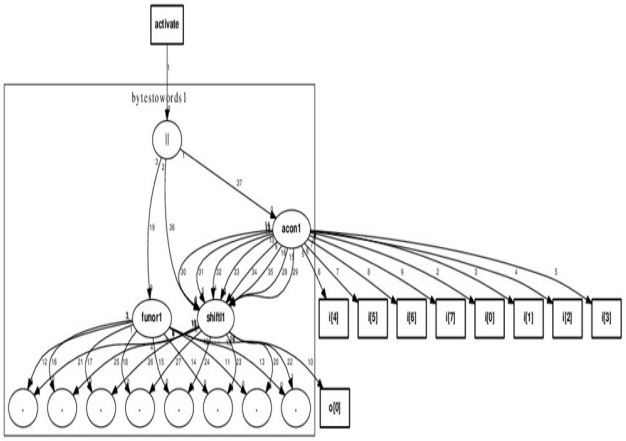

Figura 4. Diagrama de *handhake* para *BytesToWords* generado por Balsa

En el diagrama de *handshake* se pueden diferenciar cada uno de los bloques que conforman la función *BytesToWords*. Inicialmente se identifica un puerto de activación que se comunica con el operador "||".

Las unidades que estén conectadas a este operador se procesaran en paralelo. Sin embargo, existe una transición intermedia entre la función *shiftL* y funor debido a que las salidas del procedimiento *shiftL* se transmiten a la entrada de *funor*, creándose así un punto de sincronización.

Función *MIX*

MIX es el núcleo del algoritmo *Threefish*. La función $MIX_{d,j}$ tiene dos palabras de entrada (x_0,x_1) y produce dos palabras de salida (y_0,y_1) usando las siguientes relaciones:

$$y_0 := (x_0 + x_1) \mod 2^{64} \qquad (1.1)$$

$$y_1 := (x_1 <<< R_{(d \bmod 8),j}) \oplus y_0 \qquad (1.2)$$

Donde <<< es la operación de rotar hacia la izquierda. Las constantes de rotación $R_{d,j}$ se encuentran definidas en (2). Para implementar este bloque se utilizaron dos sumadores en módulo 2^{64}, la función de rotación a la izquierda rotación256 con sus respectivas constantes de rotación *cte*R256 y dos operaciones *XOR* de 64 bits, *funxor*. En la Figura 5 se muestra el diagrama de bloques de dos funciones *MIX* utilizadas en una ronda para un tamaño de bloque de 256 bits. El diagrama de *handshake* se muestra en la Figura 6.

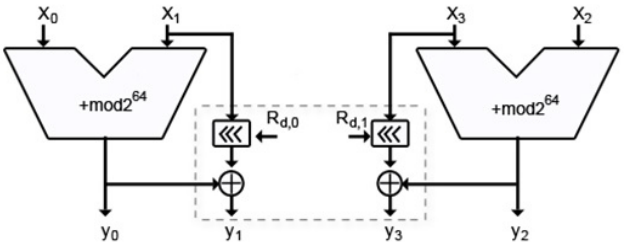

Figura 5. Diagrama de bloques de la función *MIX*

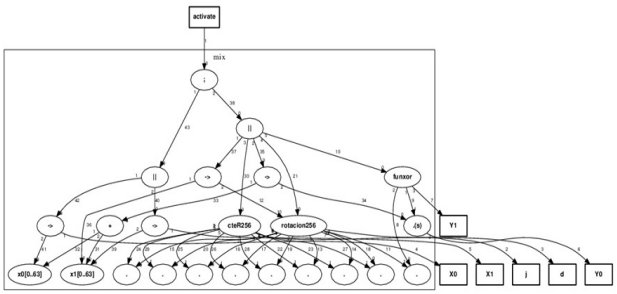

Figura 6. Diagrama de *handshake* de la función *MIX*

En el diagrama de handshake de la función MIX después de la activación se observa el operador de secuencia, ";", que se comunica con dos operadores paralelos "||", esto indica que el proceso de asignación entre las variables *x0[0..63]* y *x1[0..63]* se realiza de forma secuencial, pero al interior de cada variable la asignación es de forma paralela. Los datos almacenados en estas variables son los que se procesan junto con las constantes de rotación que se encuentran en *cte*R256 y activan los procedimientos *rotacion*256, y *funxor* para producir las salidas *Y0* y *Y1*.

Función *Permute*

Permute cambia la posición de las palabras de entrada y se define como:

$$v_{d+1,i} := f_{d,\pi(i)} \qquad \text{para } i = 0,\ldots,N_w - 1 \qquad (1.3)$$

El valor de π(i) se especifica en (2). La Figura 7 muestra el diagrama de *handshake* de la función.

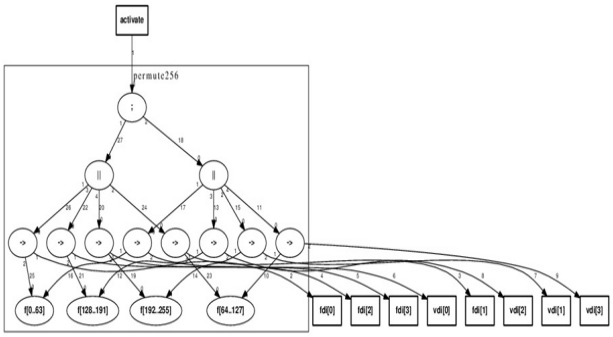

Figura 7. Diagrama de *handshake* de la función *Permute*

Este diagrama ilustra cómo se almacena la entrada *fdi* en la variable *f* dependiendo del orden determinado por la función π(i) y finalmente se asigna a la salida *vdi*. El operador ";" indica que primero se debe capturar la entrada para asignar la salida. Las operaciones de captura y asignación de datos se realizan en forma paralela mediante el operador "||".

Función *WordsToBytes*

WordsToBytes distribuye cada palabra de 64 bis en 8 bytes y la salida constituye el texto cifrado. En la Figura 8 se muestra como la función *WordsToBytes* activa el bloque *toBytes* para procesar los datos de entrada. Para mejor ilustración, el diagrama de handshake muestra únicamente el proceso de una palabra de 64 bits. También se muestra la activación del procedimiento *tobytes* que es el que realiza el procesamiento de los datos. El diagrama de *handshake* de *tobytes* es bastante denso por lo que no se anexa aquí.

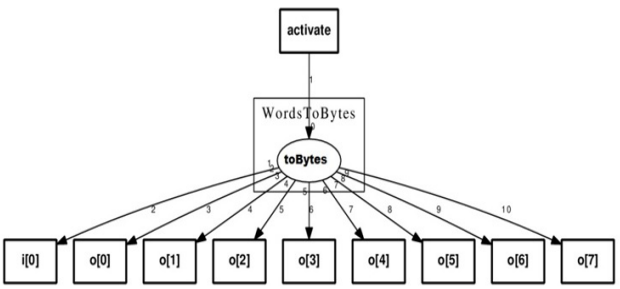

Figura 8. Diagrama de *handshake* de *WordsToBytes*

Función de generación de subclaves, *KeySchedule*
La generación de subclaves comienza definiendo dos palabras adicionales k_{Nw} y t_2 por:

$$k_{Nw} := C240 \oplus \bigoplus_{i=0}^{Nw-1} k_i \qquad t_2 := t_0 \oplus t_1 \qquad (1.4)$$

La constante C_{240} = 0x1BD11BDAA9FC1A221[1] asegura que la clave extendida no sean todos ceros. La generación de subclaves está definido por:

$$k_{s,i} := k_{(s+i) \bmod (Nw+1)} \qquad \text{para } i=0,\ldots,N_w - 4$$

$$k_{s,i} := k_{(s+i) \bmod (Nw+1)} + t_{s \bmod 3} \qquad \text{para } i = N_w - 3$$

$$k_{s,i} := k_{(s+i) \bmod (Nw+1)} + t_{(s+1) \bmod 3} \qquad \text{para } i = N_w - 2$$

$$k_{s,i} := k_{(s+i) \bmod (Nw+1)} + s \qquad \text{para } i=N_w - 1$$

Las sumas son todas en módulo 2^{64}.

En la Figura 9 se muestra el diagrama de bloques de la unidad de generación de subclaves, el diagrama de *handshake* generado por Balsa es muy denso y no se visualiza de manera clara, por lo cual no se incluye.

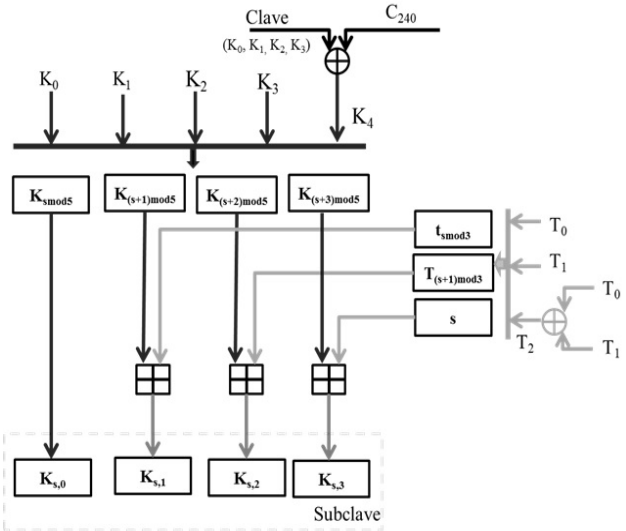

Figura 9. Diagrama del bloque de generación de subclaves.

La tabla 1 muestra el costo de área está dado por la herramienta *breeze-cost* de Balsa para cada una de las funciones de *Threefish-256*. El costo mostrado es un valor de referencia para estimar el área total, tenien-

1 C_{240} es el cifrado AES del texto plano 240 (en decimal) con los 256 bits de la clave en cero, es decir, $C_{240} = AES\text{-}256_0(240)$.

do en cuenta el área de cada circuito de handshake que compone el circuito total. Cada componente se detalla en (7). Este valor del costo es útil pues brinda información rápida sobre cómo un cambio de la descripción del circuito afecta su tamaño.

Función	Costo total (unidades)	Tiempo de finalización (ns)
BytesToWords	154,049.75	1,199,700
KeySchedule	1,244,266.75	55,500
MIX	32,212.75	70,300
Permute	6,915	4,900
WordsTobytes	36,565.25	110,500

Tabla 1. Resultado del costo funciones de transformación asíncrona.

Las funciones de transformación fueron sintetizadas con la herramienta Xilinx ISE y se implementaron en la FPGA *Virtex-5* (11) *XUPV5-LX110T-3FF136* de *Xilinx*. Los resultados de síntesis de las funciones se muestran en la Tabla 2. La Tabla 3 consigna los resultados del mapeo de las funciones asíncronas después del *place-and-route.*

Función	Retardo Máximo	Uso de memoria total
BytesToWords	317.377ns 45.512ns lógica (14.3%) 272.225ns rutas (85.7%)	427,912 KB
KeySchedule	495.325ns 75.958ns lógica (15.3%) 419.367ns rutas (84.7%)	843,838 KB
Permute	18.189ns 4.214ns lógica (23.2%) 13.975ns rutas (76.8%)	171,044 KB
WordsToBytes	379.716ns 46.886ns lógica (12.3%) 332.830ns ruteo (87.7%)	177,736 KB

Tabla 2. Resultados de la síntesis de las funciones de transformación asíncronas

Utilización lógica	BytesToWords [unidades]	KeySchedule [unidades]	Permute [unidades]	WordsToBytes [unidades]
Registros	2,200	6,024	0	128
LUTs	8,306	31,563	30	749
Slices	3,519	0	18	450

Tabla 3. Resultado de mapeo de las funciones asíncronas en la *Virtex-5*

Los circuitos se implementaron usando el protocolos de 4 fases (*four_b_rb en Balsa*) Los resultados muestran que el mayor porcentaje de retardo que se obtiene obedece a los retardos causados por las rutas de conexión en la FPGA (mayor a 76%). Esto se debe a la sobrecarga en los circuitos de *handshake* y también a que la arquitectura de la FPGA usada carece de elementos propios de los circuitos asíncronos.

CONCLUSIONES

En este trabajo se han descrito las funciones de transformación del algoritmo *Threefish* con el estilo de diseño asíncrono buscando aprovechar las ventajas que ofrece este estilo de diseño en los circuitos criptográficos. Se implementó la arquitectura asíncrona de las funciones de transformación del algoritmo *Threefish* en la FPGA *Virtex 5*, presente en el sistema de desarrollo *XUPV5-LX110T* de Xilinx. Para la síntesis de los circuitos asíncronos se utilizó el lenguaje Balsa y se utilizó la codificación *single-rail* de 4 fases. Para la implementación de las funciones aprovechó la posibilidad que ofrece Balsa para enlazarse con la herramienta CAD, Xilinx ISE.

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.

BIBLIOGRAFÍA

1. ECRIPT, [internet] "The side Channel Cryptanalysis Lounge". Disponible http://www.crypto.ruhr-uni-bochum.de/en_sclounge.html.
2. Schneier, B., Ferguson y N. & Lucks, S. (2011). "The skein hash function family". Submission to NIST (Round 3). Available at: http://www.skein-hash.info/sites/default/files/skein1.3.pdf.
3. NIST, (2010). "Cryptographic Hash Algorithm Competition. National Institute of Standards and Technology". Disponible en: http://csrc.nist.gov/groups/ST/hash/sha-3/index.html.
4. Jacobson, H. (1996). "Asynchronous Circuit Design A Case Study Of A Framework Called Ack". Department of Computer Engineering Lulea University of Technology. Disponible en: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.100.1114
5. Alarcón, Rubén V. (2003). "Síntesis de Circuitos Digitales Asíncronos Aplicados en Comunicación". Ph.D. disertación, Universitat Autonoma de Barcelona. ISBN: 8468841862. Disponible en: http://www.tdx.cat/bitstream/handle/10803/5342/rvam1de1.pdf?sequence=1
6. Sparsø, J. y Furver, S. (2001). "Principles of Asynchronous Circuits Design- A System Perspective". Kluwer Academics Publishers. ISBN 0-7923-7613-7. Disponible en: http://intranet.cs.man.ac.uk/apt/async/pubwork/index.html
7. 7. Edwards, D., Bardsley, A., Janin L., Plana, L. y Toms, W. (2006). "Balsa: a Tutorial Guide". Versión 3.5. Disponible en: http://intranet.cs.man.ac.uk/apt/projects/tools/balsa/
8. 8. Nieto, R. y Bernal, A. (2010). "Arquitectura para la Implementación Asíncrona de la Función de Sustitución de Byte del Algoritmo de Rijndael". En Dyna vol.77, n.162, pp. 281-291.
9. 9. Komatsu, Y., Hariyama, M. y Kameyama, M. (2012). "Architecture of an Asynchronous FPGA for Handshake-Component-Based Design". IEICE Transactions on Information and Systems, Vol.E96-D No.8 pp.1632-1644.
10. 10. Bardsley A., (2000) "Implementing Balsa handshake Circuits", Ph.D. dissertation, faculty of Science & Engineering, University of Manchester. Disponible en: ftp://ftp.cs.man.ac.uk/pub/amulet/theses/bardsley_phd.pdf
11. 11. Baliñas, J. FPGA *Virtex* V de *Xilinx*. (2009). Disponible en: http://delep.uah.es/antigua/PLANES%20ANTIGUOS/I.ECA/2/Dise%F1o%20de%20Ctos%20y%20Sist%20Ecos/Apuntes/Virtex-5_JBS.pdf