RIELAC, Vol.XXXIV 3/2013 p.41-51 Septiembre - Diciembre ISSN: 1815-5928

RIELAC
Revista de Ingeniería Electrónica, Automática y Comunicaciones

Diseño e integración de algoritmos criptográficos en sistemas empotrados sobre FPGA

Alejandro Cabrera Aldaya[1], Alejandro José Cabrera Sarmiento[2]

[1]Complejo de Investigaciones Tecnológicas Integradas, acabrera@udio.cujae.edu.cu

[2]Instituto Superior Politécnico "José Antonio Echeverría", Facultad de Ingeniería Eléctrica, alex@electrica.cujae.edu.cu

RESUMEN / *ABSTRACT*

En este trabajo se integran implementaciones hardware de algoritmos criptográficos a la biblioteca OpenSSL la cual es utilizada por aplicaciones sobre el sistema operativo Linux para asegurar redes TCP/IP. Los algoritmos implementados son el AES y las funciones resumen SHA-1 y SHA-256. Estos algoritmos son implementados como coprocesadores del procesador MicroBlaze utilizando interfaces FSL para el intercambio de datos entre ellos. Estos coprocesadores son integrados dentro de la biblioteca OpenSSL considerando la naturaleza multitarea del sistema operativo Linux, por lo que se selecciona un mecanismo de sincronización para controlar el acceso a estos dispositivos. Además son presentados los resultados de velocidad alcanzados por los coprocesadores integrados en la biblioteca utilizando la herramienta *speed* de la misma. Finalmente es presentado el impacto de estos coprocesadores en la velocidad de transmisión a través de una red privada virtual utilizando la herramienta OpenVPN.

Palabras claves: FPGA, Microblaze, Linux, criptografía, OpenSSl

In this paper hardware implementations of cryptographic algorithms are integrated to the OpenSSL library which is used by the Linux operating system applications to protect TCP/IP networks. The selected algorithms are AES and the hash functions SHA-1 and SHA-256. These algorithms are implemented in hardware as coprocessors of Xilinx MicroBlaze soft processor core using the FSL interface for data exchange. These coprocessors are integrated inside the OpenSSL library considering Linux operating system multitasking, so a synchronization technique is selected to control the access to the hardware coprocessors. The speed gain achieved is measured using the OpenSSL's speed utility. Finally a virtual private network is configured using OpenVPN in order to compare the transmission rate using the coprocessors and without them.

Key words: FPG, Microblaze, Linux, cryptography, OpenSSL

Design and Integration of Cryptographic Algorithms on FPGA-Based Embedded Systems

INTRODUCCIÓN

Debido al creciente aumento de la complejidad de los sistemas empotrados y la posibilidad de comunicación a través de redes de comunicaciones, la seguridad de las mismas se convierte en un aspecto a tener en cuenta y en donde entran a jugar protocolos criptográficos que solamente habían sido utilizados en sistemas basados en dispositivos de computo tradicionales, como computadoras personales, servidores, etc. con capacidades de procesamiento muy superiores a las que se pueden desarrollar en un sistema empotrado [1]. Esto conlleva a la implementación de estos algoritmos en arquitecturas no convencionales y con diferentes potencialidades, por ejemplo, tanto en microcontroladores de 8 bits basados en arquitectura 8051 de Intel, como en sistemas empotrados basados en el procesador MicroBlaze de Xilinx, haciendo esta tarea no homogénea y dependiente de la arquitectura y de los recursos del sistema de procesamiento [2].

Un aspecto muy importante a tener en cuenta en la implementación de algoritmos criptográficos es la cantidad de recursos de cómputo que puede ser necesario destinar a la ejecución de los mismos, los cuales en algunos casos no son despreciables, como se puede apreciar en [1, 3, 4]. Esto da lugar a que sea necesario acelerar mediante hardware dedicado la ejecución de aquellos algoritmos cuyo tiempo de procesamiento es considerable en un procesador empotrado, permitiendo así reducir la carga del procesador y con ello destinar sus recursos a la ejecución de las aplicaciones principales del sistema. Por ejemplo, en sistemas empotrados donde sea necesario garantizar la seguridad de la comunicación entre diferentes dispositivos conectados a través de una red TCP/IP, es necesario implementar algoritmos criptográficos [5]. Esta tarea no es la principal que debe llevar a cabo este sistema, por lo que la cantidad de tiempo de procesamiento utilizada por la misma debe ser reducida para asegurar que las aplicaciones principales del sistema tengan a su disposición la mayor capacidad de cómputo posible, situación que conlleva a la necesidad de la implementación hardware de los algoritmos criptográficos.

En este trabajo se implementan algoritmos criptográficos utilizando la lógica reconfigurable de los FPGA de Xilinx para luego ser integrados en un sistema empotrado donde el procesador MicroBlaze es la unidad central de procesamiento. El objetivo de esta integración es evaluar el impacto de la aceleración de estos algoritmos durante su utilización en protocolos seguros de comunicaciones basados en TCP/IP.

La organización de este artículo es la siguiente: en la primera sección se brinda una introducción al protocolo SSL/TLS para establecer comunicaciones seguras con el objetivo de seleccionar los algoritmos a implementar en hardware. En la segunda sección se introduce el sistema empotrado de estudio, comentando sus componentes y funciones principales. En la tercera sección se aborda el diseño e implementación de los algoritmos criptográficos seleccionados, así como su integración en el sistema empotrado descrito anteriormente. En la cuarta sección se expone el mecanismo de integración de estos algoritmos en la biblioteca OpenSSL la cual es utilizada por aplicaciones para establecer comunicaciones seguras sobre el sistema operativo Linux. Finalmente se exponen los resultados de la aceleración de los algoritmos en el sistema y se resaltan las principales conclusiones de la investigación.

SELECCIÓN DE LOS ALGORIMOS CRIPTOGRÁFICOS

El protocolo SSL (*Secure Socket Layer*) fue creado desde el principio de la década de los noventa inicialmente por la compañía Netscape. Al finalizar esta década fue estandarizado por la IETF (*Internet Engineering Task Force*) como TLS (*Transport Layer Security*). Este protocolo permite establecer comunicaciones seguras entre dos nodos a través de redes IP [5].

El mismo se divide en dos etapas fundamentales: inicio de sesión e intercambio de datos. El propósito del inicio de sesión es lograr que los nodos que desean establecer una comunicación segura acuerden las claves secretas necesarias para enviar y recibir datos durante la segunda etapa del protocolo. Para llevar a cabo esta tarea se utilizan algoritmos criptográficos de intercambio de claves y de firma digital, ambos basados en criptografía asimétrica (clave de cifrado difiere de la clave de descifrado). La segunda etapa utiliza las claves generadas durante el inicio de sesión para proteger los datos intercambiados entre los dos nodos. Durante esta etapa se utilizan algoritmos de cifrado simétrico y de chequeo de integridad [6].

En esta sección se seleccionarán los algoritmos criptográficos, utilizados en el protocolo TLS, a implementar en hardware utilizando los recursos del FPGA. Para realizar esta selección se tuvieron en cuenta los siguientes parámetros:

- Cantidad de tiempo consumido por la ejecución del algoritmo.
- Frecuencia de utilización de los algoritmos.
- Selección realizada por otros fabricantes y desarrolladores.

La criptografía asimétrica consume más tiempo de procesamiento con respecto al resto de las primitivas criptográficas utilizadas por el protocolo TLS. Una comparación de la misma con respecto a otros algoritmos se muestra en la Tabla 1. Como se puede apreciar en esta tabla, los algoritmos RSA (llamado así por las iniciales de los creadores, *Rivest-Shamir-Adleman*) y ECDSA (*Elliptic Curve Digital Signature Algorithm*), algoritmos de firmas digitales basados en RSA y curvas elípticas respectivamente, consumen mucho más tiempo de procesamiento que el AES-256-CBC (cifrado simétrico) y la función hash SHA-512. Por otra parte, los algoritmos RSA y ECDSA (basados en criptografía asimétrica) solamente son utilizados durante el inicio de una sesión TLS, por lo cual la ejecución de estos algoritmos no es muy frecuente en sistemas empotrados como el utilizado durante esta investigación.

Tabla 1. Consumo de tiempo de distintos algoritmos presentes en el protocolo TLS.

RSA-4096	ECDSA-571	AES-256	SHA-512
20 s (firma)	8,3 s (verificación)	6 ms (1500 bytes)	18 ms (1500 bytes)

Otro de los aspectos que se han tenido en cuenta durante la selección de los algoritmos a implementar con los recursos hardware del FPGA está relacionado con decisiones que han tomado algunos fabricantes e investigadores en relación a este tema. Es muy común encontrar reportes de implementaciones muy rápidas de los algoritmos que utilizan criptografía asimétrica para aquellos sistemas de cómputo que necesiten manejar varias conexiones TLS al mismo tiempo, como es el caso de servidores de redes privadas virtuales, servidores de transferencias bancarias, etc. [7]. Por otra parte, los algoritmos de cifrado simétrico y las funciones hash (los utilizados durante la etapa de intercambio de datos en TLS) se encuentran implementados en sistemas de cómputo que tienden a manejar menor cantidad de conexiones con relación a la cantidad de información que intercambian con el otro extremo de la conexión [8].

Por estas razones se ha seleccionado acelerar los algoritmos que intervienen en la etapa del intercambio de datos del protocolo TLS, es decir un algoritmo de cifrado simétrico y una función resumen (hash), la cual es utilizada para calcular un código de autenticación del mensaje utilizando el algoritmo HMAC (*Keyed-Hashed Message Authentication Code*).

El algoritmo AES-256 y las funciones resumen (hash) SHA-1 y SHA-256 fueron las seleccionadas durante esta investigación. Estos algoritmos son ampliamente utilizados y están estandarizados por lo que poseen una amplia utilización a nivel mundial [9-11]. Más detalles sobre el proceso de selección de los algoritmos a implementar pueden ser encontrados en [12].

IMPLEMENTACIÓN HARDWARE DE LOS ALGORITMOS AES-256, SHA-1 Y SHA-256

En la Figura 1 se muestra un diagrama del sistema empotrado utilizado durante esta investigación. En el mismo se muestra el procesador MicroBlaze así como un conjunto de módulos hardware que permiten la ejecución del sistema operativo Linux y el establecimiento de conexiones TCP/IP tal y como se describe en [13].

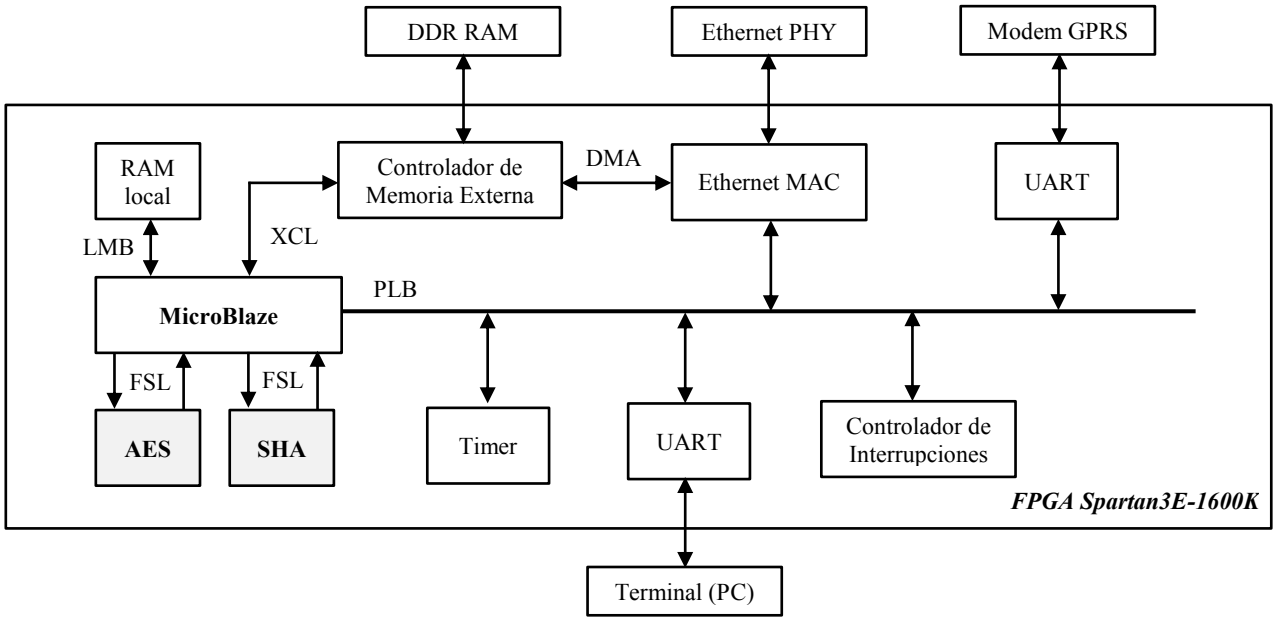

Figura 1. Diagrama en bloques del sistema empotrado utilizado.

El procesador MicroBlaze es capaz de comunicarse con sus periféricos a través de los buses LMB (*Local Memory Bus*) y PLB (*Peripheral Local Bus*) [14]. Además este procesador dispone de la interfaz FSL (*Fast Simplex Link*) dedicada para la expansión de las funcionalidades del procesador a través de coprocesadores hardware [15]. Los coprocesadores de los algoritmos criptográficos implementados utilizan esta interfaz para intercambiar datos con el microprocesador debido a que esta interfaz es la más veloz que posee este procesador [16].

El núcleo de la interfaz FSL es una FIFO compartida entre el procesador y el coprocesador, lo que permite el intercambio de datos de una forma directa y rápida utilizando instrucciones de escritura y lectura de la interfaz [17].

Implementación hardware del algoritmo AES-256

El algoritmo AES (*Advanced Encryption Standard*) es un algoritmo de cifrado simétrico de gran aceptación a nivel mundial debido a que no se han encontrado debilidades en el mismo que comprometan su seguridad [9]. Este algoritmo cifra y descifra la información dividiendo los datos a procesar en bloques de 128 bits. El algoritmo AES es capaz de cifrar información con claves de 128, 192 y 256 bits, lo cual lo convierte en un algoritmo fácilmente adaptable a las necesidades de seguridad del sistema donde vaya ser utilizado.

La estructura algorítmica del AES es sencilla pues está basado en cuatro transformaciones fundamentales: ShiftRows, SubBytes, MixColumns, y AddRoundKey, las cuales se le aplican a un bloque de 128 bits llamado estado. El estado se puede representar como una matriz de cuatro filas por cuatro columnas, donde cada celda corresponde a un byte del estado. La transformación ShiftRows, rota un byte de cada fila un determinado número de localizaciones. La transformación SubBytes sustituye cada byte del estado por otro a través de una tabla definida en la descripción del algoritmo. La transformación AddRoundKey es el resultado del OR-exclusivo entre el estado y la subclave de ronda correspondiente. La transformación MixColumn se obtiene al multiplicar cada columna del estado por una matriz de coeficientes utilizando la aritmética de campos finitos utilizada por este algoritmo [9]. Estas transformaciones son aplicadas al bloque a procesar de forma repetitiva, donde el número de iteraciones utilizadas depende del tamaño de clave seleccionado. Las transformaciones son simples de implementar en hardware lo que permite una implementación del mismo con pocos recursos del FPGA. En [12] y en [18] se encuentra una descripción detallada del diseño de este coprocesador.

Se ha seleccionado el tamaño de clave de 256 bits debido a que es el que presenta mayor seguridad y como se expone en [18] la diferencia de la latencia de ejecución del algoritmo no sobrepasa los 20 ciclos de reloj ocupando la misma cantidad de recursos lógicos del FPGA.

En la Figura 2 se puede apreciar un diagrama de la estructura interna de la implementación hardware del algoritmo AES. A la izquierda de la misma se encuentran los bloques encargados de almacenar el bloque que está siendo procesado.

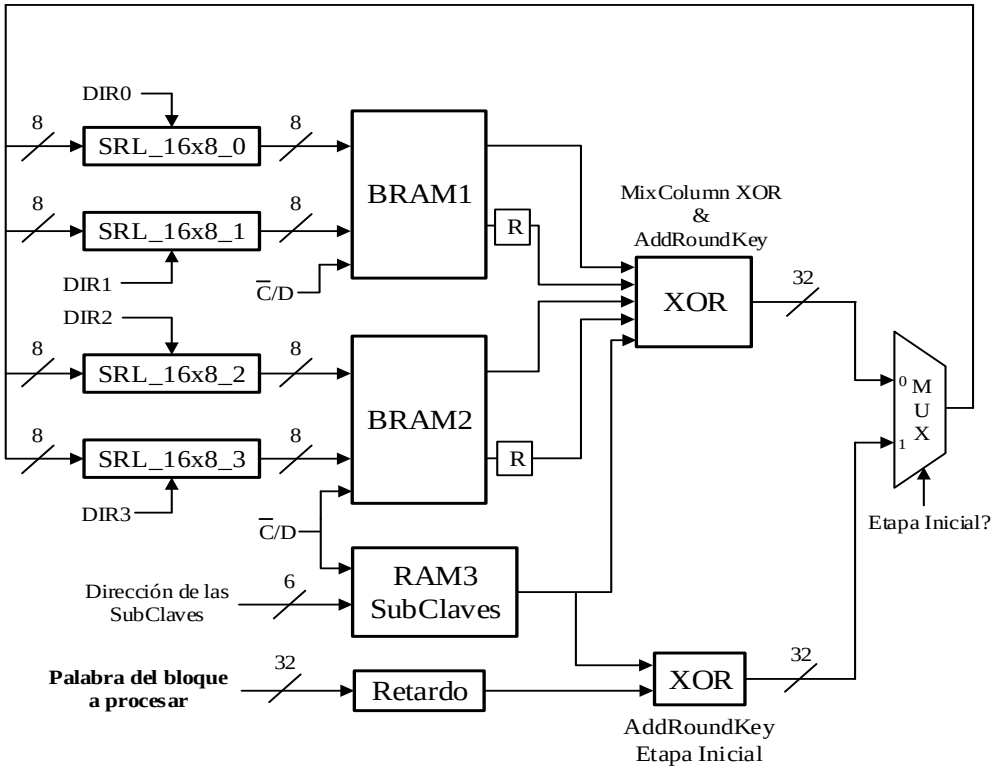

Figura 2. Diagrama del coprocesador del algoritmo AES.

Este almacenamiento se lleva a cabo utilizando registros de desplazamiento direccionables (SRL_16x8_i), los cuales permiten la implementación de las transformaciones ShiftRows y su inversa solamente modificando la dirección de salida de cada registro [19]. Los bloques de memoria RAM (BRAM1 y BRAM2) son utilizados para almacenar las tablas propuestas en [20] para implementar las transformaciones SubBytes y MixColumns del algoritmo, permitiendo ahorrar recursos lógicos del FPGA a cambio de la utilización de dos bloques de memoria del dispositivo. La última transformación es la operación AddRoundKey, la cual es una operación XOR entre el bloque que se está procesando y una subclave de iteración generada previamente y almacenada en la memoria RAM3 de la Figura 2 [18].

Las entradas que se pueden observar en la Figura 2 son manejadas por una máquina de estados finitos. Además, esta máquina de estados permite la comunicación con el procesador MicroBlaze a través de las interfaces FSL.

El diseño de este coprocesador ocupa 163 slices y tres bloques de memoria RAM de un FPGA de la familia Spartan3, siendo este consumo equivalente a una interfaz UART simple, lo que destaca su bajo consumo de recursos lógicos [18].

Implementación hardware de las funciones SHA-1 y SHA-256

Las funciones resumen (hash) permiten obtener un "resumen" de longitud fija de una trama de bits, permitiendo de esta forma comprobar la integridad de los datos. Si los datos a "resumir" son "mezclados" con alguna clave secreta solamente conocida por entidades confiables, entonces es posible confiar en el origen de los datos. El algoritmo encargado de "mezclar" el resumen de un mensaje con una clave secreta se conoce como HMAC. Este algoritmo es genérico para cualquier función resumen que se utilice, por lo que los detalles del mismo no son relevantes para esta investigación [21].

Estas funciones son utilizadas por el algoritmo HMAC para generar un código de autenticación de mensaje y de esta forma autenticar al remitente del mismo [21].

Durante esta investigación se han implementado dos funciones hash (SHA-1 y SHA-256) con el objetivo de abarcar la versión actual del protocolo TLS y las posteriores [6].

En la Figura 3 se muestra el modo de operación de estas funciones. Primeramente se divide el mensaje a "resumir" en bloques de 512 bits y luego cada bloque es procesado consecutivamente utilizándose el resumen del bloque anterior como entrada para el cálculo del siguiente. De esta forma la salida del último bloque es el resumen del mensaje original.

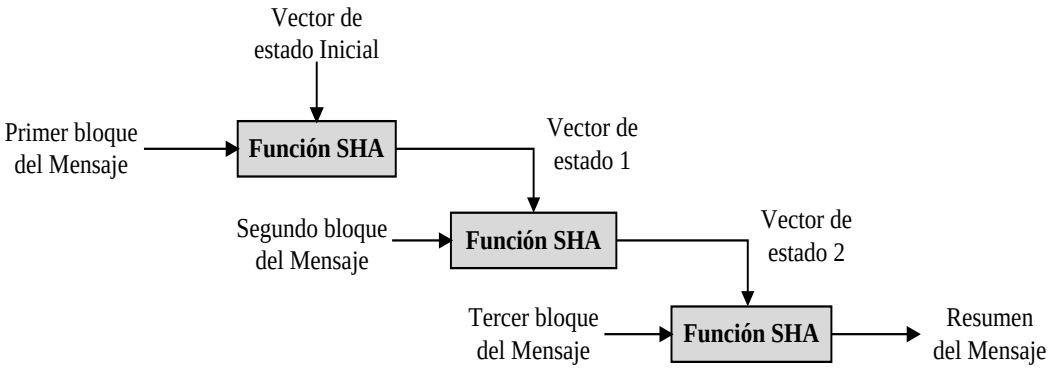

Figura 3. Algoritmo para calcular el "resumen" de un mensaje de tres bloques.

Debido a que solamente es necesario obtener el resumen del mensaje y no los resúmenes intermedios de cada bloque, es conveniente no emplear tiempo en la lectura de los resultados intermedios con el objetivo de alcanzar una mayor velocidad de cómputo.

En el caso del coprocesador del AES descrito anteriormente, el procesador MicroBlaze debe enviarle un bloque de 128 bits hacia el coprocesador, al terminar el procesamiento los resultados son enviados hacia el procesador a través de una interfaz FSL, procedimiento que debe repetirse tantas veces como bloques contenga el mensaje.

Debido al algoritmo de las funciones resumen explicado anteriormente y mostrado en la Figura 3, la lectura de los resultados intermedios solamente implica una pérdida de tiempo durante la ejecución del algoritmo. Con el objetivo de eliminar esta problemática se propone el circuito mostrado en la Figura 4 el cual es capaz de vaciar la FIFO de la interfaz FSL de salida del coprocesador (a través de un reset) al recibir un nuevo bloque por parte del procesador. De esta forma el procesador puede enviar todos los bloques de forma consecutiva y solamente leer el resultado del último bloque que equivale al resumen del mensaje sin necesidad de leer los resultados intermedios, reduciendo de esta forma la latencia inherente a estas lecturas.

Las operaciones matemáticas involucradas en las funciones SHA-1 y SHA-256 son: adición de números de 32 bits, operaciones XOR, rotaciones y desplazamientos, las cuales son sencillas de implementar en hardware. Para la implementación de los coprocesadores de estas funciones se reutilizó un diseño de las mismas disponible gratuitamente en OpenCores.org [22]. Este diseño fue estudiado y modificado, con el objetivo de añadir la comunicación a través de FSL y una máquina de estados que controlase al coprocesador de la cual carecía el módulo original.

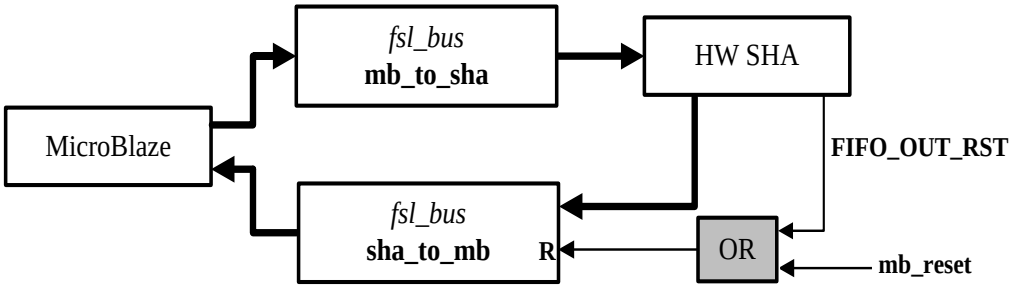

Figura 4. Circuito utilizado para vaciar la FIFO de salida del coprocesador.

El consumo de recursos de los coprocesadores de las funciones SHA-1 y SHA-256 es de 581 y 1016 slices respectivamente [12]. Estos consumos representan aproximadamente el 4 y 7 % de los recursos lógicos del FPGA utilizado durante esta investigación (Spartan3E-1600K) [19].

INTEGRACIÓN DE LOS COPROCESADORES A LA BIBLIOTECA OpenSSL

La biblioteca OpenSSL posee un mecanismo (denominado *engine*) mediante el cual es posible añadir implementaciones alternativas de los algoritmos criptográficos que esta posee. La concepción de este mecanismo es la de incorporar el acceso a los coprocesadores hardware presentes en una arquitectura con el objetivo de remplazar la implementación de los algoritmos con el objetivo de lograr mayor velocidad de procesamiento.

Esta biblioteca es ampliamente utilizada por otras aplicaciones con el objetivo de utilizar los algoritmos criptográficos de la misma como es el caso de las herramientas OpenVPN y OpenSSH [23].

En [12] se encuentra disponible una descripción detallada de este mecanismo para integrar algoritmos de cifrado simétrico y funciones hash a esta biblioteca. De forma general un *engine* contiene la declaración de un objeto cuyos métodos son específicos del tipo de algoritmo (ej. cifrado simétrico o función resumen), siendo estos llamados por la biblioteca cuando un usuario de la misma desea utilizarlo.

Un aspecto a tener en cuenta durante la integración de los coprocesadores a la biblioteca OpenSSL es la naturaleza multitarea del sistema operativo Linux donde esta biblioteca es ejecutada. Esto implica que varias aplicaciones pueden desear utilizar el coprocesador concurrentemente, por lo cual el acceso a los coprocesadores debe ser sincronizado de tal forma de evitar un acceso simultaneo descontrolado. En el sistema operativo Linux las aplicaciones cuando desean intercambiar información con un dispositivo hardware lo hacen a través de drivers de los mismos, resolviéndose el problema de la sincronización planteado anteriormente [24]. Sin embargo, la utilización de drivers para acceder a los coprocesadores a través de las interfaces FSL correspondientes presenta el problema que reduce considerablemente el rendimiento alcanzado por la utilización del coprocesador. Esta razón (la cual fue verificada experimentalmente como parte de la investigación) ha provocado que actualmente el driver para dispositivos FSL esté descontinuado y que la compañía Xilinx (fabricante del sistema de procesamiento MicroBlaze) haya decidido permitir el acceso a los dispositivos conectados a través de interfaces FSL directamente desde las aplicaciones de usuario, sin necesidad de utilizar un driver (en las versiones anteriores a la 8.10 de este procesador las instrucciones para acceder a las interfaces FSL son privilegiadas) [15].

Debido a que el acceso a los coprocesadores se realiza sin la utilización de un driver es necesario sincronizar desde la propia biblioteca OpenSSL el acceso a estos coprocesadores. Luego de un estudio de diferentes mecanismos se ha llegado a la conclusión de que los semáforos con nombre del estándar SUSv3 (*Single Unix Specification v3*) y las funciones para sincronizar el acceso a ficheros, son las más adecuadas para llevar a cabo esta tarea [25]. La utilización de los semáforos con nombre es preferida debido a que estos poseen una función que permite esperar un determinado tiempo por la captura de un semáforo, opción que puede ser útil para detectar y evitar abrazos fatales. Desafortunadamente la versión de la biblioteca estándar del lenguaje C utilizada durante esta investigación no posee implementada los semáforos con nombre, por lo que se ha utilizado la función *flock* disponible en el sistema operativo Linux para sincronizar el acceso a los coprocesadores. De esta forma se crea un fichero que represente a cada coprocesador y solamente puede acceder al mismo aquella aplicación que tenga el control del fichero. Cada aplicación, antes de utilizar un coprocesador, debe poseer el control del fichero que lo representa y al terminar de utilizar el coprocesador liberar el control sobre el mismo.

Resultados de la integración de los coprocesadores

A continuación se presentan algunas comparaciones de la velocidad de procesamiento de los algoritmos implementados en software en la propia biblioteca OpenSSL así como utilizando los coprocesadores diseñados durante esta investigación.

En la Tabla 2 se muestran las velocidades de ejecución de las descripciones software (SW) y hardware (HW) de los algoritmos AES-256 y HMAC utilizando las funciones SHA-1 y SHA-256. Como se puede apreciar en esta tabla, la aceleración obtenida con el coprocesador hardware del algoritmo AES-256-CBC varía entre 4,7 y 25,8 veces, siendo mayor a medida que se incrementa el tamaño de bloque.

Tabla 2. Velocidades obtenidas al integrar los algoritmos en la herramienta OpenSSL.

Implementación	Velocidad en kbytes/s para diferentes tamaños de bloques (en bytes)		
	16	**256**	**8192**
AES-256-CBC-SW	249	266	277
AES-256-CBC-HW	1 187	4 836	7 169
HMAC-SHA1-SW	87	791	1 804
HMAC-SHA1-HW	113	1 890	16 871
HMAC-SHA256-SW	32	195	310
HMAC-SHA256-HW	132	1 365	17 440

Como se puede apreciar, la aceleración de la implementación hardware del algoritmo HMAC utilizando la función resumen SHA-1 es de apenas 1,3 veces para bloques de 16 bytes, incrementándose a medida que se incrementa el tamaño del bloque, hasta alcanzar una aceleración de 9,3 veces para bloques de 8192 bytes.

En la Tabla 2 también se puede apreciar que la aceleración de la implementación hardware del algoritmo HMAC utilizando la función resumen SHA-256 es de 4,1 veces para bloques de 16 bytes, llegando a ser de 56,2 veces para bloques de 8192 bytes.

Es importante destacar la diferencia significativa de velocidad entre las implementaciones software del algoritmo HMAC, por lo que la utilización de la función software SHA-256 utilizando la implementación propia de la biblioteca OpenSSL resultaría en una reducción de rendimiento con respecto a la utilización de la función SHA-1. Sin embargo, debido a que las velocidades alcanzadas utilizando los coprocesadores hardware de ambas funciones son similares, es posible aumentar el nivel de seguridad utilizando el coprocesador de la función SHA-256 sin comprometer el rendimiento del sistema.

La Tabla 3 muestra la velocidad de ejecución de las implementaciones hardware de los algoritmos utilizando o no la función *flock*, permitiendo apreciar el impacto (en la velocidad) de la utilización de esta función para sincronizar el acceso a los coprocesadores por parte de diferentes aplicaciones.

Tabla 3. Impacto de la utilización de la función *flock* para la sincronización del acceso a los coprocesadores.

Implementaciones HW con y sin *flock*	Velocidad en kbytes/s para diferentes tamaños de bloques (en bytes)		
	16	**256**	**8192**
AES-256-CBC	1 347	5 658	7 017
AES-256-CBC-FLOCK	1 187	4 836	7 169
HMAC-SHA1	156	2 364	19 501
HMAC-SHA1-FLOCK	113	1 890	16 871
HMAC-SHA256	164	2 557	21 041
HMAC-SHA256-FLOCK	132	1 365	17 440

Como se puede apreciar, el impacto de la utilización de la función *flock* no es significativo, aunque en general hace disminuir ligeramente la velocidad de ejecución de los algoritmos. Nótese que, aún con la utilización de la función *flock*, se obtienen incrementos de velocidad con respecto a la implementación software de los algoritmos.

Por otra parte, la herramienta OpenVPN permite implementar redes privadas virtuales (VPN, *Virtual Private Network*) a nivel de usuario utilizando los protocolos SSL/TLS. Esta herramienta utiliza la biblioteca OpenSSL para la implementación de estos protocolos y de los algoritmos criptográficos involucrados. En la Tabla 4 se muestran las velocidades alcanzadas con diferentes configuraciones de la herramienta OpenVPN. La primera línea muestra la velocidad alcanzada sin la utilización de criptografía para proteger el túnel de comunicaciones entre los dos extremos de la VPN, mientras que en las restantes se puede apreciar la velocidad utilizando la implementación software de los algoritmos y los coprocesadores diseñados durante esta investigación.

Como se puede apreciar en la Tabla 4 la utilización de los coprocesadores no implica un aumento en la velocidad de la comunicación de más del doble con respecto a la utilización de las implementaciones software. Esto se debe a que la herramienta OpenVPN necesita realizar varias transferencias de datos entre el espacio de usuario y el kernel del sistema operativo para poder encapsular las tramas de red, siendo estas operaciones costosas en tiempo y las principales responsables de que no se observe un incremento notable en la velocidad. Sin embargo como se puede apreciar en esta tabla, la velocidad se multiplica 1,8 veces cuando se utilizan los algoritmos AES y SHA-256 con respecto a la implementación software, además la velocidad alcanzada utilizando la función SHA-1 y SHA-256 es prácticamente la misma, por lo que es posible aumentar la seguridad del sistema sin comprometer el rendimiento del sistema si se utilizan los coprocesadores criptográficos.

Tabla 4. Impacto de los coprocesadores en la aceleración de redes privadas virtuales utilizando OpenVPN.

Configuración de OpenVPN	Velocidad alcanzada (Mbps)
Sin criptografía	6,72
SW (AES-SHA1)	2,35
HW (AES-SHA1)	3,51
SW (AES-SHA256)	1,93
HW (AES-SHA256)	3,49

CONCLUSIONES

Después de haber concluido el proceso de investigación, los resultados más significativos obtenidos han sido los siguientes:

- Se han desarrollado coprocesadores hardware para el algoritmo de cifrado simétrico AES y las funciones resumen SHA-1 y SHA-256.
- El coprocesador AES consume pocos recursos del FPGA, lo que permite su utilización en sistemas empotrados basados en FPGA de bajo y mediano coste.
- Los coprocesadores de las funciones resumen SHA-1 y SHA-256 poseen un consumo de recursos mayor en comparación con el del algoritmo AES, pero aun así pueden ser considerados apropiados para ser incluidos en sistemas empotrados en FPGA de bajo a mediano costo.
- La utilización de los coprocesadores en entornos multitarea requiere la utilización de técnicas de sincronización de tareas para garantizar un correcto funcionamiento de los mismos.
- Es posible aprovechar al máximo la velocidad de ejecución de los coprocesadores si se procesan tamaños de datos iguales o mayores a 8192 bytes.
- La utilización de la función *flock* no implica una pérdida considerable en el rendimiento obtenido por los coprocesadores.
- No es necesario implementar el intercambio de datos entre MicroBlaze y los coprocesadores a través de una interfaz más rápida (como por ejemplo DMA) para acelerar la transmisión de datos a través de redes privadas virtuales utilizando la herramienta OpenVPN, pues la misma no es capaz de aprovechar completamente las velocidades obtenidas a través de la interfaz FSL.

– Es posible aumentar la seguridad (utilizando algoritmos computacionalmente más costosos en software) a través de la utilización de los coprocesadores sin comprometer el rendimiento del sistema.

REFERENCIAS

1. **Argyroudis, P. G.; Verma, R.; Tewari, H. and O'Mahony, D.**: "Performance Analysis of Cryptographic Protocols on Handheld Devices," in Proceedings of the Third IEEE International Symposium on Network Computing and Applications, 2004, pp. 169–174.
2. **Torres López, A. and Cuiman Márquez, R.**: "Implementación de dos Esquemas de Firma Digital sobre Hardware Reconfigurable," Tesis de Diploma, Instituto Superior Politécnico José Antonio Echeverría (CUJAE), 2009.
3. **Xenakis, C.; Laoutaris, N.; Merakos, L. and Stavrakakis, I.**: "A generic characterization of the overheads imposed by IPsec and associated cryptographic algorithms," Computer Networks, vol. 50, no. 17, pp. 3225–3241, 2005.
4. **Liu, W.; Luo, R. and Yang, H.**: "Cryptography Overhead Evaluation and Analysis for Wireless Sensor Networks," in Communications and Mobile Computing, 2009, pp. 496–501.
5. **Kayfman, C.; Perlman, R. and Speciner, M.**: Network Security: Private Communicacion in a Public World, 2nd ed. Prentice Hall, 2002.
6. **Dierks, T. and Allen, C.**: "The TLS Protocol Version 1.0." Internet Engineering Task Force, pp. 1–81, 1999.
7. **Laue, R.; Molter, H. G.; Rieder, F.; Huss, S. A. and Saxena, K.**: "A Novel Multiple Core Co-Processor Architecture for Efficient Server-based Public Key Cryptographic Applications," in IEEE Computer Society Annual Symposium on VLSI, 2008, pp. 87–92.
8. **Broadcom Inc.**: "BCM5365 Product Datasheet," Datasheet BCM5365, 2003.
9. **National Institute of Standard and Technology (NIST)**: "Advanced Encryption Standard." National Institute of Standards and Technlogy (NIST), 2001.
10. **National Institute of Standards and Technology (NIST)**: "Secure Hash Standard," vol. 2. National Institute of Standards and Technlogy (NIST), 2008.
11. **Intel Inc.**: "Intel's s Advanced Encryption Standard (AES) Instructions Set," Israel, 2009.
12. **Cabrera Aldaya, A.**: "Integración de Algoritmos Criptográficos en Sistemas Empotrados sobre FPGA," Tesis de Maestría, Instituto Superior Politécnico José Antonio Echeverría, 2012.
13. **Rodríguez Chong, J. D.**: "Evaluación y Desarrollo de Aplicaciones para el Sistema Operativo Petalinux," Tesis de Diploma, Instituto Superior Politécnico José Antonio Echeverría (CUJAE), 2010.
14. **Cabrera Aldaya, A.**: "Controlador Maestro basado en FPGA para un Sistema Inteligente de Transporte," Tesis de Diploma, Instituto Superior Politécnico José Antonio Echeverría (CUJAE), 2009.
15. **Xilinx Inc.**: "UG081 - MicroBlaze Processor Reference Guide (v12.0)." 2011.
16. **Williams, J.; Bergmann, N. and Xie, X.**: "FIFO Communication Models in Operating Systems for Reconfigurable Computing," in 13th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'05), 2005, pp. 0–1.
17. **Xilinx Inc.**: "DS499 - Fast Simplex Link v2.0 Bus Datasheet." pp. 1–9, 2010.
18. **Cabrera Aldaya, A. and Cabrera Sarmiento, A. J.**: "Implementación Híbrida Hardware/Software del Algoritmo Rijndael," in IV Simposio Internacional de Electrónica: diseño, aplicaciones, técnicas avanzadas y retos actuales, 2011.
19. **Xilinx Inc.**: "DS312-2 - Spartan3E FPGA Family: Functional Description." pp. 1–96, 2005.
20. **Daernen, J. and Rijmen, V.**: The Design of Rijndael. Springer, 2002.
21. **National Institute of Standards and Technlogy (NIST)**: "The Keyed-Hash Message Authentication Code." National Institute of Standards and Technlogy (NIST), 2002.
22. **Nugroho, A. E.**: "Nugroho Free Hash Cores," *OpenCores.org*, 2010. [Online]. Available: http://opencores.org/project,nfhc.
23. **Feilner, M.**: OpenVPN - Building and Integrating Virtual Private Networks. Packt Publishing Ltd, 2006.
24. **Corbet, J.; Rubini, A. and Kroah-Hartman, G.**: Linux Device Drivers, 3rd ed. O'Reilly Media Inc., 2005.
25. **Kerrisk, M.**: The Linux Programming Interface. No Starch Press, Inc., 2010.

AUTORES

Alejandro Cabrera Aldaya, Ingeniero en Automática, Máster en Sistemas Digitales. Complejo de Investigaciones Tecnológicas Integradas (CITI), acabrera@udio.cujae.edu.cu. Actualmente desarrolla su tema de investigación doctoral sobre implementación de algoritmos criptográficos mediante hardware reconfigurable.

Alejandro José Cabrera Sarmiento, Ingeniero Electricista, Doctor en Ciencias Técnicas. Profesor Titular, Departamento de Automática y Computación, Instituto Superior Politécnico "José Antonio Echeverría, Cuba, alex@electrica.cujae.edu.cu.