REVISTA DIGITAL DE LAS TECNOLOGÍAS DE LA INFORMACIÓN Y LAS COMUNICACIONES
TELEM@TICA
Revista Telem@tica. Vol. 13. No. 1, enero-abril, 2014, p. 46-62 ISSN 1729-3804

COMPONENTES DE CIFRADO SIMÉTRICO SOBRE MICROBLAZE BASADOS EN LOS ESTÁNDARES AES Y GOST

COMPONENTS OF SYMMETRIC CIPHER DEPLOY MICROBLAZE BASED AES AND GOST STANDART

Ing. Ernesto Gil Aranguren 1, MSc. Humberto Díaz Pando 2, Ing. Yosbel Martínez García 3
1 Complejo de Investigaciones Tecnológicas Integradas, Cuba, egil@udio.cujae.edu.cu, Puerta Cerrada #413 apto 20 entre Figuras y Chamorro
2 Facultad de Ingeniería Informática (ISPJAE), Cuba, hdiazp@ceis.cujae.edu.cu
3 Complejo de Investigaciones Tecnológicas Integradas, Cuba, ymartinez@udio.cujae.edu.cu

RESUMEN: Es un hecho que la PC es un entorno inseguro, que puede estar sujeto a muchos ataques de intrusos. Algunos de estos ataques podrían obtener la clave u otro elemento secreto, necesario para garantizar la confidencialidad de una información determinada mediante algún algoritmo criptográfico simétrico. Por lo que es necesario implementar componentes propios, externos a este entorno inseguro, que contengan las funcionalidades criptográficas.
En este trabajo se desarrollan dos componentes de cifrado simétricos sobre FPGA de Xilinx. Estos implementan los algoritmos AES y GOST sobre Microblaze. En una primera parte se exponen los aspectos básicos de la criptografía simétrica y el flujo de desarrollo de los sistemas embebidos. Se realiza un análisis de los algoritmos para observar el despliegue de los mismos sobre Microblaze obteniéndose una arquitectura básica inicial. En el proceso de validación de esta, se realizaron un conjunto de cambios que derivaron en una nueva arquitectura, la cual incluye un módulo hardware que apoya al componente AES en su ejecución.
Palabras Clave: FPGA, Criptografía simétrica, AES, GOST, Microblaze.

ABSTRACT: It is a fact that the PC is an unsafe environment, which may be subject to many attacks from intruders. Some of these attacks could obtain the secret key or other item necessary to ensure the confidentiality of certain information by a symmetric cryptographic algorithm. So it is necessary to implement your own components, external to this insecure environment, containing cryptographic functionality.
In this paper we develop two symmetric encryption components on FPGA from Xilinx. These implement the algorithms AES and GOST on Microblaze. The first part outlines the basics of symmetric cryptography and the flow of embedded systems development. Was performed an analysis of algorithms to monitor the deployment of the same on Microblaze, obtaining an initial basic architecture. In the validation process of this, there were a set of changes that led to a new architecture, which includes a hardware module that supports AES component in its execution.

Key Words: FPGA, Symmetric Cryptograph, AES, GOST, Microblaze.

1. INTRODUCCIÓN

La criptografía simétrica es un subconjunto de la criptografía que se encarga principalmente del objetivo confidencialidad, ya que intenta impedir que entidades no autorizadas puedan leer cierta información. Para poder garantizar este objetivo, los sistemas criptográficos simétricos utilizan un secreto compartido sólo por las entidades que se comunican, el que si es violado entonces se incrementan las probabilidades de un ataque.

El secreto compartido es, en muchos casos, la clave y además algunos otros elementos. Se pueden encontrar en la actualidad muchos algoritmos y muchas soluciones informáticas que los utilizan. Sin embargo, muchos de estos algoritmos son ejecutados sobre el ordenador. Precisamente, este hecho contribuye a una posible violación del secreto.

Es conocido que los sistemas operativos poseen brechas de seguridad, que por lo general son utilizadas por los atacantes para intentar descubrir los elementos secretos. Por otro lado es necesario cerciorarse de la legitimidad de las soluciones que soportan criptografía simétrica pues en muchas ocasiones no se puede acceder al código fuente, por lo que no se puede asegurar que una solución haga lo que se espera de ella.

En la actualidad se manejan algunas soluciones que impiden estos escenarios. Entre ellas se encuentra, la de utilizar implementaciones propias en dispositivos hardware externos. Estos dispositivos por lo general se pueden clasificar como Sistemas Embebidos y se denominan Token criptográficos.

Un sistema embebido se compone de un circuito integrado y de un procesador que ejecutara el software. Entre los tipos de circuitos integrados el que más se ajusta a las características de las soluciones criptográficas es el FPGA (Hardware Reconfigurable) ya que permite manejar tamaños de palabras por encima de los 64 bits, el tope del tamaño de palabra de un microprocesador.

Por otro lado el FPGA es flexible, lo cual es sustancialmente importante por el hecho de que los algoritmos criptográficos simétricos son actualizados, o pueden ser sustituidos por otros. Además, permite un diseño híbrido, lo que significa que además de un procesador se pueden agregar otros componentes hardware que apoyen la ejecución del software en el mismo.

Como plataforma de procesamiento se propone utilizar Microblaze, el cual es un procesador embebido de tipo RISC (Reduced Instruction Set Computer) optimizado para las FPGA de Xilinx. Este procesador es en realidad un soft-core, lo que permite trasladar el diseño entre familias de dispositivos sin problemas de compatibilidad y permite configurar los recursos utilizados en dependencia del diseño.

Problema a resolver: No se dispone de una implementación propia de los algoritmos AES y GOST para Microblaze

Objetivos:

- Desarrollar un componente propio para el algoritmo AES sobre FPGA orientado a co-diseño.
- Desarrollar un componente propio para el algoritmo GOST sobre Microblaze.

Objetivos específicos:

- Implementar el algoritmo AES sobre Microblaze.
- Implementar el algoritmo GOST sobre Microblaze.
- Validar la implementación de AES y GOST.

2. CONTENIDO

Los estándares de cifrado simétrico se encuentran definidos y explicados: en el caso del AES, en [1] y el GOST en [2]. No es objetivo de este trabajo profundizar en el funcionamiento de los algoritmos sino en lograr una implementación ajustada a las características de los sistemas embebidos. Sin embargo, vale destacar un trabajo publicado en [3], donde se hace referencia a una implementación de AES sobre FPGA utilizando tablas para reducir los cálculos propios del algoritmo. No obstante, esta fue una implementación hardware; de ahí que se tenga que confirmar su despliegue sobre Microblaze.

2.1 Sistemas Embebidos

Un Sistema Embebido (del inglés, Embedded System) se puede definir como un sistema que tiene empotrado o embebido tres componentes fundamentales: hardware para brindar funcionalidades similares a una computadora, software que ejecuta un grupo de tareas y por último, un sistema operativo de tiempo real el cual supervisa el software que se ejecuta en el hardware y organiza el acceso a los recursos del sistema de acuerdo a restricciones de prioridad y tiempo [4]. Aunque otro autor plantea que un Sistema Embebido es a grandes rasgos cualquier sistema de cómputo que no sea uno de escritorio, laptop o servidor [5].

Estos sistemas están atados a fuertes restricciones en su diseño, cuyo objetivo principal es mantener su bajo costo. Para lograr tal objetivo, los diseñadores de estos sistemas deben enfrentar un problema complejo y es que es necesario implementar el SE que optimice de forma simultánea varias métricas de diseño. Se define cómo métrica de diseño a una característica medible en la implementación de un SE[5]. Entre las métricas más comunes se encuentran: Costo por Unidad, Costo NRE, Tamaño, Rendimiento, Potencia, Flexibilidad, Tiempo de Mercado, Tiempo de prototipado, Seguridad [5].

2.1.1 Diseño basado en Plataforma

En los últimos años ha cobrado auge otro concepto vinculado a la mejora de los procesos de diseño e implementación de SE: este concepto es el de diseño basado en plataforma (de sus siglas en inglés PBD, Platform-Based Design). Este diseño provee al diseñador de un conjunto de "librerías de componentes y un marco arqui-tectural de trabajo consistente en un conjunto integrado de software y hardware, componentes virtuales, modelos, herramientas, librerías y metodología para soportar un desarrollo rápido a través de una arquitectura inicial, verificación, simulación e integración del sistema". Con este concepto, al igual que con los lenguajes, se pretende elevar el nivel de abstracción del diseñador con el objetivo de centrarse en el cumplimiento de las funcionalidades que se especifiquen [6].

Bajo este enfoque, el diseñador cuenta ya con una arquitectura inicial definida en la cual puede estar presente un procesador de propósito general, un sistema operativo de tiempo real (RTOS), periféricos, memoria y buses. Lo que queda por parte del diseñador es particularizar esta arquitectura inicial añadiendo procesadores de propósito específico, ya sea por el uso de bibliotecas IP o diseñándolos él

mismo, bloques lógicos dentro del IC o escribiendo el software a la medida que utilice los dispositivos que se incluye en el sistema para darles respuesta a las especificaciones capturadas [6].

El PBD es una metodología que se centra en el medio o "meet-in-the-middle", donde el punto de entrada es la propia plataforma o arquitectura inicial (Figura 1). En este punto el diseñador puede ir "hacia abajo" cuando tiene que diseñar nuevos periféricos o procesadores específicos; o puede ir "hacia arriba" cuando se escribe código software que utilice los componentes o la plataforma creada para la aplicación [6].

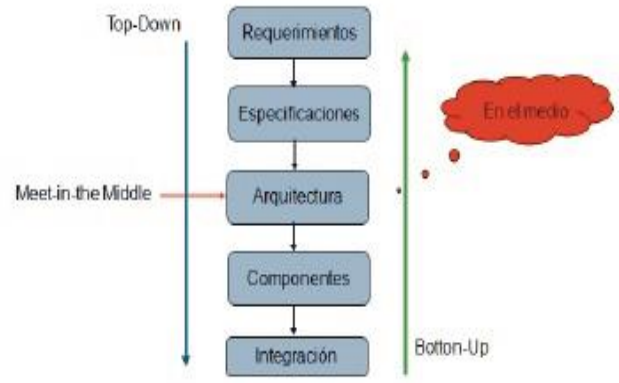


Figura 1. Diseño Basado en Plataforma, su ubicación en los niveles de abstracción.

La ventaja más importante de utilizar el PBD es la disminución del tiempo de desarrollo hasta poder crear un SoC (del inglés System on Chip)1 en pocos días. Permite además abstraer al diseñador de los detalles de la implementación permitiéndole centrarse en el desarrollo de la aplicación. Los detalles de la tecnología están ocultos y solo se puede particularizar la plataforma de referencia [6].

2.1.2 Co-diseño Hardware-Software

El co-diseño hardware-software no es más que el diseño en conjunto de componentes de hardware y software con el objetivo de garantizar las funcionalidades y restricciones del sistema, explotando la correlación entre el hardware y el software [7].

Existen varios trabajos donde se describe el flujo de actividades a seguir durante el proceso de co-diseño [7-9], aunque cabe señalar que el flujo varía de uno a otro en dependencia del objetivo que persiga el trabajo.

En [9] se especifica un flujo básico inicial, el cual está representado en la Figura 2. Como se puede apreciar este parte de una especificación del sistema y concluye con la verificación del diseño, pasando el particionado del sistema, la descripción del hardware y el software, hasta la síntesis (tanto del hardware como del software) y su integración.

El proceso de diseño comienza con una especificación del sistema, capturada en un modelo y/o lenguaje de especificación. Luego, el diseñador basado en su experiencia, en suposiciones de un análisis a priori del algoritmo y/o en los resultados obtenidos de un Profiling realizado al mismo, determina cuales partes del algoritmo se pudieran realizar en software y cuales parte en hardware. Una vez terminada la

síntesis, existen algunas plataformas que permiten la simulación de hardware y software en un entorno integrado. Sin embargo en ocasiones la simulación debe ser llevada de manera independiente. Por último, los resultados de la simulación deben estar acordes como con los requisitos funcionales y las restricciones definidas en la especificación inicial [9]

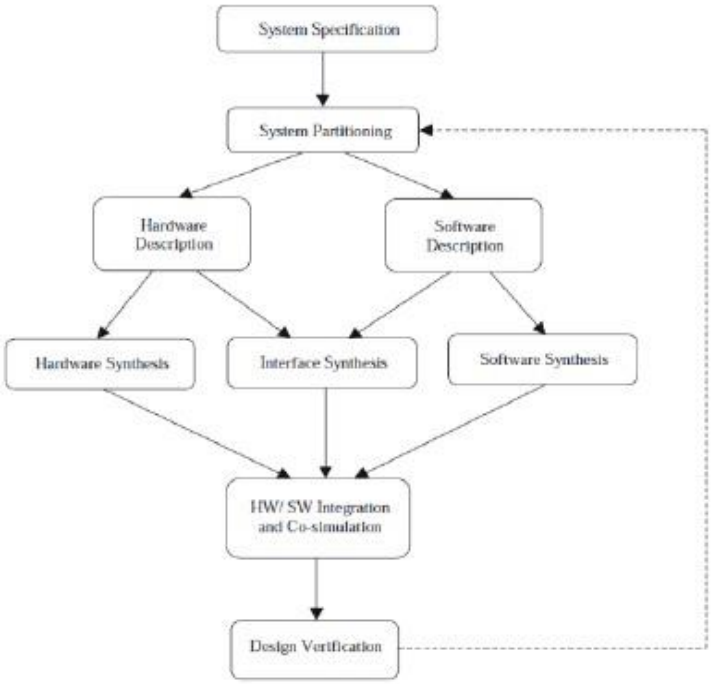


Figura 2. Flujo de trabajo del co-diseño.

Si no se logra cumplir con todos los requisitos, el proceso completo comienza a partir del particionado del sistema [9].

2.2 Partición Hardware-Software

Los análisis de los algoritmos previos para realizar el particionado hardware-software, vital para el dicho proceso, se encuentran descritos en [10].

En el ámbito del proceso de diseño y refinamiento del co-diseño, una de las etapas más importantes es la del particionado hardware-software. En esta etapa se analiza qué operaciones deberían ser implementadas en software y cuáles en hardware.

Para poder realizar el Profiling es necesario partir de una arquitectura básica de inicio (Figura 3). Esta arquitectura estará formada solamente por los elementos necesarios para ejecutar la aplicación software y será implementada sobre una FPGA de la familia Spartan-3E de Xilinx.

El núcleo de la arquitectura estará formado por Microblaze como microprocesador encargado de ejecutar la aplicación software, que cuenta con 32KB de memoria local. Para permitir la depuración y puesta a punto del software, es necesario incluir en la arquitectura un módulo de depuración (MDM) y un periférico serie para comunicarse con la computadora (RS232). Por último, para medir los tiempos en

el Profiling es necesario incluir un temporizador (Timer) en la arquitectura. Para interconectar todos los componentes de la arquitectura se empleó el bus PLB.

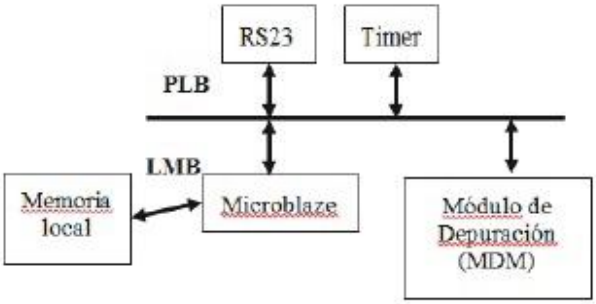


Figura 3. Arquitectura básica de partida.

Luego de realizar el proceso de Profiling, se obtiene un conjunto de datos interesantes detallados mejor en [10]. Tabla I se encuentran los resultados generales, donde se puede observar cómo en el caso del AES, las funciones de cifrado y descifrado significaron más del 50% del tiempo de ejecución, lo que no se comporta así con la implementación de GOST, pues posee un diseño sencillo que se ajusta muy bien a las características de Microblaze.

Tabla I. Resultados generales del Profiling.

Algoritmo	Función	% de tiempo de ejecución
AES-Cifrado	rijndaelEncrypt	60.4
AES-Descifrado	rijndaelDecrypt	50.5
GOST	GOST-RawCrypt	14.53

En la búsqueda de una implementación de los algoritmos que se desplegará mejor sobre Microblaze, se utilizó una técnica software conocida como loop unrolling. Esta técnica se basa en la reducción de las iteraciones dentro de la ejecución de algoritmos, puesto que los ambos algoritmos presentan n rondas, y las mismas se implementan en forma de iteraciones. Sin embargo en ocasiones es mejor realizar estas rondas sin bloques de iteraciones. Esto consiste en hacer más extenso un bucle simple reduciendo la sobrecarga por comparación del bucle y permitiendo una mejor concurrencia [11, 12].

Tabla II. Resumen de los experimentos realizados.

Experimento	Config. de hardware	Tipo de AES	Tipo de optimización	Cifrado con *loop unrolling*
AES-1	Inicial	Estándar	Estándar	Si
AES-2	Inicial	Estándar	Estándar	No
AES-3	Inicial	Estándar	Velocidad	Si
AES-4	Inicial	Estándar	Velocidad	No
AES-5	Inicial	Por tablas	Tamaño	Si
AES-6	Inicial	Por tablas	Tamaño	No
GOST-1	Inicial	-	Estándar	Si
GOST-2	Inicial	-	Estándar	No
GOST-3	Inicial	-	Velocidad	Si
AES-7	Barrel Shifter	Por tablas	Tamaño	Si
AES-8	Barrel Shifter	Por tablas	Tamaño	No
AES-9	Barrel shifter y co-procesador AES	Utilizando co-procesador AES	Estándar	-
AES-10	Barrel shifter y co-procesador AES	Utilizando co-procesador AES	Velocidad	-

2.3 Experimentos y Resultados

Los resultados de los experimentos descritos en este epígrafe se expresan teniendo en cuenta las tres métricas principales el área ocupada por el diseño, la cantidad de ciclos de reloj y el factor área x tiempo, este último representado por la cantidad de ciclos.

Para validar la arquitectura se desarrollaron 13 experimentos diferentes, agrupados en tres configuraciones de hardware diferentes y por el algoritmo al que pertenecen.

Por otro lado, se realizó una serie de experimentos previos con el objetivo de constatar si la propuesta hecha por [3] para un diseño de hardware es factible en una implementación software además de verificar la influencia de los parámetros de compilación en la ejecución de las diferentes variantes.

Para todos los experimentos del componente AES, se utilizaron tamaños de clave de 256-bits y en modo CBC, lo que implicó, también la utilización de un vector de inicialización (IV), además del texto plano. Los experimentos del componente GOST tuvieron como parámetros claves estándar de 256 bits y en modo ECB, por lo que no fue necesario ningún otro parámetro que el texto plano inicial.

En la Tabla II se puede apreciar cada una de las configuraciones de hardware y parámetros de compilación de cada uno de los experimentos agrupados por el algoritmo correspondiente.

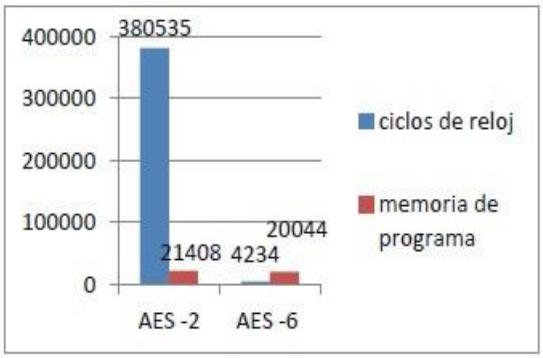
La configuración inicial de hardware se refiere a la arquitectura primaria propuesta, en la que se encuentran el procesador Microblaze, el bus PLB y el puerto de salida RS232 como elementos

principales. En el caso de la Configuración #2, es casi la misma que la inicial sólo que se configura además el Barrel Shifter, un recurso hardware que realiza desplazamientos de bits en un solo ciclo de reloj. Para la Configuración #3, se utilizan todos los elementos anteriores, y además se agrega un módulo hardware que realiza el proceso de cifrado y descifrado de AES. Este módulo hardware, se conecta a Microblaze, mediante el bus FSL, de ahí que ese puerto también se agrega a esta configuración.

En general se observa un creciente consumo de slices, al punto de llegar casi al 45% de la disponibilidad de este modelo de FPGA, lo cual debe tenerse en cuenta pues este componente pudiera no estar aislado y este consumo podría implicar problemas para integrarlo con otros.

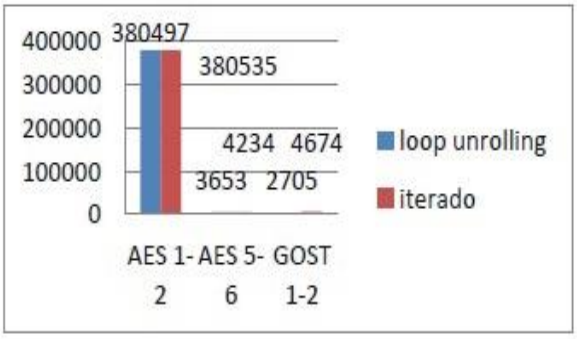
2.3.1 Configuración Inicial

Los resultados mostrados en la Fígura 4 se obtuvieron de la ejecución del proceso de cifrado de AES en su implementación estándar [1] y luego en la utilización de tablas. En dicho gráfico se puede apreciar una disminución sustancial de la cantidad de ciclos de reloj, aunque el experimento AES-6 fue realizado bajo una compilación optimizada para reducir el tamaño, debido a que con una compilación normal el tamaño del programa excedía los 32 KB de Microblaze.

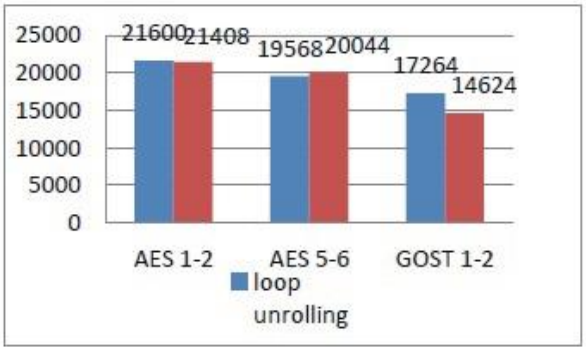


Fígura 4. Comparación entre los experimentos del tipo software: AES-2 y AES-6.

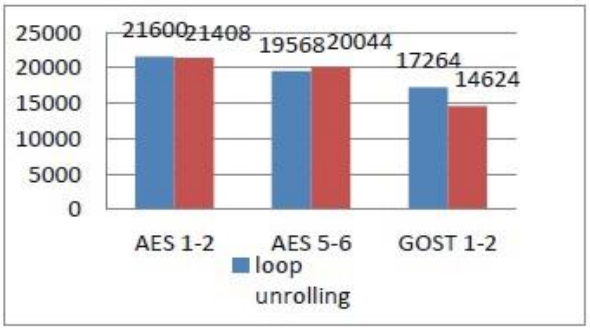
Otros resultados se obtuvieron de comparar la implementación de los algoritmos utilizando loop unrrolling contra la variante que implementa bucles. En caso de utilizar la primera variante se obtiene una mejora mínima de los tiempos de cifrado, debido a que Microblaze contiene un pipeline de cinco etapas, a expensas de un incremento del tamaño del programa. Sin embargo, en la segunda variante el procesador debe vaciar el pipeline cada vez que encuentra un salto, como cuando se termina una iteración en un bucle.



Fígura 5. Comparación en ciclos de reloj entre las diferentes variantes de los algoritmos.



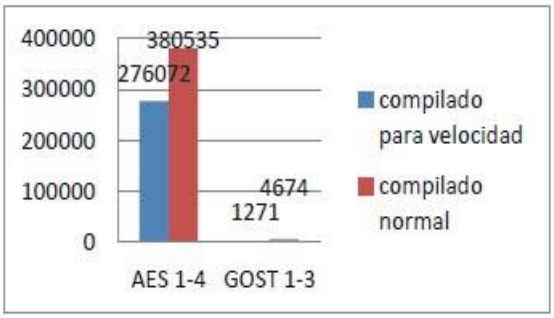
Fígura 6 Resultados de la cantidad de ciclos de reloj y la memoria de programa de cada experimento involucrado.



Fígura 7. Memoria de programa utilizada por cada experimento.

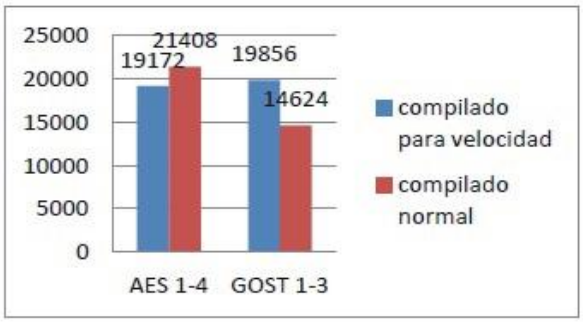
En los dos gráficos anteriores se logró apreciar la diferencia en cuanto a ciclos de reloj y memoria de programa de ambas variantes. Sin embargo, no tienen significación desde el punto de vista práctico, pues la mayor reducción en ciclos de reloj es la del GOST con 1969 ciclos, lo que representa el 0,004 % de los 50 MHz (velocidad del reloj) que posee Microblaze. Además, si este componente debe unirse a otros para complementar funcionalidades criptográficas, el mismo debe ser lo más compacto posible, por lo que no es práctica esta propuesta.

Los experimentos realizados con optimización para velocidad tenían la intención de ver hasta donde era capaz de obtenerse una implementación que consumiera menos ciclos de reloj (Fígura 8, Fígura 9), claramente a expensas de un aumento considerable de la memoria de programa en algunos casos.



Fígura 8. Cantidad de ciclos de reloj de cada experimento, utilizando optimización para velocidad y compilación normal.

Como se puede apreciar en estos valores (Fígura 8, Fígura 9) es posible mejorar la cantidad de ciclos de reloj de las implementaciones con esta variante de optimización pero siempre aumentando el uso de la memoria del programa, pues en ocasiones este tipo de optimización produce programas con un tamaño que excede los 32 KB disponibles en el sistema.

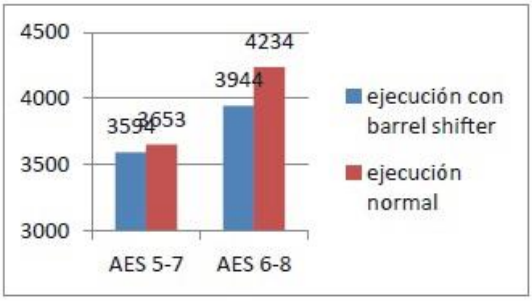


Fígura 9. Uso de la memoria de programa de los experimentos compilados para velocidad y compilados de manera estándar.

2.3.2 Configuración #2

Los resultados de los experimentos de esta configuración se basan en ejecutar aquellas variantes de los algoritmos que requieran gran cantidad de desplazamientos y compararlos con los resultados en la ejecución inicial.

La Fígura 10 muestra cómo el uso del Barrel Shifter mejora aún más la variante basada en tablas, obteniéndose mejores resultados en cuanto a los ciclos de reloj. Una vez más se pretende ver cómo influye la ejecución secuencial y la ejecución iterativa del proceso de rondas, aunque como se puede constatar la diferencia sigue siendo casi insignificante. Por otro lado se puede adelantar que el uso de componentes hardware externos apoyando las operaciones criptográficas es una de las ventajas de las FPGA más explotadas pues permite obtener tiempos de respuesta similares a los de una PC.



Fígura 10. Muestra la diferencia entre la ejecución de las variantes software de AES, utilizando o no el barrell shifter.

2.3.3 Arquitectura Hardware-Software

Luego de ejecutado un conjunto de experimentos sobre la arquitectura inicial, se realiza una modificación a la misma, como parte de la nueva configuración de hardware para la siguiente serie de experimentos.

En la Figura 11 está representada la nueva arquitectura. En esta se encuentra el microprocesador que será el encargado de ejecutar las instrucciones software perteneciente a los algoritmos AES y GOST y que está conectado por el bus MLB a la memoria local del mismo. El co-procesador AES es el encargado de las funcionalidades de rijndaelEncrypt y rijndaelDecrypt, debido a que se demostró en el Profiling que estas eran las más costosas. Este coprocesador está conectado mediante el bus FSL con Microblaze, a acusa de la rapidez del mismo. El módulo de entrada/salida hace posible la comunicación entre el sistema y el exterior a través de una interfaz serie RS232 y se comunica con Microblaze mediante el bus PLB. En el caso de GOST, no fue necesario implementar ningún co-procesador hardware.

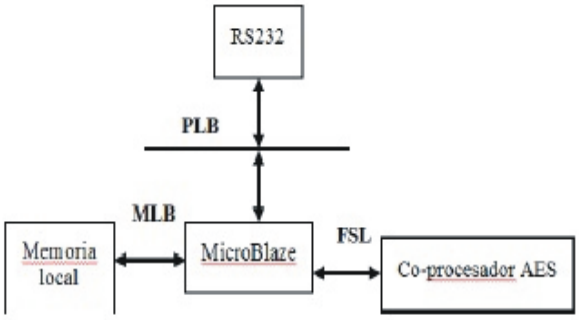
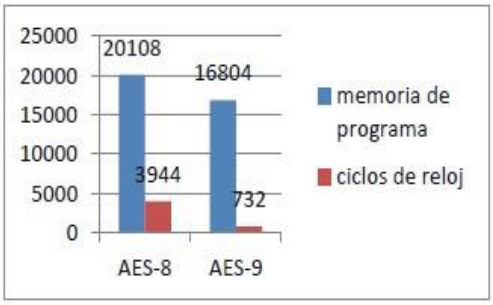


Figura 11. Arquitectura propuesta para el sistema embebido.

2.3.4 Configuración #3

La Configuración #3 incluye a la configuración inicial, el Barrell Shifter y además utiliza un co-procesador (recurso hardware)[13] que tiene como tarea realizar el proceso de cifrado y descifrado de AES.



Fígura 12. Diferencias entre los experimentos que utilizan el barrell shifter y el co-procesador AES

La Gráfico 12 muestra la disminución en la cantidad de ciclos de reloj y la memoria del programa cuando se utiliza el co-procesador AES. La diferencia en el consumo de memoria no es significativa debido a que el experimento AES-8 fue compilado con optimización de tamaño y el AES-9 no. En el caso de la cantidad de ciclos, se puede apreciar la caída brusca de los valores, lo cual confirma las sospechas de que un diseño hibrido de AES resulta beneficioso.

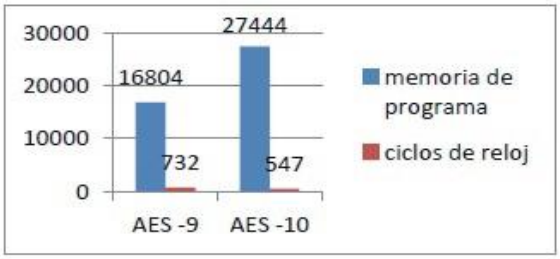


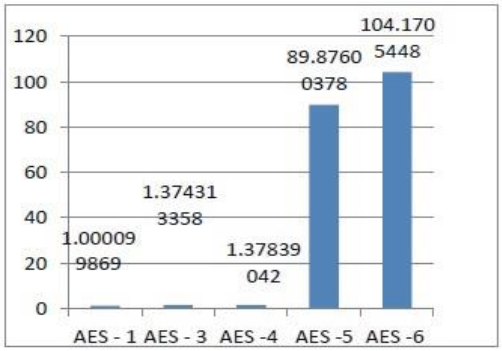
Gráfico 8. Muestra las diferencias entre los experimentos que utilizan compilado normal y compilado para velocidad.

La Fígura 13 muestra la comparación entre AES-9 y AES-10 observándose como la variante optimizada para velocidad incrementa sustancialmente el uso de la memoria de programa del experimento AES-10. Por otra parte se puede apreciar también una disminución en la cantidad de ciclos de reloj, sin embargo esta reducción, que es del 30%, implica consumir cerca del 86% de la memoria disponible. Esta situación pudiera no ser la mejor pues, como se ha explicado anteriormente, este componente pudiera estar junto a otros, por lo que debe existir un equilibrio entre memoria de programa y ciclos de reloj.

2.3.5 Resultados finales

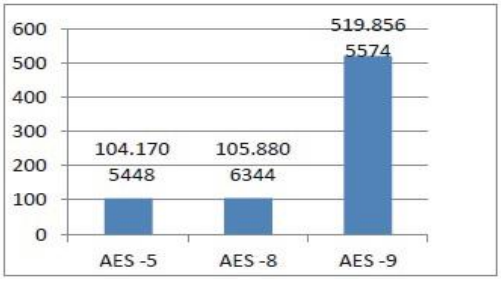
Teniendo en cuenta el conjunto de experimentos realizados sería interesante observar algunas métricas globales donde se comparen los experimentos de cada configuración de hardware.

Una métrica sería la aceleración de los ciclos de reloj, la cual se define como la división entre la cantidad de ciclos del experimento más lento, con el resto. En la Fígura 14 se pude apreciar que solamente con la variante de AES por ta-blas rotadas se aceleró en 104 unidades con respecto al AES-2, el cual es la variante más lenta.



Fígura 14. Aceleración de los ciclos de reloj para los experimentos software

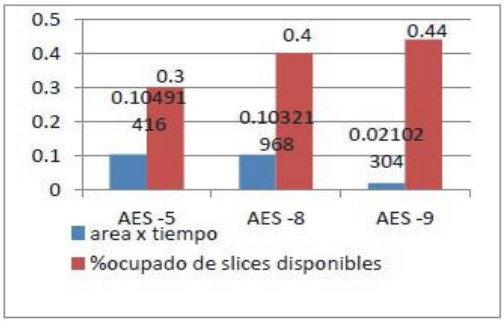
Por otro lado en la Fígura 15, se muestran los experimentos con mejor aceleración por configuración de hardware, donde puede demostrarse que el uso del co-procesador AES ha sido determinante en cuanto a lograr una aceleración final de 520 unidades.



Fígura 15. Aceleración de los ciclos de reloj de los mejores experimentos de cada configuración hardware.

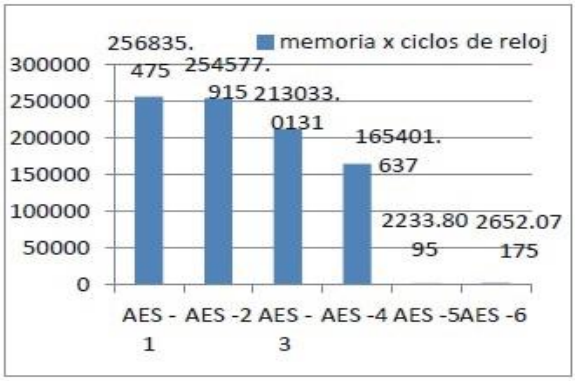
Otros resultados obtenidos fueron el cálculo del área x tiempo para los experimentos de cada configuración hardware. Se realizó para el algoritmo AES pues es el que posee el co-procesador hardware. El área, como se sabe es la cantidad de slices ocupados y el tiempo se resuelve al dividir la cantidad de ciclos de reloj entre la frecuencia de Microblaze (50 Mhz).

En la Fígura 16 demuestra cómo se puede sacrificar área de la FPGA para ganar en velocidad de la aplicación. En el caso de AES-5 y AES-8 los valores son parecidos. Pese a que el uso del Barrell Shifter disminuyó la cantidad de ciclos, esta disminución no fue significativa a favor del aumento de slices utilizados. Sin embargo, aun cuando el aumento de la cantidad de slices es solo del 4% para el experimento AES-9, se puede observar una caída brusca de la métrica área x tiempo. La principal razón es la enorme aceleración que se logra con el co-procesador AES.



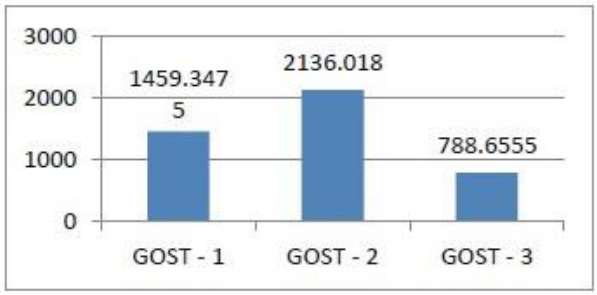
Fígura 16. Comparación entre el área x tiempo y la cantidad de slices ocupados de los experimentos de AES.

Algunos valores interesantes se pudieran obtener de multiplicar el porcentaje de memoria de Microblaze consumida por cada experimento y la cantidad de ciclos de reloj de cada uno (Fígura 17, Fígura 18). Esto se realiza con el objetivo de ver cuáles de los experimentos consumen menos memoria y al mismo tiempo poseen una buena velocidad de implementación. De esta manera se puede tener una noción de hasta qué punto perder memoria para ganar en velocidad y viceversa.



Fígura 17. Por ciento de memoria consumida x ciclos de reloj de los experimentos de AES.

En la Figura 17 se puede apreciar como solamente en software, el uso de la variante de tablas con rotaciones junto a la optimización por área hace que la métrica anterior sea mejor para el experimento AES-5. La caída en los valores se debe en su mayor parte a la disminución de la cantidad de ciclos de reloj, que es donde esta variante hace la diferencia con respecto a la variante inicial.



Fígura 18. Por ciento de memoria consumida x ciclos de reloj de los experimentos de GOST.

En el caso del GOST (Fígura 18) a pesar de su relativa sencillez, se puede apreciar como el experimento GOST-3 es el de mejores resulta dos. Esta situación es producto de la baja complejidad de sus operaciones, lo que implica que cuando se optimiza para velocidad, se reduce drásticamente la cantidad de ciclos de reloj en casi el 60%; pero aumenta notablemente la memoria de programa, indicando que una implementación sobre hardware al igual que en el AES, sería lo más lógico y reduciría de paso la cantidad de memoria consumida.

CONCLUSIONES

A partir de los resultados de este trabajo se arriban a las siguientes conclusiones:

- Se logró implementar y validar el componente propio de AES sobre Microblaze, utilizando un co-procesador para las funcionalidades de cifrado y descifrado.

- Se logró implementar y validar el componente propio de GOST sobre Microblaze, en el caso de este algoritmo se determinó no utilizar nada más que la implementación sobre software.

- La utilización de la tecnología de FPGA ciertamente permite la flexibilidad de incluir co-procesadores que apoyen la ejecución de algoritmos criptográficos.

- El diseño hibrido de la solución demostró que se puede llegar a un compromiso entre el área de la FPGA utilizada y los tiempos de respuesta de los algoritmos.

- Existe una mejoría en los tiempos de respuesta en las variantes de loop unrolling con respecto a las iteradas, aunque la disminución no llega a ser significativa, y que propicia además un incremento en la memoria de programa.

- Los parámetros de compilación ciertamente pueden afectar tanto los ciclos de reloj, como la memoria de programa.

REFERENCIAS

1. FIPS-197, F.I.P.S.P., Announcing the ADVANCED ENCRYPTION STANDARD (AES). 2001.

2. Science, D.o.C. and U.o. Wollongong, Soviet Encryption Algorithm (1994).

3. Ga¨el Rouvroy, F.o.-X.S., Jean-Jacques Quisquater and Jean-Didier Legat Compact and Eficient Encryption/Decryption Module for FPGA Implementation of the AES Rijndael VeryWell Suited for Small Embedded Applications.

4. Kamal, R. Embedded Systems (2008).

5. F. Vahid, T.G., Embedded System Design: A UnifiedHardware/Software Approach. 1999, Department of Computer Science and Engineering.

6. Vincentelli, A.S. Platform-based Design.

7. Shaout, A., A.H. El-Mousa, and K. Mattar, Specification and Modeling of HW/SW CO-Design for Heterogeneous Embedded Systems. 2009.

8. Kocher, P., Security as a New Dimension in Embedded System Design. 2004.

9. Adhipathi, P., Model based approach to Hardware/Software Partitioning of SOC Designs. 2004.

10. Gil, E., COMPONENTES DE CIFRADO SIMÉTRICO SOBRE MICROBLAZE BASADOS EN LOS ESTÁNDARES AES Y GOST. 2012, Instituto Superior Politécnico José Antonio Echeverría: La Habana. p. 65.

11. Corporation, N. Compiler Optimizations. 2012 [cited 2012 6 de Junio]; Available from: http://www.compileroptimizations.com/category/loop_unrolling.htm.

12. ibiblio.org. CODE OPTIMIZATION - USER TECHNIQUES. 2012 [cited 2012 6 de Junio]; Available from: http://www.ibiblio.org/pub/languages/fortran/ch1-9.html.

13. Cabrera, A., Controlador Maestro basado en FPGA para un Sistema Inteligente de Transporte. 2009.