IMPLEMENTACIÓN DEL ALGORITMO DE CIFRADO TRIVIUM EN UN SISTEMA EMBEBIDO

AN IMPLEMENTATION OF THE TRIVIUM ENCRYPTION ALGORITHM IN AN EMBEDDED SYSTEM

Ishai Gun Roffe
Universidad Autónoma Metropolitana
Ishaigun@gmail.com

Oscar Alvarado-Nava
Universidad Autónoma Metropolitana
oan@azc.uam.mx

Eduardo Rodríguez Martínez
Universidad Autónoma Metropolitana
erm@azc.uam.mx

Andrés Ferreyra Ramírez
Universidad Autónoma Metropolitana
fra@azc.uam.mx

Resumen

En el presente trabajo se muestra la implementación de un sistema embebido capaz de cifrar datos a través del algoritmo TRIVIUM utilizando llaves creadas con criptografía de curvas elípticas. El sistema de cifrado fue implementado bajo el codiseño hardware-software donde las partes computacionalmente costosas fueron llevadas a módulos hardware y agregados a un sistema de cómputo tradicional CPU-RAM, donde un programa convencional se encarga de orquestar el trabajo. Los resultados obtenidos muestran que es posible mejorar el rendimiento de una aplicación al utilizar el codiseño hardware-software y además se pueden tener otros beneficios al utilizar tecnologías de implementación que integran todos los módulos del sistema en un único circuito integrado, como un bajo consumo de energía, características ideales para sistemas inalámbricos, móviles o portátiles.

Palabras Claves: Criptografía, sistemas reconfigrables, sistemas embebidos.

Abstract

The present work shows the implementation of an embedded system able to encrypt data using the TRIVIUM algorithm, with keys generated by means of elliptic curves. The system was designed using hardware-software codesing principles, where computationally expensive modules were implemented in hardware and added to a traditional computing system (CPU-RAM). A conventional program running on the computing system is responsible for orchestrating the work between hardware modules and the CPU. Obtained results show it is possible to improve an application performance by using hardware-software codesign. Additionally, several other benefits can be gained by integrating such solution in a system-on-a-chip such as low power consumption, ideal to wireless and portable systems.

Keywords: Cryptography, embedded systems, system on chip.

1. Introducción

En la actualidad, la gran cantidad de información intercambiada entre varios dispositivos, así como el crecimiento de Internet, exigen el desarrollo de dispositivos que garanticen comunicaciones seguras, evitando que agentes malintencionados utilicen datos confidenciales. De hecho, la seguridad de la información es uno de los principales desafíos a abordar en el campo de la IT (*Information technology*). Debido a las fuertes limitaciones de recursos en algunas aplicaciones IT, se han propuesto algoritmos criptográficos que ofrecen la llamada criptografía ligera [Maimut, 2012]. El cifrado simétrico ligero se está generalizando cada vez más en aplicaciones de bajo consumo de energía como los sistemas portátiles e inalámbricos. Trivium es uno de los algoritmos de cifrado de flujo liviano con un perfil de hardware preseleccionado para el proyecto eSTREAM. Ha habido varias mejoras de Trivium con técnicas de paralelización [Wang, 2015] tanto para mejorar el tiempo de ejecución del cifrado como para reducir el consumo dinámico de energía.

Por otro lado, a medida que avanza la tecnología de sistemas en un chip (SOC, System On Chip), cada día es posible integrar más elementos de procesamiento en un solo circuito integrado, incluidos lógica reprogramable, varios núcleos de procesador, celdas de memoria y elementos de procesamiento analógicos. Ahora

los diseñadores de aplicaciones en SOC deben considerar no solo el diseño de hardware sino también el diseño de software. Un punto destacado de la tecnología de diseño de SOC ha sido el constante decremento del consumo de energía, mediante la reducción del tamaño de los elementos reconfigurables y de procesamiento, y la consecuente disminución del voltaje operativo, al tiempo que se mantiene el rendimiento del sistema [Pu, 2017][Wang, 2015][ul Haq, 2009].

Proceso de cifrado

El cifrado o encriptación es el proceso reversible de alterar un mensaje mediante un algoritmo y una clave o llave, para que de esta forma sea ilegible. Para recuperar el mensaje original, se debe utilizar el mismo algoritmo y la clave correspondiente. De esta manera se puede tener un control sobre quién puede tener acceso a la información que se quiere mantener en secreto. Actualmente estos sistemas son muy utilizados para transmisión de mensajes en una red no segura.

Existen dos tipos de cifrado:

- Con el cifrado simétrico se necesita la misma llave tanto para cifrar como para descifrar.
- En el cifrado de llave pública, se utiliza una llave de libre acceso “pública” para cifrar la información y solo el que recibe la información cifrada puede descifrarla con una llave “privada”, las llaves son diferentes.

Existen también maneras seguras de establecer una llave simétrica a través de un canal no seguro, una de ellas es el método de Diffie-Hellman, donde intercambiando datos públicos y realizando operaciones con datos secretos, dos nodos pueden llegar a calcular una clave simétrica secreta para intercambiar datos cifrados mediante un algoritmo de cifrado simétrico de manera segura.

Criptografía de curva elíptica

Uno de los principales problemas que tienen los algoritmos criptográficos asimétricos es la longitud de las llaves, estas generalmente son muy largas para poder ofrecer una buena seguridad, por ejemplo, algoritmos de amplio uso como el

RSA pueden llegar hasta los 1000 bits. Trabajar con cifras tan grandes puede ser computacionalmente costoso para procesadores con arquitectura de 32 o 64 bits y aún más para soluciones embebidas.

Para este proyecto se decidió utilizar criptografía de curva elíptica ECC (*Elliptic curve cryptography*) para realizar la resolución de las llaves ya que este nos ofrece la misma seguridad que los algoritmos anteriores usando datos de longitud mucho menor y aun así nos ofrece la misma seguridad [Fournaris, 2015]. Las curvas elípticas son polinomios con la forma mostrada en la ecuación 1, la cual genera una curva como la mostrada en la figura 1.

$$y^2 = x^3 + ax + b \quad donde \quad a, b \in R \tag{1}$$

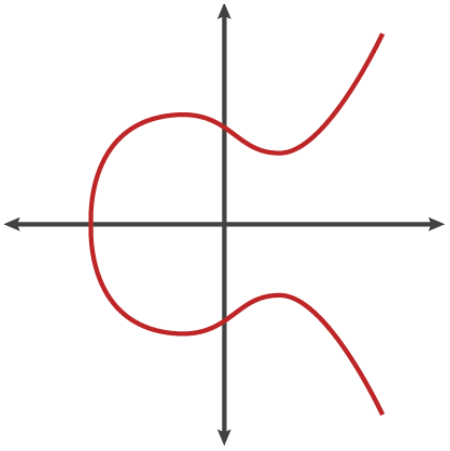

Figura 1 Curva elíptica.

Para poder ser utilizadas en criptografía, las curvas deben ser calculadas en módulo p (donde p es un número primo), la curva elíptica definida en $Z_p = 0,1,2,\cdots,p-1$, para $p < 3$, es el conjunto de todos los pares $(x, y) \in Z_p$ que cumplan con $y^2 = x^3 + ax + b$ mod p.

Tomando en cuenta un punto infinito θ que cumple con la condición $4a^3 + 27b^2 \neq \theta$ mod p, existen dos tipos de operaciones que se pueden realizar con dos puntos que pertenecen a la curva, la suma y la multiplicación. Dado un punto $P = (x, y)$ en la curva elíptica, las operaciones aritméticas cumplen la propiedad de simetría expresada como el conjunto de identidades:

- $-P = (x, -y)$

- $P + \theta = P$
- $P + P = 2P$.

La operación de suma se realiza de la siguiente manera. Sean $P = (x_1, y_1), Q = (x_2, y_2)$, y $S = (x_3, y_3)$ puntos sobre la curva elíptica, con $P \neq Q$. Si $S = P + Q$, se cumplen las siguientes identidades:

- $S = \frac{y_2 - y_1}{x_2 - x_1} \bmod p$
- $x_3 = S^2 - x_1 - x_2 \bmod p$
- $y_3 = S(x_1 - x_3) - y_1 \bmod p$

Mientras que las operaciones de multiplicación se realizan. Sean $P = (x_1, y_1), Q = (x_2, y_2)$, y $S = (x_3, y_3)$ puntos sobre la curva elíptica, con $P = Q$. Si $S = P \times Q = 2P$, se cumplen las siguientes identidades:

- $S = \frac{3x_1^2 + a}{2y_1} \bmod p$
- $x_3 = S^2 - x_1 - x_2 \bmod p$
- $y_3 = S(x_1 - x_3) - y_1 \bmod p$

Los puntos que forman parte de una curva elíptica junto con el elemento θ crean grupos cíclicos al realizar la operación de multiplicación a un punto de la curva; al multiplicar un punto por sí mismo las suficientes veces da por resultado el elemento θ, $dP = \theta$. El número exacto de puntos dentro de una curva elíptica E ($|E|$) está dado por el Teorema de Hasse: $P + 1 - 2\sqrt{P} \leq |E| \leq P + 1 + 2\sqrt{2}$.
Podemos ver que $|E|$ tiene un valor cercano a P. El algoritmo de la curva elíptica basa su seguridad en la enorme dificultad para resolver el siguiente problema: $P + P + \cdots + P = dP = T$.

Resolución de llaves

El algoritmo de la curva elíptica que se utilizará es en particular el protocolo Diffie-Hellman para la resolución de llaves, esto nos permite conseguir una llave secreta sin que tenga que ser enviada del emisor al receptor.

El protocolo funciona de la siguiente manera: Dados una curva elíptica *E*, un número primo p y un punto $P = (x_p, y_p)$, todos datos públicos:

- El Emisor escoge cualquier entero a tal que $a \in \{2,3,4,\cdots,|E|-1\}$.
- El Emisor calcula $A = aP = (x_A, y_A)$ y lo envía al Receptor.
- El Receptor escoge cualquier entero b tal que $b \in \{2,3,4,\cdots,|E|-1\}$.
- El Receptor calcula $B = bP = (x_B, y_B)$ y lo envía al Emisor.
- El Emisor calcula $aB = T_{Emisor}$.
- El Receptor calcula $bA = T_{Receptor}$.

Al final podremos observar que $T_{Emisor} = T_{Receptor}$, por lo que cualquier componente de ambos vectores puede ser utilizada como llave secreta.

Algoritmo TRIVIUM

TRIVIUM es un algoritmo de encriptación simétrica creado por Christophe De Canniere y Bart Preneel para el proyecto eSTREAM el cual se llevó a cabo entre los años 2004 y 2008. El algoritmo TRIVIUM está diseñado especialmente para ser implementado fácilmente en hardware [2], es altamente paralelizable y de licencia libre, por esos motivos fue seleccionado para este proyecto.

TRIVIUM es un algoritmo de cifrado de tipo *Stream Cipher*, esto quiere decir que es capaz de generar un *keystream* de tamaño 2^{64} bits a partir de una llave privada de 80 bits y de un vector inicial de 80 bits. El algoritmo de cifrado se divide en dos partes; la inicialización del estado interno y la generación del *keystream*. El estado interno está formado por 288 bits.

El proceso de inicialización del estado interno *s* consiste en los siguientes pasos:

- Se copia la llave secreta *K* a las primeras 80 localidades del estado interno y se asigna cero a las localidades de 81 a 93:$(s_1, s_2, s_3, \cdots, s_{93}) \leftarrow (K_1, K_2, \cdots, K_{80}, 0, \cdots, 0)$.
- Se copia el vector inicial *IV* a las localidades 94 a 173 del estado interno y se asigna cero a las localidades de 174 a 177: $(s_{94}, s_{95}, s_{96}, \cdots, s_{177}) \leftarrow (IV_1, IV_2, \cdots, IV_{80}, 0, \cdots, 0)$.

- Se asigna cero a las localidades 178 a 285 y uno a las localidades 286 a 288 del estado interno:$(s_{178}, s_{179}, s_{180}, \cdots, s_{288}) \leftarrow (0, \cdots, 0,1,1,1)$

El proceso de generación de llave se hace una vez inicializado el estado interno *s* y se realiza con el algoritmo mostrado en la figura 2, donde la operación $a + b$ equivale a la operación lógica OR, y la operación $a \cdot b$ equivale a la operación lógica AND.

```
01 for i=1 to N do
02   t1 ← s66 + s93
03   t2 ← s162 + s177
04   t3 ← s243 + s288
05   zi ← t1 + t2 + t3
06   t1 ← t1 + s91·s92 + s171
07   t2 ← t2 + s175·s176 + s264
08   t3 ← t3 + s286·s287 + s69
09   (s1, s2, s3, s4, …, s93)←(t3, s1, s2, s3, …, s92)
10   (s94, s95, s96, s97, …, s177)←(t1, s94, s95, s96, …, s176)
11   (s178, s179, s180, s181, …, s288)← (t2, s178, s179, s180, …, s287)
12 end for
```

Figura 2 Generación de llave para keystream.

Al finalizar cada iteración, se obtiene bit del *keystream* al cual se le aplica la operación lógica OR-Exclusiva para el cifrado del texto.

Posteriormente se ejecuta el proceso de generación de *keystream* 1152 veces y se descartan los bits generados, una vez finalizado el proceso el estado interno ya se encuentra listo para generar información válida.

Algoritmo TRIVIUM paralelo

Por lo general los algoritmos de tipo *Stream Cipher* solo pueden generar un bit por ciclo, pero como se ha comentado, una de las ventajas del algoritmo de cifrado TRIVIUM es la posibilidad de paralelizar sus operaciones, por lo que es capaz de generar hasta 64 bits por ciclo.

Al analizar el algoritmo en la figura 2 se puede observar que en cada ciclo de generación del *keystream* se generan también tres valores *t,* en las líneas 02, 03, y 04, los cuales se utilizan tanto para generar el bit del *keystream* como para el estado inicial.

En estas primeras tres t se tienen por lo menos 64 bits del estado interno que no se utilizan antes de los dos valores que se requieren para calcular la respectiva t y que solamente se van recorriendo cada ciclo, por ejemplo de la s_1 a la s_{66} para la t_1, se podrían aprovechar y generar más de un trio *t* por ciclo. Por ejemplo, para generar ocho bits por ciclo se debe generar ocho tríos t:

$$t_{j,k} = \{tj, 1, tj, 2, tj, 3, tj, 4, tj, 5, tj, 6, tj, 7, tj, 8\}, \qquad j\{1,2,3\}$$

Como se muestra en la ecuación 2.

$$\begin{gathered} t1, k\ s66 - k\ +\ s93 - k \\ t2, k\ s162 - k\ +\ s177 - k \\ t3, k\ \leftarrow\ s243 - k\ +\ s288 - k\ para\ k \leftarrow \{1,2,3,4,5,6,7,8\} \end{gathered} \tag{2}$$

De esta forma se generan ocho tríos t por ciclo, por lo tanto se pueden generar ocho bits de *keystream* $z_{j,k}$ los cuales son utilizados para cifrar un byte del mensaje. Al terminar de generar el *keystream* también se debe generar los t que se introducirán al estado interno y deben ser ocho tríos también, como se muestra en ecuación 3.

$$\begin{gathered} t_{1,k}\ \leftarrow\ t_{1,k}\ +\ s_{91} - k \cdot s_{92} - k\ +\ s_{171} - k \\ t_{2,k}\ \leftarrow\ t_{2,k}\ +\ s_{175} - k \cdot s_{176} - k\ +\ s_{264} - k \\ t_{3,k}\ \leftarrow\ t_{3,k}\ +\ s_{286} - k \cdot s_{287} - k\ +\ s_{69} - k \\ s_1, s_2, s_3, s_4, \dots, s_{93} \leftarrow t_{3,1}, t_{3,2}, \dots, t_{3,8}, s_1, s_2, s_3, \dots, s_{85} \\ s_{94}, s_{95}, s_{96}, s_{97}, \dots, s_{177} \leftarrow t_{1,1}, t_{1,2}, \dots, t_{1,8}, s_{94}, s_{95}, s_{96}, \dots, s_{169} \\ s_{178}, s_{179}, s_{180}, s_{181}, \dots, s_{288} \leftarrow t_{2,1}, t_{2,2}, \dots, t_{2,8}, s_{178}, s_{179}, s_{180}, \dots, s_{280} \\ Para \qquad k \rightarrow \{1,2,3,4,5,6,7,8\} \end{gathered} \tag{3}$$

Al introducir j tríos de bits al estado interno se tiene la ventaja adicional de que la inicialización de éste será mucho más rápida y en vez de tener que esperar 1152 ciclos solo tomará $1152/j$ ciclos para inicializarse.

Se decidió que el módulo de TRIVIUM fuera diseñado para calcular 32 bits de *keystream* por ciclo, por lo cual la arquitectura del CPU deberá ser de 32 bits para obtener un mejor desempeño.

2. Métodos

Codiseño hardware-software

Para la implementación del sistema de cifrado, se utilizó la metodología de co-diseño hardware-software. Esta metodología permite el desarrollo de sistemas donde las partes más costosas en cómputo de una aplicación son implementadas en hardware, dejando las menos costosas software. De esta forma, un programa ejecutado en un sistema de CPU-RAM puede delegar trabajo computacionalmente costoso a unidades implementadas en hardware de propósito específico. Este tipo de diseño tiene varias ventajas, las más importantes son mejorar el tiempo de ejecución de la aplicación. Otra ventaja importante es que se sigue aprovechando la gran flexibilidad que el software ofrece en el diseño de aplicaciones, así como el potencial paralelismo del hardware minimizando la complejidad de su diseño.

Después de realizar un análisis de la aplicación, se decidió diseñar dos módulos de hardware, el TRIVIUM y el ECC, los cuales serán integrados a un sistema de cómputo CPU-RAM. La propuesta del codiseño se muestra en la figura 3.

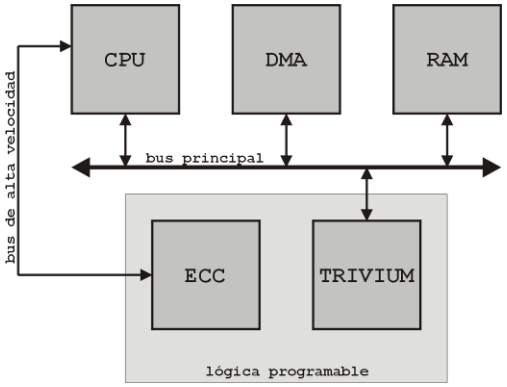

Figura 3 Diagrama a bloques del sistema digital propuesto.

El módulo marcado como CPU representa un procesador convencional, el cual puede contar con uno o varios núcleos. El módulo ECC (Curva Elíptica) implementa las operaciones de multiplicación de punto en la curva elíptica para la generación de la llave [Sutter, 2013][Qingxian, 2005]. El módulo TRIVIUM es la implementación del algoritmo paralelo de cifrado explicado en la sección 2B considerando las características de explicadas en [Jafarpour, 2011] y será el encargado del proceso

de cifrado. El módulo DMA (*Direct Memory Access*) o acceso directo a la memoria, permite al módulo TRIVIUM el acceso a la memoria principal para recibir los datos a cifrar y regresar los datos cifrados, de manera independiente al CPU.

Los módulos CPU, DMA, RAM y TRIVIUM están interconectados a través de un bus convencional. El módulo ECC se interconecta con el CPU por medio de un bus de alta velocidad. Esto se debe a que para el proceso de cifrado el módulo ECC será accedido solamente para la generación de la llave, mientras que el módulo TRIVIUM deberá realizar un mayor número de accesos a la memoria principal. Para acceder a los módulos de hardware desde un programa, el ECC será mapeado en memoria, mientras que el TRIVIUM será accedido a través del espacio de direcciones del DMA.

Software

Un programa será cargado a la memoria principal y ejecutado en el CPU para realizar las siguientes actividades: inicialmente enviará los parámetros al ECC para que realice las operaciones necesarias para la generación de la llave. El programa esperará la señal de interrupción que indica que los resultados están listos para ser leídos por el CPU y ser llevados a la memoria principal.

Calculada la llave, el programa la envía al módulo de TRIVIUM, el cual la almacena en un registro. Ahora, el programa inicia la transferencia de datos a cifrar, desde un archivo alojado en un dispositivo de almacenamiento con acceso USB, hacia la memoria principal y de ahí hacia el módulo de TRIVIUM. Como esta transferencia se realiza a través del DMA, no es necesario direccionar los dispositivos, así, la lectura y escritura de la memoria principal del sistema se realiza de manera totalmente independiente al CPU. El programa también es el encargado de llevar los datos cifrados a un archivo alojado en el dispositivo de almacenamiento.

4. Resultados

Implementación del sistema

El sistema fue implementado en la tarjeta de desarrollo ZYBO de la compañía Xilinx. Esta tarjeta de desarrollo cuenta con el SoC (System on Chip) Zynq-

XC7Z010, el cual es un sistema embebido que combina entre otros recursos, una zona de lógica reconfigurable, un procesador incrustado con dos núcleos ARM Cortex-A9 a 650 Mhz, módulos de memoria DRAM y un módulo DMA para la comunicación entre a periféricos. Un esquema del sistema implementado en el Zynq se puede observar en la figura 4.

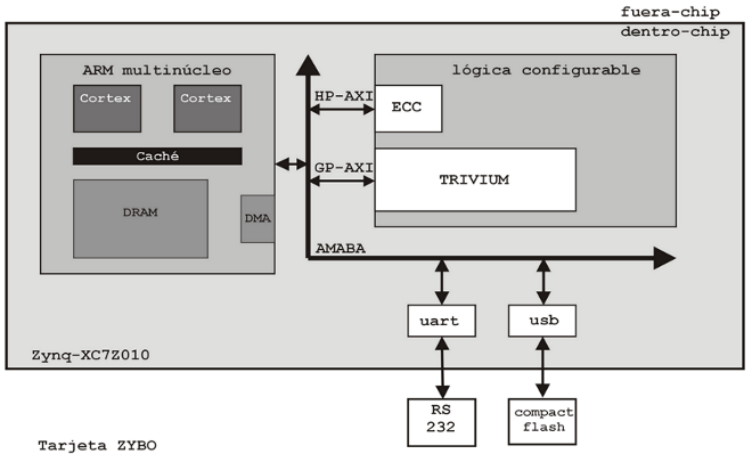

Figura 4 Sistema digital implementado en la tarjeta de desarrollo Zybo.

Para el desarrollo tanto del hardware como del software se utilizó el ambiente de trabajo Vivado de Xilinx. Vivado es un IDE o ambiente integrado de desarrollo para FPGAs de mayor capacidad y dispositivos que incluyan tanto la síntesis de alto nivel como el desarrollo de aplicaciones hechas en lenguajes de programación como C, C++ o Python. Los módulos ECC y TRIVIUM fueron desarrollados a través del lenguaje de descripción de hardware VHDL y encapsulados como dispositivos de propiedad intelectual, conocidos como IPs. Estos IPs fueron sintetizados en la zona reconfigurable del Zynq.

El CPU ARM contiene dos núcleos Cortex, memoria caché y bloques de memoria DRAM. Este CPU se conecta a través del bus AMABA (*Advanced Microcontroller Bus Architecture*) a los diferentes dispositivos. Existen dos tipos de puertos para conectar los IPs de la lógica configurable con el CPU: HP-AXI (High-Performance Ports) y GP-AXI (General-Purpose Ports), la diferencia radica en que los GP-AXI son direccionados por el DMA. El IP-ECC fue conectado al puerto de alta velocidad y el IP-TRIVIUM a un puerto de propósito general.

Se agregó un dispositivo UART (*Universal Asynchronous Receiver Transmitter*) para utilizarla como consola del sistema, esto es, para que el sistema pudiera enviar y recibir cadenas de caracteres a través de un dispositivo RS-232.

En la figura 5 se muestra un esquema de interconexión de los módulos sintetizados por la herramienta Vivado, en la cual se pueden identificar los IPs desarrollados y la parte de procesamiento del Zynq, así como su interconexión.

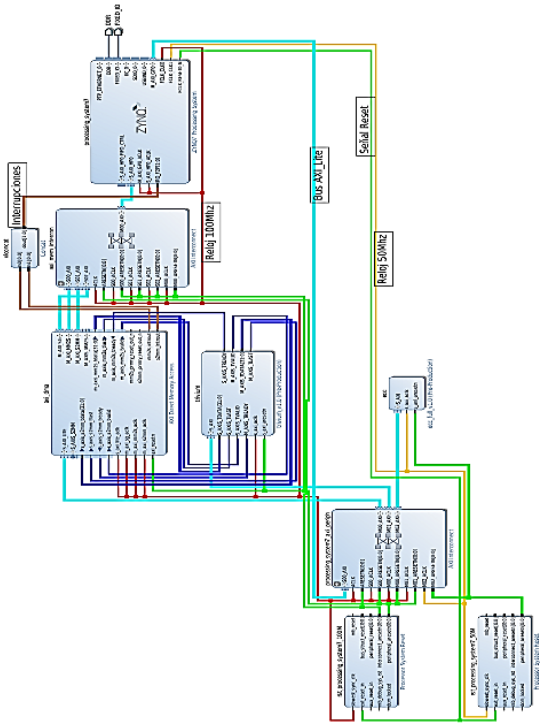

Figura 5 Esquema de interconexión entre las entidades de diseño del sistema digital.

Software de la aplicación

El programa fue desarrollado en lenguaje C y a través de un compilador cruzado se llevó a la arquitectura objetivo ARM. Además, el programa fue enlazado de manera estática con módulos de biblioteca para la entrada y salida estándar, ofrecidas por Xilinx. Estos módulos de biblioteca incluyen funciones especializadas en lenguaje C para el acceso a los IPs, con definiciones como las siguientes:

- Xil_Out32(*dirección, dato*) → Escribe dato a una dirección del sistema, a cada IP se le asigna un espacio de direcciones.
- Xil_In32(dirección) → Lee un dato de una dirección del sistema.

El programa se encarga de llevar a cabo las tareas descritas en la sección 3B. Una tarea adicional que realiza el programa es contabilizar el tiempo de procesamiento de cada IP. Para ello, se utilizó la función de biblioteca XTime_GetTime(), la cual es capaza de contabilizar el tiempo del CPU en microsegundos.

4. Discusión.

Recursos utilizados

Los recursos utilizados de la lógica programable muestran en la tabla 1.

Tabla 1 Recursos utilizados en la implementación del sistema digital.

LUT %	LUT's	FF %	FF's	BRAM %	BRAM's
53.307	9382	28.531	10043	3.333	2

Los recursos utilizados en la síntesis de los IPs incluyen LUT (*Look up tables*) tablas lógicas de verdad, FF (*Flip flops*) para registros y retención de señales y BRAM (B*lock RAM*) bloques de memoria RAM los cuales son necesarios para la ejecución del programa.

Tiempo de ejecución

Para comparar el rendimiento del sistema desarrollado (tabla 2), se programó el algoritmo TRIVIUM en C y se ejecutó en dos equipos con CPU de arquitectura x86 de gama media.

Tabla 2 Comparativa del tiempo de ejecución en tres arquitecturas diferentes.

	Intel 520M [µs]	AMD FX8350 [µs]	Zynq [µs]
	5876000	6687000	21388
	5905000	6715000	21392
	5934000	6700000	21380
Promedio	**5905000**	**6700666.67**	**21386.67**

El experimento consistió en codificar un archivo de un Megabyte de información y se contabilizó el tiempo de cifrado. Se realizó la misma prueba tres veces y se utiliza el promedio de estas como el resultado final.

Podemos considerar otra métrica comparativa si definimos el tiempo que toma codificar un Megabyte de información como t, podemos calcular la velocidad v de cifrado en Mb/s como $v = 1/t$, con lo cual se obtienen los resultados mostrados en la tabla 3.

Tabla 3 Comparativa del desempeño del sistema digital en tres diferentes arquitecturas.

Intel 520M [Mb/s]	AMD FX8350 [Mb/s]	Zynq [Mb/s]	
00.16930	00.14920	46.7581	

5. Conclusiones

Se puede observar una aceleración considerable al cifrar por medio del sistema embebido, esto es debido principalmente a que la parte computacionalmente más costosa del algoritmo TRIVIUM está diseñado en hardware.

Una estrategia para aumentar aún más el rendimiento de la aplicación sería sintetizar más de un módulo TRIVIUM y hacerlos trabajar de manera concurrente aprovechando tanto el inherente paralelismo del hardware como que el algoritmo de cifrado TRIVIUM es altamente paralelizable.

A pesar de que el Zynq es un sistema embebido de gama baja, es capaz de contener todo nuestro sistema desarrollado, sin embargo, las rutas de interconexión entre los CLB no son distribuidas de la manera adecuada y podrían generar problemas de sincronización. Este problema no se puede solucionar, aunque se reduzca la velocidad de operación de la lógica programable, el problema sólo se puede resolver utilizando una tarjeta con mayores recursos como por ejemplo la Zedboard de Xilinx.

6. Bibliografía y Referencias

[1] Fournaris, A.P., Zafeirakis, I., Koulamas, C., Sklavos, N., Koufopavlou, O., "Designing efficient elliptic Curve Diffie-Hellman accelerators for embedded systems," 2015 IEEE International Symposium on Circuits and Systems (ISCAS), pp.2025-2028, 24-27 May 2015.

[2] Jafarpour, A., Mahdlo, A., Akbari, A., Kianfar, K., "Grain and Trivium ciphers implementation algorithm in FPGA chip and AVR micro controller," 2011 IEEE

International Conference on Computer Applications and Industrial Electronics (ICCAIE), pp.656-658, 4-7 Dec. 2011.

[3] Maimut, D., Ouafi, K., "Lightweight Cryptography for RFID Tags," in IEEE Security & Privacy, vol. 10, no. 2, pp. 76-79, March-April 2012. doi: 10.1109/MSP.2012.43.

[4] Pu, Y., Samson, G., Shi, C., Park, D., Easton, K., Beraha, R., Hadi, J., "Blackghost: An ultra-low-power all-in-one 28nm CMOS SoC for Internet-of-Things," 2017 IEEE Symposium in Low-Power and High-Speed Chips (COOL CHIPS), Yokohama, 2017, pp. 1-3. doi: 10.1109/CoolChips.2017.7946384.

[5] Qingxian, W., "The application of elliptic curves cryptography in embedded systems," Second International Conference on Embedded Software and Systems, pp. 16-18, Dec. 2005.

[6] Sutter, G.D.; Deschamps, J.; Imana, J.L., "Efficient Elliptic Curve Point Multiplication Using Digit-Serial Binary Field Operations," IEEE Transactions on Industrial Electronics, vol. 60, no.1, pp.217-225, Jan. 2013.

[7] Ul Haq, E., Hafeez, M. K., Khan, M. S., Sial, S., Riazuddin, A., "FPGA implementation of a low power, processor-independent and reusable System-on-Chip platform," 2009 International Conference on Emerging Technologies, Islamabad, 2009, pp. 337-341. doi: 10.1109/ICET.2009.5353150.

[8] Wang, Z. C., Dai, Z. Bin, "High-speed realization of trivium based on multi-core cryptographic processor," 2015 IEEE 11th International Conference on ASIC (ASICON), Chengdu, 2015, pp. 1-4. doi: 10.1109/ASICON.2015.7517000.

[9] Wang, X., et al., "A multi-level collaboration low-power design based on embedded system," 2015 28th IEEE International System-on-Chip Conference (SOCC), Beijing, 2015, pp. 186-190. doi: 10.1109/SOCC.2015.7406937.