Ilustración del uso de pipeline

Dr. Oscar Leopoldo Pérez Castañeda [1], M.C. Jesús Daniel Pérez Castañeda[2], M.C. José Enrique Salinas Carrillo[3], Jaime Leonardo Huerta Valencia[4], Ángel Joaquín Fabián Rosales[5]

Resumen: En un sistema de procesamiento de datos, un parámetro importante es el throughput, esto es, el número de datos procesados por unidad de tiempo. Éste es logrado aplicando la técnica de pipeline o arquitecrura segmentada. Esta métrica es muy utilizada debido a que tiende a aumentar el throughput, incrementando la cantidad de recursos consumidos para su implementación. Dependiendo del parámetro a privilegiar, sea el tiempo de ejecución o los recursos para su implementación, se escoge entre una implementación en hardware como es el ASIC (Application Specific Integrated Circuits, por sus siglas en inglés) o el software usando un microprocesador. Sin embargo, el FPGA (Field Programmable Gate Arrays, por sus siglas en inglés) se sitúa entre estos dos últimos dispositivos, el ASIC y microprocesador. El FPGA supera la rigidez en cuanto a la implementación de un diseño, trabaja a una frecuencia muy cercana a la de los ASICs, y sus recursos son reconfigurables, como lo es la memoria del microprocesador. Esto permite utilizar la técnica de *pipeline* a un precio no tan elevado, a una frecuencia de trabajo bastante aceptable y con un *throughput* relevante. Este artículo ilustra la utilización del pipeline en un FPGA, mostrando ventajas y desventajas de esta técnica para mejorar el throughput, utilizando el algoritmo AES (Advanced Encryption Standard, por sus siglas en inglés).
Palabras clave: Arquitectura segmentada, ASIC, FPGA, microprocesador, pipeline.

Illustration of pipeline use

Abstract: In a data processing system, throughput is considered an important parameter, that is, the number of data processed per unit of time. Throughput is achieved by applying the pipeline technique or segmented architecture. This metric is widely used because it tends to increase throughput, adding the number of resources consumed for its implementation. Depending on the parameter chosen (execution time or the resources used), it is either implemented in the hardware such as ASIC (Application Specific Integrated Circuits) or in the software using a microprocessor. However, the FPGA (Field Programmable Gate Arrays) is configured inbetween these last two devices (ASIC and microprocessor). The FPGA overcomes the rigidity in the implementation of a design, works at a frequency very close to that of the ASIC's, and its resources are reconfigurable, as is the memory of the microprocessor. This allows you to use the pipeline technique at a lower price, at a fairly acceptable working frequency and with a competitive throughput. This article illustrates the use of the pipeline in an FPGA, showing advantages and disadvantages of this technique to improve throughput, using the AES (Advanced Encryption Standard) algorithm.
Keywords: Segmented architecture, ASIC, FPGA, microprocessor, pipeline.

Introducción

El *pipeline* es una técnica utilizada para el aumento del *throughput* (Andonov et al., 2001). Sin embargo, se aplica generalmente a algoritmos implementados en software y a algoritmos regulares; en otras palabras, algoritmos que no presenten saltos (Biddiscombe et al., 2007; Hodjat y Verbauwhede, 2004; Chodowiec et al., 2001) y que además no presenten dependencia de datos para no interrumpir la ejecución de la instrucción o tarea.

La implementación de *pipeline* en hardware eleva los costos considerablemente, esto debido a la replicación de bloques a conectar según el número de etapas o segmentos en que es segmentado el algoritmo o parte de éste (Hodjat y Verbauwhede, 2004; Chodowiec et al., 2001). Esto hace complicado, al menos en la práctica, implementar esta técnica tan relevante. Sin embargo, los FPGAs superan con relativa facilidad estas restricciones. Por tanto, el objetivo de este trabajo es el de ilustrar los beneficios y costos de una arquitectura segmentada estática lineal, *pipeline*, utilizando el algoritmo AES en un FPGA.

[1] El Dr. Oscar Leopoldo Pérez Castañeda es profesor del Tecnológico Nacional de México-Instituto Tecnológico de Tehuacán, Puebla, México, del área de Ingeniería Eléctrica-Electrónica, oscarleopoldo.pc@tehuacan.tecnm.mx **(autor corresponsal)**
[2] El M.C. Jesús Daniel Pérez Castañeda es profesor del Tecnológico Nacional de México-Instituto Tecnológico de Tehuacán, Puebla, México, del área de Ingeniería Eléctrica-Electrónica, jesusdaniel.pc@tehuacan.tecnm.mx
[3] El M.C. José Enrique Salinas Carrillo es profesor del Tecnológico Nacional de México-Instituto Tecnológico de Tehuacán, Puebla, México, del área de Ingeniería Civil, joseenrique.sc@tehuacan.tecnm.mx
[4] Jaime Leonardo Huerta Valencia es alumno del Tecnológico Nacional de México-Instituto Tecnológico de Tehuacán, Puebla, México, del área de ingeniería Eléctrica-Electrónica, L16360690@tehuacan.tecnm.mx
[5] Ángel Joaquín Fabián Rosales es alumno del Tecnológico Nacional de México-Instituto Tecnológico de Tehuacán, Puebla, México, del área de ingeniería Eléctrica-Electrónica, L16360686@tehuacan.tecnm.mx

La mayoría de trabajos relacionados con la técnica de *pipeline* o segmentación, mencionan las ventajas y desventajas de esta técnica para el mejoramiento del procesamiento de datos por unidad de tiempo, esto es, el *througput*, ya sea en software como en hardware (Hodjat y Verbauwhede, 2004. Este trabajo presenta los fundamentos de esta técnica de mejoramiento del *throughput* y valida cuantitativamente las bases de la misma, utilizando un FPGA, el cual ofrece recursos reconfigurables en hardware. Además induce al lector a la utilización de esta técnica así como del dispositivo (FPGA).

Antecedentes

Hace algunos años se presentaron escenarios reales donde algunos procesadores, disponiendo de frecuencias de reloj a la mitad de otras, ejecutaban a casi la misma velocidad la mayoría de los mismos programas; tal fue el caso del procesador IBM POWER2, que a tan sólo 135 MHz competía con el ALPHA 21164 a 400 MHz en operaciones de punto flotante (Etiemble, 2018). Aunque este hecho pareciera fuera de toda lógica, ya que existe una relación de 2 a 1 en cuanto a la frecuencia de reloj entre ambos procesadores, fue un hecho real y validado. La justificación de esta aparente contradicción fue el tipo de arquitectura implementada en cada microprocesador. Se sabe que las instrucciones en un microprocesador se ejecutan de manera secuencial, es decir, una después de la otra, pero esto no siempre ha sido así. A mediados de los años 80s se empezó a desarrollar e implementar la técnica de pipeline en los microprocesadores, generando resultados relevantes, de forma tal que esta técnica ha sido llevada a otro tipo de dispositivos como los microcontroladores (Datasheet MICROCHIP 18F4550), tal es el caso de los microcontroladores de MICROCHIP de la familia 18F4550 (Microchip, 2006), los cuales disponen de una arquitectura segmentada de cuatro etapas, y últimamente en los FPGAs (Sneha, 2018).

La arquitectura *pipeline* (segmentada) surge por la necesidad de aumentar la velocidad de procesamiento de datos (Forodvd, 2009). El *pipeline* es una forma relativamente económica de hacer paralelismo temporal en las computadoras (Pawel et. al. 2001). El funcionamiento del *pipeline* está basado en las líneas de ensamblaje de una planta industrial.

A continuación se describen las dos arquitecturas, una sin *pipeline* y la segunda haciendo uso de ésta. Para el microprocesador con arquitectura sin *pipeline*, supone que el ciclo de instrucción está formado por cuatro etapas, esto es, primero la instrucción es buscada (fetched), después decodificada, seguido de ello es ejecutada por una unidad funcional apropiada, y finalmente el resultado es almacenado en algún lugar, siendo así las cuatro etapas: búsqueda (**B**), decodificación (**D**), ejecución (**E**), y almacenamiento (**A**). Con este esquema, este microprocesador simple toma 4 ciclos para ejecutar una instrucción. Esto se muestra en la Figura 1.

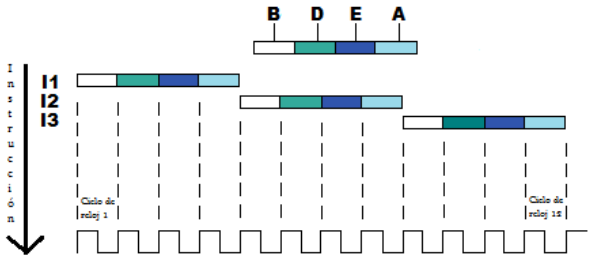

Figura 1. Flujo de instrucciones de un microprocesador secuencial, (Mitnik 2008).

Para el microprocesador con arquitectura *pipeline*, suponer que se utilizan las mismas cuatro etapas de la ejecución de instrucción del ejemplo anterior. Se denota la i-ésima instrucción como I_i. Entonces, en un caso ideal, en el primer ciclo se ejecuta la parte de búsqueda (**B**) de la instrucción $\mathbf{I_1}$; en el segundo ciclo, en el mismo ciclo de reloj 2, se ejecutan la parte de decodificación de $\mathbf{I_1}$ así como la parte de búsqueda de $\mathbf{I_2}$; para el tercer ciclo de reloj, se ejecutan en el mismo intervalo, la parte de ejecución de $\mathbf{I_1}$, la parte de decodificación de $\mathbf{I_2}$, y la parte de búsqueda de $\mathbf{I_3}$. En el cuarto ciclo de *pipeline*, se ejecutan en el mismo intervalo, la parte de almacenamiento de $\mathbf{I_1}$, la parte de ejecución de $\mathbf{I_2}$, parte de decodificación de $\mathbf{I_3}$ así como la parte de búsqueda de $\mathbf{I_4}$, dándose así el traslape de instrucciones, la ejecución temporal, y la concurrencia. En el siguiente ciclo de reloj, se procesan el almacenamiento de $\mathbf{I_2}$, la ejecución de $\mathbf{I_3}$, la decodificación de $\mathbf{I_4}$ y la búsqueda de $\mathbf{I_5}$. A partir del quinto ciclo de reloj, se observa lo siguiente:

- La $\mathbf{I_1}$ ha sido completamente ejecutada, se está realizando el almacenamiento (**A**) de $\mathbf{I_2}$, la ejecución (**E**) de $\mathbf{I_3}$, la decodificación (**D**) de $\mathbf{I_4}$, y la búsqueda (**B**) de $\mathbf{I_5}$.
- Una instrucción $\mathbf{I_i}$, una vez que se ha llenado el *pipeline*, se dispone en cada ciclo de una instrucción $\mathbf{I_i}$ completamente ejecutada. Para este ejemplo, en el quinto ciclo de reloj se dispone de $\mathbf{I_1}$, en el sexto ciclo

se dispone de $\mathbf{I_2}$ completamente ejecutada, en el ciclo 6 se tiene que $\mathbf{I_3}$ ha sido ejecutada completamente, y así sucesivamente.

La arquitectura *pipeline* ofrece una mejora de rendimiento en el procesamiento de datos, ya que una vez lleno el *pipeline*, éste ofrece una instrucción $\mathbf{I_i}$ completamente ejecutada en un ciclo de *pipeline*. Comparándolo con la arquitectura sin *pipeline*, la mejora ha sido en un factor de 4, el número de etapas utilizadas en este ejemplo, para la implementación del *pipeline*. Es conveniente hacer notar que el *pipeline* no reduce el tiempo de ejecución de una instrucción individual sino que incrementa el número de instrucciones que son ejecutadas simultáneamente, y la velocidad a la cual éstas son iniciadas y terminadas. En el ejemplo anterior, a todas las instrucciones les toma cuatro ciclos para que sean ejecutadas, esto es, toman el mismo tiempo de ejecución que en una arquitectura sin *pipeline*. Pero una vez lleno el *pipeline*, en cada ciclo se dispone de una instrucción. Por ejemplo, en la arquitectura sin *pipeline*, la instrucción $\mathbf{I_3}$ será completamente ejecutada hasta el ciclo de reloj $\mathbf{I_2}$, mientras que en la arquitectura con *pipeline* lo hará en el ciclo de reloj 6, incrementándose así el número de instrucciones ejecutadas en el mismo intervalo de tiempo. Esto se muestra en la Figura 2.

Las etapas representan circuitos que ejecutan operaciones aritméticas y/o lógicas sobre el conjunto de datos que fluyen a través de la línea denominada pipe (tubo). Cada etapa está separada por registros de muy alta velocidad, *latches*, que almacenan resultados intermedios entre cada etapa. Esto permite incrementar el *throughput* del sistema de manera considerable. La velocidad, en un sistema secuencial síncrono está limitada por aspectos de tipo tecnológico, consumo de energía, restricción algorítmica, y arquitectura, por mencionar algunos. Segmentar una arquitectura es dividirla en segmentos o etapas. Cada etapa (segmento) está definida por un registro que almacena los datos a procesar y otro que almacena los resultados. Un ejemplo de esta arquitectura se muestra en la Figura 3.

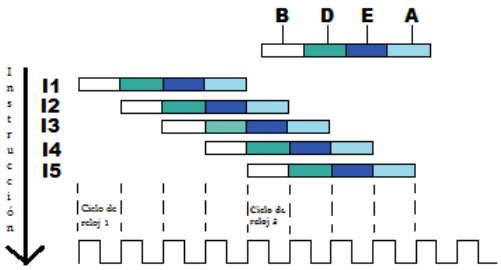

Figura 2. Flujo de instrucciones de un microprocesador pipeline, (Mitnik 2008).

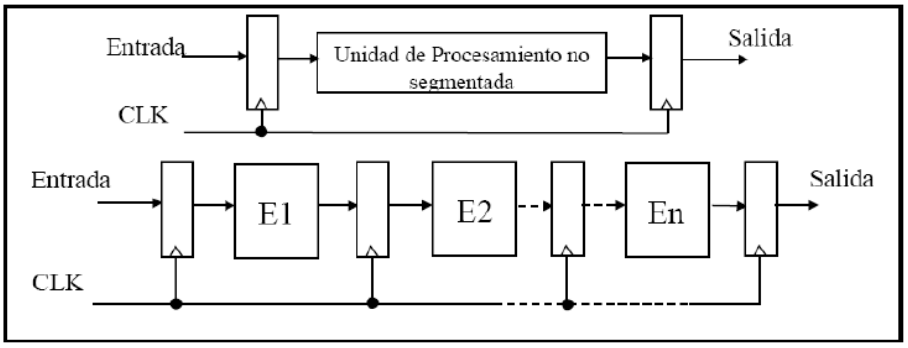

Figura 3. Ejemplo de una arquitectura pipeline o segmentada en n etapas o segmentos, (Forodvd 2009).

Las arquitecturas segmentadas o pipeline se clasifican en general como arquitecturas lineales o no lineales. Una arquitectura lineal está formada por un conjunto de etapas conectadas en cascada, sin ciclos de retroalimentación (*feedback*) de datos. En una arquitectura no lineal pueden existir ciclos hacia delante o hacia atrás (*feedback* o *feedforward*). Atendiendo a la función que realizan, se pueden clasificar en estática o dinámica. La estática realiza una función fija sobre un conjunto de datos de entrada, mientras que la dinámica realiza diferentes funciones sobre un conjunto de datos de entrada.

Definiciones básicas

Partiendo de la arquitectura genérica de un *pipeline* estático, como el representado en la Figura 4, se definen los siguientes términos:

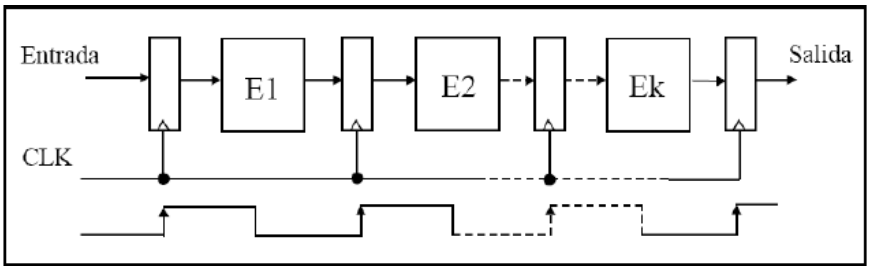

Figura 4. Implementación Pipeline, (Forodvd 2009).

Frecuencia máxima ($f_{máx}$)

$$f_{máx} = \frac{1}{T_{mín}} \qquad (1)$$

donde $T_{mín}$ es el período mínimo.

Speedup S_E (aceleración)

En un pipeline lineal estático de **E** etapas y **D** datos a procesar, el número de ciclos de reloj necesarios son: ***E + (D-1)***. En una arquitectura no pipeline será ***E*D***. Así que, la aceleración, Speedup (S_E), se puede expresar como:

$$S_E = \frac{E \cdot D}{E + (D-1)} \qquad (2)$$

Si el número de datos **D** a procesar es elevado, esto es, $\mathbf{D} \to \infty$ entonces $\mathbf{S_E} = \mathbf{S_{E\_máx}} \sim \mathbf{E}$, es decir, se obtiene una aceleración en el procesamiento de datos de un factor de **E**.

$$S_E = \lim_{D\to\infty} \frac{E \cdot D}{E + (D-1)} = \lim_{D\to\infty} \frac{E}{\frac{E}{D} + (1 - \frac{1}{D})} \cong \frac{E}{0+1} \cong E \qquad (3)$$

*La Eficiencia (**$\mathbf{E_f}$**)* se puede ver como la comparación entre la aceleración real y la ideal.

$$E_f = \frac{S_E}{E} = \frac{\frac{E \cdot D}{E + (D-1)}}{\frac{E}{1}} \qquad (4)$$

Si $\mathbf{D} \to \infty$ entonces $\mathbf{E_f} \sim 1$, lo que significa que la arquitectura alcanza su valor máximo de eficiencia ya que el valor de la eficiencia de un sistema es: $0 \leq \mathbf{E_f} \leq 1$. Una eficiencia con un valor aproximado a cero o cero, es un sistema poco o nada eficiente mientras que un sistema con un valor de eficiencia cercano o igual a 1 se considera como altamente eficiente. Así que,

$$E_f = \lim_{D\to\infty} \frac{\frac{E \cdot D}{E + (D-1)}}{\frac{E}{1}} = \lim_{D\to\infty} \frac{D}{E + (D-1)} \cong \lim_{D\to\infty} \frac{1}{\frac{E}{D} + (1 - \frac{1}{D})} \cong 1 \qquad (5)$$

Throughput

Se define como el número de datos procesados por unidad de tiempo. $T_H = f$, siempre que se mantenga un flujo constante de datos de entrada.

$$T_H = \frac{D}{[E + (D-1)] \cdot T} = \frac{D \cdot f}{E + (D-1)} \qquad (6)$$

Si $\mathbf{D} \to \infty$ entonces $\mathbf{T_H} = \mathbf{f}$.

$$T_H = \lim_{D\to\infty} \frac{D \cdot f}{E + (D-1)} = \lim_{D\to\infty} \frac{f}{\frac{E}{D} + (1 - \frac{1}{D})} \cong \frac{f}{0+1} \cong f \qquad (7)$$

Consideraciones

Para que se puedan obtener las ventajas de una arquitectura pipeline es necesario que el algoritmo cumpla con ciertas características.

1. El algoritmo debe de ser regular, esto es, que no presente saltos o brincos.
2. Una tarea **T**, se puede subdividir en un conjunto de **k** sub-tareas $\{T_1,...,T_k\}$. La relación de precedencia de este conjunto implica que una tarea Tj no puede comenzar hasta que otra tarea anterior Ti ($i < j$) no haya terminado. Las interdependencias de todas las sub-tareas forman el grafo de precedencia. Con una relación de precedencia lineal, la tarea Tj no puede comenzar hasta que todas las sub-tareas anteriores {Ti para todo i no mayor a j} terminen. Un *pipeline* lineal puede procesar una sucesión de sub-tareas que tengan un grafo de precedencia lineal.

Metodología

Lo primero que se realizó para investigar las ventajas y desventajas reales de la implementación de una arquitectura segmentada fue buscar un algoritmo regular, y el elegido fue el AES (Advanced Encryption System). Seguido de ello, se implementó el algoritmo en una computadora, utilizando el lenguaje de programación C. Entonces, se procedió a calcular el *throughput*. Como tercer paso, se implementó en hardware el mismo algoritmo sin la aplicación de pipeline. Después, se aplicó la técnica de pipeline y dividiendo en 10 etapas o segmentos (k = 10) el algoritmo de acuerdo al funcionamiento del AES. Luego se calculó el *throughput* alcanzado en software así como los recursos utilizados en hardware CLBs (Configurable Logic Blocks, por sus siglás en inglés). Finalmente se cuantificaron y compararon los valores de *throughput* alcanzados, así como los recursos consumidos para cada implementación.

Algoritmo AES

Este algoritmo consiste de dos partes: la parte de encriptación o codificación de datos y la parte de des-encriptación o decodificación de datos. La primera parte a su vez, se divide en lo que se le conoce técnicamente como la generación de la llave (Generation of the key) y la segunda de la codificación propiamente dicha de los datos, técnicamente conocida como la parte de Cipher. Para el algoritmo AES, el número de rondas, iteraciones o vueltas a ser realizadas durante la ejecución del mismo está en función del tamaño de la llave (key). El número de rondas está representado por N_r, donde $N_r = 10$ cuando $N_k = 4$, número de palabras de 32 bits de la llave. En este trabajo se utilizó $N_r = 10$ con $N_k = 4$. La operación del algoritmo de manera general se muestra en la Figura 5. Es necesario indicar que sólo se trabajó en la parte de encriptación, ya que la parte de des-encriptación es el proceso inverso, Federal Information (2001).

Ventajas del AES

A continuación se presentan algunas ventajas por las cuales el AES es utilizado.

1. La versatilidad de este algoritmo es una de las razones por la que es utilizado, ya que puede ser implementado en hardware y software.
2. Las diferentes longitudes de la llave (key) de un ancho de 128, 192 y 256 bits para su encriptación, lo hace más difícil de descifar.
3. La naturaleza open source del algortimo, le da un plus para su extenso uso en todo el mundo.
4. La robustes del algoritmo lo posiciona como uno de los preferidos. Para una llave (key) de 128 bits, se necesitan al menos 2^{128} intentos para descifrarlo. Los dos tipos de ataque en un algoritmo de cifrado simétrico son el cifroanálisis y la fuerza bruta (que implica probar todas las llaves posibles). El tiempo que tomaría descubrir mediante ataques de fuerza bruta una llave privada de diferentes tamaños es de 5.4×10^{24} años (Vargas, 2019).. Considerando que el tamaño de la llave es de 128 bits, el número de posibles llaves es $2^{128} = 3.4 \times 10^{38}$. Ahora, si el tiempo requerido en el peor caso para encontrar una llave por fuerza bruta fuera de 2^{127} ms, lo cual es muy significativo, entonces, 2^{127} ms = $2^{127} \times (1 \times 10^{-3}) = 1.7 \times 10^{35}$ s. Entonces el tiempo requerido para intentar descifrar al AES en estas condiciones, es de 5.4×10^{24} años, lo cual es desarrollado a continuación.

$$3.4x10^{38} * \ 1.7x10^{32}\ s * \frac{1\ min}{60\ s} * \frac{1\ hs}{60\ min} * \frac{1\ día}{24\ hs} * \frac{1\ año}{365\ días} = 5.4x10^{24}\ años$$

Le sugerimos al lector que a partir de este valor calcule cuántos milenios le tomaría tal descifrado. ¡Así es! ¡difícil de creer!

Todos los anteriores beneficios del AES lo han colocado como protocolo de seguridad en diferentes áreas de aplicación como comunicación inalámbrica, transacciones financieras, almacenamiento de datos encriptados, protección de código de algunos microcontroladores, evaluaciones académicas y otras.

Desventajas

Como todo algoritmo, el AES presenta algunas desventajas, las cuales se mencionan a continuación.

1. Usa demasiada álgebra simple.
2. Es difícil de implementar en software.
3. Cada bloque siempre es implementado de la misma manera.

Por todo lo anteriormente mencionado, el AES fue seleccionado para este trabajo.

Procedimiento

Se consideraron 10 etapas para el algoritmo, esto es, el valor de **E** = 10. Según la ecuación (3), la aceleración (*speedup*) es igual a 10 cuando el número **D** de datos tiende a infinito, si **E** = 10; **Ef** ~ 1 según la ecuación (5) y el throughput ~ f, siendo entonces la frecuencia del dispositivo en que se implementa el algoritmo.

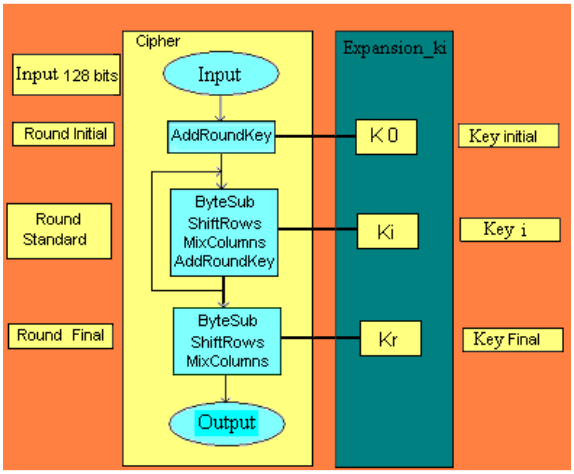

Figura 5. Muestra las dos partes que componen el algoritmo AES.

Para calcular el número de datos **D** a procesar que cumpla la ecuación a esta propuesta, se tiene a partir de la ecuación (2):

$$D = \frac{(S_E(1-E))}{S_E - E} \qquad (8)$$

Debido a que se busca que el número de datos **D** a procesar se aproxime a un valor real, para probar el pipeline, se analiza la ecuación (8). Analizando el denominador de la ecuación (8), se perciben los siguientes casos:

i. $\mathbf{S_E} = \mathbf{E}$, lo cual es un caso no real, ya que esto hace que el valor de **D** en la ecuación valga infinito y no se dispone de tal número de datos en la práctica.

ii. $\mathbf{S_E} \neq \mathbf{E}$, lo que nos lleva a otros dos casos:

   a. $\mathbf{S_E} > \mathbf{E}$. En este caso, el numerador de la ecuación (8) se vuelve negativo ya que E > 1 y, entonces se tiene un número **D** datos negativos, lo cual no es posible.

   b. $\mathbf{S_E} < \mathbf{E}$, es el único caso posible de manera real.

Así que, como el número de etapas **E** del algoritmo AES seleccionado fue 10, entonces se propone que $\mathbf{S_E}$ = 9, para así cumplir con la condición de que $S_E = 9 < \mathbf{E} = 10$. Sustituyendo estos valores en la ecuación (8) se tiene que **D** = 81. Para verificar este resultado, se graficaron $\mathbf{S_E}$ vs. **D**, lo cual es mostrado en la gráfica 1.

Como se aprecia de la Figura 6, con un valor de **E** = 10 y **D** = 81, la aceleración S_E se aproxima a 9. Así pues, para obtener una aceleración $\mathbf{S_E}$ = 10, esto es, el número de etapas **E** del algoritmo AES, **D** debe tender a infinito y con **D** = 81 aporta un valor de $\mathbf{S_E}$ = 9, no se tiene el valor teórico, esto es, $\mathbf{S_E}$ = 10. Por lo que se buscó un conjunto de datos que se considerara con tendencia a infinito para obtener el valor de $\mathbf{S_E}$ = 10. Se propuso una imagen a color de tamaño no comprimido de 800x600 pixeles, 97759 bytes = 782072 bits = 6110 words of 128 bits cada una. La comparación del número de datos propuestos para **D**, esto es, 782072 y el obtenido de manera teórica **D** = 81 es 9655.19, lo cual bien puede considerarse como que **D** ~ ∞.

Pruebas y Resultados

El algoritmo fue implementado en una tarjeta RC203 de Celoxica, que cuenta con un FPGA XC2V6000, Xilinx.Virtex-II (2005). Con el propósito de alcanzar valores altos de *throughput*, diferentes maneras de ejecutar el algoritmo AES han sido propuestas.

Figura 6. Aceleración $\mathbf{S_E}$ en función del número de datos **D**.

Todas ellas basadas en lo que se le conoce como Lazo Iterativo (Iterative Looping), Chodowiec et al. 2001. Cuando el dispositivo hardware no tiene suficiente espacio o recursos para desenrollar el lazo o ciclo completo, entonces se aplica este método. Por el contrario, si el dispositivo hardware cuenta con los recursos suficientes entonces el lazo se desenrolla, lo que hace que ya no sea un ciclo iterativo. Otras combinaciones han sido propuestas para el AES, pero todas ellas son una variante del Lazo Iterativo, Hodjat y Verbauwhede (2004) y Chodowiec et al. 2001. Para este caso de estudio, se implementó el algoritmo con Lazo Iterativo y después Lazo desenrollado. Para este último se aplicó el pipeline con 10 segmentos o etapas (**E** = 10). Así que, se espera que el throughput mejore en un factor aproximado a diez siendo penalizado en la misma proporción en cuanto a los recursos consumidos. Con la finalidad de calcular el throughput, se utilizó como referencia procesar una imagen a color de tamaño no comprimido de 800x600 pixeles 97759 bytes = 782072 bits = 6110 words of 128 bits cada una. Para Lazo Iterativo se utilizó el mismo bloque re-utilizando los operadores. Para cuando se desenrolla el Lazo, entonces este bloque se repite diez veces. La Tabla 1 muestra los resultados obtenidos para la implementación del algoritmo AES en sus dos versiones. CLBs representa los recursos o el área utilizada, BRAM el número de Bloques RAM utilizados para la implementación. $T_{ejecución}$ es el tiempo de ejecución de cada bloque.

Estrategia	Módulo	CLBs	BRAM	T_ejecución (ns)	Throughput (Mbps)
Lazo iterativo	Generación de la llave	72	0	2.8	4273
	Cipher	321	0	3.0	
Lazo extendido (pipeline)	Generación de la llave	744			
	Cipher	2905	0	2.9	43448

Tabla 1. Comparación entre algoritmo sin pipeline y pipeline con 10 segmentos

Como puede verse de la Tabla 1, un aspecto interesante a resaltar es que aunque el número de bits a tratar en este caso de estudio no es infinito, el número de bits evaluados es suficientemente grande como para que la aceleración $\mathbf{S_E}$ sea aproximadamente igual a 10 el número de etapas y la eficiencia $\mathbf{E_f}$ tienda a 1. Esto es lo que permite que se guarde

una relación proporcional de 10 entre el número de recursos y el throughput alcanzado por cada implementación. Si bien es cierto que, segmentar el algoritmo en 10 etapas o segmentos aumenta el throughput en un factor de diez, se ve afectado de manera colateral por los recursos demandados. El costo en recursos demandados por la técnica de pipeline es la causa por la cual en ocasiones no se implementaba esta técnica en hardware.

Conclusiones

A través de la implementación del AES en un FPGA, se han verificado los dos aspectos relacionados con esta técnica: su *throughput* y sus recursos demandados. Segmentar un algoritmo en **E** etapas y utilizar **E** registros rápidos entre cada bloque, aumenta el *throughput* en un factor de **E**. La penalización es la cantidad de recursos consumidos; en este caso es **E** veces. Sin embargo, con el avance de la tecnología, los FPGAs se presentan como una alternativa a esta limitante.

Limitaciones

Actualmente el pipeline es utilizado en diversos dispositivos como microprocesadores, microcontroladores, etc. La experiencia docente nos ha mostrado lo complicado que es entender este concepto, su implementación así como su utilización. Así que decidimos utilizar un algoritmo que permitiera ilustrar el pipeline con su bondades y limitaciones. La selección del algoritmo a implementar no fue tarea sencilla, pero afortunadamente el AES permitió ser implementado tanto en Lazo Iterativo como en Pipeline, y así comparar los resultados. Nos tomó algo de tiempo entender el funcionamiento de dicho algoritmo, pero bien valió la pena el tiempo invertido. Es fascinante el funcionamiento de este algoritmo, así como el hecho de implementarlo en Pipeline.

Recomendaciones

Los resultados obtenidos sugieren la implementación en hardware del AES para obtener un mejor desempeño del sistema. Una pregunta interesenta que se plantea es la implementación del AES de manera híbrida, i.e., una parte un software y otra en hardware; tal implementación ¿ofrecerá alguna ventaja? no necesariamente desde la perspectiva del throughput. ¿Será posible implementar el AES en un microcontrolador? Esto con miras de proteger los datos entre microcontroladores y/u otros dispositivos, sabedores que la naturaleza de un microcontrolador no es la de procesar datos sino señales de control. Una pregunta práctica que surge casi el final de este trabajo es la siguiente: ¿el AES podrá ser implementado en dispositivos más compactos como para generar aplicaciones de seguridad en puertas de autos o de casas? ¿El AES ha llegado a su límite de aplicaciones o todavía se puede extender a otras áreas?

Referencias

Biddiscombe J., Geveci, B., Martin, K., oreland, K., Thompson, D. (2007). Time Dependent Processing in a Parallel Pipeline Architecture. IEEE Transactions on Visualization and Computer Graphics, vol. 13, pp. 1376-1383.

Etiemble, D. (2018). 45-year CPU evolution: one law and two equations. Second Workshop on Pioneering Processor Paradigms, Feb 2018, Vienne, Austria.

Federal Information 2001. Announcing the ADVANCED ENCRYTION STANDARD (AES). Federal Information. Processing Standards Publication 197 AES. Página disponible en: http://www.nist.gov/CryptoToolkit

Forodvd. (2009). Explicando conceptos informáticos: EL PIPELINE. 12 enero 2021, de Forodvd.com Sitio web: https://www.forodvd.com/tema/134980-explicando-conceptos-informaticos-el-pipeline/

Hodjat, A. and Verbauwhede, I. 2004. A 21.54 Gbits/s Fully Pipelined AES Processor on FPGA.

Kacarska, M.; Andonov, D.; Glamocanin, V.; Stojkovska 2001. Pipeline multiprocessor algorithm for contingency analysis in powersystems, B. Power Engineering Society Summer Meeting, IEEE Volume 3, Issue, pp: 1457 – 1461.

MICROCHIP (2006). PIC18F2455/2550/4455/4550 Data Sheet.

Mitnik, R. (2008). Arquitectura de Computadores: Pipelining. 02 enero 2021, de Pontificia Universidad Católica de Chile, Depto. Ciencia de la Computación. Sitio web: https://slideplayer.es/slide/10739416/

Pawel Chodowiec, Po Khuon, Kris Gaj. (2001). Fast implementation of secret-key block ciphers using mixed inner-and outer-round pipeling. .ACM/SIGDA International Symposium on Field Programmable Gate Arrays, FPGA, FPGA'01 Monterrey, CA.

Sneha, H (2018). The Why and How of Pipelining in FPGAs. www.allaboutcircuits.com. https://www.allaboutcircuits.com/technical-articles/why-how-pipelining-in-fpga/

Vargas J. (2019. Tesis deMaestría: Análisis de eficacia y eficiencia para un método de ciberseguridad para el protocolo de comunicación ACARS en aeronaves comerciales legado.

Xilinx.Virtex-II 2005. Platform FPGAs: Complete Data Sheet.