IMPLEMENTACIÓN DE UN ACELERADOR PARA AES DE 128 BITS EN FPGA

ÓSCAR IVÁN PARDO BOHÓRQUEZ

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FÍSICOMECÁNICAS
ESCUELA DE INGENIERÍAS
ELÉCTRICA, ELECTRÓNICA Y DE TELECOMUNICACIONES
BUCARAMANGA
2021

IMPLEMENTACIÓN DE UN ACELERADOR PARA AES DE 128 BITS EN FPGA

ÓSCAR IVÁN PARDO BOHÓRQUEZ

Trabajo de Grado para optar al título de
Ingeniero Electrónico

Director
Elkim Felipe Roa Fuentes,
Philosophy Doctor.

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FÍSICOMECÁNICAS
ESCUELA DE INGENIERÍAS
ELÉCTRICA, ELECTRÓNICA Y DE TELECOMUNICACIONES
BUCARAMANGA
2021

CONTENIDO

LISTA DE FIGURAS

pág.

LISTA DE TABLAS

pág.

RESUMEN

TÍTULO: IMPLEMENTACIÓN DE UN ACELERADOR PARA AES DE 128 BITS EN FPGA [*]

AUTOR: ÓSCAR IVÁN PARDO BOHÓRQUEZ [**]

PALABRAS CLAVE: SYSTEM-ON-CHIP, FPGA, CRIPTOGRAFÍA, AES.

DESCRIPCIÓN:

En este documento se presenta el proceso de diseño de un acelerador por medio de hardware para el algoritmo *Advance Encryption Standard* (AES) de 128 bits. Este acelerador se desarrolló para operar como el periférico en un *System-on-chip* que opera con un procesador RISC-V de 32 bits a 200MHz de frecuencia. Este acelerador tiene como finalidad mejorar el rendimiento del AES de 128 bits comparado contra una implementación por software. Se presenta el diseño del acelerador desde el diseño de los módulos y su diseño preliminar. Se presenta el proceso de mejora del diseño preliminar, se plantea un mejor pipeline el cual mejora el tiempo de encriptación y el proceso de disminución de área llevado a cabo para el diseño final. Este diseño demora 50 ciclos de reloj en realizar el proceso de encriptación, esto sin contar la demora en el tiempo de carga de información propia del bus. Este diseño, junto al *System-on-chip*, fue sintetizado para poder realizar simulaciones para corroborar el funcionamiento y la medición del tiempo de ejecución. La implementación se realiza en una ARTIX 7 FPGA trainer board. Extrayendo la información del *System-on-chip* se tiene que el acelerador demora 95 ciclos de reloj contando el tiempo de carga de información. Adicionalmente se realizó una comparación del periférico contra una implementación por software. El acelerador implementado presenta una mejora cercana al 7200 % en cantidad de datos procesados comparado con una implementación por software. También presenta una mejora en espacio utilizado cercana a la mitad, esto comparado con la literatura.

---

[*] Trabajo de grado

[**] Facultad de Ingenierías Físico-Mecánicas. Escuela de Ingenierías Eléctrica, Electrónica y Telecomunicaciones. Director: Elkim Felipe Roa Fuentes, Philosophy Doctor.

ABSTRACT

TITLE: IMPLEMENTACIÓN DE UN ACELERADOR PARA AES DE 128 BITS EN FPGA [*]

AUTHOR: ÓSCAR IVÁN PARDO BOHÓRQUEZ [**]

KEYWORDS: SYSTEM-ON-CHIP, FPGA, CRYPTOGRAPHY, AES.

DESCRIPTION:

This document presents the design process of a hardware accelerator for the 128-bit Advanced Encryption Standard (AES) algorithm. This accelerator was developed to operate as the peripheral in a *System-on-chip* operating with a 32-bit RISC-V processor at 200MHz frequency. This accelerator is intended to improve the performance of 128-bit AES compared against a software implementation. The design of the accelerator is presented from module design and preliminary design. The improvement process of the preliminary design is presented, a better pipeline is proposed which improves the encryption time and the area reduction process carried out for the final design. This design takes 50 clock cycles to perform the encryption process, without taking into account the delay in the bus information loading time. This design, together with the *System-on-chip*, was synthesized to be able to perform simulations to corroborate the operation and measurement of the execution time. The implementation is performed on an ARTIX 7 FPGA trainer board. Extracting the information from the *System-on-chip*, the accelerator takes 95 clock cycles counting the information loading time. Additionally, a comparison of the peripheral against a software implementation was performed. The implemented accelerator shows an improvement of about 7200 % in the amount of data processed compared to a software implementation. It also presents an improvement in space utilization close to half, this compared to the literature.

---

[*] Bachelor Thesis

[**] Facultad de Ingenierías Físico-Mecánicas. Escuela de Ingenierías Eléctrica, Electrónica y Telecomunicaciones. Director: Elkim Felipe Roa Fuentes, Philosophy Doctor.

INTRODUCCIÓN

La encriptación es el procedimiento mediante el cual se modifica una información de tal forma que ésta quede ilegible por medio de una clave. Esta clave suele ser un conjunto de letras que dictamina la transformación de la información original a una información cifrada. De esta forma la información queda incomprensible para cualquier mecanismo que no tenga esta clave para el proceso de desencriptación. Este proceso se ha llevado a cabo mediante diversos algoritmos de encriptación para diversos fines, uno de ellos es el algoritmo AES siendo este algoritmo en su versión de 128 bits la escogida para realizar la implementación. Partiendo de esta información, este algoritmo se ha convertido en uno de los más populares para el cifrado, por ende, ha terminado haciendo parte de múltiples sistemas digitales de comunicación. Es por esto por lo que se han venido desarrollando diferentes aproximaciones de cómo mejorar el consumo energético[1], su eficiencia operacional[2], su área[3], dando así origen a los aceleradores de encriptación.

Los aceleradores son componentes que optimizan una operación, por ejemplo, un mecanismo de encriptación, ya sea por medio de *software*[4] o por medio de *hard-*

---

[1] Manh-Diep DAO y col. "An Energy Efficient AES Encryption Core for Hardware Security Implementation in IoT Systems". En: *2018 International Conference on Advanced Technologies for Communications (ATC)*. 2018, págs. 301-304.

[2] Shuang CHEN, Wei HU y Zhenhao Li. "High Performance Data Encryption with AES Implementation on FPGA". En: *2019 IEEE 5th Intl Conference on Big Data Security on Cloud (BigDataSecurity)*. 2019, págs. 149-153.

[3] Van DAO y col. "A compact, low power AES core on 180nm CMOS process". En: *2016 International Conference on IC Design and Technology (ICICDT)*. 2016, págs. 1-5.

[4] Shay GUERON. "Ïntel Advanced Encryption Standard (AES) Instruction Set White Paper". En: 2010, págs. 16-21.

ware [5]. Un acercamiento por medio de *hardware* brinda la oportunidad de diseñar circuitos adaptados a cada necesidad, ya sea consumo de potencia, velocidad de respuesta o área de implementación. De esta forma destacan los aceleradores in-core con una unidad aritmética lógica (ALU) dedicada a realizar el proceso criptográfico con la ventaja de no tener penalizaciones de comunicación con el core. También se encuentran los aceleradores que utilizan un co-procesador de propósito específico resultando en una mejor eficiencia, pero enfrentándose a la comunicación con el procesador. Por último, el acelerador como periférico, el cual tiene un menor rendimiento por la latencia propia del bus[6].

Este reporte presenta el diseño de un acelerador para AES de 128 bits por medio de hardware, mostrando en el capítulo 2 los pasos seguidos en el proceso de diseño general, los módulos usados y su respectiva optimización hasta el acelerador final. El capítulo 3 presenta los resultados de la simulación y su comparación con una solución que solo utiliza software [7]. En el capítulo 4 se presenta la implementación en FPGA del acelerador, sus resultados y su respectiva comparación con la implementación por software. Finalmente, en los capítulos 5 y 6 se presentan los posibles trabajos futuros, así como las conclusiones del trabajo realizado, respectivamente.

---

[5] Ye YUAN y Yijun YANG. "A High Performance Encryption System Based on AES Algorithm with Novel Hardware Implementation". En: *2018 IEEE International Conference on Electron Devices and Solid State Circuits (EDSSC)*. 2018, págs. 1-2.

[6] Stephan MATHEW y col. "340mV x2013;1.1V, 289Gbps/W, 2090-gate NanoAES hardware accelerator with area-optimized encrypt/decrypt GF(24)2 polynomials in 22nm tri-gate CMOS". En: *2014 Symposium on VLSI Circuits Digest of Technical Papers*. 2014, págs. 1-2.

[7] KOKKE. *tiny-AES-c*. `https://github.com/kokke/tiny-AES-c`. [En línea]. 2019.

1. OBJETIVOS

Objetivo general

- Describir una sbox para acelerar el algoritmo de encriptado avanzado estándar (AES).

Objetivos específicos

- Describir un acelerador haciendo uso de un lenguaje de descripción de hardware.
- Validar el funcionamiento del acelerador mediante simulación.
- Validar el funcionamiento del acelerador mediante la implementación en FPGA.

2. DISEÑO DE UN ACELERADOR PARA AES

Este capítulo presenta el funcionamiento del algoritmo AES y el proceso de diseño. La descripción del acelerador se realizó haciendo uso del lenguaje de descripción de hardware Chisel[8]. También se presenta la optimización llevada a cabo en términos de área y de tiempo de ejecución.

El algoritmo AES es un mecanismo de encriptación simétrico basado en un principio conocido como red de sustitución-permutación. Este algoritmo consta de 5 módulos que se repiten a lo largo de una serie de rondas siendo estos módulos: ExpandRoundKey (ERK), AddRounKey (ARK), SubBytes (Sbox), ShiftRows (SR) y MixColumns (MC) en donde los últimos 4 modifican la palabra que se busca cifrar y el primero modifica la clave para cada ronda.

La clave es el componente que puede variar entre 128 bits, 192 bits o 256 bits siendo la clave de 128 bits la escogida para este proyecto. Este algoritmo tiene una longitud de palabra estándar de 128 bits organizando la información en una tabla de 4*4 con cada celda conteniendo 1 byte, como puede ser visto en la Figura 1. El algoritmo AES de 128 bits consta de 10 rondas dictaminadas por el algoritmo de encriptación, con la última ronda omitiendo la operación MixColumns como se puede ver en la Figura 2. El diseño del acelerador debe funcionar como un periférico de un sistema integrado en chip (SoC) basado en un procesador RISC-V de 32 bits el cual opera a una frecuencia de 200 MHz. En este diseño se busca implementar un acelerador que tenga un mejor rendimiento que el desarrollado a través de software y a su vez se busca ocupar la menor cantidad de área posible.

---

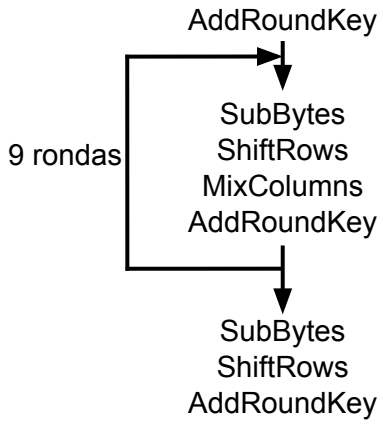
[8] U. OF CALIFORNIA IN BERKELEY. *Chisel - Constructing Hardware in a Scala Embedded Language*. `https://chisel.eecs.berkeley.edu/`. [En línea].

Figura 1. Estructura de la palabra del algoritmo AES.

byte 00	byte 01	byte 02	byte 03	byte 04	byte 05	byte 06	byte 07	byte 08	byte 09	byte 10	byte 11	byte 12	byte 13	byte 14	byte 15

byte 00	byte 04	byte 08	byte 12
byte 01	byte 05	byte 09	byte 13
byte 02	byte 06	byte 10	byte 14
byte 03	byte 07	byte 11	byte 15

Figura 2. Funcionamiento por rondas del algoritmo AES de 128 bits.



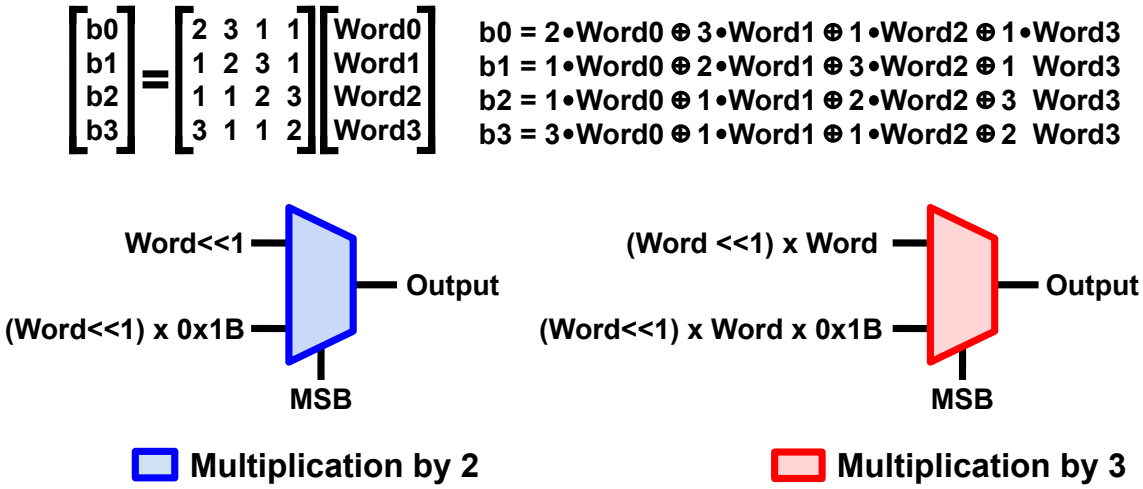
2.1. IMPLEMENTACIÓN POR HARDWARE DEL AES

Para el diseño preliminar del periférico se plantearon los módulos de tal forma que operasen a 32 bits por ciclo. El módulo ShiftRows reorganiza el orden de las filas del mensaje lo cual puede ser implementado realizando un arreglo en el orden de los bits. El módulo AddRoundKey es una serie de compuertas xor que adicionan la clave para esa ronda y la palabra a cifrar.

El módulo MixColumns es una operación matricial que transforma la columna a ope-

rar en una matriz con la particularidad que ocurre en un campo de Galois finito [9]. Esta forma de operar puede ser traducido en comportamiento circuital[10] como se puede ver en la Figura 3.

Figura 3. Implementación del módulo MixColumns.



El módulo SubBytes es una Substitution Box (Sbox) de 8b de entrada con 8b de salida. Este es el módulo que añade la sustitución al algoritmo. Tanto la entrada como la salida son interpretados como polinomios en un campo de Galvois finito. La Sbox se puede implementar de estas dos formas: Como la transformación que ocurre en el campo de Galvois ó como una LookUpTable. Debido a la complejidad circuital que presenta trabajar en un campo de Galvois, se optó por realizar una implementación basada en LookUpTables. De esta forma el módulo se puede traducir fácilmente en multiplexores. Debido a que el diseño opera a 32 bits por ciclo el módulo está

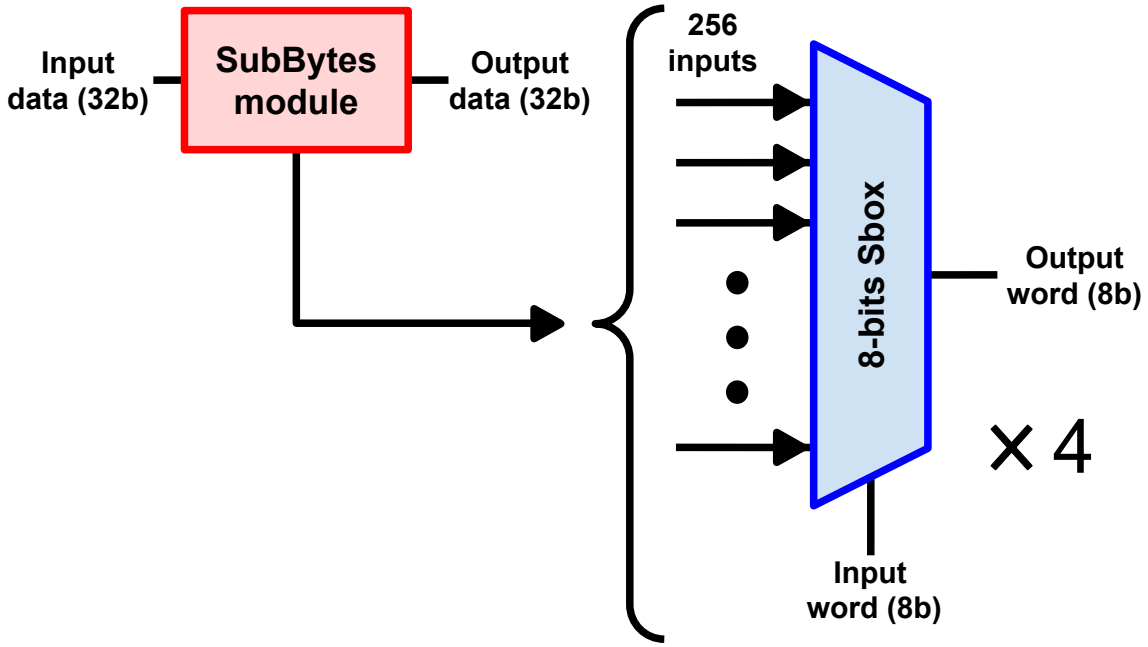
---

[9] Vincent RIJMEN y Joan DAEMEN. "Advanced Encryption Standard". En: *NIST FIPS PUB 197* (2001).

[10] PONCHO. *How to solve MixColumns*. `https://https://crypto.stackexchange.com/questions/2402/how-to-solve-mixcolumns`. [En línea]. 2014.

compuesto de 4 Sbox de 8b como puede ser visto en la Figura 4.

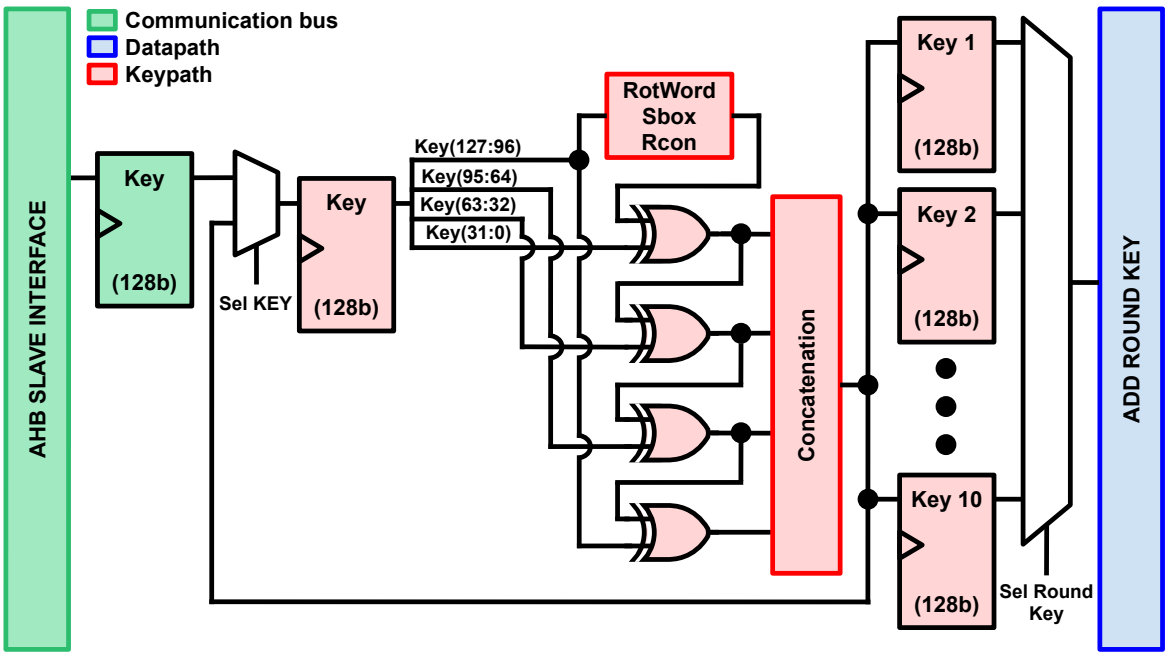
Figura 4. Diseño preliminar del módulo SubBytes.



Por último el módulo ExpandRoundKey es el encargado de expandir la clave original en varias claves que serán utilizadas en cada ronda. En la Figura 5 se puede ver la primera implementación, la cuál realiza el proceso de expansión y almacena, por medio de registros, cada una de las claves que serán utilizadas en el proceso. Del mismo modo este módulo utiliza cuatro Sbox de 8b independientes de las usadas en el módulo SubBytes.

Con estos módulos se realizó la implementación vista en la Figura 6 de la AES siguiendo lo establecido en la Figura 2. El pipeline del diseño implica que una ronda toma 7 ciclos para operar los 128 bits, 1 ciclo se ocupa en la primera mezcla con la clave original y la expansión de la llave toma 10 ciclos. Dando un total de 71 ciclos para encriptar la palabra y 10 ciclos extra debido a la expansión de la clave, pero si la clave persiste a lo largo de diferentes encriptaciones, ya no debe volver a

Figura 5. Diseño preliminar del módulo ExpandRoundKey.



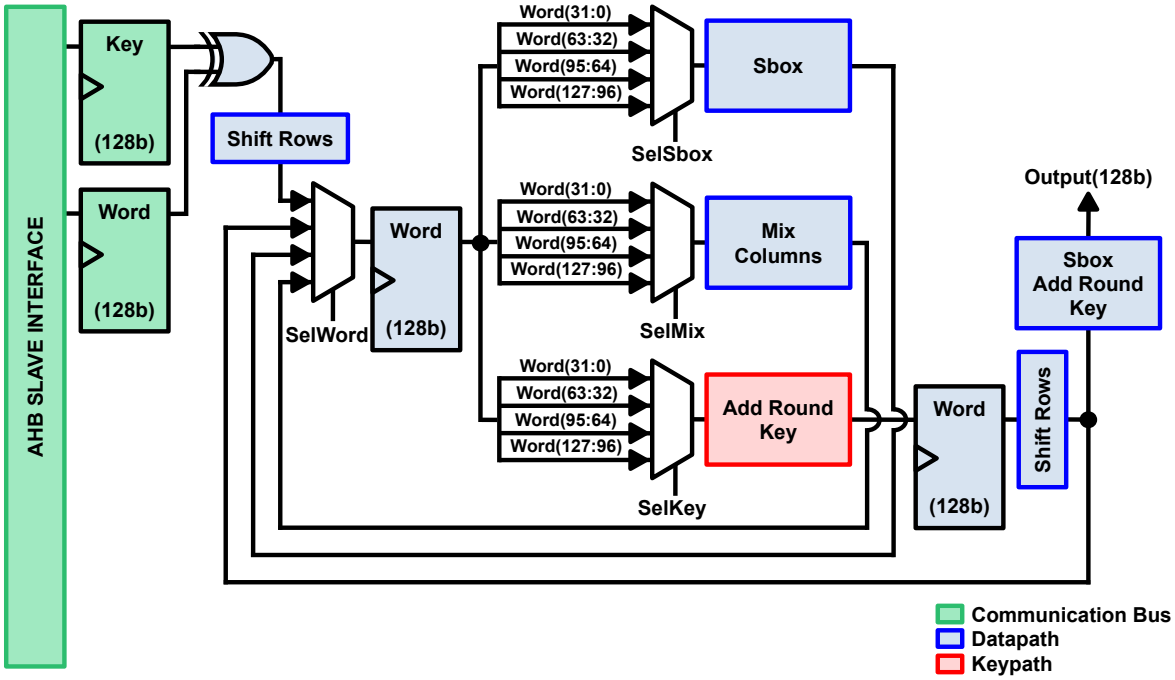
expandirse.

2.2. OPTIMIZACIÓN DE ÁREA

El diseño previo demora 81 ciclos contando la expansión de la llave y los módulos operan 32 bits de palabra por ciclo. En términos de área lo que más ocupa son los registros usados para almacenar la clave y le siguen los registros extras que se usan para almacenar la palabra. Se realizó una simulación para comprobar el correcto funcionamiento del periférico y de los módulos. Este proceso de simulación se ahondará más detalladamente en el siguiente capítulo.

Con el fin de obtener una reducción cercana a la mitad del área utilizada por el acelerador se planteó un nuevo diseño. Para realizar el diseño final se implementó un nuevo pipeline con el fin de mejorar la velocidad de operación. Este pipeline permite que dos módulos operen en el mismo ciclo de reloj y se utiliza una única serie de

Figura 6. Implementación preliminar del algoritmo AES.



registros para guardar la palabra durante todo el proceso. Se diseñó la ronda final de tal forma que agrupe el módulo SubBytes y el módulo AddRoundKey como se puede ver en la Tabla 1. Se agruparon los módulos MixColumns y AddRoundKey en el mismo ciclo como puede ser visto en la Tabla 2. Estos cambios se realizaron para reducir el tiempo de ejecución de la implementación y aumentar así el rendimiento.

Tabla 1. Diagrama de pipeline del algoritmo AES general.

1er ERK	Nueve rondas	Ronda final			
ARK/SR	-	Sbox/ARK	-	-	-
ARK/SR	-	-	Sbox/ARK	-	-
ARK/SR	-	-	-	Sbox/ARK	-
ARK/SR	-	-	-	-	Sbox/ARK
ERK	-	-	-	-	-

El módulo ExpandRoundKey fue rediseñado de tal forma que se opere cada ronda para disminuir el número de registros empleados. Esta disminución de registros im-

plica una reducción de casi la mitad en términos de área utilizada. El diseño previo empleaba 2 bloques de Sbox, uno para el datapath y otro para la expansión de la clave. Con el fin de disminuir aún mas el área se implementó un único bloque de Sbox para ser usado por los módulos ExpandRoundKey y SubBytes. La ronda fue rediseñada de tal forma que la Sbox, el submódulo que más área ocupa, opere la mayor cantidad de ciclos posibles. Esto implica el aprovechar el módulo la mayor cantidad del tiempo posible dando como resultado que la Sbox se utilice los 50 ciclos que demora el proceso de encriptación. Estos cambios pueden ser vistos en la Tabla 2.

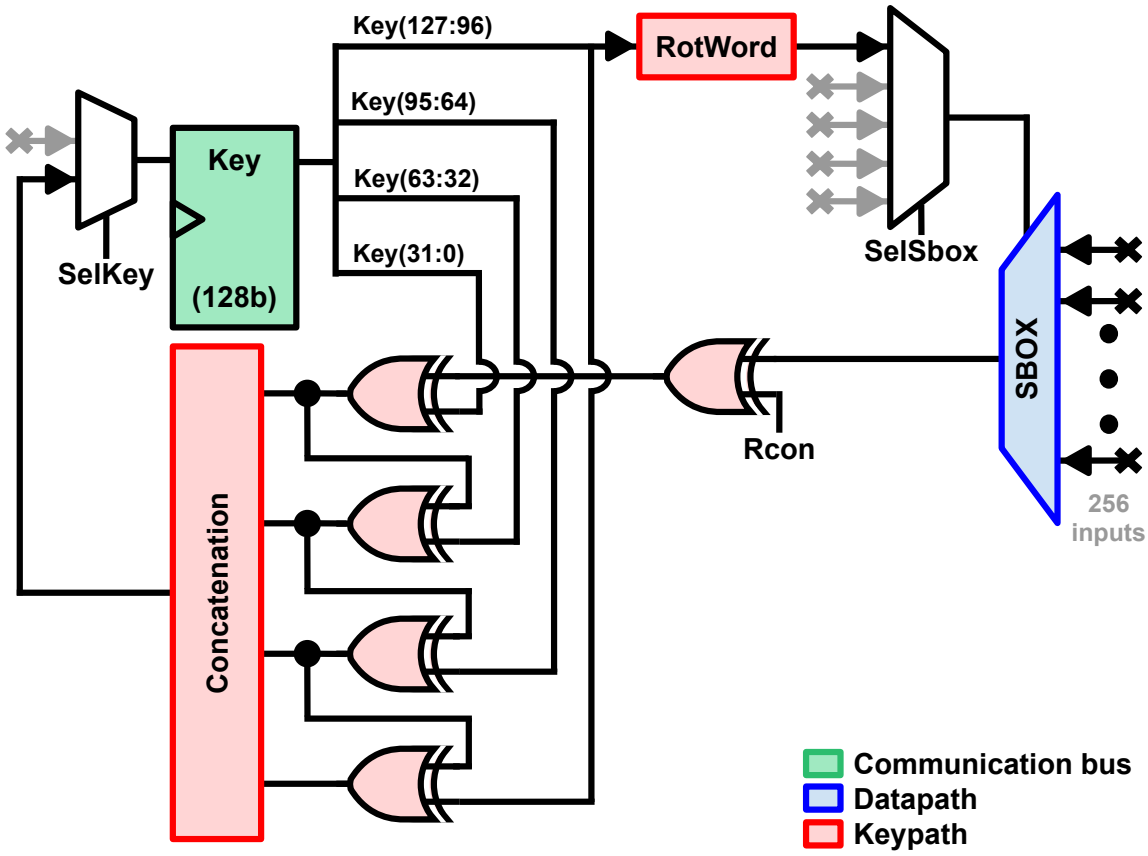
Tabla 2. Diagrama de pipeline de una ronda del algoritmo AES.

	Ronda				
Ciclos	1	2	3	4	5
word1	Sbox	MC/ARK	-	-	SR
word2	-	Sbox	MC/ARK	-	SR
word3	-	-	Sbox	MC/ARK	SR
word4	-	-	-	Sbox	MC/ARK/SR
key	-	-	-	-	ERK

En la revisión de la síntesis de la implementación preliminar se observó que el bus tiene sus propios registros que almacenan la clave y la palabra original. Con el fin de disminuir el área al mínimo se optó por usar estos registros del bus para almacenar la clave y la palabra junto a sus modificaciones y el resultado final. Este cambio ahorra los registros necesarios para almacenar 256 bits. Esta decisión implica que la clave debe ser cargada para cada encriptación, pero es una estrategia de ahorro de área. El módulo ExpandRoundKey fue rediseñado para no almacenar todas las claves de las 10 rondas. Esta expansión realiza la operación y almacena solo la clave necesaria para la ronda. Del mismo modo la Sbox necesaria para el proceso es la misma que se usa para la modificación de la palabra ahorrando área. Este diseño final puede ser visto en la Figura 7.

Para el diseño del datapath, éste se modificó de tal forma que solo usase un registro

Figura 7. Módulo ExpandRoundKey sin almacenamiento de todas las llaves.

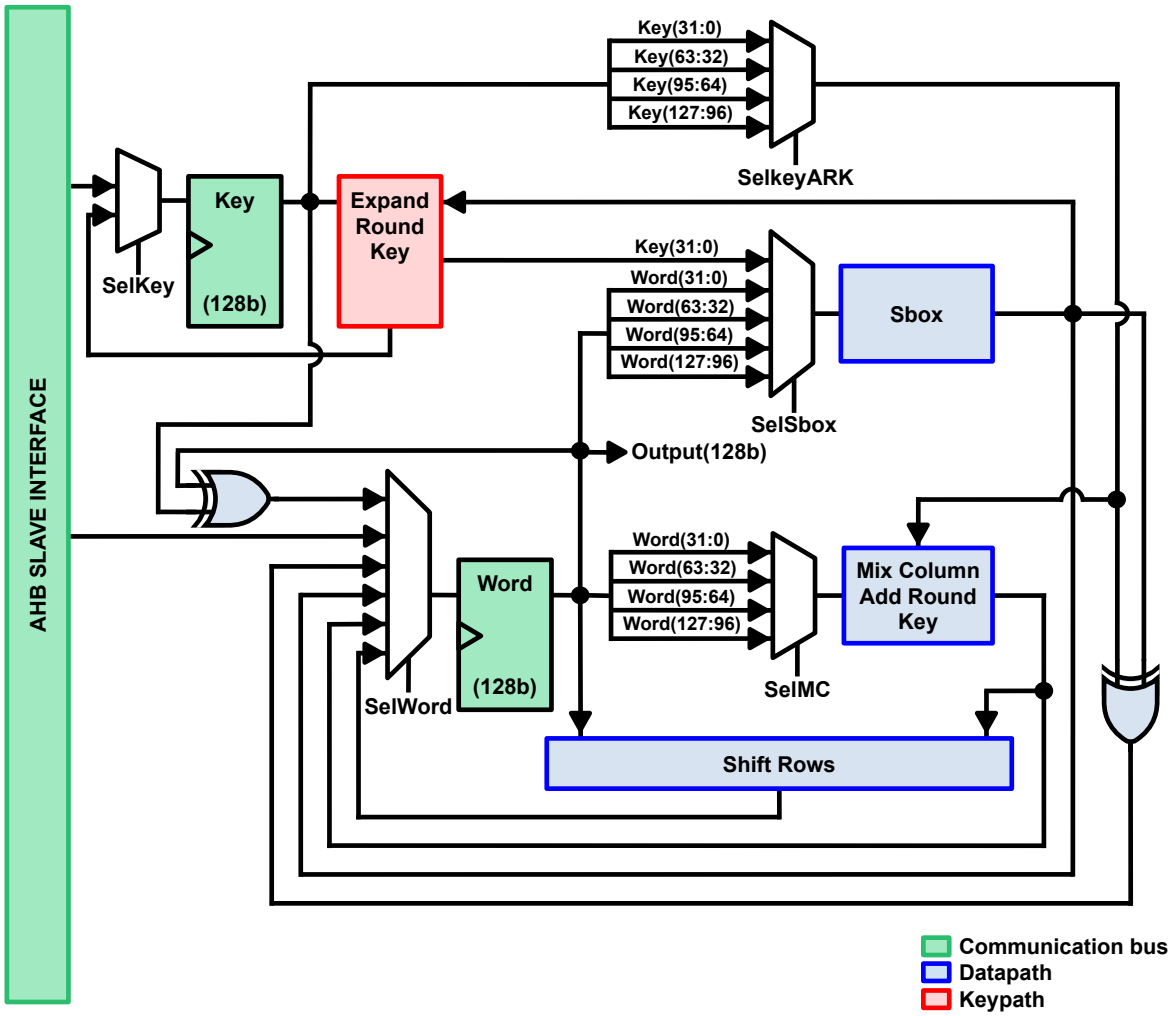


de 128 bits para almacenar la palabra. Esto implica que este registro almacena la palabra original, sus modificaciones y finalmente la palabra encriptada. También se unieron los módulos de MixColumns y AddRoundKey y se reorganizó la forma en la que funciona el módulo ShiftRows para lograr el funcionamiento establecido en la Tabla 2 siendo el resultado final visto en la Figura 8.

Esta implementación utiliza el mínimo de registros para guardar la palabra, la clave, la señal de inicio y una señal de busy, usando 258 bits de memoria. Esta implementación tiene la característica de que no puede ser detenida una vez la señal de start ha sido recibida debido a que los registros se bloquean para no recibir más información del bus. Lo anterior evita que se rompa el proceso de encriptación en

caso de que se intente reescribir la palabra o la clave. También es importante destacar que la operación de encriptar 128 bits de palabra demora 50 ciclos, aunque requiere recargar nuevamente cualquier clave, ya que ésta se pierde al momento de operarla.

Figura 8. AES total con el mínimo de registros para almacenar la información.

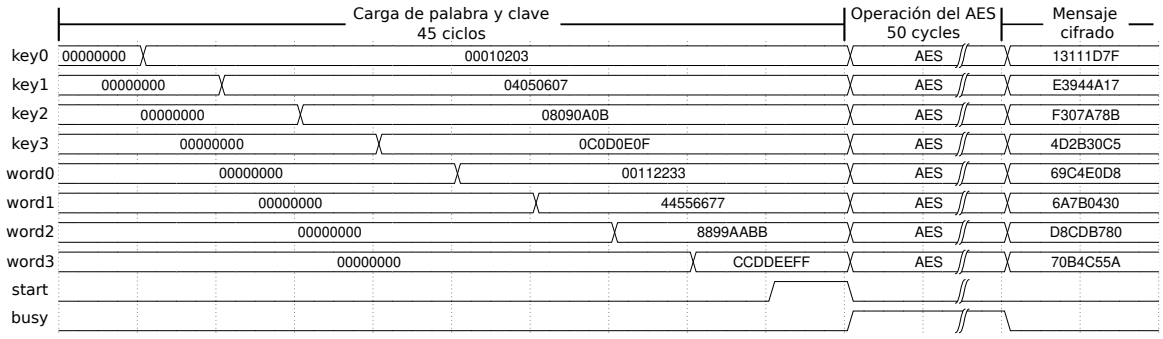


3. RESULTADOS DE SIMULACIÓN

Este capítulo presenta los resultados de simulación del periférico presentado en la sección anterior y del mismo modo presenta su respectiva comparación contra una implementación meramente de software. Este proceso se realizó sintetizando los archivos de Chisel[8] en los cuales está descrito el hardware del periférico y del SoC al cual pertenece; los resultados de la síntesis son archivos de Verilog los cuales fueron usados para realizar el proceso de simulación. Estas simulaciones fueron realizadas por medio del programa SimVision[11].

Para realizar la simulación del periférico del AES, se escribió un código que cargue en el periférico: la palabra, la clave y da la señal de arranque. Este código, escrito en C y posteriormente compilado con RISC-V de 32 bits, da como resultado lo visto en la Figura 9 donde se puede apreciar la demora en los tiempos de carga de la información; a pesar de esta demora se puede extraer el correcto funcionamiento del periférico.

Figura 9. Funcionamiento del acelerador implementado en simulación.



[11] CADENCE DESIGN SYSTEMS. *SimVision Debug A unified graphical debugging environment*. `http://https://www.cadence.com/en_US/home/tools/system-design-and-verification/debug-analysis/simvision-debug.html`. [En línea].

La medida del tiempo de ejecución se realiza desde que el *program counter* señala la carga de la información hasta que la señal de busy del AES se apaga. De esta forma se tiene en cuenta la latencia del bus para la carga de información. El tiempo de ejecución para el periférico es de 113 ciclos con un rendimiento de 269.47 Mbps a 200 MHz. Con el fin de poder realizar la comparación con una implementación puramente de software se escogió tiny-aes [7]. Este código fue compilado usando el compilador RISC-V gcc [12]. Para realizar la medición del tiempo se crea una variable de arranque y se contabiliza el tiempo hasta que se imprime el resultado de la operación. El tiempo de ejecución extraído indica una demora de 6800 ciclos de reloj por operación. Extrapolando este dato se obtiene un rendimiento de 3.764 Mbps a 200 MHz.

[12] RISC-V INTERNATIONAL. *riscv-gcc*. `https://github.com/riscv/riscv-gcc`. [En línea].

4. RESULTADOS DE IMPLEMENTACIÓN EN FPGA

Este capítulo presenta la síntesis e implementación del SoC con el periférico del AES en FPGA. El programa Vivado 2020.1 fue utilizado para realizar la síntesis y la Artix-7 FPGA Trainer Board[13] fue escogida para la implementación. Los resultados de la síntesis pueden ser vistos en la Tabla 3. Se denota el uso en el área del módulo Sbox debido a ser uno de los módulos que más área consume. Existen otras alternativas que pueden mejorar el área [14] pero operan en campos de Galois finitos aumentando la complejidad circuital.

Tabla 3. Resumen de síntesis en FPGA del periférico del AES.

Name	Slice LUTs	Slice Registers	F7 Muxes	F8 Muxes	Slices	LUT as Logic
AES	989	356	64	32	348	989
Sbox	0	0	64	32	32	0

El SoC fue implementado en la FPGA con una frecuencia operacional de 200 MHz. Se cargaron 2 códigos diferentes. El primer código verifica el funcionamiento del acelerador imprimiendo los resultados por medio de una conexión UART. El segundo código fue empleado para el cómputo del tiempo de ejecución tanto del acelerador como de la implementación por software. El SoC posee un periférico de *timer* el cual puede ser usado para medir los ciclos de reloj que toma la operación del AES pero presenta el problema de añadir ciclos de reloj extra a la medición al momento de escribir la señal de parada. Esto es debido a la demora de escritura entre el

---

[13] DILIGENT. *Nexys 4*. `https://reference.digilentinc.com/reference/programmable-logic/nexys-4/start`. [En línea].

[14] Joan BOYAR y René PERALTA. “A depth-16 circuit for the AES S-box.” En: vol. 2011. Ene. de 2011, pág. 332.

procesador y el periférico del *timer*. Esta adición extra de ciclos de reloj puede ser sustraída del tiempo total considerando que el tiempo de carga de una palabra a un periférico es de 5 ciclos. Del mismo modo, el código utiliza un *while* que lee el registro de *busy* del AES. Debido a que esta medición puede no estar sincronizada con el cambio del *busy* se pueden estar añadiendo ciclos extra de reloj. Analizando los resultados de simulación: el código usado para el periférico suma 10 ciclos mientras que el código usado con la librería *tiny-aes* suma 6 ciclos. Con el fin de obtener el tiempo final empleado, estos ciclos de reloj extra serán restados del valor indicado por el módulo *timer*.

Con este método se realiza la medición del tiempo del acelerador como periférico. La lectura del módulo *timer* indicó un valor de 110 ciclos, restando los ciclos estimados que hacen parte de la medición se tiene un tiempo de ejecución de 95 ciclos de reloj con un rendimiento de 269.47 Mbps a 200 MHz. Esta cifra concuerda con el resultado visto en simulación. El resultado leído en el módulo *timer* de la implementación por software fue de 6811. Considerando los ciclos extra se tiene entonces 6800 ciclos de reloj por operación. Extrapolando este dato se obtiene un rendimiento de 3.764 Mbps a 200 MHz. El resultado final de rendimientos puede ser visto en la Tabla 4.

Tabla 4. Rendimiento operacional de la AES a 200 MHz

Diseño	Mbps	Ciclos	Condición
Ideal	501.96	51	Carga de información en un único ciclo.
Simulación	269.47	95	Medición del tiempo de carga en software.
Implementación en FPGA	**269.47**	**95**	Utiliza el módulo timer para contabilizar los ciclos.
Software en simulación	3.76	6800	Realizado con el software tiny-aes.
Software en FPGA	**3.76**	**6800**	Utiliza el módulo timer para contabilizar los ciclos.

En la Tabla 5 se puede ver la comparación contra otros trabajos[5] [15] [16] [17].

Tabla 5. Tabla comparativa con otros trabajos AES.

Diseño	FPGA	Slices	Frecuencia MHz	Rendimiento Mbps
Y. Yuan	XC6SLX451	3854	153.3	1.57
Bouhraoua	XC2V250	30989	172.41	38.11
T. Good	XC3S4000-5	20720	240.90	30.83
McLoone	XCV812E	2457	93.90	12.02
Trabajo actual	**XC7A100T**	**1345**	**200**	**0.25**

[15] Abdelhafid BOUHRAOUA. “Design feasibility study for a 500 Gbits/s advanced encryption standard cipher/decipher engine”. En: *Computers Digital Techniques, IET* 4 (ago. de 2010), págs. 334 -348.

[16] Tim GOOD y Ma BENAISSA. “Pipelined aes on fpga with support for feedback modes (in a multichannel environment)”. En: *IET Information Security* 0 (ago. de 2007), págs. 1 -10.

[17] Maire MCLOONE y Jhon MCCANNY. “Rijndael FPGA implementation utilizing look-up tables”. En: vol. 34. Feb. de 2001, págs. 349-360.

5. TRABAJO FUTURO

El rendimiento general disminuye debido al tiempo de carga de la información como puede ser visto en la Figura 9. Esto puede ser mejorado haciendo uso de un acceso directo a memoria (DMA). Por otro lado la Sbox implementada es una LookUpTable que deja espacio a mejora. La Sbox es uno de los componentes que más influye tanto en el área como en el critical path, y si bien puede operar hasta una frecuencia de 280 MHz, según síntesis, existen otras alternativas que pueden mejorar ya sea el área[18] ó el critical path[5].

[18] David CANRIGHT. “A Very Compact Rijndael S-box”. En: (mayo de 2005), pág. 71.

6. CONCLUSIONES

Se diseñó un acelerador para el algoritmo AES de 128 bits como periférico de un SoC basado en un procesador RISC-V de 32 bits. Este acelerador se puso a prueba en simulación y en implementación en FPGA para comprobar su funcionamiento. Se comparó el funcionamiento del módulo AES de 128 bits como periférico contra una implementación por software tanto en simulación como en FPGA. Se destaca una mejoría en la cantidad de datos procesados del 7166 %. Esto implica una mejora sustancial del rendimiento, a cambio de una utilización extra de área. Se tiene que el acelerador implementado ocupa cerca de la mitad de los slices comparado con otros trabajos. Los resultados de esta comparación pueden verse en la Tabla 4.

BIBLIOGRAFÍA

BOUHRAOUA, Abdelhafid. “Design feasibility study for a 500 Gbits/s advanced encryption standard cipher/decipher engine”. En: *Computers Digital Techniques, IET* 4 (ago. de 2010), págs. 334 -348 (vid. pág. 25).

BOYAR, Joan y René PERALTA. “A depth-16 circuit for the AES S-box.” En: vol. 2011. Ene. de 2011, pág. 332 (vid. pág. 23).

CANRIGHT, David. “A Very Compact Rijndael S-box”. En: (mayo de 2005), pág. 71 (vid. pág. 26).

CHEN, Shuang, Wei HU y Zhenhao Li. “High Performance Data Encryption with AES Implementation on FPGA”. En: *2019 IEEE 5th Intl Conference on Big Data Security on Cloud (BigDataSecurity)*. 2019, págs. 149-153 (vid. pág. 8).

DAO, Manh-Diep y col. “An Energy Efficient AES Encryption Core for Hardware Security Implementation in IoT Systems”. En: *2018 International Conference on Advanced Technologies for Communications (ATC)*. 2018, págs. 301-304 (vid. pág. 8).

DAO, Van y col. “A compact, low power AES core on 180nm CMOS process”. En: *2016 International Conference on IC Design and Technology (ICICDT)*. 2016, págs. 1-5 (vid. pág. 8).

DILIGENT. *Nexys 4*. `https://reference.digilentinc.com/reference/programmable-logic/nexys-4/start`. [En línea] (vid. pág. 23).

GOOD, Tim y Ma BENAISSA. “Pipelined aes on fpga with support for feedback modes (in a multi-channel environment)”. En: *IET Information Security* 0 (ago. de 2007), págs. 1 -10 (vid. pág. 25).

GUERON, Shay. “Ïntel Advanced Encryption Standard (AES) Instruction Set White Paper”. En: 2010, págs. 16-21 (vid. pág. 8).

INTERNATIONAL, RISC-V. *riscv-gcc*. `https://github.com/riscv/riscv-gcc`. [En línea] (vid. pág. 22).

KOKKE. *tiny-AES-c*. `https://github.com/kokke/tiny-AES-c`. [En línea]. 2019 (vid. págs. 9, 22).

MATHEW, Stephan y col. “340mV x2013;1.1V, 289Gbps/W, 2090-gate NanoAES hardware accelerator with area-optimized encrypt/decrypt GF(24)2 polynomials in 22nm tri-gate CMOS”. En: *2014 Symposium on VLSI Circuits Digest of Technical Papers*. 2014, págs. 1-2 (vid. pág. 9).

MCLOONE, Maire y Jhon MCCANNY. “Rijndael FPGA implementation utilizing look-up tables”. En: vol. 34. Feb. de 2001, págs. 349-360 (vid. pág. 25).

PONCHO. *How to solve MixColumns*. `https://https://crypto.stackexchange.com/questions/2402/how-to-solve-mixcolumns`. [En línea]. 2014 (vid. pág. 13).

RIJMEN, Vincent y Joan DAEMEN. “Advanced Encryption Standard”. En: *NIST FIPS PUB 197* (2001) (vid. pág. 13).

SYSTEMS, CADENCE DESIGN. *SimVision Debug A unified graphical debugging environment*. `http://https://www.cadence.com/en_US/home/tools/system-`

design-and-verification/debug-analysis/simvision-debug.html. [En línea] (vid. pág. 21).

U. OF CALIFORNIA IN BERKELEY. *Chisel - Constructing Hardware in a Scala Embedded Language*. https://chisel.eecs.berkeley.edu/. [En línea] (vid. págs. 11, 21).

YUAN, Ye y Yijun YANG. "A High Performance Encryption System Based on AES Algorithm with Novel Hardware Implementation". En: *2018 IEEE International Conference on Electron Devices and Solid State Circuits (EDSSC)*. 2018, págs. 1-2 (vid. págs. 9, 25, 26).