PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

FACULTAD DE CIENCIAS E INGENIERÍA

DISEÑO DE UNA ARQUITECTURA DE CODIFICACIÓN/DECODIFICACIÓN DE ACUERDO AL ESTÁNDAR DE ENCRIPTACIÓN AES

Tesis para optar el Título de Ingeniero Electrónico, que presenta el bachiller:

Carlo Santiago Madera Vivar

ASESOR: Mg. Ing. Mario Andrés Raffo Jara

Lima, 2020

Resumen

El presente trabajo consiste en el diseño de un circuito digital para codificación y decodificación del algoritmo de encriptación AES (Advanced Encryption Standard) [1] para la implementación en FPGA de tecnología 90 nm como el Cyclone II y Virtex IV de las compañias Altera y Xilinx respectivamente. Este algoritmo consta de cuatro bloques, los cuales son AddRoundKey, SubBytes e InvSubBytes, ShiftRows e InvShiftRows y MixColumns e InvMixColumns.

El diseño del bloque SubBytes e InvSubBytes fue adaptado del diseño propuesto por Wolkerstorfer [1] usando la descomposición aritmética de $GF((2^4)^2)$. De igual manera, el diseño del bloque MixColumns e InvMixColumns fue adapatado del diseño propuesto por Satoh [2] usando la técnica de descomposición matricial.

Los bloques AddRoundKey, ShiftRows e InvShiftRows y el bloque completo AES fueron diseñados usando diversas técnicas de optimización como paralelismo de operaciones *(pipeline)*, FSMD y ASMD. El presente trabajo compara dos arquitecturas propuestas para algoritmo AES utilizando cero , una y dos etapas de pipeline en el bloque SubBytes e InvSubBytes. Referente a las arquitecturas, la primera se realizó usando la técnica de FSMD, mientras que la segunda se realizó usando la técnica de ASMD.

Se realizó la verificación funcional del circuito usando la herramienta de simulación ModelSim de la empresa MentorGraphics. Posteriormente se comparó los resultados con el documento del estándar de encriptación AES del NIST [3] obteniendo resultados exitosos.

Los requerimientos más importantes para este diseño son la alta velocidad de transmisión de datos *(throughput)* y el menor consumo de área. En base a esto, se realizó el análisis de síntesis y se obtuvieron los siguientes resultados. Para una arquitectura en ASMD se obtuvo hasta 0.382 Mbits/LUT y 182.538 MHz usando la plataforma Virtex IV; mientras que para una plataforma Cyclone II se obtuvo 0.162 Mbits/LE y 122.9 MHz. Respecto a la arquitectura FSMD se obtuvo hasta 0.305 Mbits/LUT y 185.895 MHz usando la plataforma Virtex IV; mientras que para una plataforma Cyclone II se obtuvo 0.159 Mbits/LE y 122.26 MHz. De acuerdo a estos resultados, se comprueba que la mejor técnica para realizar el diseño del algoritmo AES es la de ASMD.

[1] A continuación los nombres de los mnemónicos se encuentran en el índice de abreviaturas.

Dedicado a mi familia y en especial a mis padres, Elsa y Freddy, por el apoyo incondicional durante mi etapa universitaria.

A Santiago, mi hermano, por haberme enseñado ser un ejemplo a seguir.

A mi mascota Lazu por su alegría en los momentos más difíciles.

A Bonnie, mi fiel compañera, por su apoyo y aliento de superación día a día.

Agradezco a mis amigos de la especialidad de Ingeniería Electrónica, Fernando, Victor, Johan, Julian, Andrés y demás por su apoyo a lo largo de la carrera.

A todos mis colegas del grupo de Microelectrónica, Kelvin, Jair, Frank, Armando, Alejandro, Gabriel, Haris, Marco, Salvador, Stefano, Edward, Sammy, Guillermo, Christian y demás por su amistad y constante dedicación al grupo de investigación.

A todos mis profesores que he tenido a lo largo de la carrera, en especial al profesor Carlos Silva por haberme permitido ser parte de un extraordinario grupo de investigación como es el Grupo de Microelectrónica.

A mi querido amigo y asesor Mario Raffo, por su apoyo, dedicación, consejos y continua motivación para poder finalizar satisfactoriamente este proyecto.

Sólo una cosa vuelve un sueño imposible: el miedo a fracasar.

Paulo Coelho

Índice general

Índice de figuras

Índice de tablas

Índice de abreviaturas

AES Advanced Encryption Standard. i, iv, v, vii, 1–7, 10, 12, 17–19, 33, 34, 39, 40, 42, 44–46

ARM Advanced RISC Machine. 5

ASIC Application Specific Integrated Circuit. 1, 4, 5, 30

ASM Algorithmic State Machine. 17

ASMD Algorithmic State Machine con Datapath. i, 1, 33, 44, 45

DES Data Encryption Standard. 3, 4

FPGA Field Programmable Gate Array. i, 1, 2, 4–6, 20, 44

FSMD Finite State Machine con Datapath. i, v, 1, 17, 26, 33, 34, 44, 45

LE Logical Elements. i, 42

LUT Look Up table. i, 20, 42

NFC Near Field Communication. 6

NIST Instituto Nacional de Tecnología. i, vii, 1, 4, 18, 39, 40, 45, 46

PUCP Pontificia Universidad Católica del Perú. 2

ROM Read Only Memory. v, 20

RTL Register Transfer Level. 33

SoC System on Chip. 5

TSMC Taiwan Semiconductor Manufacturing Company, Limited. 1, 42

UMC United Microelectronics. 42

WBAN Wireless Body Area Network. 3, 4

Introducción

Actualmente, el crecimiento de las telecomunicaciones ha llevado consigo al desarrollo de un mundo tecnológico donde gran cantidad de información se transmite a través de las comunicaciones inalámbricas. Paralelo a esto, se ha observado un aumento sustancial en diversas técnicas de robo de información por parte de los hackers para acceder a información secreta como cuentas de bancos, información médica, información secreta de países, etc. Debido a esto, en el año 1997, el NIST de los Estados Unidos anunció un proyecto para implementar un algoritmo de cifrado avanzado capaz de garantizar la privacidad de información confidencial del gobierno, el mismo fue denominado como AES, siendo actualmente uno de los algoritmos más empleados en seguridad de información.

Como consecuencia de la rápida acogida del algoritmo AES, muchos científicos e investigadores se han enfocado en desarrollar diversas implementaciones de manera que este algoritmo pueda ser usado en diversas aplicaciones. Hoy en día, existen diversas investigaciones que demuestran la eficiente implementación del AES tanto en software como en hardware. Sin embargo, debido al desarrollo de la microelectrónica y la complejidad del algoritmo, se ha demostrado una gran superioridad de la implementación en hardware con respecto al de software, como ejemplo se tienen a los FPGA y los ASIC. Adicionalmente, diversos diseños del algoritmo AES dependen de la aplicación en la que se va a emplear, estos están orientados al consumo de energía, uso de área y velocidad de transmisión de datos.

La presente tesis se encarga de realizar el diseño del codificador y decodificador del algoritmo AES orientado a consumir la menor área y la mayor frecuencia de operación. Con respecto al diseño, se usarán diversas técnicas como pipeline, FSMD y ASMD, las cuales se implementarán en FPGAs de tecnología 90 nm Cyclone II y Virtex IV de las compañías Altera y Xilinx respectivamente. Posteriormente, se realizará la verificación funcional del circuito y el análisis de síntesis, evaluando parámetros como área, frecuencia de operación, entre otros.

Esta tesis tiene por objetivo desarrollar una primera etapa para un futuro desarrollo de un ASIC de tecnología TSMC 90 nm para encriptación AES propuesto por el Laboratorio de

Microelectrónica de la PUCP, por ello se emplearán FPGAs de tecnología 90 nm.

La estructura del trabajo de tesis es descrita a continuación. En el capítulo 1 se presenta el marco problemático del tema de tesis. El capítulo 2 aborda con mayor detalle el enfoque conceptual del bloque AES. El capítulo 3 muestra el diseño de cada uno de los bloques del AES, así como, el bloque completo. El capítulo 4 muestra la verificación funcional y el análisis de síntesis de los diseños propuestos. Finalmente se presentan las conclusiones y recomendaciones.

Capítulo 1

Marco Problemático

1.1. Fundamento Teórico

Actualmente existe una variedad de dispositivos que poseen un algoritmo de encriptación como celulares, *digital wallets*, *WBAN's*, *smartcards*, entre otros. Debido al desarrollo de estos dispositivos y al avance de diversas técnicas de ataques proveniente de los hackers se requiere algoritmos de encriptación más seguros [7]. Es así como, se establece el algoritmo de encriptación AES, teniendo una serie de mejoras respecto a su predecesor DES, como se detalla en la Tabla 1.1.

Tabla 1.1: Comparación entre AES y DES [5]

Factores	**AES**	**DES**
Año	2002	1976
Longitud de la llave	128, 192 o 256 bits	56 bits
Tamaño del bloque	128 bits	64 bits
Primitivas de encriptación	Sustitución, desplazamiento y mezcla	Sustitución y permutación
Algoritmo	Abierto	Abierto
Racionalidad del algoritmo	Abierto	Cerrado
Tiempo requerido para revisar todas las posibles llaves, referencia de 50 billones de llaves por segundo	AES-128: $5x10^{21}$ años	400 días

En la Tabla 1.1 se puede observar que las mejoras importantes del algoritmo AES son el aumento del tamaño del bloque y de la llave de cifrado. Actualmente las implementaciones del algoritmo AES pueden realizarse de manera eficiente tanto en software como en hardware. Sin embargo, se prefiere una implementación en hardware debido a que este posee un mejor desempeño, como el aumento de la frecuencia de operación.

Tabla 1.2: Comparación entre FPGA y ASIC [6]

ASIC	**FPGA**
Se configura una sola vez, dependiendo de la aplicación	Puede ser reconfigurado varias veces
Diseño basado en limitaciones orientados al consumo de área, consumo de energía y velocidad de transmisión de datos	Diseño flexible
Posee un costo elevado en comparación de los FPGA´s	Posee un costo más barato en comparación de los ASIC´s
Se utiliza para aplicaciones específicas	Se utiliza para desarrollo de prototipos de circuitos

Dentro de las posibles implementaciones en hardware, se tienen a los FPGA's y los ASIC's. La Tabla 1.2 ilustra las principales diferencias entre estos dos dispositivos. En forma general, existe gran diferencia entre la realización de un diseño para un FPGA y para un ASIC. El proceso del diseño en ASIC define diversas limitaciones exigentes orientadas a la aplicación del dispositivo en sí; mientras que en un FPGA se puede tener un diseño flexible, debido a que estos poseen elementos lógicos ya definidos, los mismos que son dependientes de la tecnología del FPGA. Las limitaciones en el diseño de un ASIC que poseen una mayor relevancia en comparación de las limitaciones de un FPGA son el uso de área, consumo de energía y la velocidad de transmisión de datos, esto es debido a que una implementación en ASIC es más costosa. El presente trabajo tiene como finalidad la implementación en FPGA's de tecnología 90 nm como primera etapa de un proyecto de solución en ASIC, sin embargo, siempre que sea pertinente se realizarán comparaciones con ASIC's. El diseño está orientado en tener la mayor velocidad de transmisión de datos y un bajo consumo de área, de esta manera se puede incorporar en diversas aplicaciones como los dispositivos antes mencionados.

1.2. Estado del Arte

El constante avance de las telecomunicaciones, ha llevado consigo a un incremento sustancial en la protección de comunicaciones. Es así como, el 26 de mayo de 2002, el NIST de EE.UU. pone en vigencia el estándar avanzado de encriptación AES, siendo este un algoritmo más seguro que sus predecesores DES y Triple DES [8]. La posterior adopción mundial del estándar AES ha derivado en la necesidad de diseñar e implementar dispositivos capaces de garantizar la comunicación segura de datos. Sin embargo, en el proceso del diseño se presentan limitantes importantes derivadas de la aplicación del dispositivo en sí. Una de estas aplicaciones es el diseño de bajo consumo de energía para dispositivos *WBAN* [9].

Existen diversos tipos de plataformas donde se puede implementar de forma eficiente el algoritmo de cifrado AES, tanto software como hardware [10]. Como ejemplo de esto se puede mencionar a los microprocesadores ARM, FPGA, SoC, ASIC, entre otros. Cada uno de estos dispositivos posee características específicas, como frecuencia de operación, desempeño, velocidad y consumo de energía [11]. De acuerdo a los resultados obtenidos por diversos diseños en software [12], [13], [14], se concluye que una implementación en software sería apropiada para un dispositivo de bajos requerimientos, sin embargo, si se desea un alto desempeño y alta frecuencia de operación es recomendable la implementación en hardware, ya que se puede hacer uso de diversas técnicas de optimización como el paralelismo de operaciones, así como, el uso del hardware específico *(customizable)*.

Hoy en día, el diseño e implementación del cifrado AES depende extremadamente de la aplicación en la que se le va a emplear. Estos diseños están orientados al consumo de energía, consumo de área y la velocidad de transmisión de datos [10]. Asimismo, debido a que dos de los bloques del algoritmo AES realizan operaciones aritméticas en campo finito o campo Galois, existen diversas técnicas y algoritmos para optimizar operaciones especificas como la multiplicación y la multiplicación inversa, las cuales afectan el rendimiento del bloque AES.

1.3. Importancia y justificación

El avance tecnológico de los dispositivos electrónicos se ve afectado directamente debido a la ley de Moore, la cual detallaba inicialmente que el número de componentes en un área específica se duplicaría cada año. En 1975 se realizó una modificación la cual mencionaba que en un circuito integrado el número de componentes en un área llegaría a duplicarse cada 18 meses a 2 años aproximadamente [15], [16], [17]. Esto se ve reflejado en la actualidad, ya que cada vez se busca que los dispositivos electrónicos sean más pequeños y que tengan la misma o mayor cantidad de funciones. En términos generales, el diseño e implementación del algoritmo AES sobre un ASIC o FPGA tiene una relevancia importante en la actualidad, debido a que con el avance tecnológico se requieren dispositivos que posean circuitos integrados orientados a tareas específicas. Tal es el caso de la actual tendencia del Internet de las cosas, la cual se refiere a la interconexión digital de objetos cotidianos con el internet, como la lavadora, cocina, televisión, aire acondicionado, ventanas, etc [18].

La importancia en la comunicación segura de información tiene un impacto mundial. Esto se puede observar cotidianamente, por ejemplo, en la elección de un patrón de celular, con la finalidad de que otra persona no pueda acceder a nuestra información, o la elección de una clave

para nuestra tarjeta de débito o crédito protegiendo las transacciones monetarias. Desde este punto de vista y tomando en consideración el avance tecnológico es sumamente pertinente el diseño de algoritmos de encriptación de información más seguros, siendo que la mayor eficiencia de estos se encuentre en implementaciones en hardware [12], [13]. Es por ello que la presente tesis se enfoca en el diseño de una arquitectura de un codificador/decodificador del estándar de criptografía AES para una implementación en FPGA.

Hoy en día, muchas entidades bancarias están innovando en tecnologías que permitan una mayor facilidad e interacción con el usuario. Tal es el caso de Digital Wallet, la cual permite realizar transacciones bancarias desde un smartphone por medio de la tecnología NFC. A nivel global, estas entidades manejan exorbitantes sumas de dinero diariamente, por lo que personas o grupos externos tratan con mayor frecuencia de tener acceso a su información y podría tener consecuencias catastróficas. Debido a esto, surge la necesidad de implementar algoritmos de encriptación de información, con el objetivo de dar una mayor seguridad entre transacciones bancarias.

1.4. Objetivos

1.4.1. Objetivo general

- Diseñar una arquitectura de codificador/decodificador de acuerdo al estándar de encriptación AES a ser implementado en FPGA's de tecnología de 90 nm, Cyclone II y Virtex IV de las compañías Altera y Xilinx respectivamente.

1.4.2. Objetivos específicos

- Aprender el funcionamiento interno del algoritmo de encriptación AES.
- Aprender y manejar de manera correcta el lenguaje de programación orientado a descripción de hardware (Verilog), para la descripción y verificación del circuito propuesto como se realiza en la industria o a nivel de posgrado.
- Entender el funcionamiento interno del algoritmo de encriptación AES.
- Aprender sobre operaciones en campo finito y como se implementan en hardware.
- Realizar el diseño de cada uno de los bloques del codificador/decodificador AES, mediante técnicas que permitan obtener una mayor eficiencia en términos de *throughput* y área.

Capítulo 2

Enfoque conceptual del codificador/decodificador AES

El presente capítulo tiene por objetivo brindar conceptos básicos referentes al bloque AES, los cuales son indispensables para poder realizar un correcto diseño en hardware. Dentro del bloque AES se tienen bloques que utilizan operaciones en campo finito o campo Galois. Debido a esto se realiza una pequeña introducción orientada a operaciones fundamentales como suma, resta y multiplicación, luego se procede a explicar cada uno de los bloques del AES.

2.1. Operaciones básicas en campo Galois $GF(2^8)$

Las operaciones matemáticas en campo finito tienen la propiedad de que el resultado que se obtiene a partir de cualquier operación con elementos del campo debe estar constituido de los elementos del mismo campo.
El rango de valores del campo finito $GF(2^8)$ está dado por la expresión 2.1, donde b_7, b_6, b_5, b_4, b_3, b_2, b_1, b_0 son bits que toman el valor de 0 o 1.

$$b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x + b_0 \tag{2.1}$$

2.1.1. Suma y resta en campo Galois $GF(2^8)$

La suma y resta en $GF(2^8)$ se caracteriza por tener el mismo resultado y está dado por el operador lógico *or* exclusivo bit a bit. Esta operación se representa por el símbolo $\oplus$.

Ejemplo:

$$\text{Sea:} \quad a(x) = x^7 + x^5 + x^4 + x^2 + x + 1 \quad y \quad b(x) = x^3 + x^2 + x$$

Notación polinómica: $a(x) \oplus b(x) = x^7 + x^5 + x^4 + x^3 + 1$

Notación binaria: $a(x) \oplus b(x) = 10110111 \oplus 00001110 = 10111001$

Notación hexadecimal: $a(x) \oplus b(x) = B7 \oplus 0E = B9$

2.1.2. Multiplicación en campo Galois $GF(2^8)$

La multiplicación en $GF(2^8)$ se calcula como el residuo de la multiplicación binaria de dos componentes, teniendo en cuenta la suma y resta en campo finito, con el polinomio irreductible F(x), tal como se muestra en la expresión 2.2. Esta operación se representa por el símbolo •.

$$F(x) = x^8 + x^4 + x^3 + x + 1 = 11B \tag{2.2}$$

Ejemplo:

Sea: $a(x) = x^6 + x^5 + x^4 + x^2 + x \quad y \quad b(x) = x^7 + x^2 + 1$

$$a(x)b(x) = x^{13} + x^{12} + x^{11} + x^9 + x^7 + x^5 + x^3 + x^2 + x$$

Notación polinómica: $c(x) = a(x)b(x) \quad mod \quad x^8 + x^4 + x^3 + x + 1 = x^7 + x^5 + x^2 + x$

Notación binaria: $c(x) = a(x) \bullet b(x) = 10100110$

Notación binaria: $c(x) = a(x) \bullet b(x) = A6$

A partir de la multiplicación en campo finito se puede obtener el valor inverso de una componente. Esto se basa en la propiedad de que cualquier número no nulo posee un inverso multiplicativo tal que al multiplicarlo por sí mismo se obtiene como resultado el valor de 1, tal como se muestra en la expresión 2.3.

$$A \bullet A^{-1} = 1 \tag{2.3}$$

2.1.3. Polinomios con coeficientes en campo Galois $GF(2^8)$

Otra representación de un polinomio en $GF(2^8)$ es su representación con coeficientes que son elementos de campo, es decir cada coeficiente representa un byte, tal como se muestra en la expresión 2.4.

$$d(x) = d_3x^3 + d_2x^2 + d_1x + d_0 \tag{2.4}$$

Se observa que el polinomio en esta sección es diferente que las anteriores representaciones, sin embargo ambos usan el mismo factor indeterminado x. Las operaciones de suma y resta son

idénticas de las operaciones anteriores, sin embargo, la operación de multiplicación emplea otro polinomio irreductible m(x), tal como se muestra en la expresión 2.5. La multiplicación en este tipo de representación se representa por el símbolo $\otimes$.

$$m(x) = x^4 + 1. \tag{2.5}$$

A continuación, se muestra el procedimiento para hallar la multiplicación en esta representación:

Sea: $\quad a(x) = a_3x^3 + a_2x^2 + a_1x + a_0 \quad y \quad b(x) = b_3x^3 + b_2x^2 + b_1x + b_0$

Primero: $\quad c(x) = a(x) \bullet b(x)$

$$c(x) = c_6x^6 + c_5x^5 + c_4x^4 + c_3x^3 + c_2x^2 + c_1x + c_0$$

Donde:

$$c_0 = a_0 \bullet b_0$$
$$c_1 = a_1 \bullet b_0 \oplus a_0 \bullet b_1$$
$$c_2 = a_2 \bullet b_0 \oplus a_1 \bullet b_1 \oplus a_0 \bullet b_1$$
$$c_3 = a_3 \bullet b_0 \oplus a_2 \bullet b_1 \oplus a_1 \bullet b_2 \oplus a_0 \bullet b_3$$
$$c_4 = a_3 \bullet b_1 \oplus a_2 \bullet b_2 \oplus a_1 \bullet b_3$$
$$c_5 = a_3 \bullet b_2 \oplus a_2 \bullet b_3$$
$$c_6 = a_3 \bullet b_3$$

Segundo: Se realiza $\quad d(x) = c(x)\ mod\ m(x)$

Donde : $\quad d(x) = d_3x^3 + d_2x^2 + d_1x + d_0$

$$d_0 = a_0 \bullet b_0 \oplus a_3 \bullet b_1 \oplus a_2 \bullet b_2 \oplus a_1 \bullet b_3$$

$$d_1 = a_1 \bullet b_0 \oplus a_0 \bullet b_1 \oplus a_3 \bullet b_2 \oplus a_2 \bullet b_3$$

$$d_2 = a_2 \bullet b_0 \oplus a_1 \bullet b_1 \oplus a_0 \bullet b_2 \oplus a_3 \bullet b_3$$

$$d_3 = a_3 \bullet b_0 \oplus a_2 \bullet b_1 \oplus a_1 \bullet b_2 \oplus a_0 \bullet b_3$$

Después de haber tenido una breve introducción sobre campos finitos, se puede continuar con la explicación del bloque AES. Sin embargo si se desea estudiar más información sobre campos finitos se puede revisar las referencias [3],[19],[20],[21],[22],[23].

2.2. Especificación del bloque AES

El algoritmo AES se divide en tres procesos fundamentales: expansión de llave, cifrado y descifrado [4]. Si bien el presente trabajo se enfoca en realizar el proceso de cifrado y descifrado del algoritmo AES, se procederá a la explicación de los tres procesos.

La principal característica del algoritmo AES es el tamaño fijo del bloque de 128 bits y la llave de cifrado que puede variar entre 128, 192 y 256 bits, siendo estos llamados AES-128, AES-192 y AES-256 respectivamente [4]. El algoritmo AES posee una cantidad de iteraciones entre sus bloques dependiente del tamaño de la llave de cifrado, esta se puede observar, tal como se detalla en la Tabla .2.1.

Tabla 2.1: Comparación entre el número de iteraciones dependiendo de la longitud de la llave

Tamaño de la llave	**Tamaño del bloque**	**Cantidad de iteraciones**
128 bits	128 bits	10
192 bits	128 bits	12
256 bits	128 bits	14

El algoritmo AES posee cuatro bloques dentro del proceso de cifrado: AddRoundKey, SubBytes, ShiftRows y Mixcolumns. De igual manera posee cuatro bloques en el proceso de descifrado: AddRoundKey, InvSubBytes, InvShiftRows e InvMixcolumns. En la Figura 2.1 se muestra el proceso del algoritmo AES con una llave de cifrado de 128 bits. Se puede observar el proceso de cifrado en la parte izquierda y el proceso de descifrado en la parte derecha de la imagen.

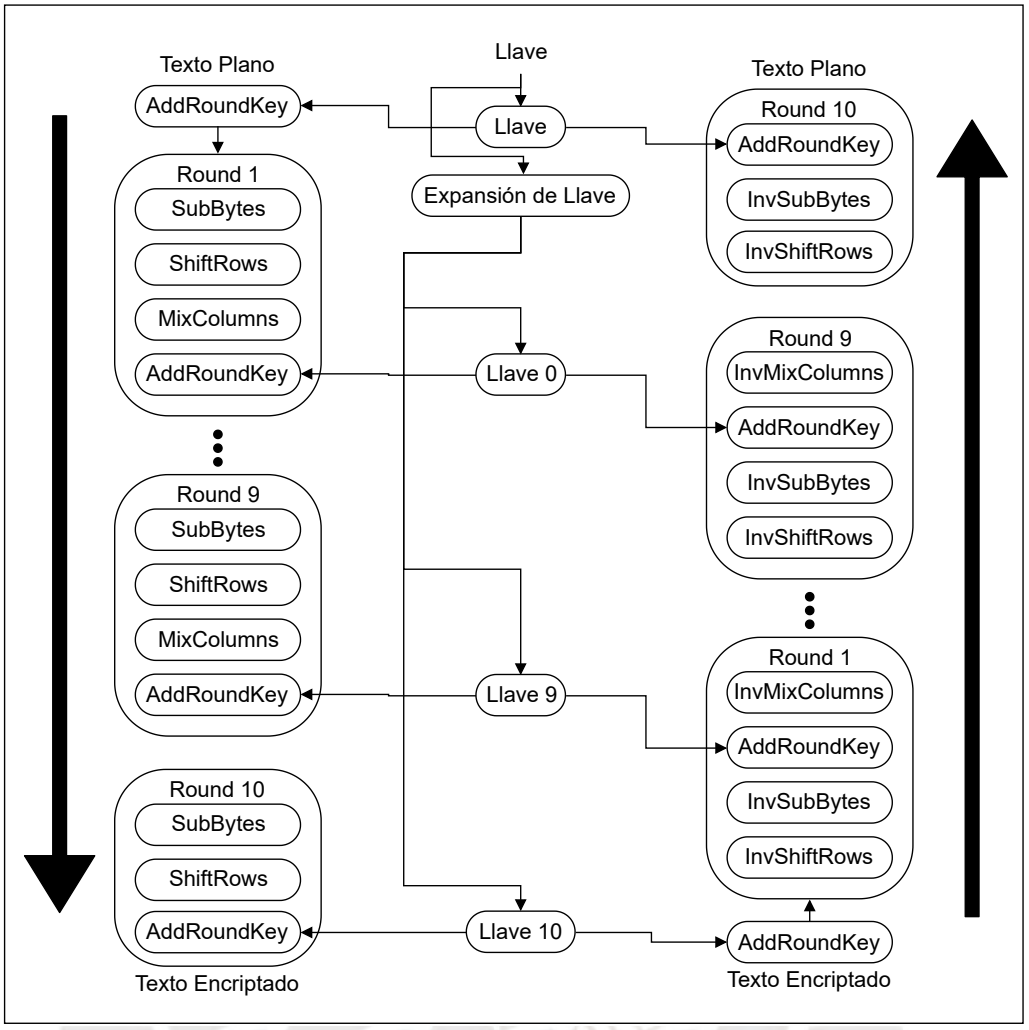

Figura 2.1: Flujo de datos del algoritmo AES-128 [4]

2.2.1. Bloque AddRoundKey

Este bloque se encarga de realizar la operación lógica or exclusivo entre cada byte del estado con un byte de sub-clave, según se muestra en la Figura 2.2.

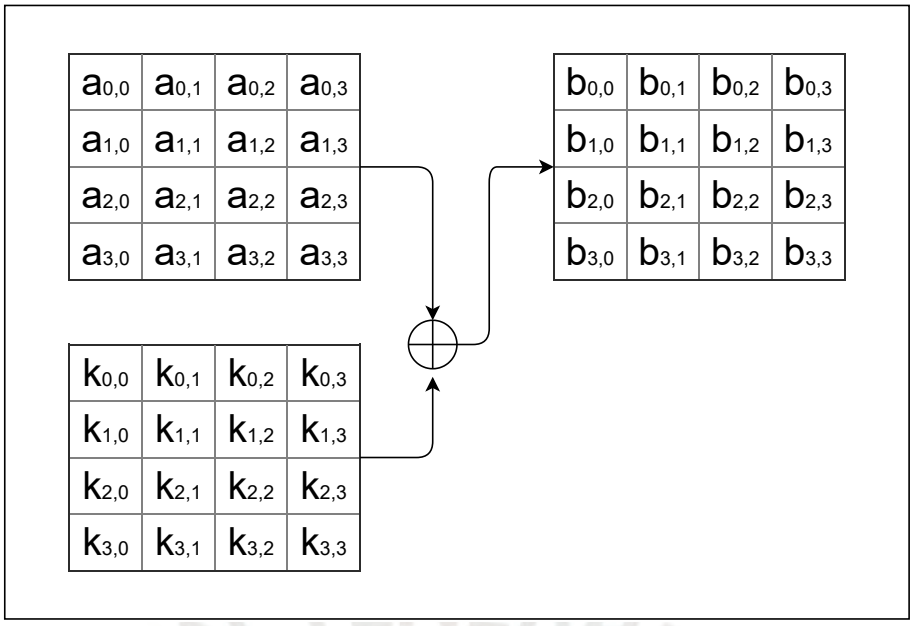

Figura 2.2: Bloque AddRoundKey

2.2.2. Bloques SubBytes e InvSubBytes

Estos bloques se encargan de sustituir cada byte del estado por el contenido de una tabla denominada SBOX (ver Tabla 2.2) o InvSBOX (ver Tabla 2.3). La primera representa la multiplicación inversa en campo finito $GF(2^8)$ cuando se encripta la señal, mientras que la segunda muestra la multiplicación inversa en campo finito $GF(2^8)$ cuando se decifra la señal. Esta transformación es una operación no lineal dentro del algoritmo AES. La Figura 2.3 muestra el bloque SubBytes / InvSubBytes.

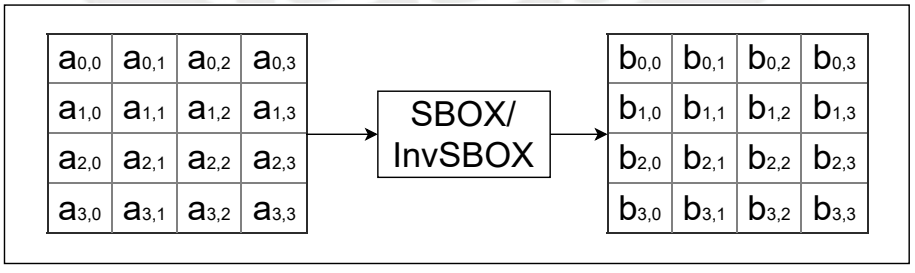

Figura 2.3: Bloque SubBytes / InvSubBytes

Tabla 2.2: Transformación SBOX

-	**00**	**01**	**02**	**03**	**04**	**05**	**06**	**07**	**08**	**09**	**0A**	**0B**	**0C**	**0D**	**0E**	**0F**
00	63	7E	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
01	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
02	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
03	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
04	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
05	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
06	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
07	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
08	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
09	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
0A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
0B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
0C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
0D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
0E	E1	F8	98	11	69	D9	8E	94	9B	16	87	E9	CE	55	28	DF
0F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

Tabla 2.3: Transformación InvSBOX [3]

-	**00**	**01**	**02**	**03**	**04**	**05**	**06**	**07**	**08**	**09**	**0A**	**0B**	**0C**	**0D**	**0E**	**0F**
00	52	09	6A	D5	30	36	A5	38	BF	40	A3	9E	81	F3	D7	FB
01	7C	E3	39	82	9B	2F	FF	87	34	8E	43	44	C4	DE	E9	CB
02	54	7B	94	32	A6	C2	23	3D	EE	4C	95	0B	42	FA	C3	4E
03	08	2E	A1	66	28	D9	24	B2	76	5B	A2	49	6D	8B	D1	25
04	72	F8	F6	64	86	68	98	16	D4	A4	5C	CC	5D	65	B6	92
05	6C	70	48	50	FD	ED	B9	DA	5E	15	46	57	A7	8D	9D	84
06	90	D8	AB	00	8C	BC	D3	0A	F7	E4	58	05	B8	B3	45	06
07	D0	2C	1E	8F	CA	3F	0F	02	C1	AF	BD	03	01	13	8A	6B
08	3A	91	11	41	4F	67	DC	EA	97	F2	CF	CE	F0	B4	E6	73
09	96	AC	74	22	E7	AD	35	85	E2	F9	37	E8	1C	75	DF	6E
0A	47	F1	1A	71	1D	29	C5	89	6F	B7	62	0E	AA	18	BE	1B
0B	FC	56	3E	4B	C6	D2	79	20	9A	DB	C0	FE	78	CD	5A	F4
0C	1F	DD	A8	33	88	07	C7	31	B1	12	10	59	27	80	EC	5F
0D	60	51	7F	A9	19	B5	4A	0D	2D	E5	7A	9F	93	C9	9C	EF
0E	A0	E0	3B	4D	AE	2A	F5	B0	C8	EB	BB	3C	83	53	99	61
0F	17	2B	04	7E	BA	77	D6	26	E1	69	14	63	55	21	0C	7D

2.2.3. Bloques ShiftRows e InvShiftRows

Estos bloques se encargan del desplazamiento de bytes a la izquierda o derecha por fila. Se desplaza un byte en la fila numero dos, dos bytes en la fila número tres y tres bytes en la fila número cuatro. El desplazamiento a la izquierda se realiza cuando se encripta la señal, mientras que se realiza un desplazamiento a la derecha cuando se decifra la señal. Las Figuras 2.4 y 2.5 muestran los bloques ShiftRows e InvShiftRows respectivamente.

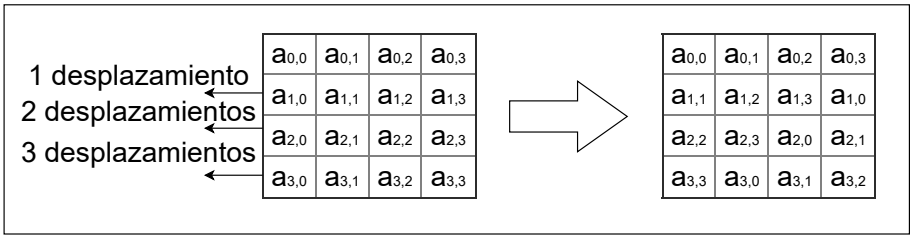

Figura 2.4: Bloque ShiftRows

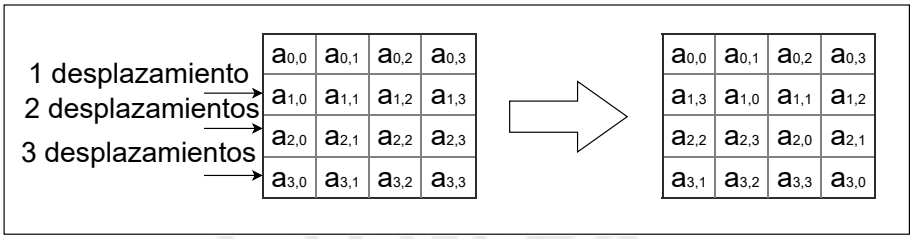

Figura 2.5: Bloque InvShiftRows

2.2.4. Bloques MixColumns e InvMixColumns

Cada columna del bloque es tratada como polinomio de campo $GF(2^8)$ y se multiplica en módulo por el polinomio constante c(x) para el bloque MixColumns, expresion 2.6 o c'(x) para el bloque InvMixColumns, expresión 2.7. Para ambos bloques se considera como polinomio irreductible el polinomio H(x), tal como se muestra en la expresión 2.8. Las Figuras 2.6 y 2.7 muestran los bloques MixColumns e InvMixColumns respectivamente.

$$c(x) = 3x^3 + x^2 + x + 2 \tag{2.6}$$

$$c'(x) = bx^3 + dx^2 + 9x + e \tag{2.7}$$

$$H(x) = x^4 + 1 \tag{2.8}$$

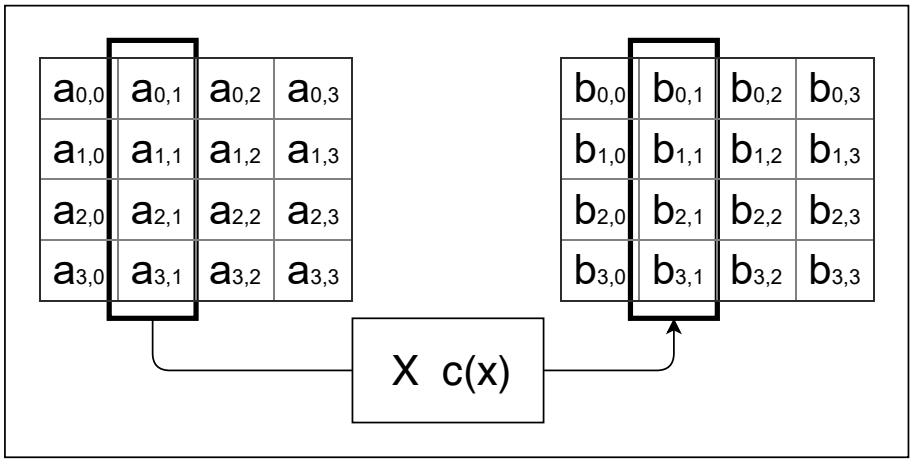

Figura 2.6: Bloque MixColumns

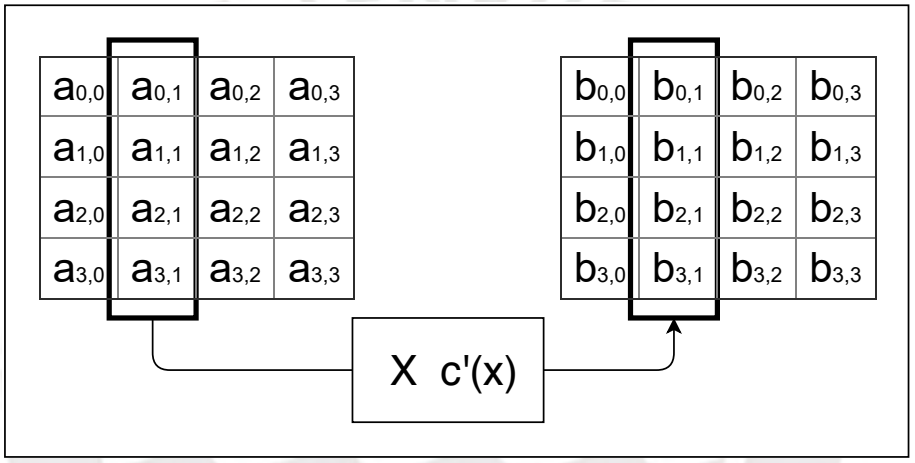

Figura 2.7: Bloque InvMixColumns

2.2.5. Expansión de la llave de cifrado

Este bloque se encarga de generar una llave de cifrado diferente para cada iteración del algoritmo de cifrado y descifrado. Estas llaves se calculan a partir de la llave original y el valor de unos valores predefinidos $Rc = 01, 02, 04, 08, 10, 20, 40, 80, 1B, 36$ con el fin de asegurar la seguridad del algoritmo. La operación del bloque se muestra en la Figura 2.8.

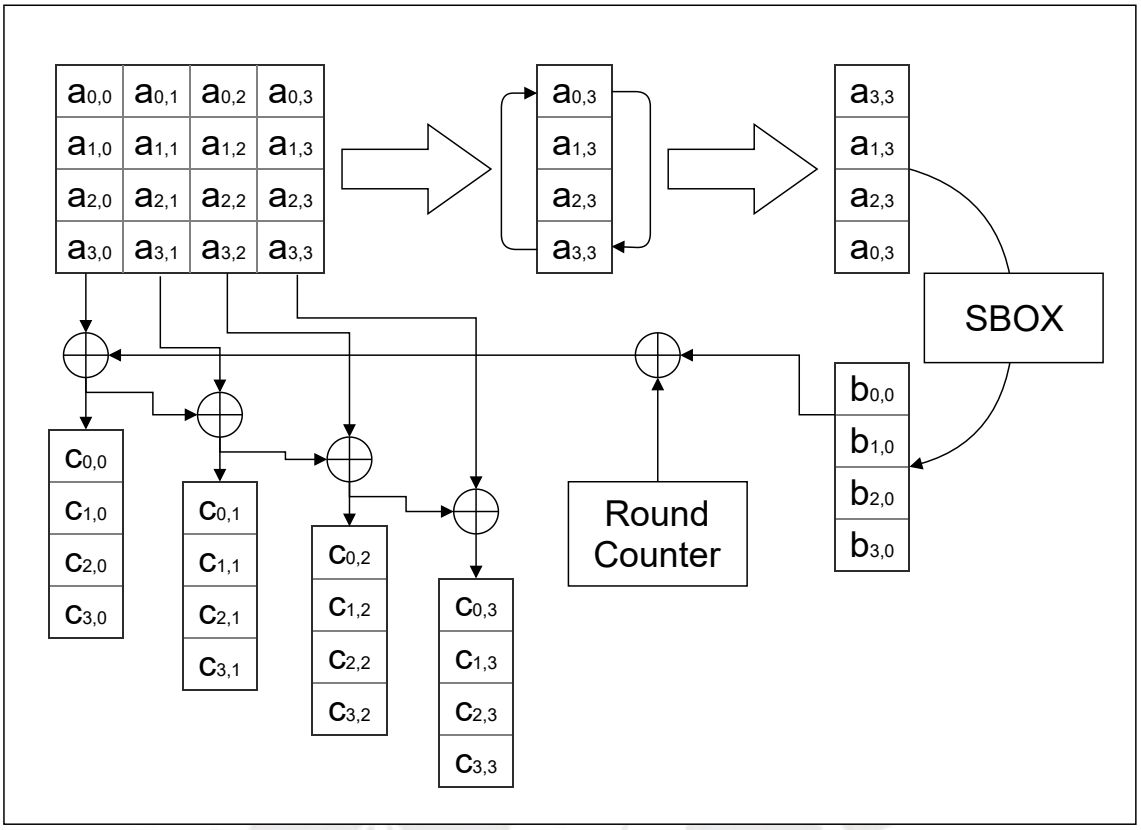

Figura 2.8: Bloque de expansión de la llave

Capítulo 3

Diseño del codificador/decodificador AES

El presente capítulo tienen por objetivo diseñar el bloque completo de codificación y decodificación AES de forma eficiente. Dentro del diseño del circuito se usarán técnicas de optimización en diseño de circuitos digitales como *pipeline*, FSMD (máquina de estados con *datapath*), ASM (máquina de estados algorítmica), entre otros.

3.1. Aspectos generales del diseño del bloque AES

La presente tesis busca realizar el diseño del bloque AES de 128 bits orientado a obtener la mayor velocidad de transmisión de datos y ocupar la menor área posible. Por tal motivo, se opta por realizar un diseño del tipo iterativo teniendo en consideración pequeñas etapas de *pipeline* para mejorar la frecuencia de operación.

Con respecto al diseño del bloque AES, este posee una interfaz de comunicación HandShake[24], por lo que se requiere que los bloques a los cuales va a ser conectado posean el mismo protocolo de información para garantizar la funcionalidad del mismo. Por otro lado, el presente diseño no considera el desarrollo del bloque KeySchedule (generación de las llaves de cifrado), este va a ser sustituido por un bloque combinacional donde se toman como constantes a todas las llaves de cifrado por cada llave original.

Actualmente, existen diversos tipos del diseño del bloque AES, y estos varían de acuerdo a la estandarización de las entradas y salidas, por ello esta estandarización se considera sumamente importante dentro del diseño general del bloque AES, ya que este posee operaciones por columna de bloque las cuales pueden afectar de forma significativa los resultados finales. Por tal motivo, la

presente tesis opta por la estandarización de entradas y salidas utilizado por el NIST [3].
Teniendo como dato de entrada $data\_in$ de 128 bits (expresión 3.1), este posee su representación por bloque de acuerdo con la Figura 3.1.

$$data\_in = \{input_{15},\ input_{14},\ input_{13},\ input_{12},\ \ldots,\ input_3,\ input_2,\ input_1,\ input_0\} \quad (3.1)$$

Como se puede observar en la expresión 3.1, $data\_in$ está conformado por 16 bytes expresados por $input\_i$, donde i puede tomar valores de 0 a 15, ver Figura 3.1.

input_15	input_11	input_7	input_3
input_14	input_10	input_6	input_2
input_13	input_9	input_5	input_1
input_12	input_8	input_4	input_0

Figura 3.1: Representación del dato de entrada ($data\_in$) en forma de bloque

De igual forma se puede expresar la llave de entrada key de 128 bits (expresión 3.2) en forma de bloque, considerando cada byte como key_i, donde i puede tomar valores de 0 a 15, ver Figura 3.2.

$$key = \{key_{15},\ key_{14},\ key_{13},\ key_{12},\ \ldots,\ key_3,\ key_2,\ key_1,\ key_0\} \quad (3.2)$$

key_15	key_11	key_7	key_3
key_14	key_10	key_6	key_2
key_13	key_9	key_5	key_1
key_12	key_8	key_4	key_0

Figura 3.2: Representación de la llave (key) en forma de bloque

En la decodificación del algoritmo AES, presentado en la sección 2 y Figura 2.1, el flujo de las transformaciones difiere respecto al flujo de la codificación, mientras que la generación de las llaves de cifrado para la codificación y decodificación siguen siendo las mismas. Sin embargo, diversas propiedades del algoritmo AES permiten que el flujo de decodificación sea equivalente al

flujo de codificación con las transformaciones reemplazadas por sus inversas. Esto se puede dar con un cambio en el bloque KeySchedule (generación de las llaves de cifrado). Las dos propiedades que permiten este cambio son:

- Las transformaciones **SubBytes()** y **ShiftRows()** pueden ser conmutadas; esto significa que la transformación **ShiftRows()** después de la transformación **SubBytes()** es equivalente a la transformación **SubBytes()** después de la transformación **ShiftRows()**. Lo mismo se puede afirmar respecto a sus inversas, **InvSubBytes()** e **InvShiftRows**.

- Las transformaciones **MixColumns()** e **InvMixColumns()** son lineares respecto a la columna de entrada. Esto significa:
  InvMixColumns (dato XOR llave generada) = **InvMixColumns** (dato) XOR **InvMixColumns** (llave generada).

Estas propiedades permiten que el orden de las transformaciones **InvSubBytes()** e **InvMixColumns()** puedan ser intercambiadas. Por otro lado, el orden de las transformaciones de **AddRoundKey()** e **InvMixColumns()** pueden también ser intercambiados, siempre que en la decodificación, los datos de las llaves generadas sean modificados usando la transformación **InvMixColumns()**[3].
El resultado de realizar estos cambios permite tener el flujo de decodificación más eficiente, ya que el diseño para realizar la codificación y decodificación en un solo bloque poseen el mismo flujo. Asimismo, se debe reestructurar el bloque KeySchedule (generación de las llaves de cifrado) para garantizar el correcto funcionamiento del bloque AES.

3.2. Diseño del bloque AddRoundKey

El comportamiento interno del bloque AddRoundKey consiste en realizar la operación *or* exclusiva entre el dato de entrada y la llave correspondiente al dato.
La expresión matemática de este bloque se muestra en la expresión 3.3.

$$data\_out \; = \; data\_in \oplus key \tag{3.3}$$

Asimismo se puede obtener el diagrama lógico del bloque, tal como se muestra en la Figura 3.3.

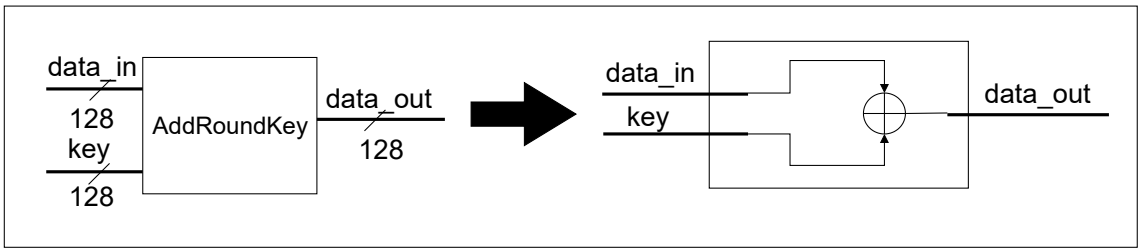

Figura 3.3: Diagrama lógico del Bloque AddRoundKey

3.3. Diseño de los bloques Subbytes e InvSubBytes

3.3.1. Alternativas del diseño del bloque SubBytes e InvSubBytes

El comportamiento interno del bloque SuBytes e InvSubBytes, está conformado por la multiplicación inversa en $GF(2^8)$. El procesamiento que realiza este bloque es complicado en comparación con los demás bloques del algoritmo AES, es por ello que existen diversos métodos para abordar este problema, como el uso de una memoria ROM para el almacenamiento de los valores de la tabla SBOX e InvSBOX, el uso del algoritmo Euclidiano Extendido y el diseño combinacional optimizando el diseño por medio de aritmética modular.

3.3.2. Memoria ROM

El uso de una memoria ROM dentro de una implementación en hardware es provechoso si se emplea en un FPGA, ya que físicamente se tiene una mayor cantidad de LUTs. Cada una de estas LUT posee una representación de tabla donde se proporciona una salida a una entrada determinada, a nivel lógico se utilizan funciones booleanas definidas arbritariamente. Como el objetivo de la tesis está enfocado en el diseño de un circuito integrado, no es conveniente utilizarlo, ya que no se tienen LUT a utilizar. [1].

3.3.3. Algoritmo Euclidiano Extendido

El algoritmo Euclidiano Extendido es una variación del algoritmo Euclidiano y se utiliza para hallar la multiplicación inversa de un número primo. Este algoritmo es capaz de hallar la multiplicación inversa en $GF(2^8)$, considerando la expresión 3.4

$$\text{Sea: } b(x) \text{ con módulo } a(x)$$

$$\text{La multiplicación inversa existirá solo si: } a(x) > b(x) \text{ y si: } [a(x), b(x)] = 1 \qquad (3.4)$$

Debido a la gran cantidad de iteraciones que utiliza el algoritmo Euclidiano Extendido se torna inadecuado para una implementación en hardware. [1],[25]

3.3.4. Diseño combinacional

Actualmente, para el diseño de la multiplicación inversa en $GF(2^8)$ se recurre al diseño combinacional dividiendo la operación en componentes de $GF(2^4)$, ya que este procedimiento es más óptimo para la implementación en hardware. [1]

El bloque SubBytes se define como la transformación inversa en $GF(2^8)$, pero estos a su vez necesitan de dos matrices propias del algoritmo, las cuales son llamadas ***affine tranformation*** e ***inverse affine transformation***. Estas transformaciones están definidas por las expresiones 3.5 y 3.6 respectivamente.

$$\begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} * \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix} \oplus \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix} \tag{3.5}$$

$$\begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \end{pmatrix} * \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix} \oplus \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \tag{3.6}$$

A partir de estas expresiones se puede obtener los diagramas lógicos, tal como se muestra en las Figuras 3.4 y 3.5.

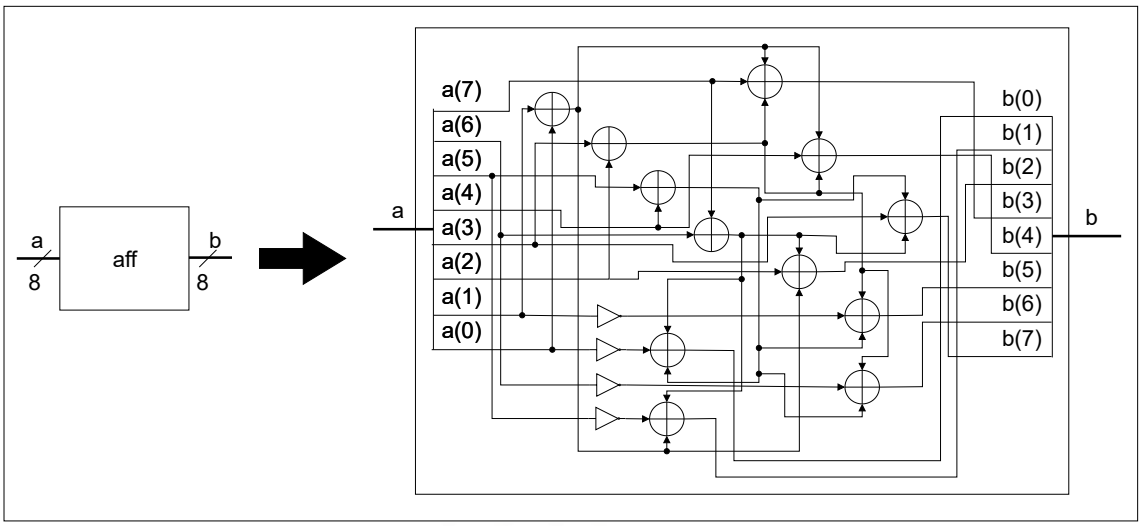

Figura 3.4: Diagrama lógico del bloque affine transformation

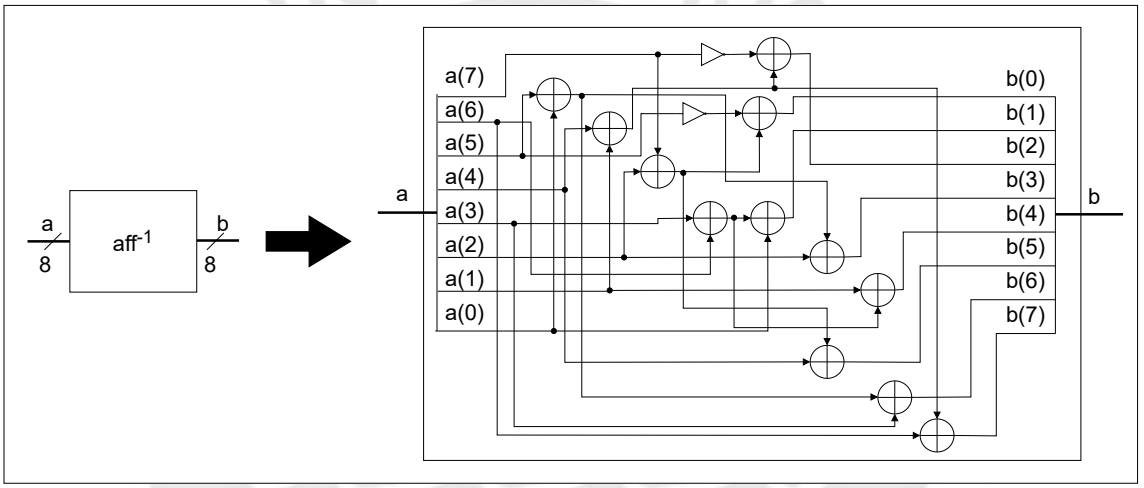

Figura 3.5: Diagrama lógico del bloque inverse affine transformation

Respecto a la multiplicación inversa en $GF(2^8)$, este se puede descomponer en una multiplicación inversa del campo $GF((2^4)^2)$. Para poder realizar operaciones en campo finito $GF(2^4)$ y estos a su vez sean equivalentes en el campo finito $GF(2^8)$, se necesita una matriz de cambio de espacio, la cual representa la transición de campos. Para calcular estas matrices se toma como referencia el polinomio irreductible dado por la ecuación 3.7 y estas se llaman ***mapping transformation*** e ***inverse mapping transformation***.

$$n(x) = x^2 + \{1\}x + \{e\} \tag{3.7}$$

Como consecuencia del cambio de espacios, todas las operaciones que se realicen en el campo finito $GF(2^4)$ necesitarán de un polinomio irreductible dado por la expresión 3.8 en caso sea

necesario.

$$m_4(x) = x^4 + x + 1 \tag{3.8}$$

Las representaciones matriciales de ***mapping transformation*** e ***inverse mapping transformation*** están representadas por las expresiones 3.9 y 3.10 respectivamente.

$$\begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \end{pmatrix} \oplus \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix} \tag{3.9}$$

$$\begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \end{pmatrix} \oplus \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix} \tag{3.10}$$

Del mismo modo, se puede obtener su diagrama lógico, tal como se muestra en las Figuras 3.6 y 3.7.

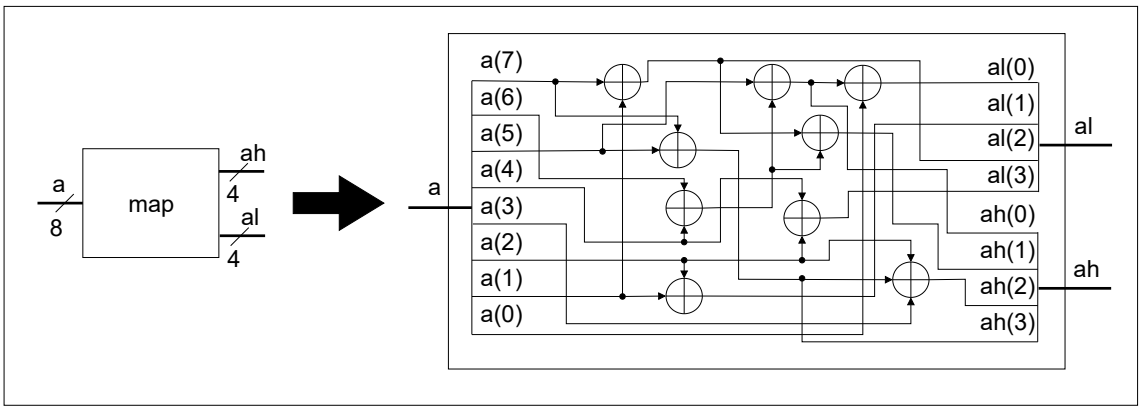

Figura 3.6: Diagrama lógico del bloque mapping transformation

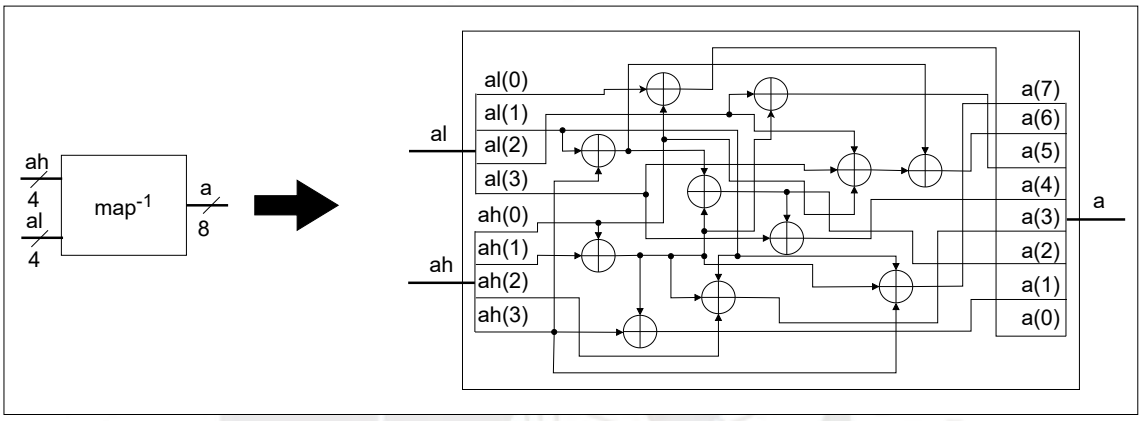

Figura 3.7: Diagrama lógico del bloque inverse mapping transformation

Para hallar la multiplicación inversa en campo finito $GF(2^4)$ se utiliza la propiedad del Algoritmo Euclidiano Extendido, el cual está definido por la expresión 3.11.

$$gcd(a, b) = ax + by \tag{3.11}$$

Para que esta propiedad ayude a calcular la multiplicación inversa, el valor de $gcd(a, b)$ debe ser igual a 1, x debe ser igual al polinomio irreductible dado por la expresión 3.7 e y debe ser remplazado por el dato de entrada ($s(x) = s_h * x + s_l$)[2] en campo finito $GF(2^4)$. El resultado de la multiplicación inversa esta dado por b. A continuación se muestra los cambios realizados en la expresión 3.12.[1], [26]

$$1 = a * n(x) + b * s(x) \tag{3.12}$$

A partir de la expresión 3.12 se puede reescribir el polinomio $n(x)$ en función de $s(x)$, tal como se muestra en la expresión 3.13. Donde $q(x)$ y $r(x)$ representa el cociente y el residuo

[2] s_h representa los cuatro bits más significativa de $s(x)$ y s_l representa los cuatro bits menos significativos de $s(x)$

respectivamente.

$$n(x) = q(x) * s(x) + r(x) \tag{3.13}$$

Después de realizar la división entre $n(x)$ y $s(x)$, se obtienen los valores de $q(x)$ y $r(x)$, tal como se muestra en las expresiones 3.14 y 3.15 respectivamente.

$$q(x) = s_h^{-1} * x + (1 + s_h^{-1} * s_l) * s_h^{-1} \tag{3.14}$$

$$r(x) = \{e\} + (1 + s_h^{-1} * s_l) * s_h^{-1} * s_l \tag{3.15}$$

Substituyendo $q(x)$ y $r(x)$ en la expresión 3.13 y multiplicando por s_h^2 se obtiene la expresión 3.16.

$$s_h^2 * n(x) = (s_h * x + (s_h + s_l)) * s(x) + (s_h^2 * \{e\} + s_h * s_l + s_l^2) \tag{3.16}$$

Remplazando θ por $(s_h^2 * \{e\} + s_h * s_l + s_l^2)$, se multiplica la expresión 3.16 por θ^{-1} y se obtiene la expresión 3.17.

$$\theta * s_h^2 * n(x) + \theta * (s_h * x + (s_h + s_l)) * s(x) = 1 \tag{3.17}$$

Al comparar la expresión 3.17 con la expresión 3.12 se obtiene el resultado de la multiplicación inversa, la cual esta dada por la expresión 3.18.

$$b = s(x)^{-1} = s_h * \theta * x + (s_h + s_l) * \theta \tag{3.18}$$

De acuerdo a la expresión 3.18 se puede obtener el diseño lógico de todos los bloques de la multiplicación inversa, las cuales están representadas por las Figuras 3.8 - 3.10.

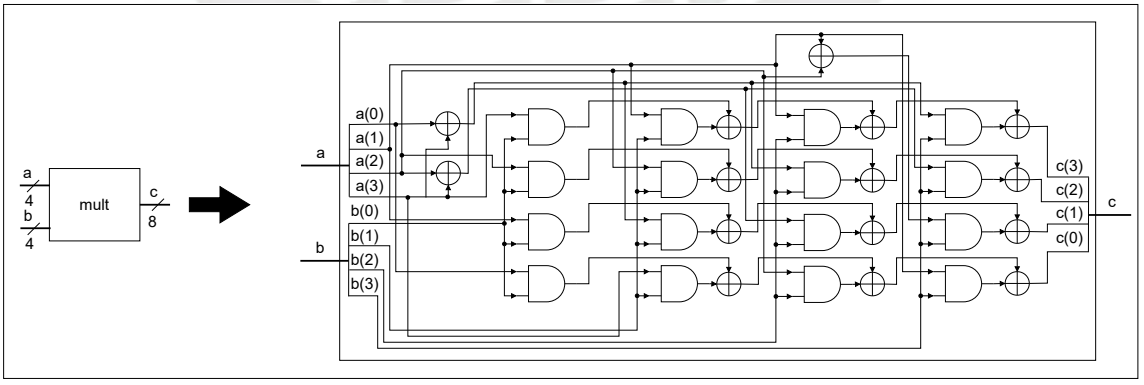

Figura 3.8: Diagrama lógico del bloque multiplicación en campo finito $GF(2^4)$

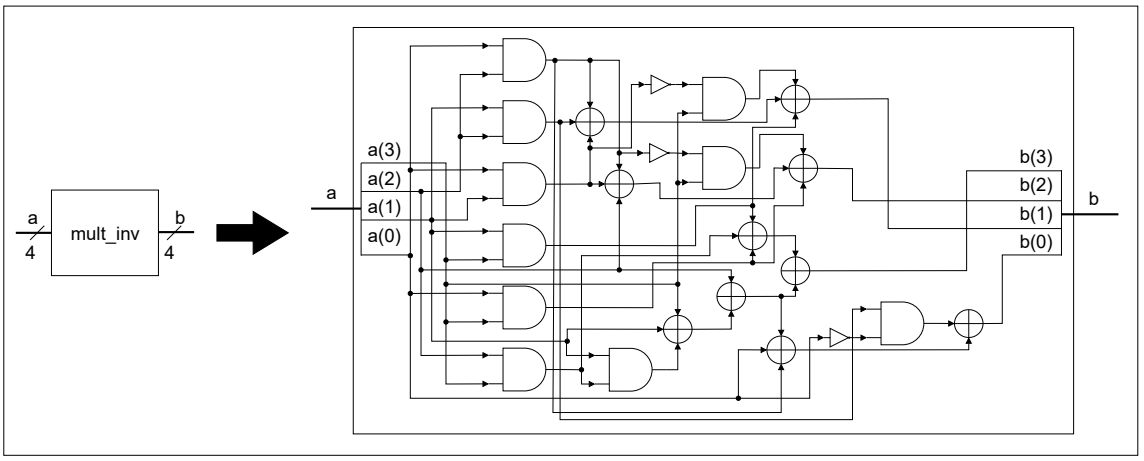

Figura 3.9: Diagrama lógico del bloque multiplicación inversa en campo finito $GF(2^4)$

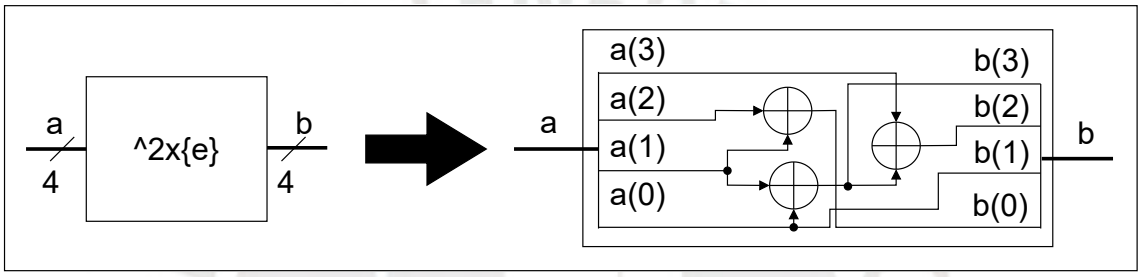

Figura 3.10: Diagrama lógico del bloque elevación cuadrada y multiplicación por la constante {e} en campo finito $GF(2^4)$

A partir de los bloques anteriores se realizó dos diseños del bloque SubBytes. El primer diseño muestra una optimización con *pipeline*; mientras que el segunda muestra una optimización con FSMD. Estos diseños se muestran en las sección 3.3.4.1 y 3.3.4.2 respectivamente.

3.3.4.1. Optimización usando pipeline

Se realizó una adaptación del circuito realizado por [1] colocando dos registros dentro del bloque, esto se observa en la Figura 3.11.

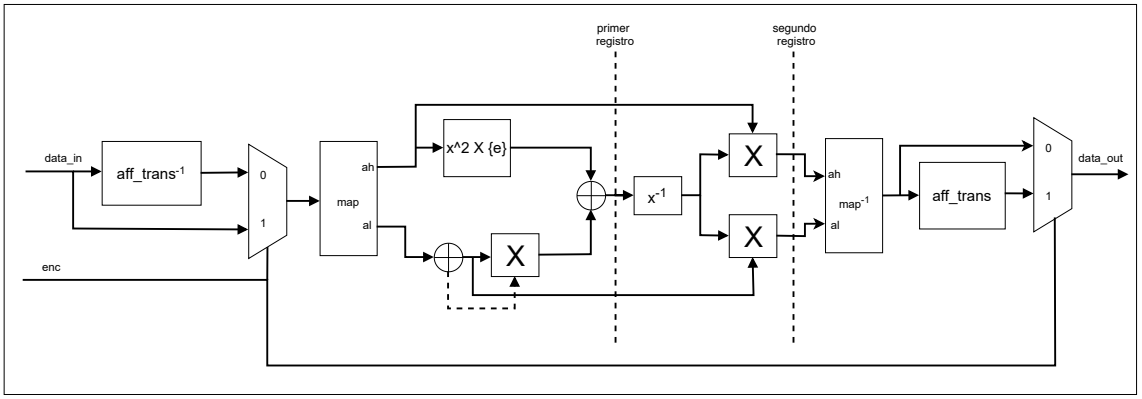

Figura 3.11: Diseño del bloque SubBytes usando la técnica *pipeline*

A partir de este diseño se obtuvieron los siguientes resultados en la Tabla 3.1

Tabla 3.1: Resultados del diseño usando pipeline

	Dispositivo	**Área (o)**	**Latencia (ns)**	**Máxima frecuencia de operación (MHz)**
Figura 3.11	Cyclone II EP2C35F672C6	91	3.64	274.26
Figura 3.11	Virtex IV XC4VLX60-12(ff668)	64	2.65	376.36

3.3.4.2. Optimización usando FSMD

Se realizó una segunda adaptación del bloque SubBytes e InvSubBytes utilizando la técnica de FSMD para optimizar el área del circuito, esto se observa en la Figura 3.12.

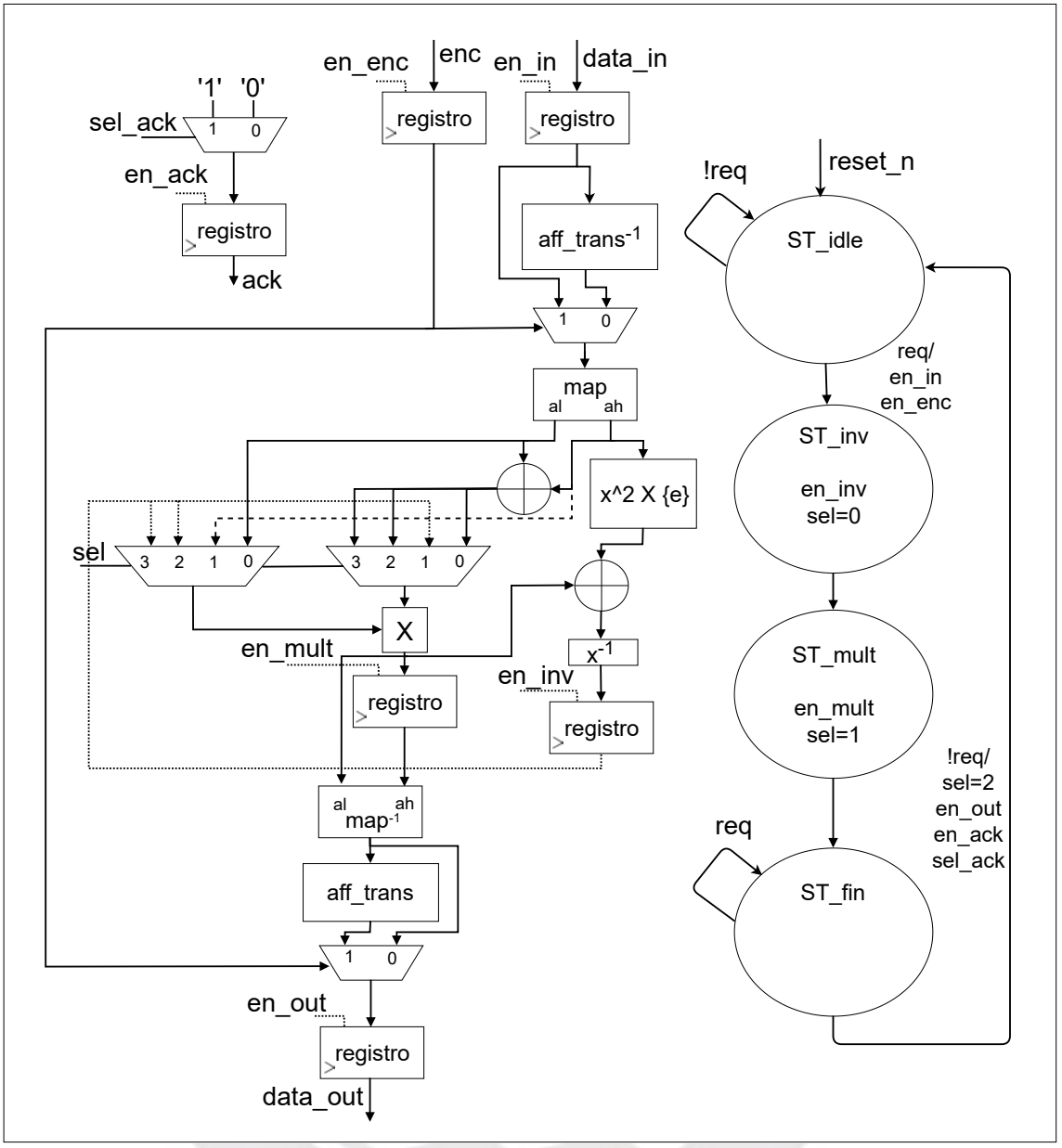

Figura 3.12: Diseño del bloque SubBytes usando FSMD

A partir de este diseño se obtuvieron los siguientes resultados en la Tabla 3.2.

Tabla 3.2: Resultados del diseño usando FSMD

	Dispositivo	**Área (o)**	**Latencia (ns)**	**Máxima frecuencia de operación (MHz)**
Figura 3.12	Cyclone II EP2C35F672C6	104	6.64	150.58
Figura 3.12	Virtex IV XC4VLX60-12(ff668)	70	6.04	165.53

Se concluye que se prefiere una optimización con *pipeline*, ya que consume una menor área y obtiene una mayor frecuencia de operación.

3.4. Diseño del bloque Mixcolumns e InvMixcolumns

Como se mencionó en el capítulo 2, los bloques Mixcolumns e Invmixcolumns se basan en multiplicaciones modulares con polinomios constantes definidos por las expresiones 3.19 y 3.20 respectivamente.

$$c(x) = 3x^3 + x^2 + x + 2 \tag{3.19}$$

$$c(x) = Bx^3 + Dx^2 + 9x + E \tag{3.20}$$

A partir de estas expresiones, se puede optar por realizar la multiplicación en forma matricial del bloque Mixcolumns e InvMixcolumns, tal como se muestran en las expresiones 3.21 y 3.22 respectivamente [2].

Multiplicación del bloque Mixcolumns:

$$\begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{pmatrix} = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \bullet \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix}$$

$$\begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{pmatrix} = \begin{pmatrix} 02 & 02 & 00 & 00 \\ 00 & 02 & 02 & 00 \\ 00 & 00 & 02 & 02 \\ 02 & 00 & 00 & 02 \end{pmatrix} \bullet \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix} \oplus \begin{pmatrix} 00 & 01 & 01 & 01 \\ 01 & 00 & 01 & 01 \\ 01 & 01 & 00 & 01 \\ 01 & 01 & 01 & 00 \end{pmatrix} \bullet \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix} \tag{3.21}$$

Multiplicación del bloque InvMixcolumns:

$$\begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{pmatrix} = \begin{pmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{pmatrix} \bullet \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix}$$

$$\begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{pmatrix} = \begin{pmatrix} 02 & 03 & 01 & 00 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \bullet \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix} \oplus \begin{pmatrix} 08 & 08 & 08 & 08 \\ 08 & 08 & 08 & 08 \\ 08 & 08 & 08 & 08 \\ 08 & 08 & 08 & 08 \end{pmatrix} \bullet \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix}$$

$$\oplus \begin{pmatrix} 04 & 00 & 04 & 00 \\ 00 & 04 & 00 & 04 \\ 04 & 00 & 04 & 00 \\ 00 & 04 & 00 & 04 \end{pmatrix} \bullet \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix} \tag{3.22}$$

Como se puede observar en las expresiones anteriores, dentro del bloque InvMixcolumns se tiene el bloque Mixcolumns, por lo que esta realimentación se considera la más óptima para una implementación en ASIC. A partir de esto, se deducen las ecuaciones correspondientes a los bloques Mixcolumns e InvMixcolumns, tal como se muestra en las expresiones 3.23 y 3.24 respectivamente.

Bloque Mixcolumns:

Considerando:

$$X_0 = a_0 \oplus a_1 \, , \, X_1 = a_1 \oplus a_2 \, , \, X_2 = a_2 \oplus a_3 \, , \, X_3 = a_3 \oplus a_0$$

Se tiene:

$$\begin{cases} b_0 &= 02 \bullet X_0 \oplus X_2 \oplus a_1 \\ b_1 &= 02 \bullet X_1 \oplus X_2 \oplus a_0 \\ b_2 &= 02 \bullet X_2 \oplus X_0 \oplus a_3 \\ b_3 &= 02 \bullet X_3 \oplus X_0 \oplus a_2 \end{cases} \tag{3.23}$$

Bloque InvMixcolumns:

Considerando:

$$Y_0 = 04 \bullet (a_1 \oplus a_3) \, , \, Y_1 = 04 \bullet (a_0 \oplus a_2) \, , \, Y_2 = 02 \bullet (Y_1 \oplus Y_0)$$

$$Z_0 = Y_2 \oplus Y_1 \, , \; Z_1 = Y_2 \oplus Y_0$$

Se tiene:

$$\begin{cases} c_0 & = b_0 \oplus Z_1 \\ c_1 & = b_1 \oplus Z_0 \\ c_2 & = b_2 \oplus Z_1 \\ c_3 & = b_3 \oplus Z_0 \end{cases} \tag{3.24}$$

Luego de obtener los resultados matemáticos de cada uno de los bloques se procede a realizar el diseño lógico de cada parte del circuito, tal como se muestra en las Figuras 3.13 - 3.18.

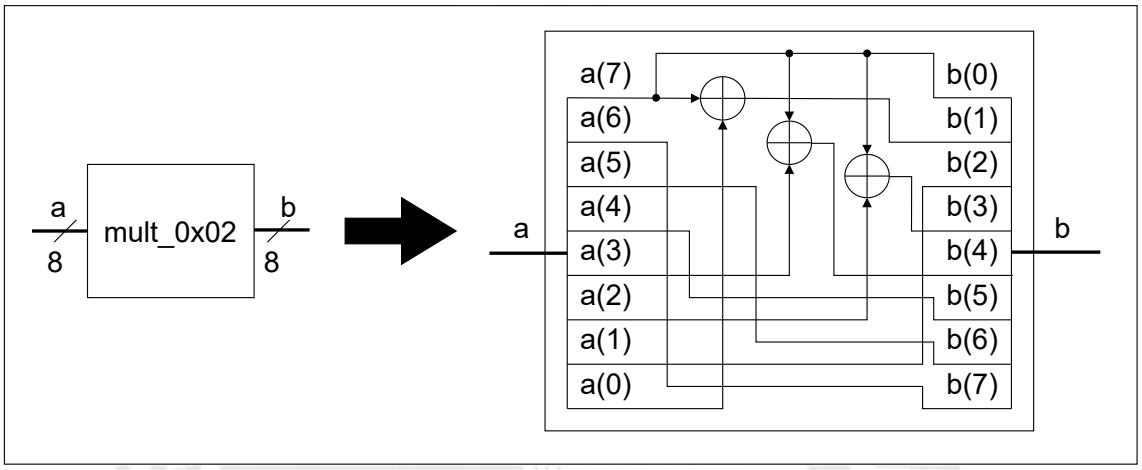

Figura 3.13: Diagrama lógico del bloque multiplicador por 02

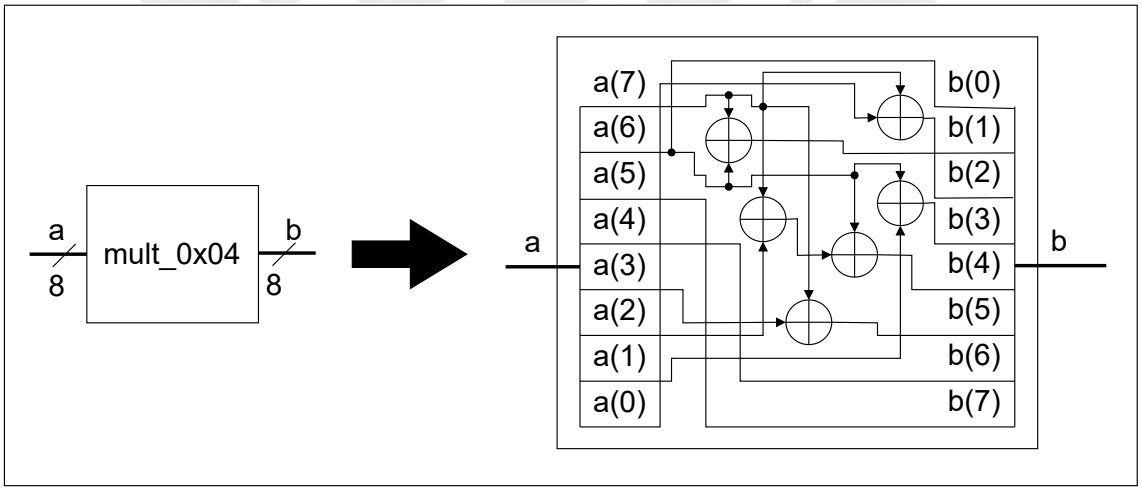

Figura 3.14: Diagrama lógico del bloque multiplicador por 04

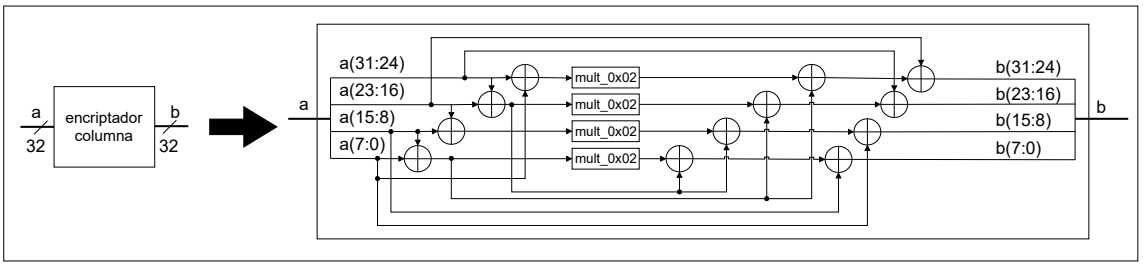

Figura 3.15: Diagrama lógico del bloque encriptador por columna

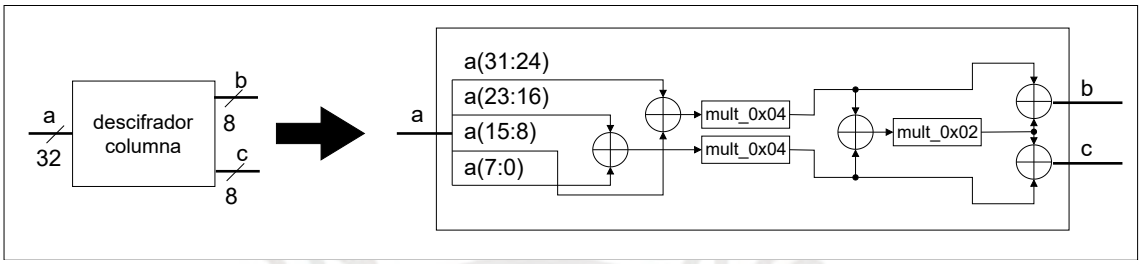

Figura 3.16: Diagrama lógico del bloque descifrador por columna

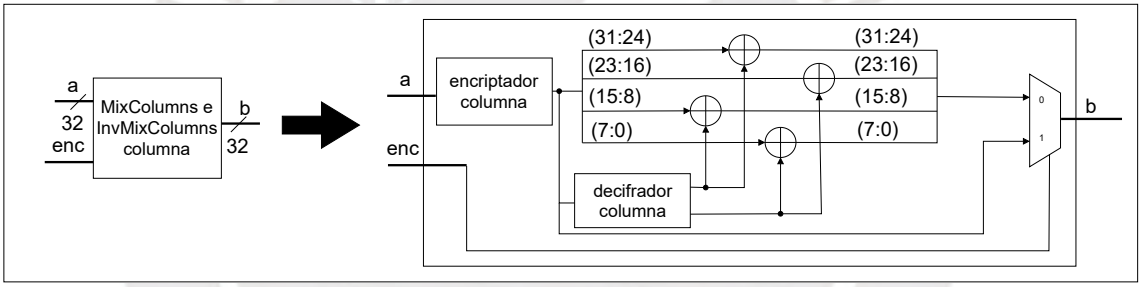

Figura 3.17: Diagrama lógico del bloque encriptador/descifrador por columna

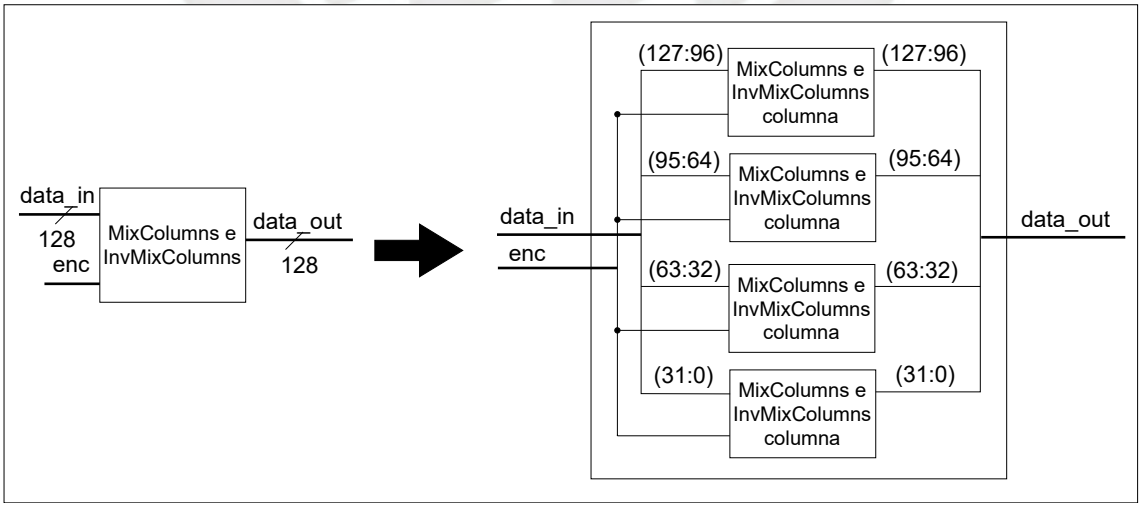

Figura 3.18: Diagrama lógico del bloque MixColumns e InvMixColumns

3.5. Diseño del bloque ShiftRows e InvShiftRows

El comportamiento interno de este bloque consiste en realizar desplazamiento de bytes por cada fila del bloque de entrada. Estos desplazamientos pueden realizarse a la derecha o a la izquierda dependiendo del modo de operación del bloque, en caso sea encriptación se realizará el desplazamiento hacia la izquierda, en caso contrario se realizará el desplazamiento hacia la derecha. El diseño lógico del circuito ShiftRows e InvShiftRows se puede apreciar en la Figura 3.19.

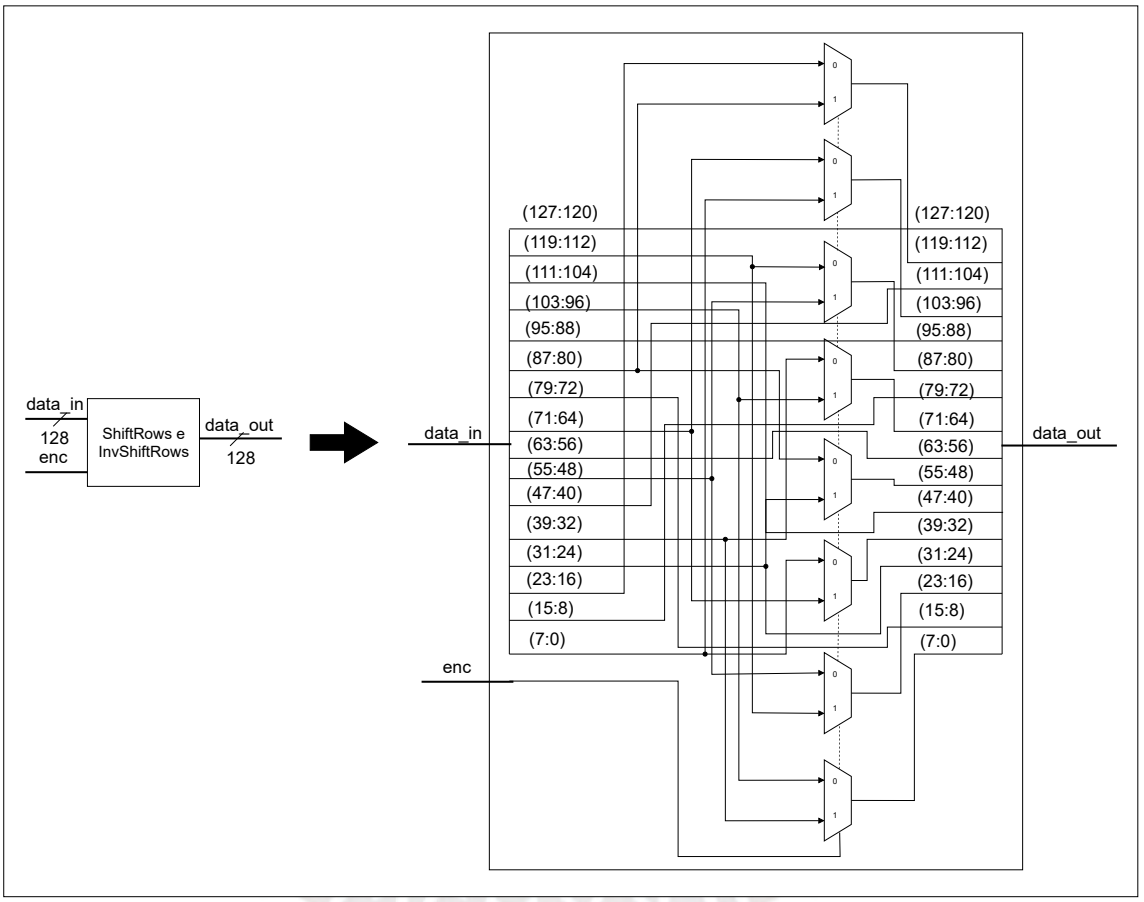

Figura 3.19: Diagrama lógico del Bloque ShiftRows e InvShiftRows

3.6. Diseño del bloque de codificación y decodificación del AES

Esta sección muestra el diseño RTL de dos arquitecturas diferentes usando las técnicas de FSMD (máquina de estado finita con *datapath*) y ASMD (máquina de estados algorítmica con *datapath*). Esto se debe a que se desea realizar una comparación en términos de área, velocidad de transmisión de datos (*throughput*) y máxima frecuencia de operación de estas arquitecturas.

3.6.1. Unión de los bloques ShiftRows e InvShiftRows, MixColumns e InvMixColumns y AddRoundKey

Debido a la complejidad del bloque SubBytes e InvSubBytes, existe una mayor probabilidad que este circuito limite la velocidad de transmisión de datos y la frecuencia de operación del circuito completo, por lo que se opta en realizar la unión de los bloques ShiftRows e InvShiftRows, MixColumns e InvMixColumns y AddRoundKey, los cuales poseen una lógica menos complicada en comparación con el bloque SubBytes e InvSubBytes, esto se realiza con la finalidad de no perder ciclos de operación en el procesamiento de datos.

La Figura 3.20 muestra la unión de los bloques ShiftRows e InvShiftRows, MixColumns e InvMixColumns y AddRoundKey dependiendo de la respectiva lógica del algoritmo. Para ello se tiene un selector (*sel*), el cual se encarga de controlar las etapas del algoritmo AES. Además, se puede apreciar que cuando *sel* sea cero, el bloque funcionará como la etapa inicial del algoritmo. De otra manera, cuando *sel* sea uno, el bloque funcionará como la etapa intermedia del algoritmo. Por último, cuando *sel* sea dos o tres, el bloque funcionará como la etapa final del circuito.

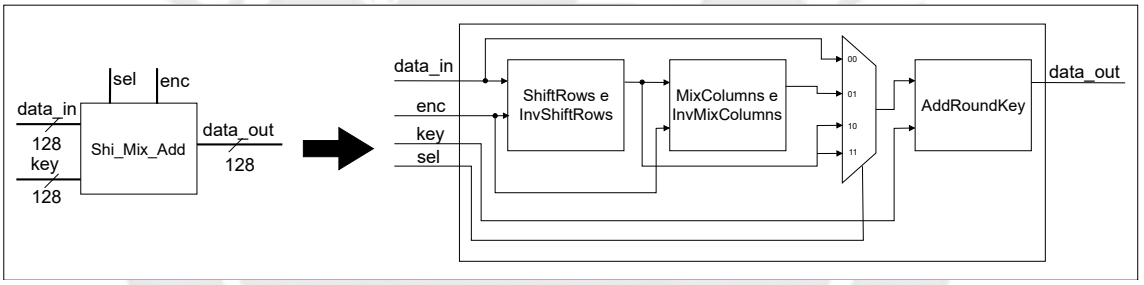

Figura 3.20: Diagrama lógico del Bloque Shift_Mix_Add

3.6.2. Diseño del bloque AES usando FSMD

La Figura 3.21 muestra el diseño del datapath propuesto. A partir de este diseño, se puede observar que se utiliza un contador de módulo 10 para contar el número de iteraciones del algoritmo. Adicionalmente, se tiene un contador de módulo N, el cual indica la cantidad de etapas de *pipeline* que posee el bloque Subbytes e InvSubBytes. Para este diseño se consideran valores de N igual a cero, uno y dos. Asimismo, las entradas están siendo registradas, esto permite garantizar el funcionamiento del bloque durante los ciclos del procesamiento de información.

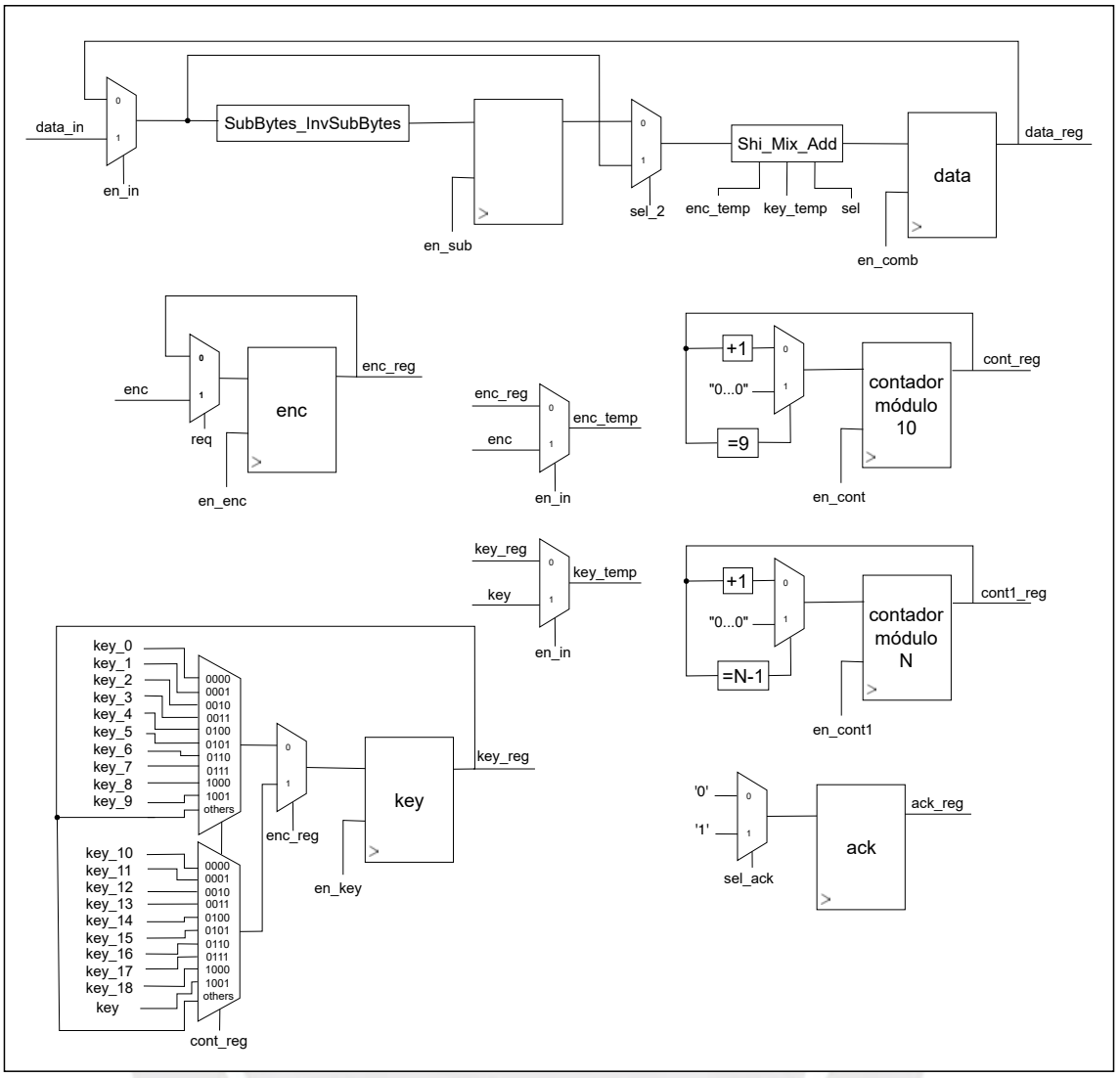

Figura 3.21: Datapath del bloque AES usando FSMD

La Figura 3.22 muestra el diseño de la máquina de estados propuesta. A partir de este diseño, se puede observar que se tienen cuatro estados, los cuales son:

- **ST_idle**: Estado en el cual espera a que la señal req se habilite para comenzar el procesamiento de datos. En caso la señal req no se active el circuito no procesará la información.

- **ST_proc_1**: Este estado representa la operación del bloque Shi_Mix_Add.

- **ST_proc_2**: Este estado representa la operación del bloque SubBytes e InvSubBytes.

- **ST_last**: Estado en el cual espera a que la señal req se deshabilite para actualizar el dato de salida. En caso la señal req se encuentre habilitada, no se actualizarán las salidas.

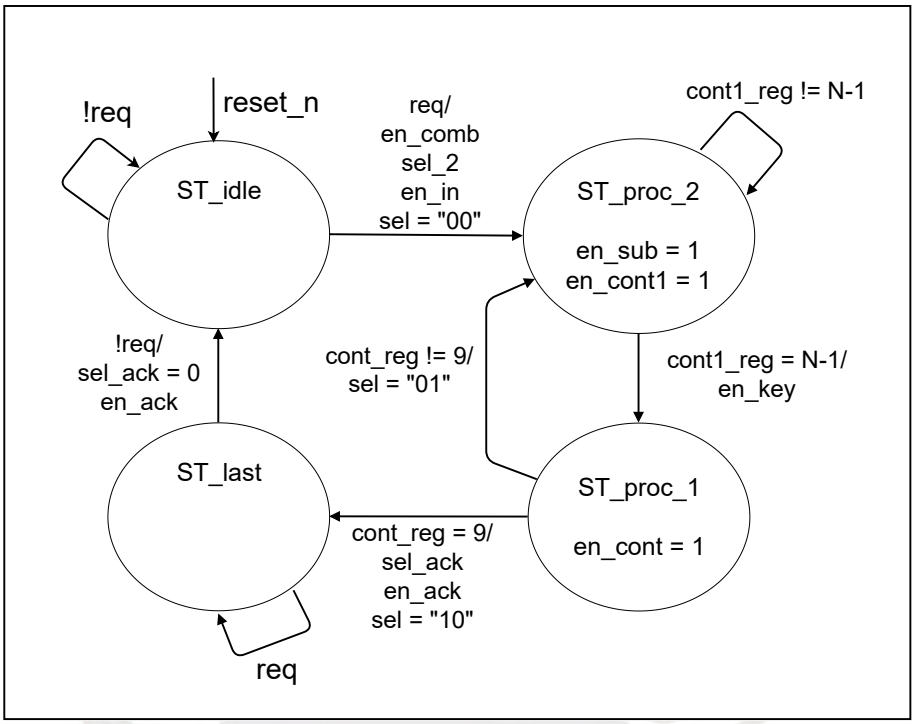

Figura 3.22: Máquina de estados del bloque AES usando FSMD

3.6.3. Diseño del bloque AES usando ASMD

La Figura 3.23 muestra el diseño del datapath propuesto usando un diseño del tipo ASM. Se observa que se utiliza un contador de módulo 10 para contar el número de iteraciones del algoritmo. Adicionalmente, se tiene un contador de módulo N, el cual indica la cantidad de etapas de *pipeline* que posee el bloque Subbytes e InvSubBytes. Para este diseño se consideran valores de N igual a cero, uno y dos. En comparación al datapath propuesto en la sección 3.6.3, este diseño cuenta con multiplexores a la entrada de cada registro, los cuales dependen del estado registrado de la máquina de estados. Esto permite realizar un diseño más flexible. Asímismo, se puede observar que las entradas están siendo registradas, esto permite garantizar el funcionamiento del bloque durante los ciclos del procesamiento de información.

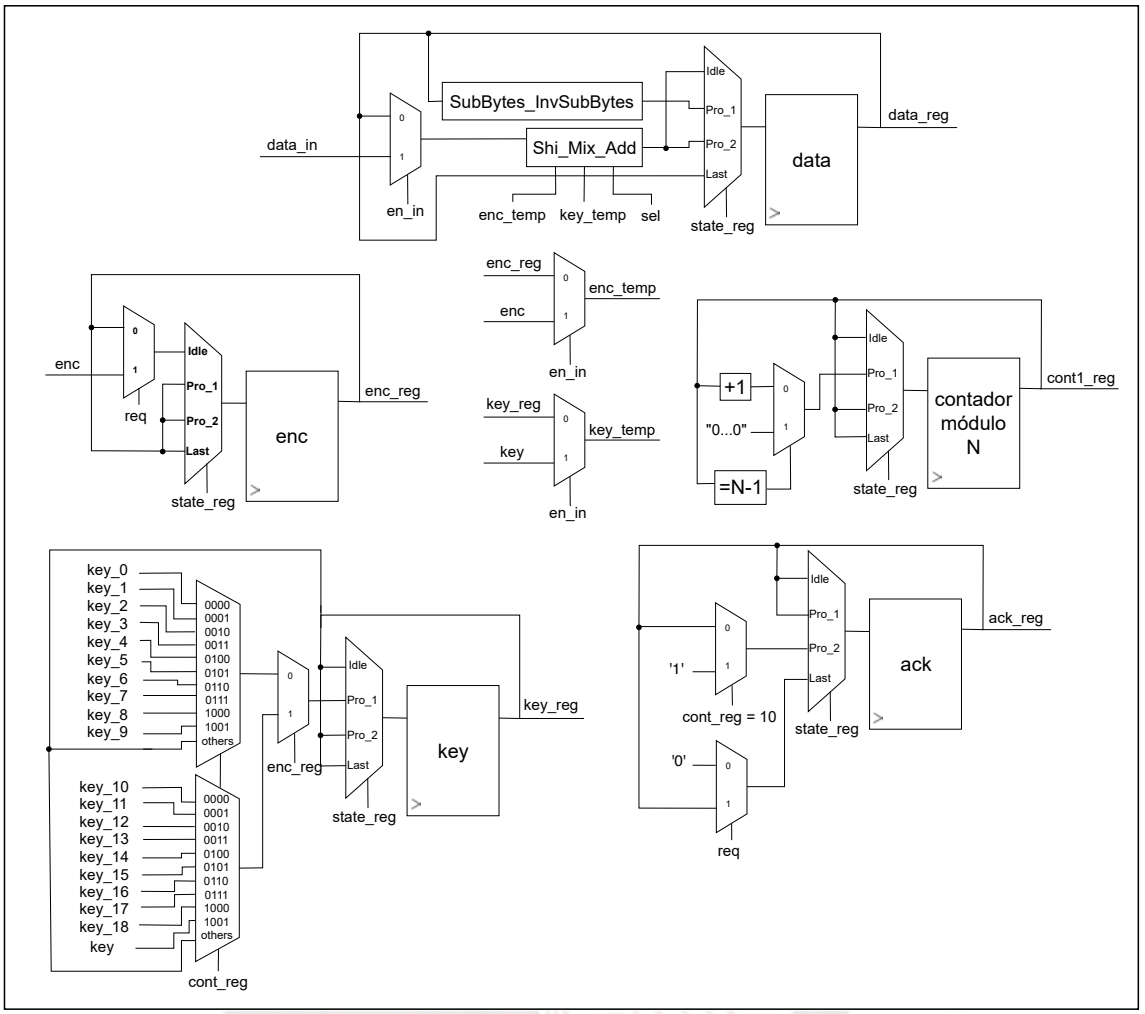

Figura 3.23: Datapath del bloque AES usando ASMD

La Figura 3.24 muestra el diseño de la máquina de estados propuesta usando el diseño del tipo ASM. A partir de este diseño, se puede observar que se tienen cuatro estados, los cuales son:

- **ST_idle**: Estado en el cual espera a que la señal req se habilite para comenzar el procesamiento de datos. En caso la señal req no se active el circuito no procesará la información.

- **ST_proc_1**: Este estado representa la operación del bloque SubBytes e InvSubBytes.

- **ST_proc_2**: Este estado representa la operación del bloque Shi_Mix_Add.

- **ST_last**: Estado en el cual espera a que la señal req se deshabilite para actualizar el dato de salida. En caso la señal req se encuentre habilitada, no se actualizarán las salidas.

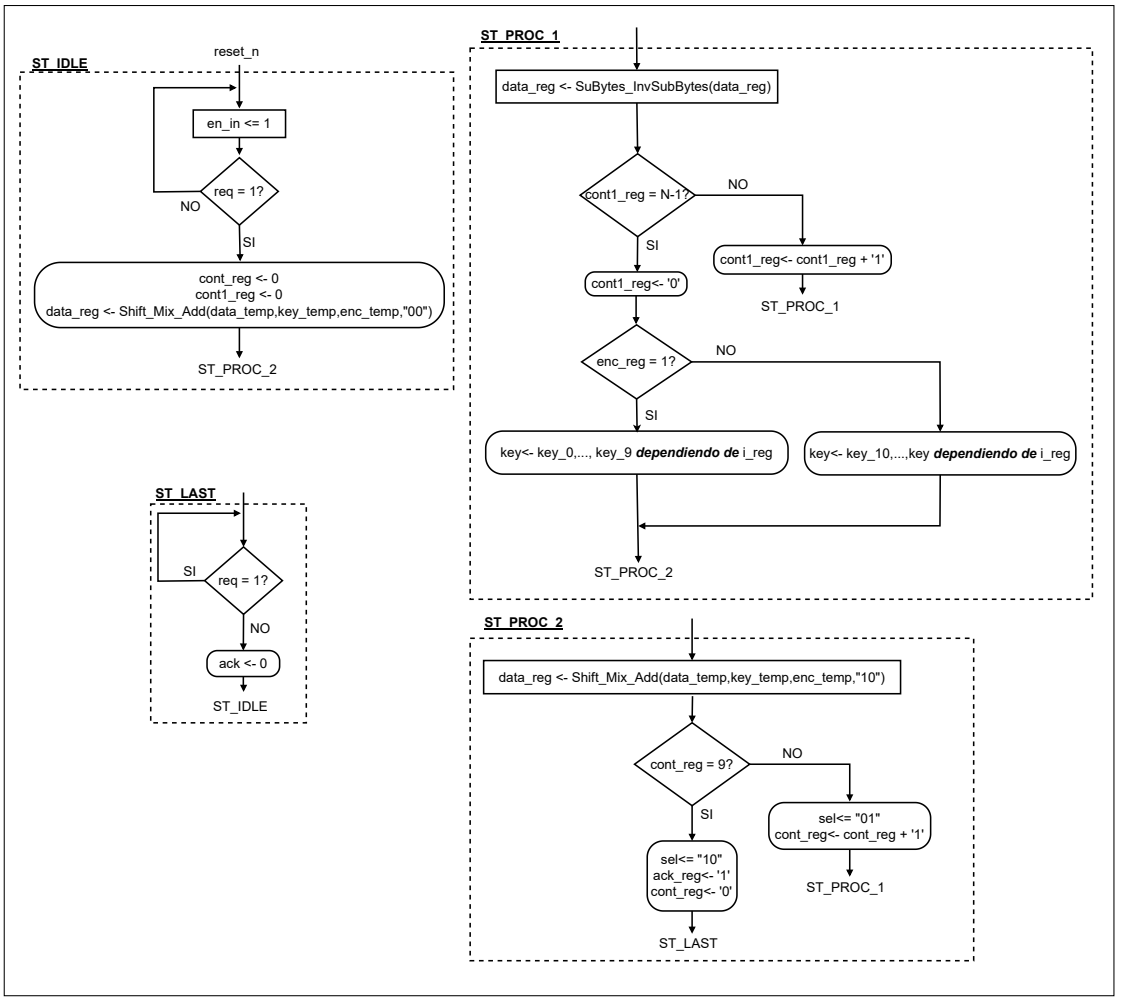

Figura 3.24: Máquina de estados del bloque AES usando ASMD

En comparación con el diseño de la sección 3.6.3, este diseño permite una una mejor visualización de las salidas por cada estado. Dentro de la Figura 3.24, los rectángulos muestran las salidas del tipo Moore y los rectángulos ovalados muestran las salidas del tipo Mealy. Asimismo, se puede observar que las flechas ($\leftarrow$) representan la actualización de un registro de manera secuencial; mientras que, las flechas ($\Leftarrow$) muestran la actualización de los registros de manera combinacional. Estas consideraciones, permiten optimizar el circuito a nivel de reducción de estados, ya que se puede diseñar una máquina de estados con salidas del tipo Moore y Mealy.

Capítulo 4

Resultados

4.1. Verificación funcional del bloque AES

De acuerdo con las arquitecturas planteadas del bloque AES en el capítulo 3 sección 3.3. Los bloques fueron simulados con la herramienta de simulación ModelSim y se obtuvieron los siguientes resultados. La Figura 4.1 muestra las entradas del bloque AES. Las Figuras 4.2 y 4.3 muestran las salidas del bloque AES en la primera iteración y la salida final, respectivamente. Cada una de estas salidas se compararon con el ejemplo del algoritmo mostrado en el documento del estándar de encriptación AES del NIST [3] y se puede observar que estas coinciden con los resultados de los diseños planteados en la sección 3.

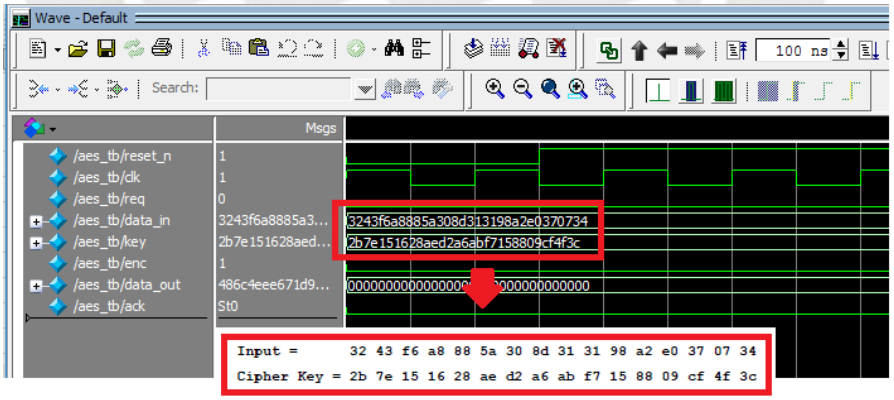

Figura 4.1: Simulación y comparación de entradas del bloque AES con el documento del NIST [3]

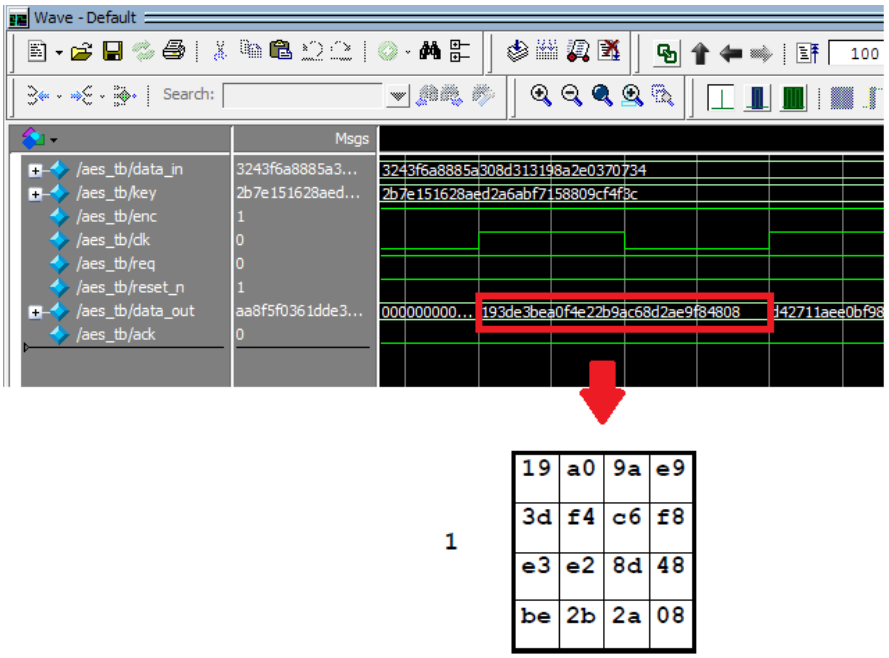

Figura 4.2: Simulación y comparación de la salida del bloque AES en la primera iteración con el documento del NIST [3]

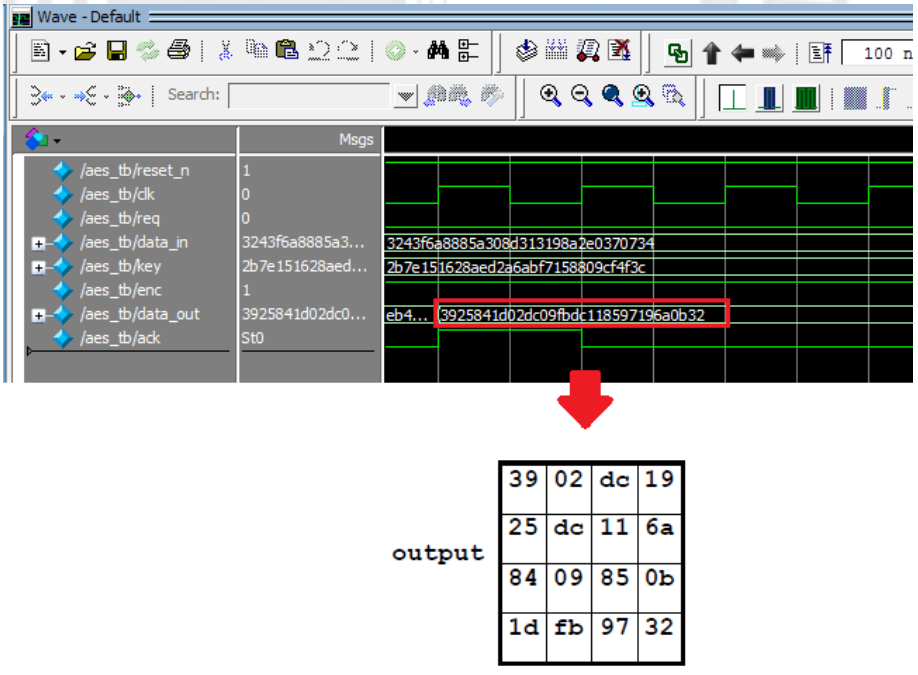

Figura 4.3: Simulación y comparación de las salidas finales del bloque AES con el documento del NIST [3]

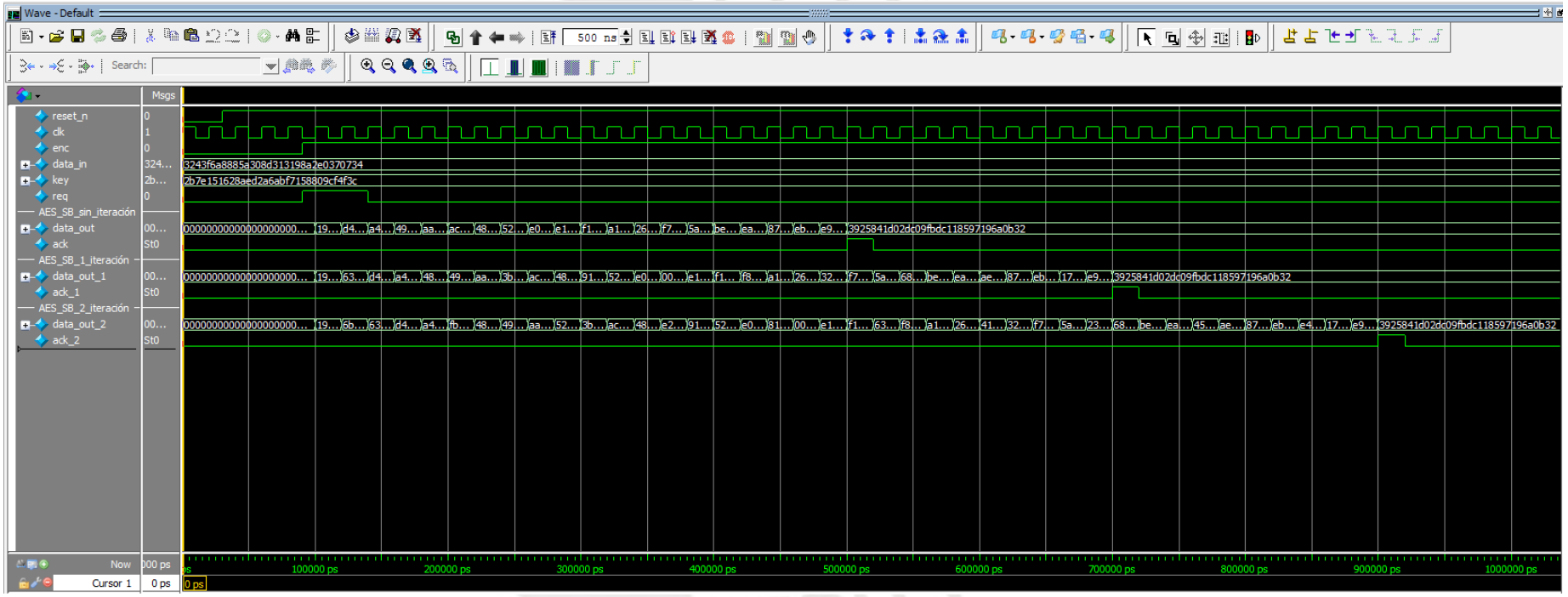

Figura 4.4: Comparación de las salidas del bloque AES considerando cero, una y dos etapas de *pipeline* en el bloque SubBytes

De manera análoga, se realizó la simulación del bloque AES considerando cero, una y dos etapas de *pipeline* en el bloque SubBytes (Figura 4.4), donde **data_out** y **ack** representan las salidas del bloque AES sin considerar ninguna etapa de *pipeline*. Las salidas **data_out_1** y **ack_1** representan las salidas del bloque AES considerando una etapa de *pipeline*. Por último, las salidas **data_out_2** y **ack_2** representan las salidas del bloque AES considerando dos etapas de *pipeline*. Como se puede observar en la imagen, los resultados de cada uno de estos bloque son iguales; sin embargo, existe retardos de 10 y 20 ciclos de reloj en los bloques AES con una y dos etapas de *pipeline* respectivamente, esto se debe a que cada etapa de *pipeline* crea un retardo de un ciclo de reloj en la salida. Considerando que el bloque SubBytes posee 10 iteraciones dentro del bloque completo del AES se puede comprobar que para una etapa de *pipeline* en el bloque SubBytes tendría por consecuencia un retardo de 10 ciclos de reloj en la salida del bloque completo. De igual manera ocurre cuando se tiene dos etapas de *pipeline*, se crea un retardo de 20 ciclos de reloj en la salida del bloque completo. Los retardos de ciclo de reloj traen como consecuencia una reducción en el *troughput*; sin embargo, se obtiene una mejora en la frecuencia de operación. En la sección 4.2 se puede observar como varían estos parámetros de acuerdo a las diferentes etapas de pipeline utilizadas.

4.2. Análisis de síntesis del bloque AES

Para realizar el análisis de síntesis del bloque AES se utilizaron las plataformas Cyclone II (TSMC 90 nm) con la herramienta Quartus de la empresa Intel y Virtex IV (UMC 90 nm) con la herramiento ISE de la empresa Xilinx. Dentro del análisis se consideran parámetros de área en LUT en el caso de utilizar la plataforma Virtex IV y LE en caso de utilizar la plataforma Cyclone II, también se consideraron parámetros a evaluar como máxima frecuencia de operación, velocidad de transmisión de datos (*throughput*), eficiencia, latencia y latencia por ciclos de reloj. Para obtener los resultados de velocidad de transmisión de datos (*throughput*) y eficiencia se utilizaron las expresiones 4.1 y 4.2 respectivamente [27], [28].

$$Throughput = (\frac{1}{máxima\ frecuencia\ de\ operación} * latencia\ por\ ciclos\ de\ reloj)^{-1} * 128 \tag{4.1}$$

$$Eficiencia = \frac{Throughput}{Área} \tag{4.2}$$

Tabla 4.1: Resultados del análisis de síntesis del bloque AES

	Dispositivo	LUT/LE	Frecuencia (MHz)	Throughput (Mbits/s)	Eficiencia (Mbits/(LUT o LE))	Latencia (ns)	Latencia por ciclos (clock)
FSMD sin iteraciones en el bloque SubBytes	Cyclone II EP2C35F672C6	3496	86.57	554.048	0.158	11.55	20
FSMD con una iteración en el bloque SubBytes	Cyclone II EP2C35F672C6	3280	122.26	521.642	0.159	8.179	30
FSMD con dos iteración en el bloque SubBytes	Cyclone II EP2C35F672C6	3303	124.02	396.864	0.12	8.063	40
FSMD sin iteraciones en el bloque SubBytes	Virtex IV XC4VLX60-12(ff668)	2248	106.523	681.74	0.303	9.387	20
FSMD con una iteración en el bloque SubBytes	Virtex IV XC4VLX60-12(ff668)	1999	137.911	588.42	0.294	7.251	30
FSMD con dos iteración en el bloque SubBytes	Virtex IV XC4VLX60-12(ff668)	1947	185.895	594.864	0.305	5.379	40
ASMD sin iteraciones en el bloque SubBytes	Cyclone II EP2C35F672C6	3578	89.35	571.84	0.159	11.19	20
ASMD con una iteración en el bloque SubBytes	Cyclone II EP2C35F672C6	3229	122.9	524.373	0.162	8.136	30
ASMD con dos iteración en el bloque SubBytes	Cyclone II EP2C35F672C6	3269	120.92	386.944	0.118	8.26	40
ASMD sin iteraciones en el bloque SubBytes	Virtex IV XC4VLX60-12(ff668)	1877	97.299	622.713	0.331	10.277	20
ASMD con una iteración en el bloque SubBytes	Virtex IV XC4VLX60-12(ff668)	2037	182.538	778.822	0.382	5.478	30
ASMD con dos iteración en el bloque SubBytes	Virtex IV XC4VLX60-12(ff668)	1938	182.466	583.891	0.301	5.48	40
Sharma [29]	Virtex IV XC4VLX60-12(ff668)	1299	141.6	1647.7	1.268	7.062	11
Rizk [30]	Virtex IV XC4VLX60-12(ff668)	1468	–	1664	1.134	–	10

De acuerdo con la tabla 4.1, se obtuvieron los siguientes resultados del análisis de síntesis. En caso del diseño basado en FSMD, se observa que con una mayor cantidad de etapas de *pipeline* en el bloque SubBytes, aumenta la máxima frecuencia de operación del circuito. Sin embargo al analizar la máxima frecuencia de operación del diseño basado en ASMD se observa que el diseño con una etapa de *pipeline* posee una mayor frecuencia de operación en comparación al diseño sin ninguna etapa de *pipeline* y al diseño con dos etapas de *pipeline*, este efecto se da debido a que existen otros bloques que estén limitando el máximo rendimiento del circuito, por ejemplo el bloque MixColumns.

Con respecto al parámetro *throughput*, este depende de la latencia por ciclos del circuito por lo que sus resultados son variables. Referente al parámetro área, este varía dependiendo a la arquitectura del FPGA.

Por último se compara los resultados de eficiencia, el cual va a decidir cual es el diseño óptimo para el algoritmo AES. Se observa que, el diseño usando ASMD en comparación con el FSMD, posee una mayor eficiencia. Ahora bien, al comparar la eficiencia con relación al diseño con diferentes etapas de *pipeline* se concluye que el diseño con una etapa de *pipeline* posee una mayor eficiencia, respecto al diseño con cero y dos etapas de *pipeline* (0.382 Mbits/ vs 0.331 Mbits/ y 0.301 Mbits/ en la plataforma Virtex IV y 0.162 Mbits/ vs 0.159 Mbits/ y 0.118 Mbits/ en la plataforma Cyclone II).

Los resultados obtenidos anteriormente fueron comparados con los diseños de Sharma [29] y Rizk [30]. Al comparar los parámetros de eficiencia y *throughput* se observa que los diseños de Sharma y Rizk son más eficientes. Sin embargo, se debe considerar que dentro de los diseños de la presente tesis no se considera el diseño del bloque generador de llaves (KeySchedule), este se considera como un bloque combinacional, el cual afecta directamente los resultados de área y máxima frecuencia de operación del circuito completo. En caso se hubiera diseñado este circuito se hubieran tenido buenas expectativas respecto al aumento de frecuencia de operación y *throughput*.

Conclusiones

- Se realizó exitosamente el diseño del codificador y decodificador AES, ya que los resultados fueron comparados con los del NIST.
- Es preferible utilizar la técnica de ASMD en comparación con la técnica de FSMD para el diseño del bloque AES, debido a que este permite obtener una mayor eficiencia.
- Es preferible un diseño con optimización en pipeline para el bloque SubBytes, ya que este usa una menor cantidad de área y posee una mayor frecuencia de operación.
- Se debe priorizar la optimización de los bloques SubBytes y MixColumns, ya que estos poseen un procesamiento más complejo en comparación de los bloques AddRoundKey y ShiftRows; limitan el desempeño del algoritmo AES.
- El bloque que une los bloques ShiftRows e InvShiftRows, MixColumns e InvMixColumns y AddroundKey, puede limitar el desempeño del bloque AES en caso se exceda la cantidad de pipelines en el bloque SubBytes. Esto se da, ya que al aumentar la cantidad de pipelines reduce el *throughput* del circuito final.
- Si bien las etapas de pipeline en el bloque SubBytes ayudan a mejorar el rendimiento del circuito completo, estos a su vez demoran la obtención del resultado final del circuito.
- La técnica de pipeline usada en el bloque SubBytes ayuda a mejorar el parámetro de frecuencia de operación. Sin embargo, no se debe abusar de esta técnica, ya que puede causar un aumento de área en el circuito completo.
- Se utilizó satisfactoriamente diversas técnicas de optimización en hardware respecto a las operaciones en campo finito o campo Galois. Como ejemplo se puede mencionar al Algoritmo Euclidiano Extendido, memorias ROM, entre otras.
- El uso de la multiplicación inversa en $GF((2^4)^2)$ para realizar el diseño del bloque SubBytes, ayuda a que el circuito posea una menor área y una mayor frecuencia de operación.

- El circuito que genera las llaves de cifrado (KeySchedule) debe tener un mayor desempeño que el proceso de cifrado y decifrado, debido a que este afecta el desempeño del circuito AES.

- El flujo de decodificación propuesto por el NIST se considera el más óptimo para realizar el diseño del codificador y decodificador AES, ya que este no permite un cambio significativo en el algoritmo general y ayuda a describir un mejor diseño digital.

- Se utilizó satisfactoriamente el lenguaje de descripción de hardware Verilog HDL, este lenguaje en comparación del VHDL permite una mayor flexibilidad en cuanto a descripción de hardware y es recomendado para realizar simulaciones de forma aleatoria.

Recomendaciones

- Diseñar el bloque SubBytes de forma que utilice descomposiciones en campo finito de $GF(((2^2)^2)^2)$, debido a que esto consume una menor cantidad de área.[2],[26],[31]

- Investigar sobre algoritmos que puedan realizar de forma eficiente la multiplicación en campo finito $GF(2)^8$.

- Utilizar el software Cadence para realizar análisis de síntesis en la tecnología TSMC 90 nm, debido a que los resultados de los software Quartus y ISE pueden tener ciertas variaciones. Sin embargo, los resultados obtenidos en Quartus pueden ser los más reales por estar el FPGA (Cyclone II) basado en la tecnología TSMC 90 nm, el cual es la misma que se planea utilizar en una futura implementación de un ASIC.

- Colocar etapas de pipeline en el bloque que combina los bloques ShiftRows e InvShiftRows, MixColumns e InvMixColumns y AddroundKey, debido a que estas pueden mejorar el desempeño del circuito en términos de área y frecuencia de operación.

- Analizar la potencia disipada del circuito completo AES, debido a ser un punto a considerar en caso se desee realizar un equipo portátil.

- Realizar el diseño del bloque generador de llaves (KeySchedule) para que pueda acoplarse al diseño del bloque codificador y decodificador AES. Tener en consideración que este bloque tiene que tener un mayor o igual desempeño en términos de área y frecuencia de operación que el bloque codificador y decodificador AES.

- Completar todo el flujo de diseño con el software Cadence para la implementación de un ASIC.

- Verificar el circuito de manera formal con ayuda de la tesis ”Functional verification framework of an AES encryption module”[32].

Bibliografía

[1] J. Wolkerstorfer, E. Oswald, and M. Lamberger, "An asic implementation of the aes sboxes," in *Cryptographers' Track at the RSA Conference*, pp. 67–78, Springer, 2002.

[2] A. Satoh, S. Morioka, K. Takano, and S. Munetoh, "A compact rijndael hardware architecture with s-box optimization," in *Asiacrypt*, vol. 2248, pp. 239–254, Springer, 2001.

[3] N. F. Pub, "197: Advanced encryption standard (aes)," *Federal information processing standards publication*, vol. 197, no. 441, p. 0311, 2001.

[4] L. A. FARIAS, "Decriptografia aes-128 em hardware usando vhdl,"

[5] Y. T. M. Vargas and H. A. M. Mnedez, "Comparación de algoritmos basados en la criptografía simétrica des, aes y 3des," *Mundo FESC*, vol. 5, no. 9, pp. 14–21, 2015.

[6] I. Kuon and J. Rose, "Measuring the gap between fpgas and asics," *IEEE Transactions on computer-aided design of integrated circuits and systems*, vol. 26, no. 2, pp. 203–215, 2007.

[7] L. Arockiam and S. Monikandan, "Data security and privacy in cloud storage using hybrid symmetric encryption algorithm," *International Journal of Advanced Research in Computer and Communication Engineering*, vol. 2, no. 8, pp. 3064–70, 2013.

[8] H. Alanazi, B. Zaidan, A. Zaidan, H. A. Jalab, M. Shabbir, Y. Al-Nabhani, *et al.*, "New comparative study between des, 3des and aes within nine factors," *arXiv preprint arXiv:1003.4085*, 2010.

[9] V.-P. Hoang, V.-L. Dao, C.-K. Pham, *et al.*, "A compact, ultra-low power aes-ccm ip core for wireless body area networks," in *Very Large Scale Integration (VLSI-SoC), 2016 IFIP/IEEE International Conference on*, pp. 1–4, IEEE, 2016.

[10] P. Ceminari, A. Arelovich, and M. Di Federico, "Diseño de tres arquitecturas para un módulo criptográfico aes," in *Biennial Congress of Argentina (ARGENCON), 2016 IEEE*, pp. 1–6, IEEE, 2016.

[11] C. E. Cano Salazar, "Diseño de una arquitectura de un filtro digital de sobre muestreo de imágenes, en factor 2, de acuerdo al formato h. 264/svc sobre fpga," 2012.

[12] A. Aggarwal and G. Singh, "Implementation of aes algorithm," 2016.

[13] G. Bertoni, L. Breveglieri, P. Fragneto, M. Macchetti, and S. Marchesin, "Efficient software implementation of aes on 32-bit platforms," in *CHES*, vol. 2523, pp. 159–171, Springer, 2002.

[14] P. R. Chodowiec *et al.*, *Comparison of the hardware performance of the AES candidates using reconfigurable hardware*. PhD thesis, George Mason University, 2002.

[15] S. E. Thompson and S. Parthasarathy, "Moore's law: the future of si microelectronics," *Materials today*, vol. 9, no. 6, pp. 20–25, 2006.

[16] J. L. Hennessy and D. A. Patterson, *Computer architecture: a quantitative approach*. Elsevier, 2011.

[17] G. E. Moore *et al.*, "Cramming more components onto integrated circuits," 1965.

[18] R. Roman, P. Najera, and J. Lopez, "Securing the internet of things," *Computer*, vol. 44, no. 9, pp. 51–58, 2011.

[19] Ç. K. Koç, "About cryptographic engineering," in *Cryptographic engineering*, pp. 1–4, Springer, 2009.

[20] W. Stallings and M. P. Tahiliani, *Cryptography and network security: principles and practice*, vol. 6. Pearson London, 2014.

[21] F. J. MacWilliams and N. J. A. Sloane, *The theory of error-correcting codes*. Elsevier, 1977.

[22] D. Canright, "A very compact rijndael s-box," 2004.

[23] C. J. Benvenuto, "Galois field in cryptography," *University of Washington*, 2012.

[24] F. Vahid and T. Givargis, *Embedded system design: a unified hardware/software introduction*. John Wiley & Sons, Inc., 2001.

[25] M. Saeed and M. S. Mian, "Methods of finding multiplicative inverses in gf (2 8)," *Computer Communications*, vol. 31, no. 17, pp. 4117–4123, 2008.

[26] X. Zhang and K. K. Parhi, "High-speed vlsi architectures for the aes algorithm," *IEEE transactions on very large scale integration (VLSI) systems*, vol. 12, no. 9, pp. 957–967, 2004.

[27] H. Qin, T. Sasao, and Y. Iguchi, "An fpga design of aes encryption circuit with 128-bit keys," in *Proceedings of the 15th ACM Great Lakes symposium on VLSI*, pp. 147–151, ACM, 2005.

[28] K. Rahimunnisa, P. Karthigaikumar, N. A. Christy, S. S. Kumar, and J. Jayakumar, "Psp: Parallel sub-pipelined architecture for high throughput aes on fpga and asic," *Central European Journal of Computer Science*, vol. 3, no. 4, pp. 173–186, 2013.

[29] V. K. Sharma, S. Kumar, and K. K. Mahapatra, "Iterative and fully pipelined high throughput efficient architectures of aes in fpga and asic," *Journal of Circuits, Systems and Computers*, vol. 25, no. 05, p. 1650049, 2016.

[30] M. Rizk and M. Morsy, "Optimized area and optimized speed hardware implementations of aes on fpga," in *Design and Test Workshop, 2007. IDT 2007. 2nd International*, pp. 207–217, IEEE, 2007.

[31] H. Lee, Y. Paik, J. Jun, Y. Han, and S. W. Kim, "High-throughput low-area design of aes using constant binary matrix-vector multiplication," *Microprocessors and Microsystems*, vol. 47, pp. 360–368, 2016.

[32] F. P. Plasencia Balabarca, "Functional verification framework of an aes encryption module," 2018.