DISEÑO E IMPLEMENTACION DE UN PROTOTIPO CRIPTOPROCESADOR AES-RIJNDAEL EN FPGA

NIDIA MARCELA PINEDA CABRERA
CÓDIGO 161000896
NIRIYINETH HAXBLEIDY VELASQUEZ ROMERO
CÓDIGO 161000895

UNIVERSIDAD DE LOS LLANOS
FACULTAD DE CIENCIAS BÁSICAS E INGENIERÍA
INGENIERÍA ELECTRÓNICA
VILLAVICENCIO
2007

DISEÑO E IMPLEMENTACION DE UN PROTOTIPO CRIPTOPROCESADOR AES-RIJNDAEL EN FPGA

NIDIA MARCELA PINEDA CABRERA
CÓDIGO 161000896
NIRIYINETH HAXBLEIDY VELASQUEZ ROMERO
CÓDIGO 161000895

Trabajo de Grado para optar al título de Ingeniero Electrónico
Modalidad Estudiantes Participantes en Proyección Social
GESTION TECNOLOGICA

HÉCTOR IVÁN REYES MONCAYO
Ingeniero Electrónico MSc
DIRECTOR

JAVIER FERNANDO CASTAÑO FORERO
Ingeniero Electrónico
Codirector

FABIAN VELASQUEZ CLAVIJO
Ingeniero Electrónico
Codirector

UNIVERSIDAD DE LOS LLANOS
FACULTAD DE CIENCIAS BÁSICAS E INGENIERÍA
INGENIERÍA ELECTRÓNICA
VILLAVICENCIO
2007

CONTENIDO

LISTA DE FIGURAS

LISTA DE TABLAS

RESUMEN

En este trabajo de grado se presenta el diseño de la arquitectura en hardware de un criptoprocesador AES-Rijndael utilizando como dispositivo lógica programable una FPGA SPARTAN-3E con lenguaje de descripción VHDL, en el proyecto se utilizó la herramienta XILINX ISE 8.2i, en la cual se sintetizó y simuló la arquitectura diseñada.

Esta implementación realiza cifrado y descifrado, soporta bloques de datos de 128 bits y una longitud de clave de 128 bits, desarrollando 10 rondas en cada proceso, siguiendo las consideraciones del estándar AES.

Opera a nivel de bytes, interpretando estos como elementos de un campo de Galois GF(2^8), utiliza el polinomio irreducible $\boldsymbol{m(x) = X^8 + X^4 + X^3 + X + 1.}$

Contiene 4 multiplicadores en el campo finito de arquitectura paralela en su proceso de descifrado.

Trabaja Comunicación RS-232 mediante módulo UART (Universal Asyncronous Receiver-Transmiter), con una tasa de transferencia de 115200 b/s.

INTRODUCCION

La seguridad informática ha adquirido gran importancia en las áreas de la industria, los negocios y en la sociedad actual. Las técnicas básicas que se requieren para proteger la información corresponden al campo de la criptografía, debido a que ésta es aplicada a la seguridad y es una herramienta primordial para asegurar confidencialidad en la transmisión y almacenamiento de la información.

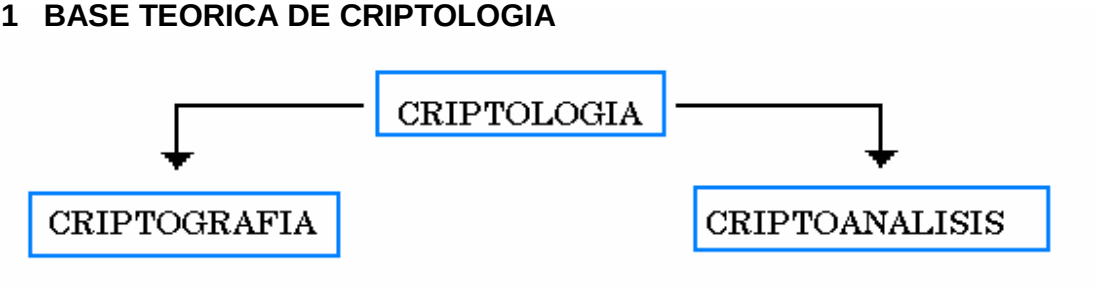
AES es el actual estándar de cifrado simétrico dispuesto por el NIST (National Institute of Standards and Technology), después de un periodo de prueba entre quince algoritmos sometidos, en Octubre de 2000 fue designado el algoritmo Rijndael como AES, el Standard reemplazó el TDES, para ser usado en los siguientes 20 años. El algoritmo Rijndael opera a nivel de bytes, interpretando estos como elementos de un campo de Galois GF(2^8), y a nivel de registros de 32 bits, considerándolos como polinomios de grado menor que 4 con coeficientes que son a su vez son polinomios en GF(2^8) [1].

En este proyecto se implementó la arquitectura hardware para el criptoprocesador Rijndael-AES en una FPGA, desarrollando el proceso de cifrado y descifrado, ejecutando una evaluación de las ventajas que tiene el algoritmo Rijndael con relación a su seguridad y optimización.

En la Universidad de los Llanos, se han desarrollado varios proyectos en la línea de telecomunicaciones. En el área de criptografía se ha realizado un prototipo de un criptoprocesador para una curva elíptica y punto base fijos sobre el campo finito GF (2^7) [11].

1. MARCO TEORICO

1.1 BASE TEORICA DE CRIPTOLOGIA



La Criptología

Figura 1

La criptología esta formada por dos técnicas las cuales son criptografía y criptoanálisis.

1.1.1 Criptoanálisis

Es la técnica de descifrar un criptograma sin tener la autorización. Consiste en romper mensajes cifrados atacándolos y buscando el mensaje original o mensaje en claro.

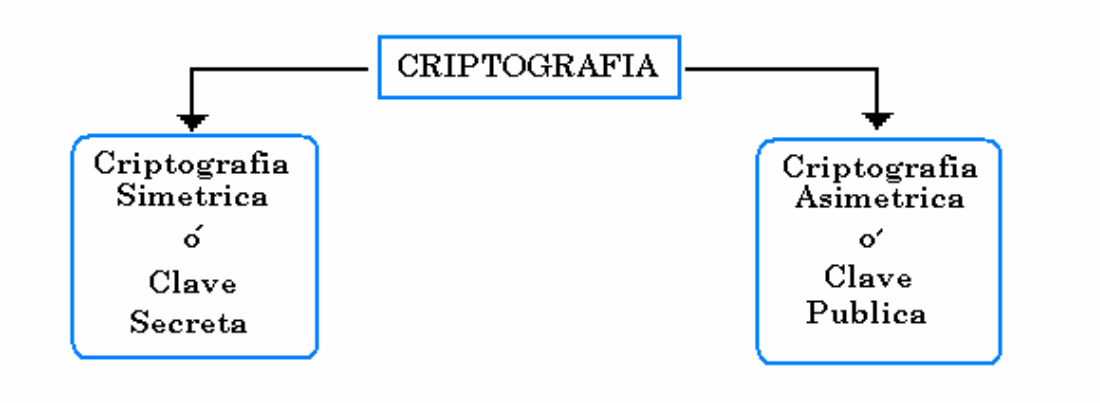
El criptoanálisis abarca diversas técnicas, muchas veces no dependen del conocimiento del algoritmo sino que mediante sistemas de aproximaciones matemáticas se puede descubrir el texto en claro o la clave. [5]

1.1.2 Criptografía

Es la técnica de convertir un texto inteligible (texto en claro) en otro, llamado criptograma, cuyo contenido de información es igual al anterior pero solo lo pueden entender las personas autorizadas. [5]

El objetivo de la criptografía es el de proporcionar comunicaciones seguras sobre canales inseguros, es decir, permitir que dos entidades, bien sean personas o aplicaciones, puedan enviarse mensajes por un canal que puede ser interceptado por una tercera entidad, de modo que sólo los destinatarios autorizados puedan leer los mensajes. Pero la criptografía no es en sí la seguridad, sólo es la

herramienta básica que utilizan mecanismos más complejos para proporcionar, además de confidencialidad, otros servicios de seguridad.



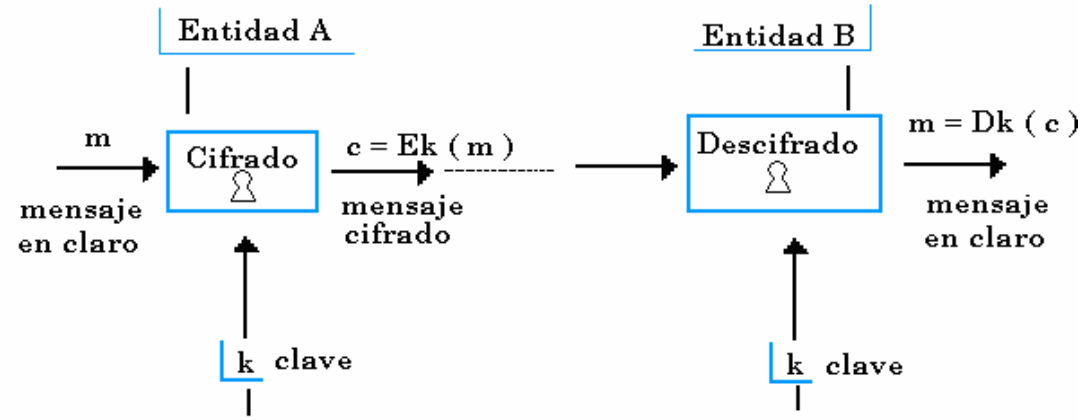
Clasificación de la criptografía
Figura 2

Las técnicas de criptografía se pueden clasificar en dos según el tipo de clave utilizado, los cuales son: Criptografía simétrica o de clave secreta y Criptografía asimétrica o de clave pública.

Nos enfocaremos en presentar información de Criptografía simétrica ya que el algoritmo Rijndael - AES se encuentra dentro de este tipo de cifrado.

1.1.3 Criptografía simétrica o de clave secreta

Existe una única clave (secreta) que comparte emisor y receptor. Con la misma clave se cifra y se descifra, por lo que la seguridad reside en mantener dicha clave en secreto. [6]



Esquema de Criptografía Simétrica
Figura 3

El proceso de obtención del mensaje de cifrado es:

$$c = E_k (m) \qquad (1)$$

Donde *k* es la clave secreta, E_k representa la operación de cifrado con esa clave y m es el mensaje en claro.

Simétricamente el proceso de recuperación del mensaje original, mediante la operación de cifrado, D_k, en el que se utiliza la misma clave secreta *k,* se representa como:

$$m = D_k (c) \qquad (2)$$

La simetría del proceso se basa en que este tipo de cifradores se comporta:

$$Dk (Ek (m)) = m \qquad (3)$$

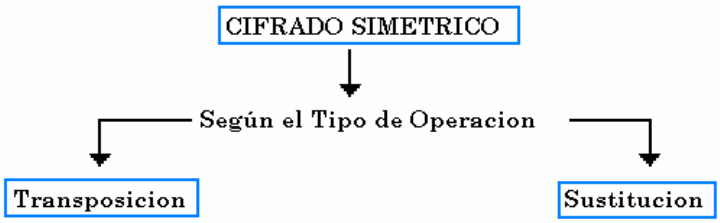
Los algoritmos de clave secreta se apoyan en operaciones algebraicas muy simples, lo que conlleva que el tiempo necesario para realizar el proceso completo de cifrado es muy pequeño. Esta característica hace que este tipo de cifrado pueda utilizarse para proporcionar confidencialidad en aplicaciones telemáticas, en las que se requiere gran velocidad en flujo de datos.

La criptografía simétrica implica que para establecer comunicación segura entre dos usuarios, es necesario que estos conozcan y compartan una clave simétrica concreta.

Dos técnicas básicas de la criptografía simétrica son: confusión y difusión.

Confusión consiste en establecer una dependencia funcional lo mas compleja posible entre la clave y el mensaje en claro y se consigue mediante la *sustitución* de unos símbolos por otros.

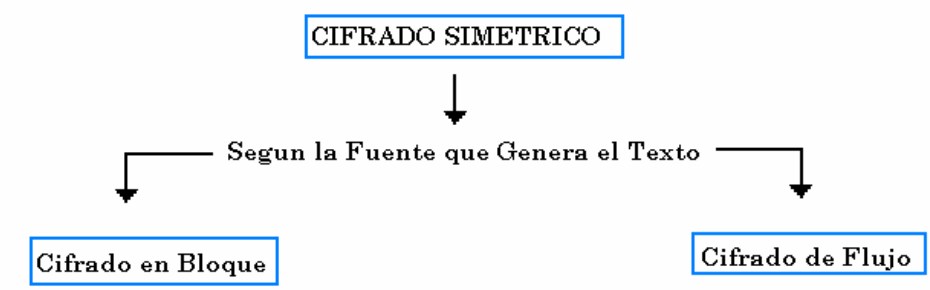
Difusión consiste en dispersar a lo largo del programa las propiedades estadísticas del mensaje en claro y se consigue mediante la *transposición* (permutación de los bits).



Operaciones del Cifrado Simétrico
Figura 4

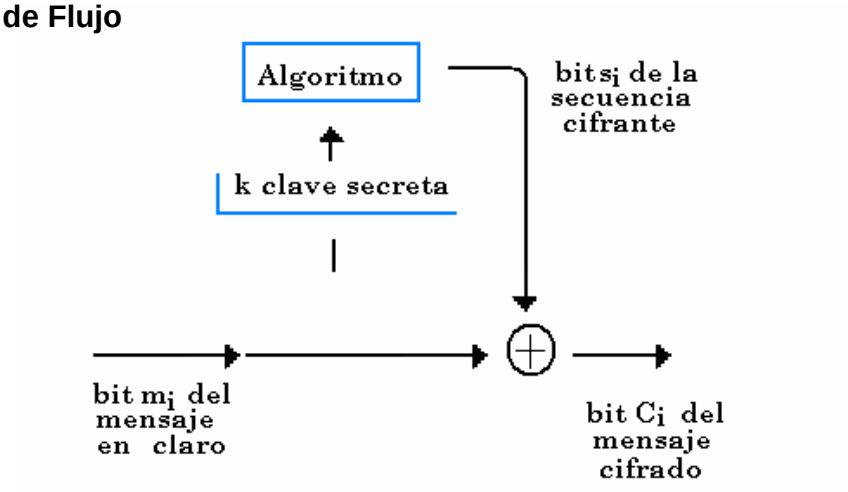
1.1.3.1 Cifradores de Flujo y Cifradores de Bloque

Dado un mensaje en claro *m*, representado mediante un número variable de octetos, a la hora de proceder a su cifrado para convertirlo en un criptograma *c* existen dos tipos básicos de cifradores, diferenciados en cuanto como dividir el mensaje en claro para abordar la tarea de transformación[8].



Tipos de Cifrado Simétrico
Figura 5

- **Cifrado de Flujo**

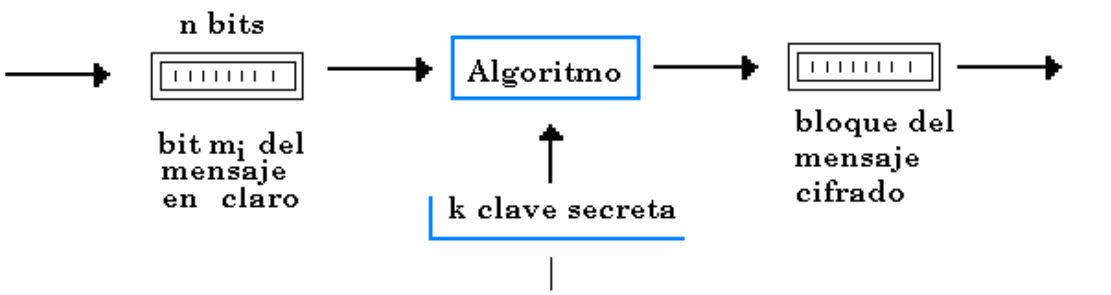


Cifrado en Flujo
Figura 6

La característica principal de este cifrado consiste en considerar el mensaje en claro como un flujo continuo de bits (o caracteres) y generar a la salida el correspondiente flujo de bits resultante de la transformación producida en el proceso de cifrado.

La ventaja de este sistema es la simplicidad en el proceso de cifrado.

- **CIFRADO EN BLOQUE**



Cifrado en Bloque
Figura 7

En este tipo de cifrado se divide el mensaje en claro en bloques de n bits cada uno, la característica principal de este tipo de cifradores consiste en que cada bloque se cifra de igual forma, independientemente del lugar que ocupe en la cadena, de manera que todos los bits del bloque se cifran conjuntamente, participando en operaciones que tratan de oscurecer las posibles relaciones que tuviesen en el mensaje original.

Los algoritmos simétricos cifran bloques de texto, su longitud puede ser variable según el algoritmo que se utilice.

En el cifrado en bloque se realizan cuatro posibles modos de operación, los cuales son:

- Electonic CodeBlock (ECB): en el se cifran los bloques por separado.
- Cipher BlockChainning (CBC): los bloques de criptograma se relacionan entre ellos mediante funciones OR EXCLUSIVA.
- Cipher feedback (CFB): se realiza una OR EXCLUSIVA entre caracteres o bits aislados del texto y las salidas del algoritmo. El algoritmo utiliza como entrada los criptogramas.
- Output feedback (OFB): igual que el CFB, se realiza una OR EXCLUSIVA entre caracteres o bits aislados de texto y las salidas del algoritmo, pero

este utiliza como entradas sus propias salidas; por tanto no depende del texto, es un generador de números aleatorios.

1.2 ALGORITMO RIJNDAEL – AES

Este algoritmo fue escogido como el sucesor de DES, después de una convocatoria que se realizó en 1997, en una primera ronda se seleccionaron 15 algoritmos de los cuales 5 fueron seleccionados en la ronda final, los cuales fueron:

- MARS, de IBM
- RC6 de RSA laboratory
- SERPENT de R. Anderson, E Biham y L. Knudsen
- TWOFISH, de B schneider y otros
- RIJNDAEL, de Joan Daemen y Vincent Rijmen

Finalmente RIJNDAEL fue escogido. Los criterios bajo los cuales se desarrolló el algoritmo seleccionado son los siguientes:

- Resistencia antes todos los ataques conocidos
- Velocidad y código compacto en su implementación sobre una amplia gama de plataformas desde dispositivos con recursos limitados hasta procesadores paralelos.
- Simplicidad de diseño
- Soportar bloques de datos de 128 bits y claves de 128, 192, y 256 bits

1.3 ESTUDIO DEL MODELO MATEMATICO DE RIJNDAEL - AES

AES es un sistema de cifrado por bloques, diseñado para manejar longitudes de clave y de bloque variables, ambas comprendidas entre los 128 y los 256 bits. Realiza varias de sus operaciones internas a nivel de byte, interpretando éstos como elementos de un cuerpo de Galois GF (2^8). El resto de operaciones se efectúan en términos de registros de 32 bits.

Este algoritmo soporta diferentes tamaños de bloque y clave, en el estándar adoptado por el Gobierno Estadounidense en noviembre de 2001 (FIPS PUB 197), se especifica una longitud fija de bloque de 128, y la longitud de clave a escoger entre 128, 192 y 256 bits. [7]

1.3.1 Campo Finitos. GF (2^m)

En el algoritmo Rijndael todos los bytes se interpretan como elementos de un campo finito, los cuales se representan mediante Campos de Galois GF(k). En este Campo existe un inverso aditivo y multiplicativo que permite cifrar y descifrar en el mismo cuerpo Zk [9].

En este algoritmo se utiliza una aritmética en módulo p sobre polinomios de grado m, siendo p un número primo. Este campo se representa como:

$$\mathbf{GF\ (p^m)} \qquad (4)$$

Los elementos de GF (p^m) son polinomios con coeficientes en Zp de grado menor que m, de esta forma:

$$\mathbf{GF\ (p^m) = \{b_0 + b_1x + b_2x^2 + \ldots\ b_{m-1}x^{m-1}\};\ b_0, b_1, b_2 \ldots b_{m-1}} \quad (5)$$

Cada elemento de GF (pm) es un resto módulo p(x), donde p(x) es un polinomio irreducible de grado m, el cual no puede ser factorizado en polinomios de grado menor que m.

Para el algoritmo Rijndael es necesario trabajar con campos de la forma GF(2^m) debido a que los coeficientes son restos del módulo 2 [0, 1], esto admite una representación binaria. Cada elemento del campo se representa con m bits y el número de elementos es 2^m.

En el algoritmo Rijndael - AES, las operaciones se realizan a nivel de byte en el campo GF (2^8). Cada byte B, se compone de los 8 bits b_7, b_6, b_5, b_4, b_3, b_2, b_1, b_0, Representado este byte como un polinomio tenemos:

$$\mathbf{b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x^1 + b_0} \quad (6)$$

El polinomio irreducible que utiliza el algoritmo Rijndael es de grado 8:

$$x^8+x^4+ x^3+ x+1 \qquad (7)$$

La representación binaria del polinomio irreducible es:

1 0 0 0 1 1 0 1 1 (8)

1.3.2 Operaciones en el Campo GF (2^8)

1.3.2.1 Suma y Resta en GF (2^8)

Las funciones de suma y resta se realizan con una operación XOR, de esta forma si los coeficientes son iguales dará como resultado 0 y si los coeficientes son diferentes el resultado será 1. Para sumar dos polinomios se aplica la operación XOR a cada elemento de los polinomios, bit a bit:

1	XOR	1	=	0
0	XOR	1	=	1
1	XOR	0	=	1
0	XOR	0	=	0

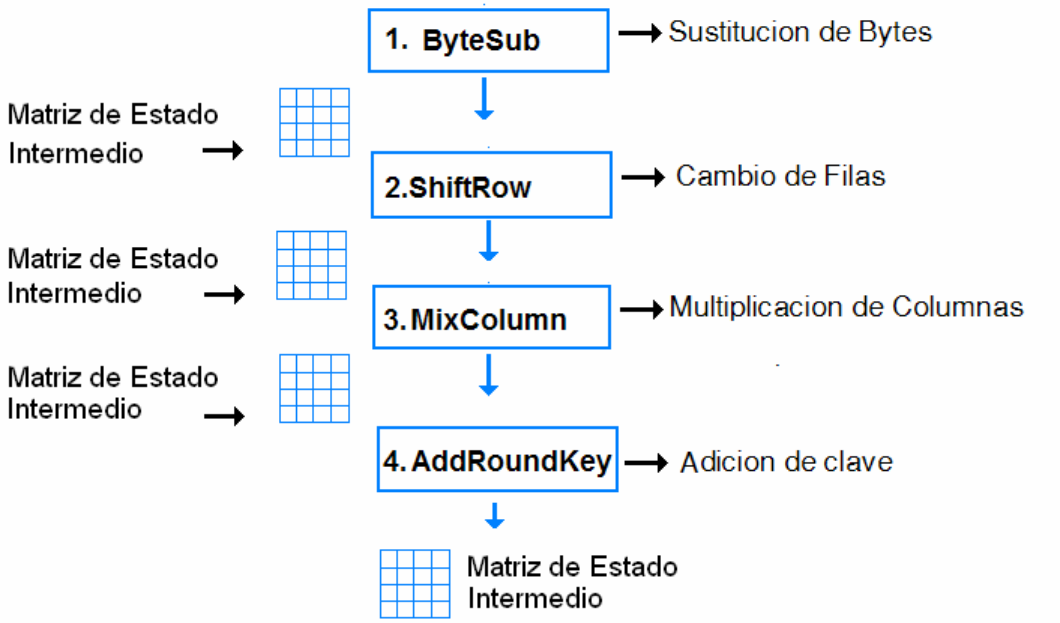
Operación XOR
Tabla 1

Esto se debe a que en el campo finito binario la suma se realiza módulo 2, lo cual coincide exactamente con la operación XOR.

1.3.2.2 Multiplicación en GF (2^8)

En la multiplicación de polinomios en GF (2^m), es posible que el resultado contenga elementos que no estén dentro del cuerpo del polinomio, es decir potencias iguales o mayores que m, por lo que se reducirá los exponentes mediante un polinomio irreducible p(x) de grado m. Para GF (2^8) la multiplicación de polinomios se realiza móulo con un polinomio irreducible de grado 8. Este polinomio irreducible se representa por m(x).

1.4 ESTRUCTURA DEL ALGORITMO RIJNDAEL – AES



Matrices de Estados Intermedios
Figura 8

El algoritmo Rijndael no posee una estructura tipo Feistel, que es la que posee el estándar DES. Esto le permite producir una mayor difusión de la información cifrada con un menor número de rondas o de aplicaciones matemáticas.

La estructura del algoritmo Rijndael está formada por un conjunto de rondas, (conjunto de iteraciones de 4 funciones matemáticas diferentes e invertibles). El algoritmo se basa en utilizar un número de rondas determinado a una información en claro para producir una información cifrada. La información generada por cada función es un resultado intermedio, que se conoce como Estado Intermedio, que se representa como un matriz rectangular de bytes, que tiene 4 filas y Nb columnas, siendo el número de columnas Nb en función del tamaño del bloque:

$$N = \text{Tamaño del Bloque Utilizado en bits} / 32 \quad (9)$$

Representación de una matriz de Estado para un tamaño de bloque de 128 bits (Nb = 4), sería:

$a_{0.0}$	$a_{0.1}$	$a_{0.2}$	$a_{0.3}$
$a_{1.0}$	$a_{1.1}$	$a_{1.2}$	$a_{1.3}$
$a_{2.0}$	$a_{2.1}$	$a_{2.2}$	$a_{2.3}$
$a_{3.0}$	$a_{3.1}$	$a_{3.2}$	$a_{3.3}$

Matriz de Estado de un bloque de 128 bits
Tabla 2

La clave del sistema se representa con una estructura análoga a la del Estado, es decir, se representa mediante una matriz rectangular de bytes de 4 filas y Nk columnas, siendo el número de columnas Nk en función del tamaño de la clave:

$$Nk = \text{Tamaño de la Clave en bits} / 32 \quad (10)$$

La representación de una clave de 128 bits (Nk = 4), en forma de matriz rectangular sería:

$k_{0.0}$	$k_{0.1}$	$k_{0.2}$	$k_{0.3}$
$k_{1.0}$	$k_{1.1}$	$k_{1.2}$	$k_{1.3}$
$k_{2.0}$	$k_{2.1}$	$k_{2.2}$	$k_{2.3}$
$k_{3.0}$	$k_{3.1}$	$k_{3.2}$	$k_{3.3}$

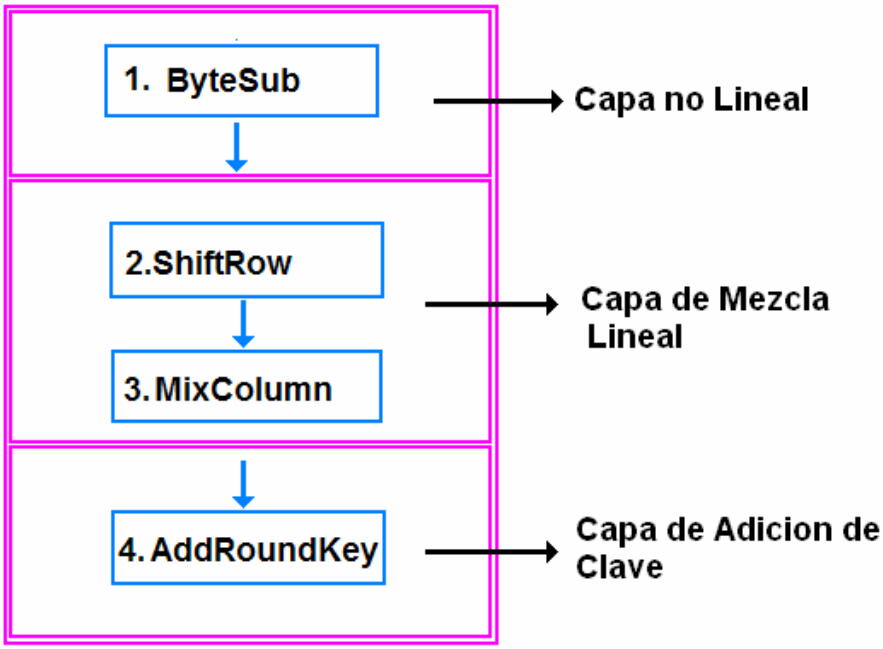
Matriz de Estado de una clave de 128 bits
Tabla 3

El bloque que se va a cifrar o descifrar se traslada byte a byte sobre la matriz de Estado, siguiendo la secuencia a0,0, a1,0, a2,0, a 3,0, a 0,1,..., a 3,4, y análogamente los bytes de la clave se copian en la matriz de la clave siguiendo el mismo criterio, k 0,0, k 1,0, k 2,0, k 3,0, k 0,1 ... k 3,3.

Cada ronda esta compuesta por cuatro operaciones basadas en transformaciones uniformes e invertibles que son llamadas capas, que han sido diseñadas para resistir ante los criptoanálisis lineal y diferencial, las cuales son:

- **Capa de mezcla lineal:** garantiza un alto nivel de difusión a lo largo de las múltiples rondas.
- **Capa no lineal:** Consiste en la aplicación de cajas – s en paralelo que tiene propiedades optimas de no linealidad.
- **Capa de adición de clave:** Se trata de una operación OR exclusivo entre el estado intermedio y la clave propia de cada ronda.

Cada ronda cuenta con 4 etapas, que son:



Ronda Básica
Figura 9

1.4.1 BYTESUB: Sustitución de Byte

Consiste en una sustitución no lineal que se aplica a cada byte de la matriz de Estado de forma independiente, generando un nuevo byte.

Esta transformación consiste en la sustitución de cada byte por el resultado de aplicarle la tabla de sustitución S-Box. Esta tabla lógicamente es invertible y se construye mediante dos transformaciones:

1ª Transformación: Cada byte es considerado como un elemento en GF (2^8) que genera el polinomio irreducible m(x)= x8 + x4 + x3 + x + 1, siendo sustituido por su inversa multiplicativa. El valor cero no se altera, así que se sustituye por si mismo debido a que no tiene inverso multiplicativo.

2ª Transformación: Al resultado de la primera transformación se le aplica la siguiente transformación similar en GF (2), siendo x0, x1, x2, x3, x4, x5, x6 y x7 los bits del byte resultante de la primera transformación, es y0, y1, y2, y3, y4, y5, y6 e y7 los bits del resultado final de la transformación ByteSub.

$$\begin{bmatrix} Y_0 \\ Y_1 \\ Y_2 \\ Y_3 \\ Y_4 \\ Y_5 \\ Y_6 \\ Y_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

Matriz de Transformación ByteSub
Figura 10

La siguiente tabla determina la operación Bytesub:

HEX		Y															
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
X	0	63	7C	77	7B	F2	6B	6F	C5	30	O1	67	2B	FE	D7	AB	76
	1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
	2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
	3	O4	C7	23	C3	18	96	O5	9A	O7	12	80	E2	EB	27	B2	75
	4	O9	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
	5	53	D1	OO	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
	6	D0	EF	AA	FB	43	4D	33	85	45	F9	O2	7F	50	3C	9F	A8
	7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
	8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
	9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	OB	DB
	A	E0	32	3A	OA	49	O6	24	5C	C2	D3	AC	62	91	95	E4	79
	B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	O8
	C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
	D	70	3E	B5	66	48	O3	F6	0E	61	35	57	B9	86	C1	1D	9E
	E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
	F	8C	A1	89	OD	BF	E6	42	68	41	99	2D	OF	B0	54	BB	16

Sustitución S-BOX para un byte genérico xy. (hexadecimal)
Tabla 4

La siguiente tabla determina la operación inversa de Bytesub:

HEX		Y															
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
X	0	52	O9	6A	D5	30	36	A5	38	BF	40	A3	9E	81	F3	D7	FB
	1	7C	E3	39	82	9B	2F	FF	87	34	8E	43	44	C4	DE	E9	CB
	2	54	7B	94	32	A6	C2	23	3D	EE	4C	95	OB	42	FA	C3	4E
	3	O8	2E	A1	66	28	D9	24	B2	76	5B	A2	49	6D	8B	D1	25
	4	72	F8	F6	64	86	68	98	16	D4	A4	5C	CC	5D	65	B6	92
	5	6C	70	48	50	FD	ED	B9	DA	5E	15	46	57	A7	8D	9D	84
	6	90	D8	AB	OO	8C	BC	D3	OA	F7	E4	58	O5	B8	B3	45	O6
	7	D0	2C	1E	8F	CA	3F	OF	O2	C1	AF	BD	O3	O1	13	8A	6B
	8	3A	91	11	41	4F	67	DC	EA	97	F2	CF	CE	F0	D4	E6	73
	9	96	AC	74	22	E7	AD	35	85	E2	F9	37	E8	1C	75	DF	6E
	A	47	F1	1A	71	1D	29	C5	89	6F	B7	62	OE	AA	18	BE	1B
	B	FC	56	3E	4B	C6	62	79	20	9A	DB	C0	FE	78	CD	5A	F4
	C	1F	DD	A8	33	88	O7	C7	31	B1	12	10	59	27	80	EC	5F
	D	60	51	7F	A9	19	B5	A4	OD	2D	E5	7A	9F	93	C9	9C	EF
	E	A0	E0	3B	4D	AE	2A	F5	B0	C8	EB	BB	3E	83	53	99	61
	F	17	2B	O4	7E	BA	77	D6	26	E1	69	14	63	55	21	OC	7D

Inversa S –BOX
Tabla 5

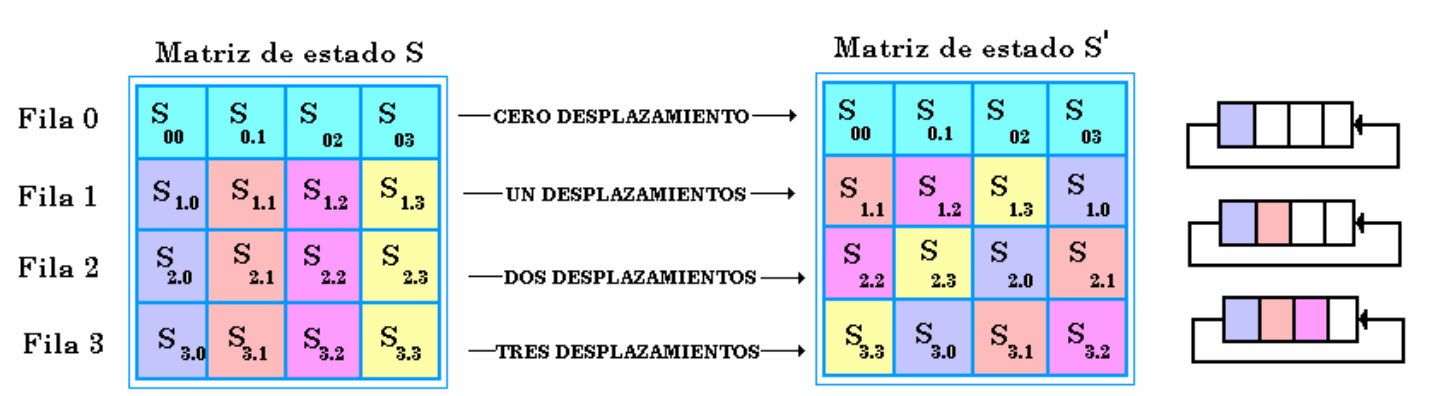
1.4.2 SHIFTROW: Desplazamiento de filas

En esta transformación lo que se hace es girar cíclicamente las filas del estado intermedio varios lugares a la izquierda, dependiendo de la posición de la fila y el tamaño de bloque cifrado. Cuando son bloques de 128 o 196 bits la primera fila (fila 0) no se desplaza, la fila 1 se desplaza cíclicamente un lugar, la fila 2 dos lugares y la fila 3 tres lugares. Cuando los bloques son de 256 estos desplazamientos son algo diferentes; 0, 1, 3 y 4 lugares respectivamente.

Tamaño de Bloque	C1	C2	C3
128 bits (Nb = 4)	1	2	3
192 bits (Nb = 6)	1	2	3
256 bits (Nb = 8)	1	3	4

Valores de Ci según el tamaño de bloque
Tabla 6

Efecto de la transformación ShiftRow



Función ShiftRow para bloque de 128 bits
Tabla 7

1.4.3 **MIXCOLUMN:** Mezcla de columnas

Esta transformación actúa sobre los bytes de una misma columna de la matriz de Estado que tiene a la entrada. En realidad esta función permite una mezcla de los bytes de las columnas.

Esta transformación considera las columnas de bytes como polinomios cuyos coeficientes pertenecen a GF(2^8), es decir, también son polinomios.

La función MixColumn consiste en multiplicar las columnas de bytes módulo x^4+1 por el polinomio c(x).

Matemáticamente c(x) esta representado por:

$$C(x) = 03_{16}x^3 + 01_{16}x^2 + 01_{16}x + 02_{16} \qquad (11)$$

Este polinomio c(x) es coprimo con x^4+1, lo que permite que sea invertible.

En forma algebraica esta función se puede representa como:

$$s'(x) = c(x) \otimes s(x) \qquad (12)$$

Donde s´(x) representa la matriz de Estado resultante de esta transformación y s(x) la matriz de Estado entrante.

Expresado en forma matricial, donde “c” representa el índice de la columna que se procesa queda de la siguiente forma:

$$\begin{bmatrix} S'_{0,c} \\ S'_{1,c} \\ S'_{2,c} \\ S'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} S_{0,c} \\ S_{1,c} \\ S_{2,c} \\ S_{3,c} \end{bmatrix}$$

Matriz de transformación MixColumn
Figura 11

Desarrollando la matriz, se observa como cada byte nuevo de la matriz de Estado es una combinación de varios bytes de las distintas filas que forman una columna específica:

$$S_{0,c} = (\{02\} \bullet S_{0,c}) \oplus (\{03\} \bullet S_{1,c}) \oplus S_{2,c} \oplus S_{3,c}$$

$$S_{1,c} = S_{0,c} \oplus (\{02\} \bullet S_{1,c}) \oplus (\{03\} \bullet S_{2,c}) \oplus S_{3,c}$$

$$S_{2,c} = S_{0,c} \oplus S_{1,c} \oplus (\{02\} \bullet S_{2,c}) \oplus (\{03\} \bullet S_{3,c})$$

$$S_{3,c} = (\{02\} \bullet S_{0,c}) \oplus S_{1,c} \oplus S_{2,c} \oplus (\{03\} \bullet S_{3,c})$$

Operaciones de Transformación
Figura 12

Para descifrar o invertir esta transformación, se realiza el mismo procedimiento pero con el polinomio d(x), que es el inverso de c(x).

La inversa de Mezclar Columnas se obtiene de multiplicar cada columna de la matriz de estado por el polinomio d(x):

$$d(x) = 0B_{16}x^3 + 0D_{16}x^2 + 09_{16}x + 0E_{16} \quad (13)$$

Este polinomio, permite realizar la inversa del polinomio c(x), cumpliendo que:

$$C(x) \oplus d(x) = 01 \quad (14)$$

La transformación inversa se puede mostrar de forma matricial así:

$$S(x) = d(x) \oplus S'(x) \quad (15)$$

$$\begin{bmatrix} S_{0,c} \\ S_{1,c} \\ S_{2,c} \\ S_{3,c} \end{bmatrix} = \begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \begin{bmatrix} S'_{0,c} \\ S'_{1,c} \\ S'_{2,c} \\ S'_{3,c} \end{bmatrix}$$

Transformación Inversa
Figura 13

Los resultados de esta multiplicación, dan como resultado la siguiente sustitución de los bytes:

$$S_{0,c} = (\{0E\} \bullet S'_{0,c}) \oplus (\{0B\} \bullet S'_{1,c}) \oplus (\{0D\} \bullet S'_{2,c}) \oplus (\{09\} \bullet S'_{3,c})$$

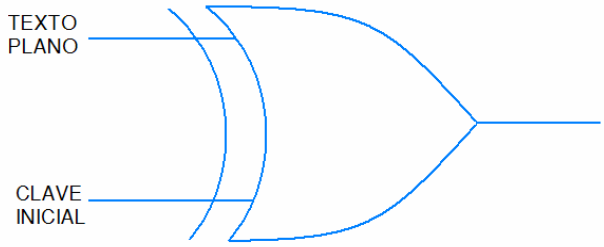
$$S_{1,c} = (\{09\} \bullet S'_{0,c}) \oplus (\{0E\} \bullet S'_{1,c}) \oplus (\{0B\} \bullet S'_{2,c}) \oplus (\{0D\} \bullet S'_{3,c})$$

$$S_{2,c} = (\{0D\} \bullet S'_{0,c}) \oplus (\{09\} \bullet S'_{1,c}) \oplus (\{0E\} \bullet S'_{2,c}) \oplus (\{0B\} \bullet S'_{3,c})$$

$$S_{3,c} = (\{0B\} \bullet S'_{0,c}) \oplus (\{0D\} \bullet S'_{1,c}) \oplus (\{09\} \bullet S'_{2,c}) \oplus (\{0E\} \bullet S'_{3,c})$$

Operaciones de Transformación Inversa
Figura 14

1.4.4 **ADDROUNDKEY:** Adición de la clave de ciclo



Operación Xor
Figura 15

Esta ultima transformación consiste en realizar una operación OR-Exclusiva entre la matriz de Estado que proviene de la transformación anterior (Función MixColumn) y la subclave generada a partir de la clave del sistema para esa ronda.
El bloque resultante de esta transformación, es la nueva matriz de Estado para la siguiente ronda. Hallando el bloque o dato de salida del algoritmo, si es la última vuelta.

El número total de bits necesarios para generar todas las subclaves, depende del número de rondas que se aplique al algoritmo y del tamaño del bloque utilizado.
El número total de bits de subclaves necesarios es calculado así:

$$\text{Numero Total Bits Subclaves} = 32 * Nb * (Nr + 1) \quad (16)$$

Entonces, el número total de bits de subclaves es igual al tamaño del bloque empleado por el número de vueltas del algoritmo (las Nr vueltas del algoritmo más la ronda inicial).

Bloque / Clave	Nk = 4(128 Bits)	Nk = 6(192 Bits)	Nk =6(192 Bits)
Nb = 4(128 Bits)	1408Bits(Nr=10)	1664Bits(Nr=12)	1920Bits(Nr=14)
Nb = 6(192 Bits)	2304Bits(Nr=12)	1496Bits(Nr=12)	2880Bits(Nr=14)
Nb = 8(256 Bits)	3840Bits(Nr=14)	3328Bits(Nr=14)	3840Bits(Nr=14)

Nº de bits de subclaves para tamaños estándar de clave y bloque
Tabla 8

Es muy importante el procedimiento para generar los bytes que forman las subclaves para cada vuelta (RoundKeys), bytes que se derivan de la clave principal K. Para esto se utiliza dos funciones auxiliares: La función de selección y la función de expansión.

1.4.5 Función de selección de clave

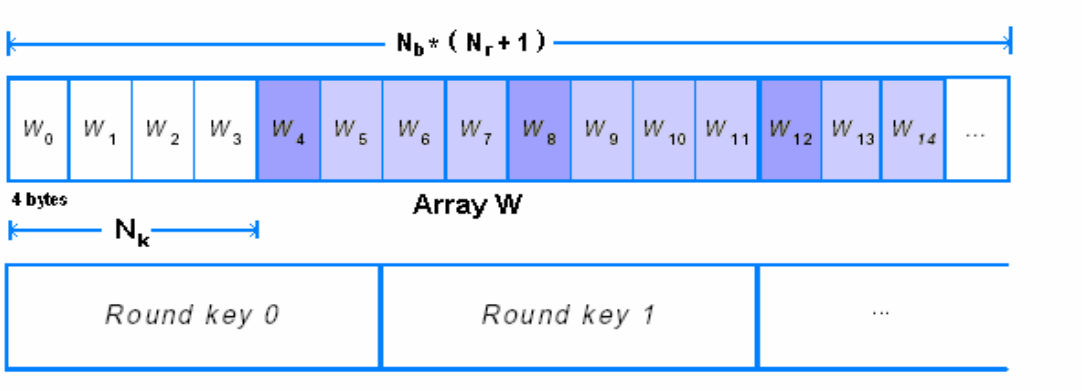
Esta función toma consecutivamente de la secuencia obtenida por la función de expansión de clave bytes que va asignado a cada subclave Ki, para formar bloques del mismo tamaño que la matriz de estado. Es decir, toma Nb*4 bytes para cada vuelta.

La generación de la claves (expansión de clave) para el proceso de descifrado se hace de igual forma al proceso de cifrado. La diferencia esta en la función de selección de clave. En el proceso de descifrado se toman bytes de la lista de claves desde los valores finales hasta llegar a los iniciales, que es la propia clave de usuario. Es decir, la última subclave que se utilizó para cifrar, será la primera que se utilizará para descifrar. [9]

1.4.6 Función de expansión de clave.

Esta función permite generar bytes útiles como subclaves a partir de la clave de sistema K. Además se puede describir como un arreglo (array) lineal, denominado W, de palabras de 4 bytes y con una longitud de Nb*(Nr+1).

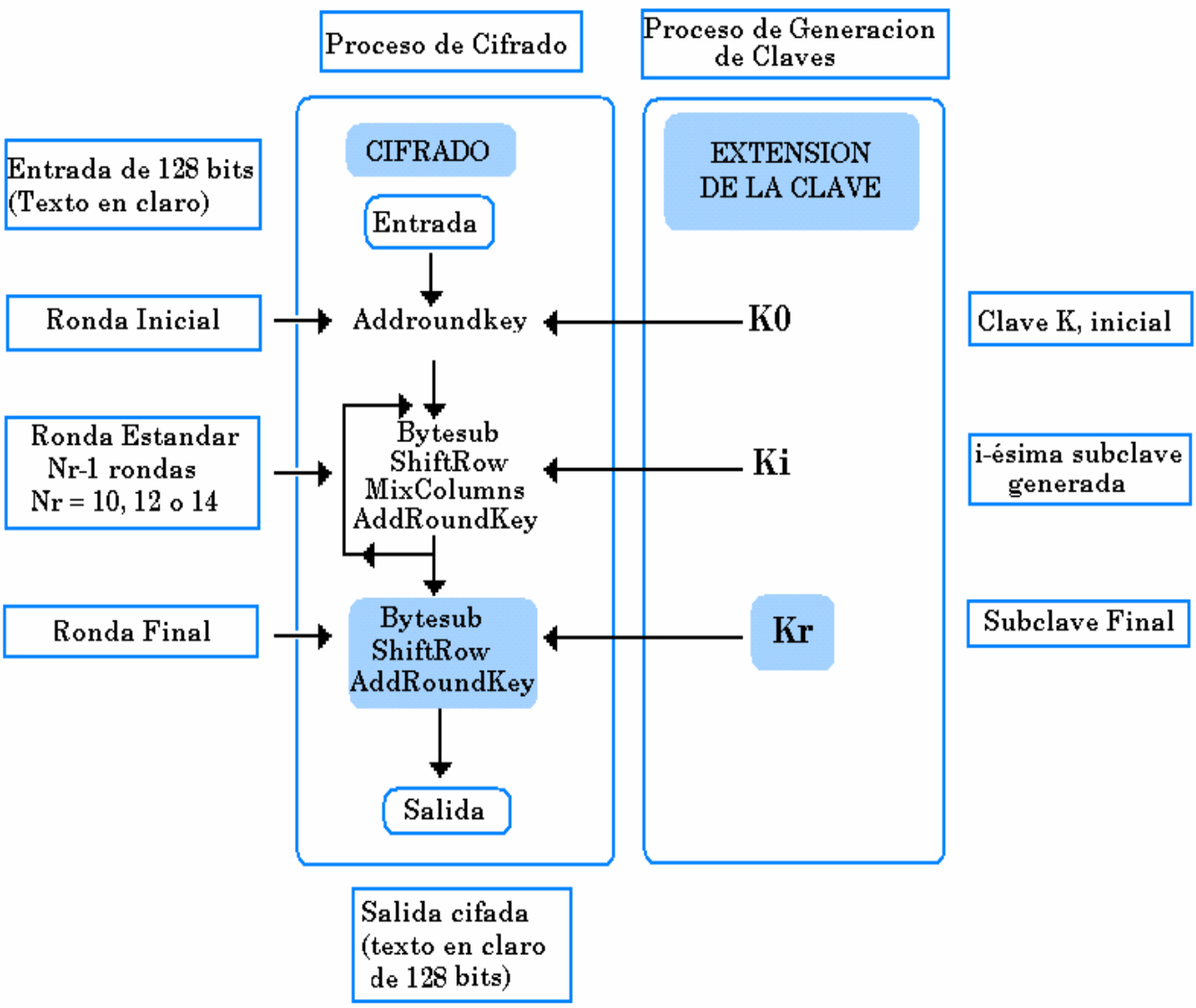
Las primeras Nk palabras de este arreglo (array) contienen la clave de cifrado, ya que la clave del usuario se mapea el arreglo (array) W, mientras que el resto de palabras se van generando a partir de estas primeras Nk palabras [9].



Ejemplo de subclaves y clave de expansión para Nb=6 y Nk=4
Figura 16

La función de la clave depende de Nk.

1.5 ESQUEMA DE CIFRADO Y DESCIFRADO



Estructura Completa del Algoritmo Rijndael - AES
Figura 17

1.5.1 Proceso de Cifrado

El algoritmo para cifrar un bloque realiza primero la transformación Addrounkey de adición con la primera clave, luego se llevan a cabo varias rondas cuyo numero n es 9 si tanto el bloque como la clave son de 128 bits, 12 si la clave o el bloque

son de 192 bits y ninguno de ellos supera esta longitud, 13 si la clave o el bloque son de 256 bits.

Se realizan las cuatro etapas que constituyen el ciclo básico de Rijndael, las cuales son Bytesub, Shiftrow, Mixcolumn y Addroundkey.

Además se realiza un ciclo extra compuesto por las transformaciones Bytesub, Shiftrow, Addroundkey sin usar la transformación Mixcolumn.

Para obtener las subclaves que se aplican en cada ciclo se genera una serie de expansiones de la clave original, cada una del mismo tamaño que ésta. Las expansiones se realizan mediante un procedimiento en la especificación del algoritmo, que consiste en añadir palabras de cuatro octetos obtenidas mediante operaciones XOR entre otras partes prefijadas de la expansión anterior. Para obtener la primera palabra de cuatro octetos e iniciar la expansión, se hace uso de la transformación mediante uso de cajas – S, que opera en el paso Bytesub y de una operación XOR con una constante específica predeterminada en función del ciclo [8].

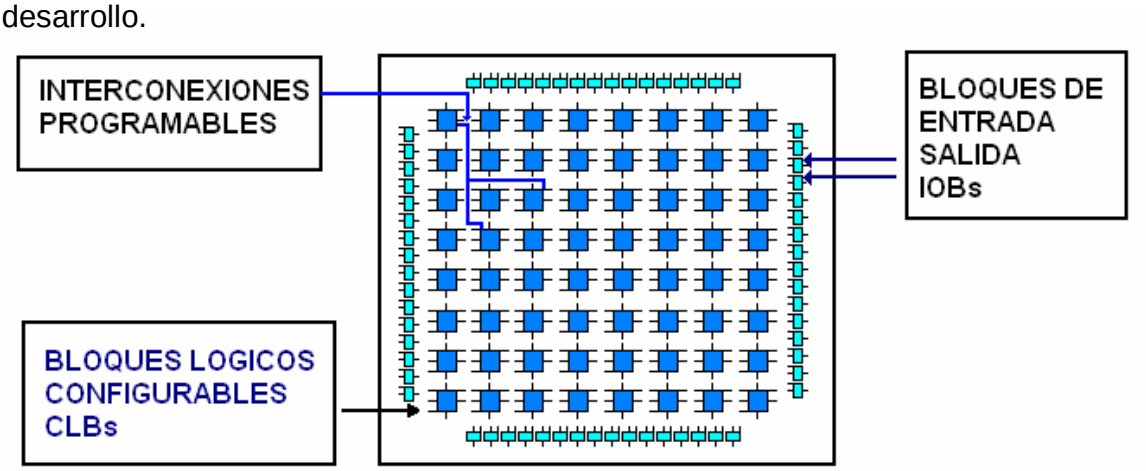
1.5.2 Proceso de Descifrado

En el proceso de descifrado se procede en el orden inverso al cifrado, realizando también en cada paso la operación inversa. La transformación Addroundkey, por estar constituida por una operación XOR, es inversa de si misma, pero en el caso de las otras tres transformaciones es necesario usar la operación inversa.
La generación de subclaves para este proceso se lleva a cabo de la misma a la que se hizo en el proceso de cifrado solo que el orden de la utilización de las claves es también inverso.

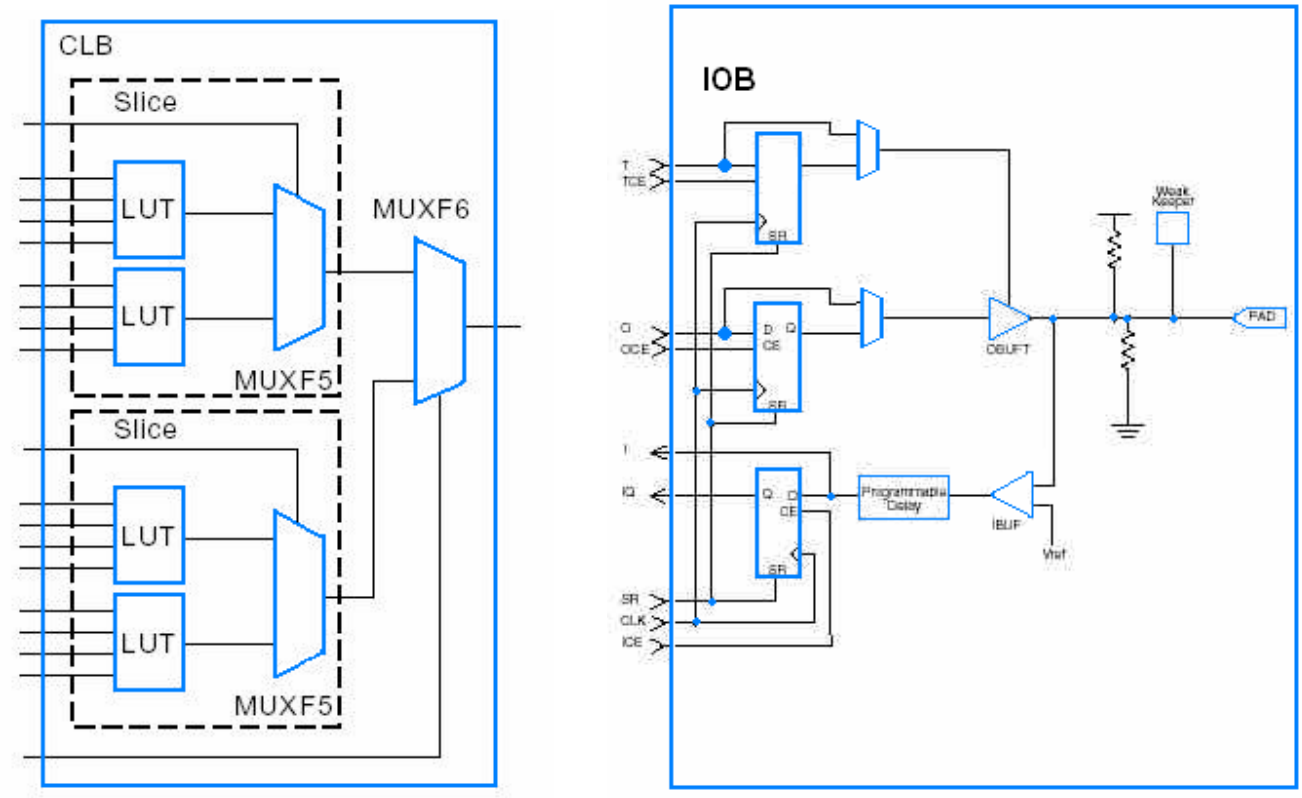
1.6 DISPOSITIVOS FPGA

Las FPGA’s son dispositivos programables de alta capacidad. Están diseñados para cumplir una función específica de acuerdo con las necesidades, los cuales están sustituyendo a la lógica tradicional o estándar en algunas aplicaciones.

Las principales ventajas que presenta la utilización de las FPGA's son: su funcionalidad, fiabilidad, miniaturización, robustez, bajo consumo y sencillez de desarrollo.



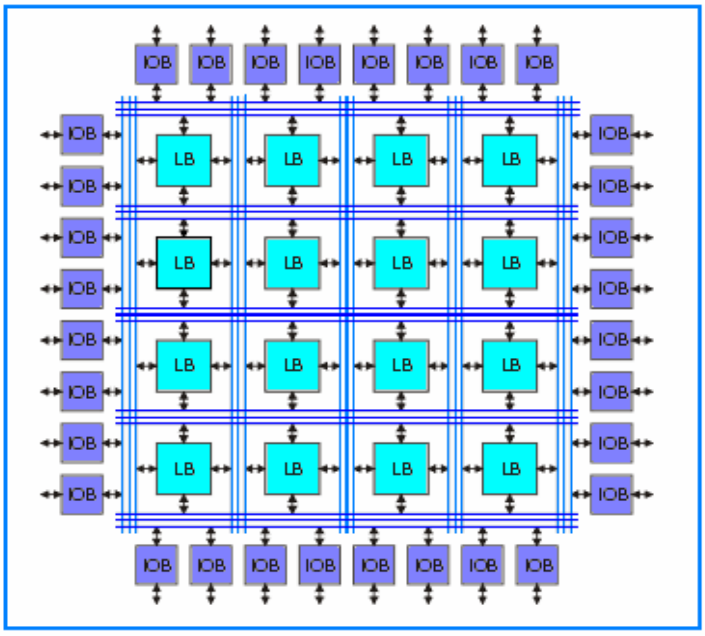
Arquitectura básica de un FPGA
Figura 18



Estructura interna a nivel lógico
Figura 19

Los elementos básicos que constituyen una FPGA de Xilinx son los siguientes:

- Bloques lógicos, su estructura y contenido se denomina arquitectura. Suelen incluir biestables para facilitar la implementación de circuitos secuenciales. Otros módulos de importancia son los bloques de Entrada/Salida.
- Recursos de interconexión, su estructura y contenido se llama arquitectura de rutado.
- Memoria RAM, que se carga durante el RESET para configurar bloques y conectarlos.



Estructura general de una FPGA XILINX
Figura 20

1.7 LENGUAJE VHDL

VHDL es un lenguaje de descripción de hardware para modelar sistemas digitales. Definido por IEEE (Institute of Electrical and Electronics Engineers) (ANSI/IEEE 1076-1993).

En VHDL hay varias formas en las que se puede diseñar un circuito las cuales son:

- Funcional**:** describe la forma en que se comporta el circuito. Es la forma que más se parece a los lenguajes de software debido a que la descripción es

secuencial. Estas sentencias secuenciales se encuentran dentro de los procesos en VHDL.

- Flujo de datos: describe asignaciones en paralelo de señales.

- Estructural: se describe el circuito con componentes. Forman un diseño de jerarquía superior
- Mixta: combinación de todas o algunas de las formas de diseño.

En VHDL también existen formas metódicas para el diseño de máquinas de estados, filtros digitales, bancos de pruebas etc.

Un diseño en VHDL posee dos partes principales:

- Entidad, es como una caja negra en la que se definen entradas y salidas pero no se tiene acceso al interior, y es lo que usa cuanto se reutiliza un diseño dentro de otro.
- Arquitectura, que es donde se describe el diseño.
- Otros elementos del lenguaje son las librerías, paquetes, funciones.

AES usa campos aritméticos finitos para sus operaciones, una de las características de los campos finitos aritméticos es la adición y sustracción que son hechas por operaciones XOR de dos entradas, esta operación no produce bit de acarreo, la velocidad de las operaciones computacionales es considerable, haciendo el algoritmo AES un buen candidato para FPGAs

METODOLOGIA

2.1 FASES DEL PROYECTO

2.1.1 Estudio temático y análisis de los conceptos preliminares

En esta fase se estudiaron los conceptos sobre criptología y los requerimientos hardware del sistema a implementar.

2.1.1.1 Análisis de seguridad informática en las telecomunicaciones

Se estudiaron los aspectos básicos de los servicios de la seguridad informática y algunas implicaciones en las telecomunicaciones.

Los componentes de un sistema informático están expuestos a ataques, por lo cual los datos y la información son los principales aspectos de protección de las técnicas de seguridad. La seguridad informática se dedica principalmente a proteger la confidencialidad, la integridad y disponibilidad de la información, así como también brinda los servicios de autentificación, no repudio y controles de acceso para que la información que se considera importante no sea fácil de de acceder por cualquier persona que no se encuentre acreditada .

2.1.1.2 Análisis del Estándar de Encriptación Avanzado AES

Se analizó la estructura del algoritmo Rijndael, en el cual esta basado el estándar de encriptación avanzado AES, esta estructura consiste en dos partes, la primera es el proceso de cifrado y la segunda el proceso de generación de subclaves. Se eligió trabajar con el mínimo recomendado por AES, con texto pleno de 128 bits y clave de igual longitud.

2.1.1.3 Análisis de los conceptos matemáticos requeridos para la implementación del algoritmo Rijndael

En esta etapa se estudiaron los elementos matemáticos necesarios para poder Implementar AES. Los conceptos que se analizan son: grupos, anillos, el campo finito GF (2^8) y teoría de Galois.

CAMPO: Es un anillo de división conmutativo, es decir, en el que todo elemento distinto de cero es invertible respecto del producto.

Un Campo es una estructura algebraica en la cual las operaciones de adición, substracción, multiplicación, y división (excepto la división por cero) se pueden realizar y cumplen las propiedades asociativa, conmutativa y distributiva. Ejemplo: los números reales (a excepción del 0) forman un campo.

CAMPO DE GALOIS (GF): Es un campo finito binario de la forma p^m donde p es un numero primo y m regularmente debe serlo, la idea de los campos de Galois es que basado en un campo pequeño F (p), se pueda hacer una extensión del campo para que en vez de tener p elementos tenga p^m elementos, esto es soportado por la Teoría de Galois en honor al matemático Evariste Galois quien fue el primero en estudiar este tipo de estructuras.

ANILLO: Es una estructura algebraica formada por un conjunto y dos operaciones (Suma y producto) que cumplen las propiedades distributiva, conmutativa, asociativa y la existencia del elemento identidad. Ejemplo: El anillo formado por los números enteros con respecto a las operaciones suma y producto clásicas. Los enteros no son un campo porque no esta definida la operación de inversión, puesto que por ejemplo el inverso de 3 es 1/3 (3 * 1/3 = 1), pero 1/3 no pertenece a los enteros.

COPRIMO: Se conoce también como primo relativo, dos números a y b son coprimos o primos relativos si se cumple que MCD (a, b) =1.

POLINOMIO IRREDUCIBLE: Es un polinomio definido sobre un campo, que no tiene raíces en el mismo, es decir no se puede factorizar.

2.1.1.4 Estudio de los requerimientos de la implementación del hardware

Se estudió la estructura del hardware reconfìgurable (FPGA), con el fin de realizar una descripción del dispositivo de una forma óptima, para llevar a cabo esto se estudiaron el lenguaje de descripción VHDL y los entornos de descripción.

VHDL permite realizar una descripción interna en un formato equivalente a nivel de compuertas, este formato es óptimo para las FPGA.

VHDL es uno de los pocos lenguajes de descripción de hardware, el cual se encuentra dividido en componentes, circuitos o sistemas, en una parte externa o visible nombre de entidades y conexiones y en una parte interna u oculta algoritmos de entidades e implementaciones

2.1.2 Fase de diseño

2.1.2.1 Diseño del Criptoprocesador

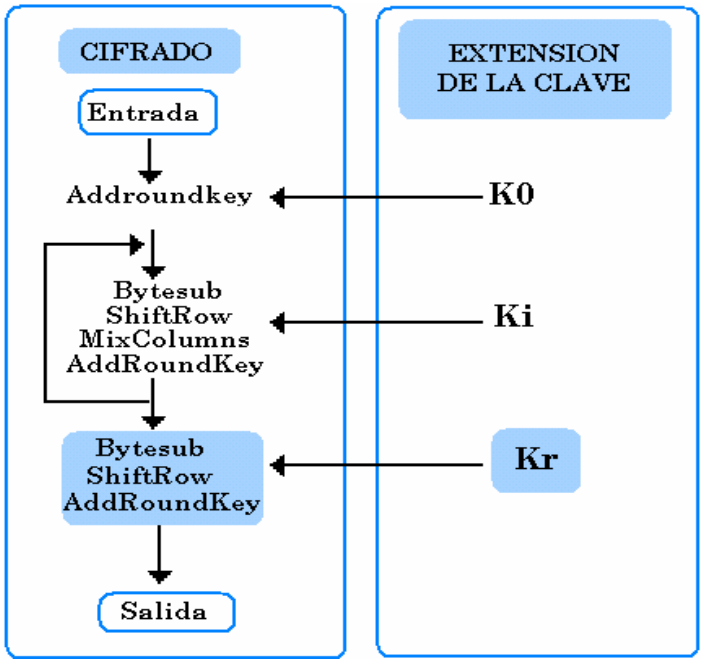
En esta fase de diseño se tienen en cuenta la toda la estructura del algoritmo Rijndael, los elementos matemáticos y las características de la FPGA.

Principalmente se divide en dos grandes bloques que son:

- Bloque de CIFRADO
- Bloque de DESCIFRADO

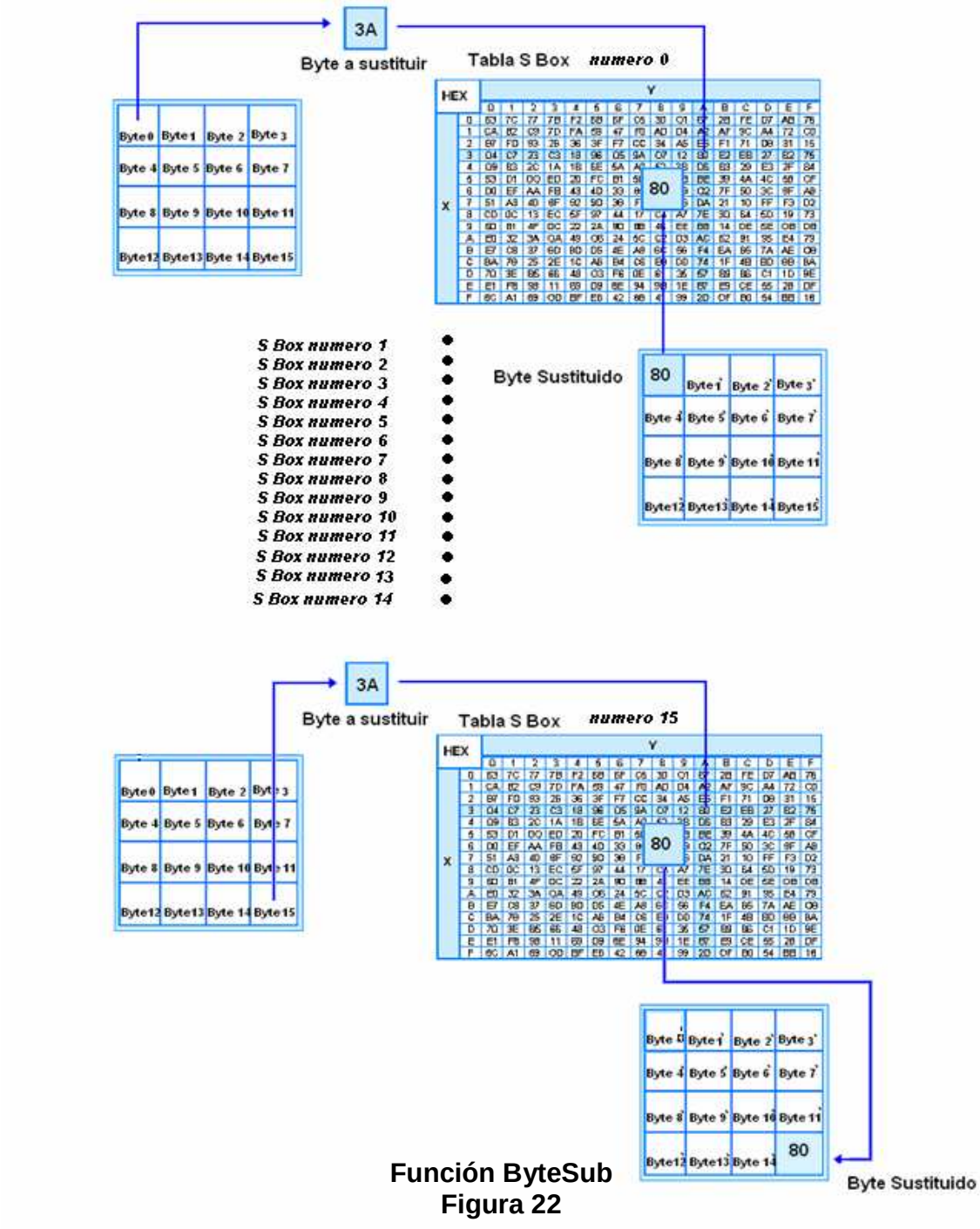
2.1.2.1.1 BLOQUE DE CIFRADO

Este bloque se encuentra constituido por otros bloques que conforman todo el proceso de cifrado del algoritmo AES – Rijndael.



Proceso de cifrado
Figura 21

2.1.2.1.1.1 BLOQUE BYTESUB-SHIFTROW

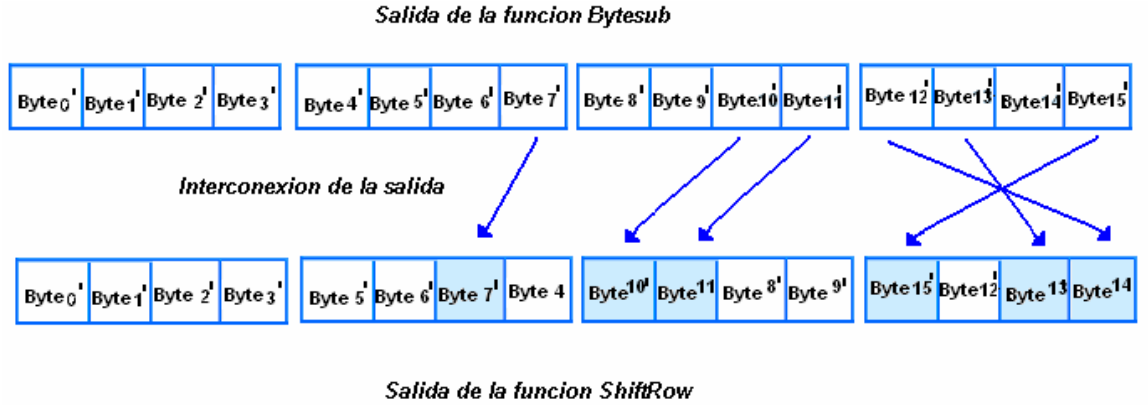


Función ByteSub
Figura 22

En este bloque se implementaron dos funciones: bytesub y shiftrow.

La primera función es una transformación no lineal de sustitución de bytes en la cual utiliza una tabla S-box (Tabla 4) en el momento del cifrado, las cuales están implementadas en la FPGA ya que no se utilizan recursos externos.

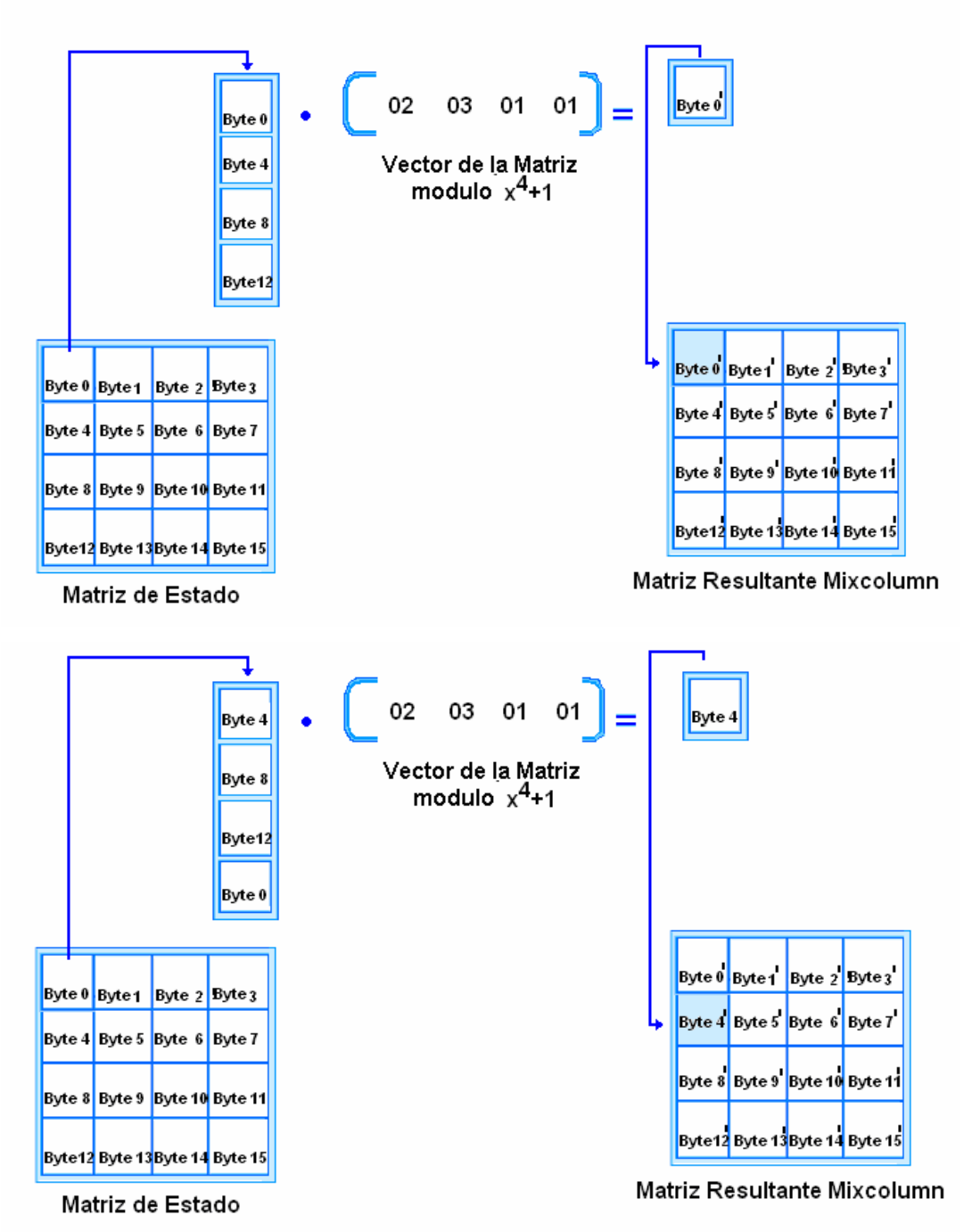
La descripción en hardware que se diseñó para esta etapa consiste en generar 16 tablas S-box (aprovechando la capacidad de paralelismo), el proceso se desarrolla a nivel de bytes, se planteó una memoria ROM de 256 posiciones la cual es la tabla S-box y funciona con cada byte de entrada correspondiente a la matriz de estado a transformar, se toma como dirección para la S-box, dicha dirección contiene un valor (byte) el cual va ser el byte de salida resultante (se genera sustitución) y así mismo se producen hasta completar los 128bits (16 bytes).



Función ShiftRow
Figura 23

En la siguiente función se realiza una reconexión de las salidas la función ByteSub en la cual se efectúa una reconexión en la segunda fila, que esta constituida por los byte 4, byte 5, byte 6 y byte 7; dos reconexiones en la tercera fila, que esta constituida por los byte 8, byte 9, byte 10 y byte 11; tres reconexión en la cuarta fila, que esta constituida por los byte 12, byte 13, byte 14 y byte 15 y la primera fila no sufre ninguna modificación. Esta operación se realiza en hardware sin costo de elementos adicionales.

2.1.2.1.1.2 BLOQUE MIXCOLUMN



Función MixColumn
Figura 24

La función MixColumn en la etapa de cifrado consiste en multiplicar las columnas de bytes módulo x^4+1 por el polinomio c(x).
Matemáticamente c(x) esta representado por:

$$C(x) = 03_{16}x^3 + 01_{16}x^2 + 01_{16}x + 02_{16} \quad (17)$$

La descripción en hardware que se diseñó para esta etapa fue desarrollando operaciones a nivel de bytes, en la operación mixcolumn como su mismo nombre lo dice es una mezcla de columnas, cada columna esta constituida por 32 bits (4 bytes), los cuales están distribuidos

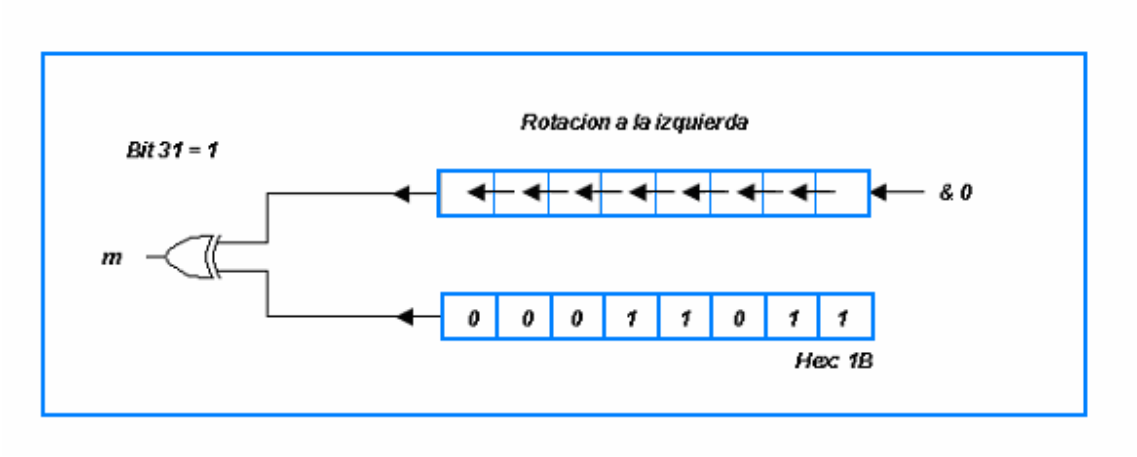
- byte 1 (31 downto 24)
- byte 2 (23 downto 16)
- byte 3 (15 downto 8)
- byte 4 (7 downto 0)

La primera operación a ejecutar es realizar la multiplicación del byte 1 por (02_{16}) ó multiplicación por **X**

x· a(x) **siendo a =** $a_7x^7 + a_6x^6 + a_5x^5 + a_4x^4 + a_3x^3 + a_2x^2 + a_1x^1 + a_0x^0$

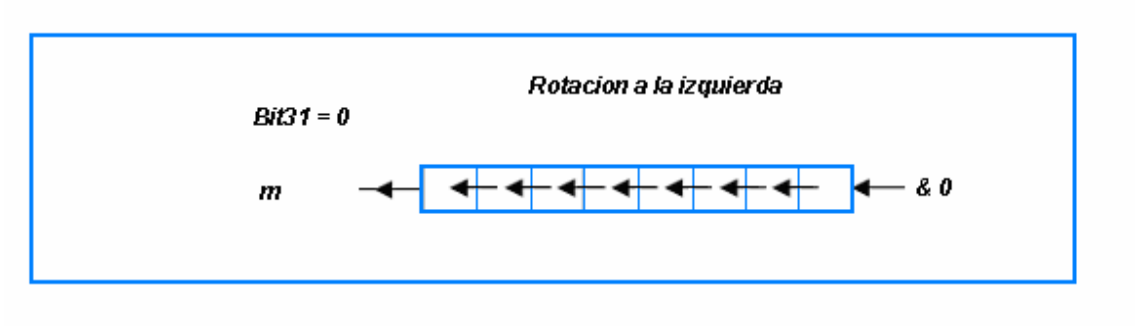
dando como resultado $a_8x^8 + a_7x^7 + a_6x^6 + a_5x^5 + a_4x^4 + a_3x^3 + a_2x^2 + a_1x^1 + a_0x^0$

Se tiene en cuenta el bit 31, si este bit se encuentra en un estado lógico '1' se efectúa un corrimiento hacia la izquierda y luego este bit resultante se le realiza una xor con **1B** (hex), obteniendo el polinomio resultante con un grado menor a 8. Esta operación en bytes se denomina Xtime.



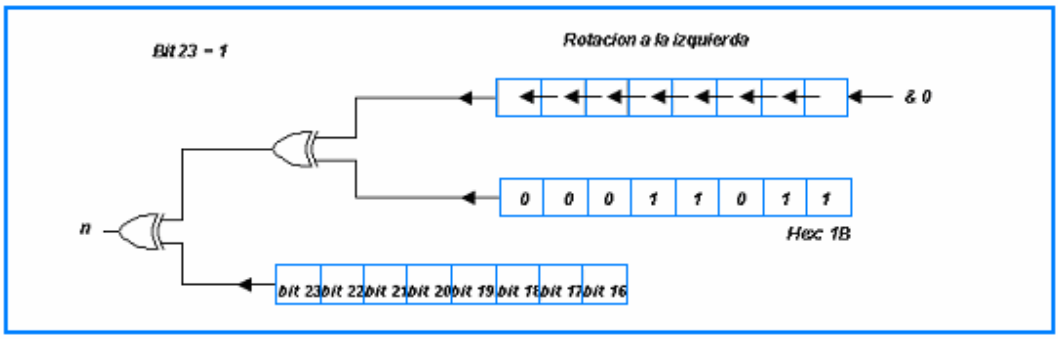
Multiplicación por Dos Bit 31=1
Figura 25

Si este bit 31 se encuentra en un estado lógico '0' solo se efectúa un corrimiento hacia la izquierda.



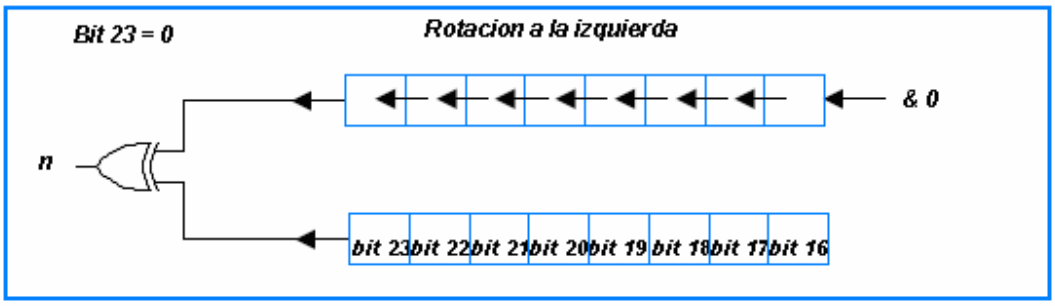
Multiplicación por Dos Bit 31=0
Figura 26

La siguiente operación a ejecutar es realizar la multiplicación del byte 2 por (03_{16})
(X+1)· a(x) siendo a = $a_7x^7 + a_6x^6 + a_5x^5 + a_4x^4 + a_3x^3 + a_2x^2 + a_1x^1 + a_0x^0$
dando como resultado $a_8x^8 + a_0x^0$
Se tiene en cuenta el bit 23, si este bit se encuentra en un estado lógico '1' se efectúa el mismo procedimiento que al multiplicar por **X** luego este bit resultante se le realiza una xor con el byte2 (original) , obteniendo el polinomio resultante con un de grado menor a 8.



Multiplicación por Tres Bit 23=1
Figura 27

Si este bit 23 se encuentra en un estado lógico '0' el mismo procedimiento que al multiplicar por **X** luego este bit resultante se le realiza una xor con el byte2 (original).



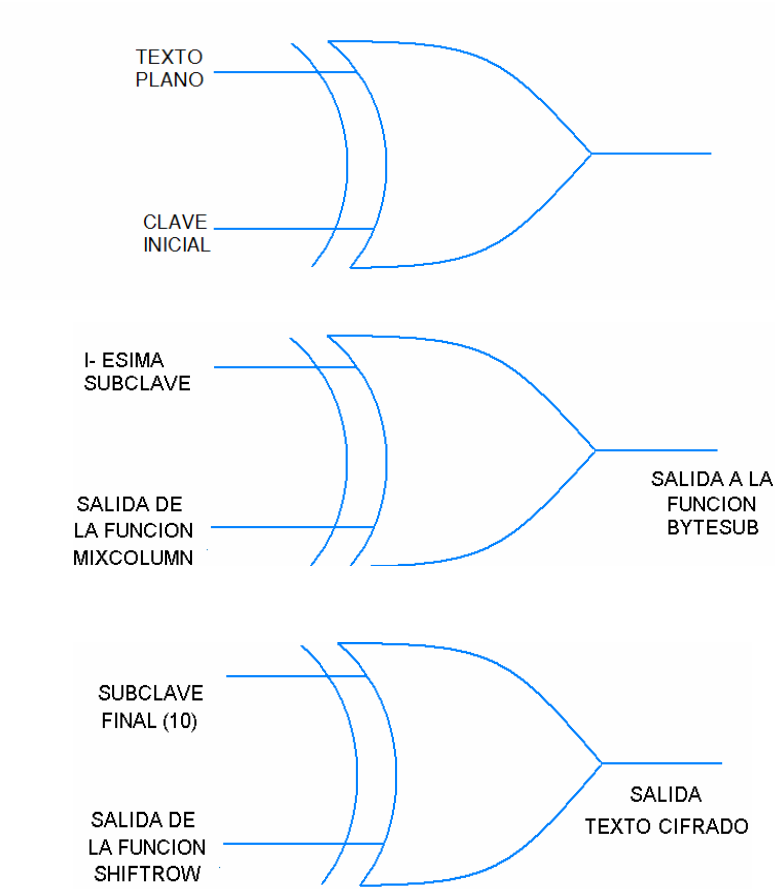
Multiplicación por Tres Bit 23=0
Figura 28

El byte 3 y byte 4 no se les multiplica por ningún factor ya que cumplen la propiedad neutra de la multiplicación.

Luego para obtener un byte resultante se realiza la operación xor entre el byte resultante de la multiplicación por (02_{16}), el byte resultante de la multiplicación por (03_{16}), el byte 3 y byte 4.

Seguido a estos procedimientos se realizan unos corrimientos dentro de cada una de las cuatro columnas para poder obtener los 16 bytes (128 bits) que conforman la matriz intermedia necesaria para realizar la siguiente operación que es addroundkey.

2.1.2.1.1.3 BLOQUE ADDROUNDKEY

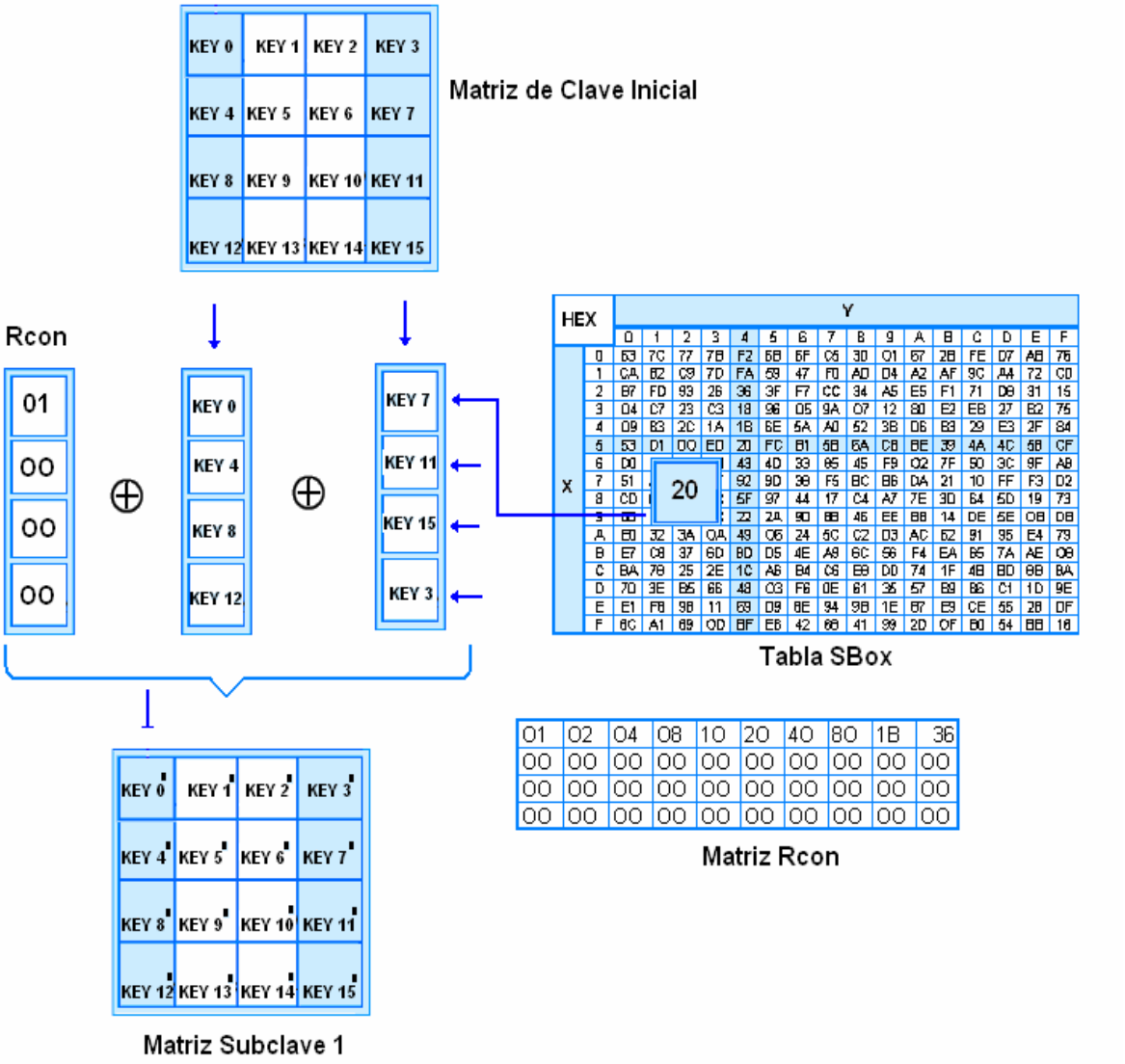


Bloque Addroundkey
Figura 29

Inicialmente esta función consiste en realizar una operación OR-Exclusiva entre el texto plano y la clave inicial (Ronda Inicial); durante el proceso de cifrado también se realiza una operación OR-Exclusiva entre la matriz de Estado que proviene de la transformación anterior (Función MixColumn) y la subclave generada a partir de la clave del sistema para esa ronda (Ronda Estandar). El bloque resultante de esta transformación, es la nueva matriz de Estado para la siguiente ronda.

En la ronda final también se efectúa una operación OR-Exclusiva entre la subclave final y la matriz de Estado que proviene de la transformación anterior (Función ShifRow), hallando dato de salida del algoritmo (Dato Cifrado).

2.1.2.1.1.4 BLOQUE DE GENERACION DE CLAVES



Generación de subclaves
Figura 30

En este bloque se generan las 10 subclaves necesarias para el proceso de cifrado a partir de la clave inicial.

El proceso consiste en trabajar a nivel de 32 bits (por columnas), se aborda eligiendo la cuarta columna de la matriz de clave inicial, la cual esta constituida por los bytes 3, 7, 11 y 15, se realiza una rotación del primer byte quedando la columna formada con bytes 7, 11, 15 y 3 (RotWord). A esta columna resultante se

le realiza la sustitución mediante la tabla S-box (SRotWord), luego se realizan operaciones xor entre la columna sustituida (SRotWord), Rcon y la primera columna de la matriz dando como resultado la primera columna de la matriz de la primera subclave. Las siguientes tres columnas se determinan de la siguiente manera:

$$W_i = W_{i-1} \text{ XOR } W_{i-4} \qquad (18)$$

El proceso continúa de igual forma hasta obtener las 10 matrices de las subclaves. En donde Rcon se obtiene de la siguiente forma:

Rcon [i] = (RC (i), 00, 00, 00)

RC [i] = 1

RC [i] = X * RC [i -1] = X^{i-1}

RC [1] = 1

RC [2] = X = 2

RC [3] = X^2 = 4

RC [4] = X^3 = 8

RC [5] = X^4 = 10

RC [6] = X^5 = 20

RC [7] = X^6 = 40

RC [8] = X^7 = 80

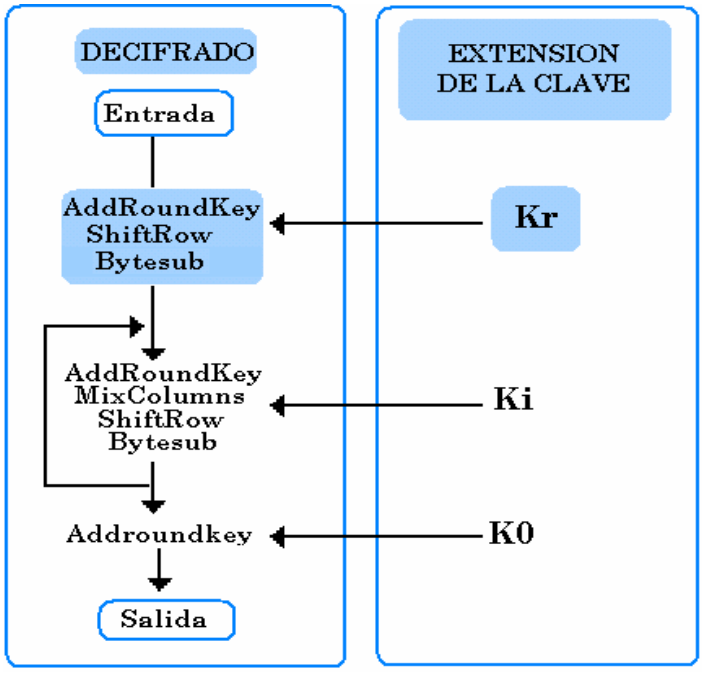
RC [9] = $X^8 = X^4 + X^3 + X + 1$ = 1B

RC [10] = $X^3 = X^5 + X^4 + X^2 + X$ = 36

[2]

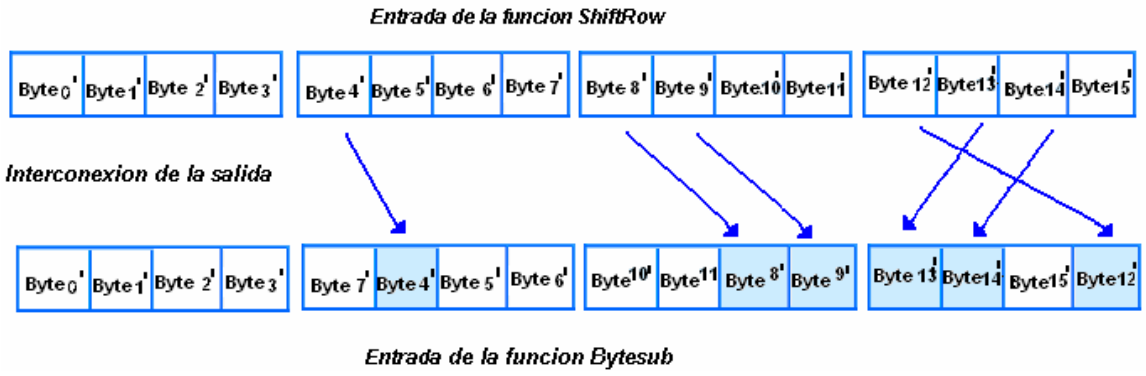
2.1.2.1.2 BLOQUE DE DESCIFRADO

Este bloque se encuentra constituido por los mismos bloques que conforman todo el proceso de cifrado del algoritmo AES – Rijndael solo que en diferente orden.



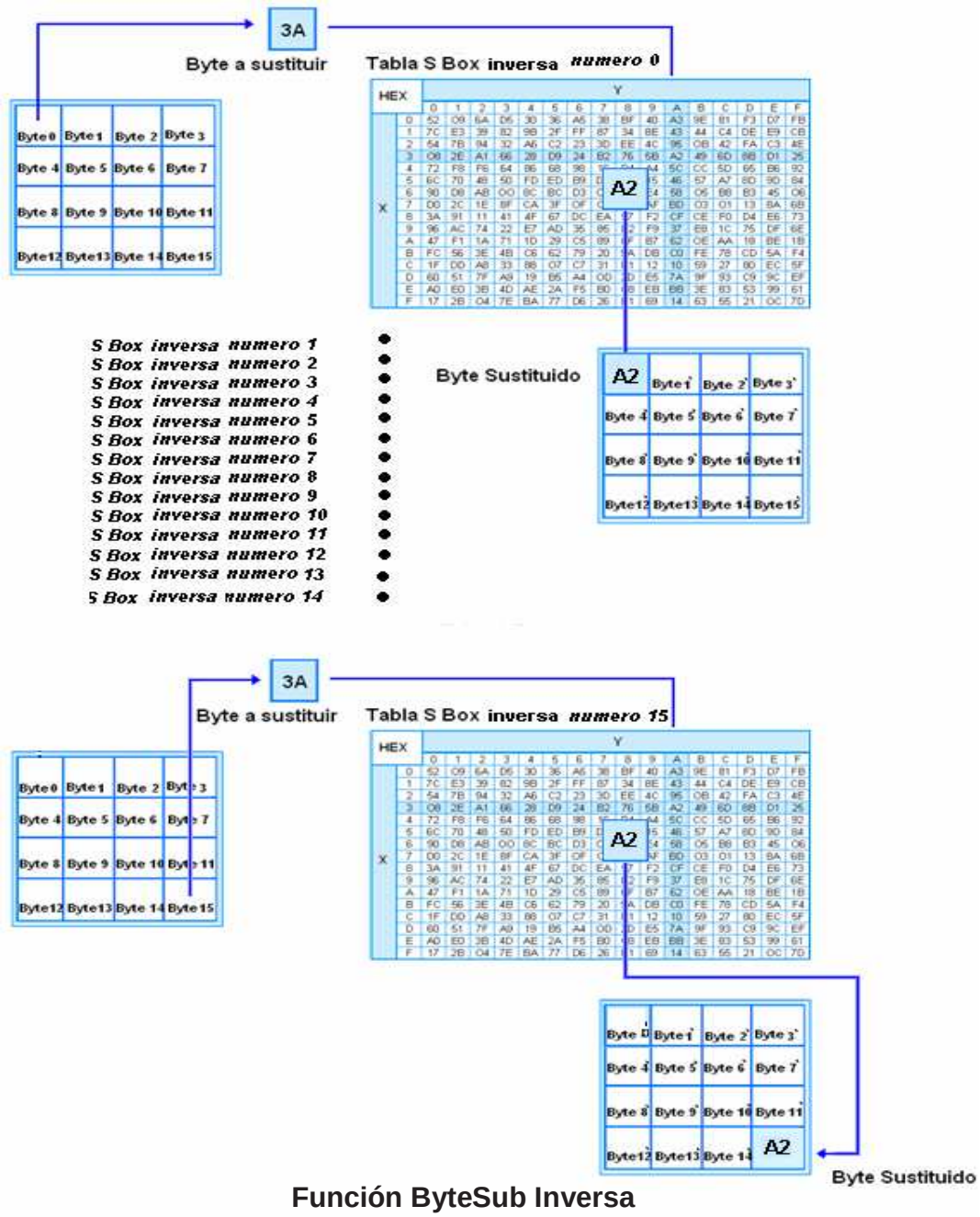
Proceso de descifrado
Figura 31

2.1.2.1.2.1 BLOQUE BYTESUB-SHIFTROW



Función ShiftRow Inversa
Figura 32

En esta función se realiza una reconexión de las entradas la función ShiftRow, en el que se genera la salida para la función ByteSub efectuando una reconexión en la segunda fila, que esta constituida por los byte 4, byte 5, byte 6 y byte 7; dos reconexión en la tercera fila, que esta constituida por los byte 8, byte 9, byte 10 y byte 11; tres reconexión en la cuarta fila, que esta constituida por los byte 12, byte 13, byte 14 y byte 15 y la primera fila no sufre ninguna modificación.



Función ByteSub Inversa
Figura 33

La función ByteSub es una transformación no lineal de sustitución de bytes en la cual utiliza una tabla s-box inversa (Tabla 5), las cuales están implementadas en la FPGA ya que no se utilizan recursos externos.

La descripción en hardware que se diseño para esta etapa consiste en generar 16 tablas s-box inversa (paralelismo), el proceso se desarrolla a nivel de bytes, se planteo una memoria ROM de 256 posiciones la cual es la tabla s-box inversa y radica en que cada byte de entrada correspondiente a la matriz de estado a transformar es una dirección para la s-box inversa, dicha dirección contiene un valor (byte) el cual va ser el byte de salida resultante (se genera sustitución) y así mismo se producen hasta completar los 128bits (16 bytes).

2.1.2.1.2.2 BLOQUE MIXCOLUMN

En el bloque de descifrado fue necesario el desarrollo de multiplicadores en la función de MixColumn debido a que las multiplicaciones no se lograban simplificar como en la etapa de cifrado.

2.1.2.1.2.2.1 MULTIPLICADOR GF [2^8]

Para implementar las operaciones básicas del algoritmo Rinjdael tal como MixColumn es necesario utilizar la operación aritmética de la multiplicación en el campo finito GF (2^8), debido a que esta se constituye como base para el desarrollo de las transformaciones del algoritmo.

La reducción modular por m(x) asegura que el polinomio resultante sea de grado menor a 8 y así puede ser por un byte, AES utiliza como polinomio irreducible a:

$$m(x) = X^8 + X^4 + X^3 + X + 1 \qquad (18)$$

Binario: ***100011011*** *Decimal****: 283***

Hexadecimal: ***11b*** *Orden: 51*

Las operaciones aritméticas suma y multiplicación cumplen las siguientes propiedades:

a + b = b + a	**Conmutativa**	a * b = b * a
a + [b + c] = [a + b] + c	**Asociativa**	a * [b * c] = [a * b] * c
a + 0 = a	**Neutro**	a * 1 = a
a + (-a) = 0	**Inverso**	a * 1/a = 1

Propiedades de la Suma y Multiplicación
Tabla 9

$$m(x) = X^8 + X^4 + X^3 + X + 1$$
$$X^8 = X^4 + X^3 + X + 1$$

La inversa de Mezclar Columnas se obtiene de multiplicar cada columna de la matriz de estado por el polinomio d(x):

$$d(x) = 0B_{16}x^3 + 0D_{16}x^2 + 09_{16}x + 0E_{16} \quad (19)$$

Este polinomio, permite realizar la inversa del polinomio c(x), cumpliendo que:

$$C(x) \oplus d(x) = 01 \quad (20)$$

La transformación inversa se puede mostrar de forma matricial así:

$$S(x) = d(x) \oplus S'(x) \quad (21)$$

$$\begin{bmatrix} S_{0,c} \\ S_{1,c} \\ S_{2,c} \\ S_{3,c} \end{bmatrix} = \begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \begin{bmatrix} S'_{0,c} \\ S'_{1,c} \\ S'_{2,c} \\ S'_{3,c} \end{bmatrix}$$

Transformación Inversa
Figura 34

Para desarrollar esta función se diseñaron 4 multiplicadores:

- Multiplicador 0E

Teniendo dos elementos **a** y **b** que pertenecen al campo Finito **GF (2^8),** siendo **b 0E** (hexadecimal) **ó 00001110** (binario)**:**

a = $a_7z^7 + a_6z^6 + a_5z^5 + a_4z^4 + a_3z^3 + a_2z^2 + a_1z^1 + a_0z^0$

b = $b_3z^3 + b_2z^2 + b_1z^1$

a * b = $a_7 b_3z^{10} + a_7 b_2z^9 + a_7 b_1z^8 + a_6 b_3z^9 + a_6 b_2z^8 + a_6 b_1z^7 + a_5 b_3z^8 + a_5 b_2z^7 + a_5 b_1z^6 + a_4 b_3z^7 + a_4 b_2z^6 + a_4 b_1z^5 + a_3 b_3z^6 + a_3 b_2z^5 + a_3 b_1z^4 + a_2 b_3z^5 + a_2 b_2z^4 + a_2 b_1z^3 + a_1 b_3z^4 + a_1 b_2z^3 + a_1 b_1z^2 + a_0 b_3z^3 + a_0 b_2z^2 + a_0 b_1z^1$

Como resultado de la multiplicación existen elementos que se salen del campo GF (2^8), es necesario utilizar el polinomio irreducible $X^8 = X^4 + X^3 + X + 1$ y *realizar* las siguientes sustituciones:

$z^{10} = z^6 + z^5 + z^3 + z^2$

$z^9 = z^5 + z^4 + z^2 + z^1$

Desarrollando el reemplazo se obtiene:

a * b = $a_7 b_3z^6 + a_7 b_3z^5 + a_7 b_3z^3 + a_7 b_3z^2 + a_6 b_3z^5 + a_6 b_3z^4 + a_6 b_3z^2 + a_6 b_3z^1 + a_5 b_3z^4 + a_5 b_3z^3 + a_5 b_3z^1 + a_5 b_3z^0 + a_4 b_3z^7 + a_3 b_3z^6 + a_2 b_3z^5 + a_1 b_3z^4 + a_0 b_3z^3 + a_7 b_2z^5 + a_7 b_2z^4 + a_7 b_2z^2 + a_7 b_2z^1 + a_6 b_2z^4 + a_6 b_2z^3 + a_6 b_2z^1 + a_6 b_2z^0 + a_5 b_2z^7 + a_4 b_2z^6 + a_3 b_2z^5 + a_2 b_2z^4 + a_1 b_2z^3 + a_0 b_2z^2 + a_7 b_1z^4 + a_7 b_1z^3 + a_7 b_1z^1 + a_7 b_1z^0 + a_6 b_1z^7 + a_5 b_1z^6 + a_4 b_1z^5 + a_3 b_1z^4 + a_2 b_1z^3 + a_1 b_1z^2 + a_0 b_1z^1.$

a * b = $\mathbf{z^0}\,[a_5b_3 + a_6b_2 + a_7b_1] +$

$\mathbf{z^1}\,[a_6b_3 + a_5b_3 + a_7b_2 + a_6b_2 + a_7b_1 + a_0b_1] +$

$\mathbf{z^2}\,[a_7b_3 + a_6b_3 + a_7b_2 + a_0b_2 + a_1b_1] +$

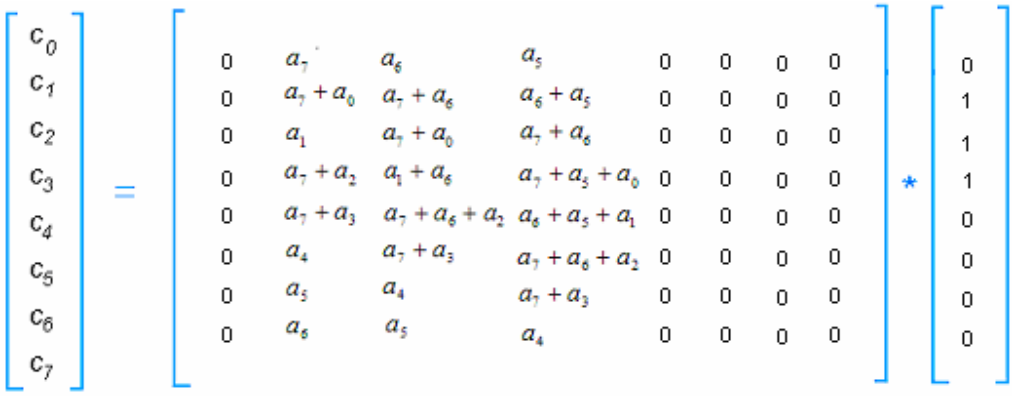
$\mathbf{z^3}\,[a_7b_3 + a_5b_3 + a_0b_3 + a_6b_2 + a_1b_2 + a_7b_1 + a_2b_1] +$

$\mathbf{z^4}\,[a_6b_3 + a_5b_3 + a_1b_3 + a_7b_2 + a_6b_2 + a_2b_2 + a_7b_1 + a_3b_1] +$

$\mathbf{z^5}\,[a_7b_3 + a_6b_3 + a_2b_3 + a_7b_2 + a_3b_2 + a_4b_1] +$

$\mathbf{z^6}\,[a_7b_3 + a_3b_3 + a_4b_2 + a_5b_1] +$

$\mathbf{z^7}\,[a_4b_3 + a_5b_2 + a_6b_1]$



Multiplicador 0E en forma matricial
Figura 35

- Multiplicador 0B

Teniendo dos elementos ***a*** *y* ***b*** *que pertenecen al campo Finito* ***GF*** (2^8)***,*** *siendo* ***b 0B*** *(hexadecimal)* ***ó 00001011*** *(binario)****:***

a = $a_7z^7 + a_6z^6 + a_5z^5 + a_4z^4 + a_3z^3 + a_2z^2 + a_1z^1 + a_0z^0$

b = $b_3z^3 + b_1z^1 + b_0z^0$

a * b = $a_7 b_3z^{10} + a_7 b_1z^8 + a_7 b_0z^7 + a_6 b_3z^9 + a_6 b_1z^7 + a_6 b_0z^6 + a_5 b_3z^8 + a_5 b_1z^6 +$
$a_5 b_0z^5 + a_4 b_3z^7 + a_4 b_1z^5 + a_4 b_0z^4 + a_3 b_3z^6 + a_3 b_1z^4 + a_3 b_0z^3 + a_2 b_3z^5 +$
$a_2 b_1z^3 + a_2 b_0z^2 + a_1 b_3z^4 + a_1 b_1z^2 + a_1 b_0z^1 + a_0 b_3z^3 + a_0 b_1z^1 + a_0 b_0z^0.$

a * b = $a_7 b_3z^6 + a_7 b_3z^5 + a_7 b_3z^3 + a_7 b_3z^2 + a_6 b_3z^5 + a_6 b_3z^4 + a_6 b_3z^2 + a_6 b_3z^1 +$
$a_5 b_3z^4 + a_5 b_3z^3 + a_5 b_3z^1 + a_5 b_3z^0 + a_4 b_3z^7 + a_3 b_3z^6 + a_2 b_3z^5 + a_1 b_3z^4 +$
$a_0 b_3z^3 + a_7 b_1z^4 + a_7 b_1z^3 + a_7 b_1z^1 + a_7 b_1z^0 + a_6 b_1z^7 + a_5 b_1z^6 + a_4 b_1z^5 +$
$a_3 b_1z^4 + a_2 b_1z^3 + a_1 b_1z^2 + a_0 b_1z^1 + a_7 b_0z^7 + a_6 b_0z^6 + a_5 b_0z^5 + a_4 b_0z^4 +$
$a_3 b_0z^3 + a_2 b_0z^2 + a_1 b_0z^1 + a_0 b_0z^0.$

a * b = $\mathbf{z^0}\,[a_5b_3 + a_7b_1 + a_0b_0] +$
$\mathbf{z^1}\,[a_6b_3 + a_5b_3 + a_7b_1 + a_0b_1 + a_1b_0] +$
$\mathbf{z^2}\,[a_7b_3 + a_6b_3 + a_1b_1 + a_2b_0] +$

z^3 $[a_7b_3 + a_5b_3 + a_0b_3 + a_7b_1 + a_2b_1 + a_3b_0]$ +

z^4 $[a_6b_3 + a_5b_3 + a_1b_3 + a_7b_1 + a_3b_1 + a_4b_0]$ +

z^5 $[a_7b_3 + a_6b_3 + a_2b_3 + a_4b_1 + a_5b_0]$ +

z^6 $[a_7b_3 + a_3b_3 + a_5b_1 + a_6b_0]$ +

z^7 $[a_4b_3 + a_6b_1 + a_7b_0]$

$$\begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \\ c_6 \\ c_7 \end{bmatrix} = \begin{bmatrix} a_0 & a_7 & 0 & a_5 & 0 & 0 & 0 & 0 \\ a_1 & a_7 + a_0 & 0 & a_6 + a_5 & 0 & 0 & 0 & 0 \\ a_2 & a_1 & 0 & a_7 + a_6 & 0 & 0 & 0 & 0 \\ a_3 & a_7 + a_2 & 0 & a_7 + a_5 + a_0 & 0 & 0 & 0 & 0 \\ a_4 & a_7 + a_3 & 0 & a_6 + a_5 + a_1 & 0 & 0 & 0 & 0 \\ a_5 & a_4 & 0 & a_7 + a_6 + a_2 & 0 & 0 & 0 & 0 \\ a_6 & a_5 & 0 & a_7 + a_3 & 0 & 0 & 0 & 0 \\ a_7 & a_6 & 0 & a_4 & 0 & 0 & 0 & 0 \end{bmatrix} * \begin{bmatrix} 1 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Multiplicador 0B en forma matricial
Figura 36

- Multiplicador 0D

Teniendo dos elementos ***a*** *y* ***b*** *que pertenecen al campo Finito* ***GF*** (2^8)***,*** *siendo* ***b 0D*** *(hexadecimal)* ***ó 00001101*** *(binario)****:***

a = $a_7z^7 + a_6z^6 + a_5z^5 + a_4z^4 + a_3z^3 + a_2z^2 + a_1z^1 + a_0z^0$

b = $b_3z^3 + b_2z^2 + b_0z^0$

a * b = $a_7 b_3z^{10} + a_7 b_2z^9 + a_7 b_0z^7 + a_6 b_3z^9 + a_6 b_2z^8 + a_6 b_0z^6 + a_5 b_3z^8 + a_5 b_2z^7 +$
$a_5 b_0z^5 + a_4 b_3z^7 + a_4 b_2z^6 + a_4 b_0z^4 + a_3 b_3z^6 + a_3 b_2z^5 + a_3 b_0z^3 + a_2 b_3z^5 +$
$a_2 b_2z^4 + a_2 b_0z^2 + a_1 b_3z^4 + a_1 b_2z^3 + a_1 b_0z^1 + a_0 b_3z^3 + a_0 b_2z^2 + a_0 b_0z^0.$

a * b = $a_7 b_3z^6 + a_7 b_3z^5 + a_7 b_3z^3 + a_7 b_3z^2 + a_6 b_3z^5 + a_6 b_3z^4 + a_6 b_3z^2 + a_6 b_3z^1 +$
$a_5 b_3z^4 + a_5 b_3z^3 + a_5 b_3z^1 + a_5 b_3z^0 + a_4 b_3z^7 + a_3 b_3z^6 + a_2 b_3z^5 + a_1 b_3z^4 +$
$a_0 b_3z^3 + a_7 b_2z^5 + a_7 b_2z^4 + a_7 b_2z^2 + a_7 b_2z^1 + a_6 b_2z^4 + a_6 b_2z^3 + a_6 b_2z^1 +$
$a_6 b_2z^0 + a_5 b_2z^7 + a_4 b_2z^6 + a_3 b_2z^5 + a_2 b_2z^4 + a_1 b_2z^3 + a_0 b_2z^2 + a_7 b_0z^7 +$

$a_6\, b_0 z^6 + a_5\, b_0 z^5 + a_4\, b_0 z^4 + a_3\, b_0 z^3 + a_2\, b_0 z^2 + a_1\, b_0 z^1 + a_0\, b_0 z^0$.

a * b = $\mathbf{z^0}\ [a_5 b_3 + a_6 b_2 + a_0 b_0] +$

$\mathbf{z^1}\ [a_6 b_3 + a_5 b_3 + a_7 b_2 + a_6 b_2 + a_1 b_0] +$

$\mathbf{z^2}\ [a_7 b_3 + a_6 b_3 + a_7 b_2 + a_0 b_2 + a_2 b_0] +$

$\mathbf{z^3}\ [a_7 b_3 + a_5 b_3 + a_0 b_3 + a_6 b_2 + a_1 b_2 + a_3 b_0] +$

$\mathbf{z^4}\ [a_6 b_3 + a_5 b_3 + a_1 b_3 + a_7 b_2 + a_6 b_2 + a_2 b_2 + a_4 b_0] +$

$\mathbf{z^5}\ [a_7 b_3 + a_6 b_3 + a_2 b_3 + a_7 b_2 + a_3 b_2 + a_5 b_0] +$

$\mathbf{z^6}\ [a_7 b_3 + a_3 b_3 + a_4 b_2 + a_6 b_0] +$

$\mathbf{z^7}\ [a_4 b_3 + a_5 b_2 + a_7 b_0]$

$$\begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \\ c_6 \\ c_7 \end{bmatrix} = \begin{bmatrix} a_0 & 0 & a_6 & a_5 & 0 & 0 & 0 & 0 \\ a_1 & 0 & a_7 + a_6 & a_6 + a_5 & 0 & 0 & 0 & 0 \\ a_2 & 0 & a_7 + a_0 & a_7 + a_6 & 0 & 0 & 0 & 0 \\ a_3 & 0 & a_1 + a_6 & a_7 + a_5 + a_0 & 0 & 0 & 0 & 0 \\ a_4 & 0 & a_7 + a_6 + a_2 & a_6 + a_5 + a_1 & 0 & 0 & 0 & 0 \\ a_5 & 0 & a_7 + a_3 & a_7 + a_6 + a_2 & 0 & 0 & 0 & 0 \\ a_6 & 0 & a_4 & a_7 + a_3 & 0 & 0 & 0 & 0 \\ a_7 & 0 & a_5 & a_4 & 0 & 0 & 0 & 0 \end{bmatrix} * \begin{bmatrix} 1 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Multiplicador 0D en forma matricial
Figura 37

- Multiplicador 09

Teniendo dos elementos ***a*** *y* ***b*** *que pertenecen al campo Finito* ***GF*** $\mathbf{(2^8)}$***,*** *siendo* ***b 09*** *(hexadecimal)* ***ó 00001001*** *(binario)****:***

a = $a_7 z^7 + a_6 z^6 + a_5 z^5 + a_4 z^4 + a_3 z^3 + a_2 z^2 + a_1 z^1 + a_0 z^0$

b = $b_3 z^3 + b_0 z^0$

a * b = $a_7\, b_3 z^{10} + a_7\, b_0 z^7 + a_6\, b_3 z^9 + a_6\, b_0 z^6 + a_5\, b_3 z^8 + a_5\, b_0 z^5 + a_4\, b_3 z^7 + a_4\, b_0 z^4 +$
$a_3\, b_3 z^6 + a_3\, b_0 z^3 + a_2\, b_3 z^5 + a_2\, b_0 z^2 + a_1\, b_3 z^4 + a_1\, b_0 z^1 + a_0\, b_3 z^3 + a_0\, b_0 z^0$.

a * b = $a_7\, b_3 z^6 + a_7\, b_3 z^5 + a_7\, b_3 z^3 + a_7\, b_3 z^2 + a_6\, b_3 z^5 + a_6\, b_3 z^4 + a_6\, b_3 z^2 + a_6\, b_3 z^1 +$
$a_5\, b_3 z^4 + a_5\, b_3 z^3 + a_5\, b_3 z^1 + a_5\, b_3 z^0 + a_4\, b_3 z^7 + a_3\, b_3 z^6 + a_2\, b_3 z^5 + a_1\, b_3 z^4 +$

$a_0\, b_3 z^3 + a_7\, b_0 z^7 + a_6\, b_0 z^6 + a_5\, b_0 z^5 + a_4\, b_0 z^4 + a_3\, b_0 z^3 + a_2\, b_0 z^2 + a_1\, b_0 z^1 + a_0\, b_0 z^0.$

$$
\begin{aligned}
\mathbf{a * b} = \mathbf{z^0}\ & [a_5 b_3 + a_0 b_0] + \\
\mathbf{z^1}\ & [a_6 b_3 + a_5 b_3 + a_1 b_0] + \\
\mathbf{z^2}\ & [a_7 b_3 + a_6 b_3 + a_2 b_0] + \\
\mathbf{z^3}\ & [a_7 b_3 + a_5 b_3 + a_0 b_3 + a_3 b_0] + \\
\mathbf{z^4}\ & [a_6 b_3 + a_5 b_3 + a_1 b_3 + a_4 b_0] + \\
\mathbf{z^5}\ & [a_7 b_3 + a_6 b_3 + a_2 b_3 + a_5 b_0] + \\
\mathbf{z^6}\ & [a_7 b_3 + a_3 b_3 + a_6 b_0] + \\
\mathbf{z^7}\ & [a_4 b_3 + a_7 b_0]
\end{aligned}
$$

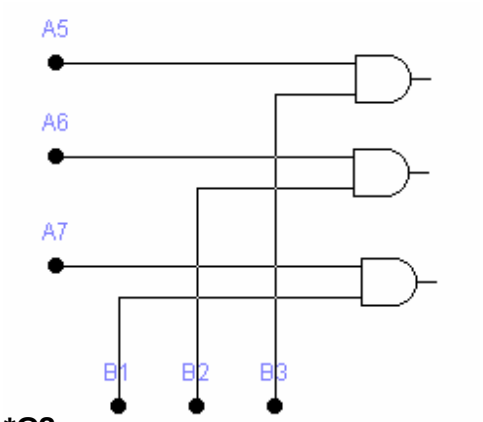
$$
\begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \\ c_6 \\ c_7 \end{bmatrix} =
\begin{bmatrix}
a_0 & 0 & 0 & a_5 & 0 & 0 & 0 \\
a_1 & 0 & 0 & a_6 + a_5 & 0 & 0 & 0 \\
a_2 & 0 & 0 & a_7 + a_6 & 0 & 0 & 0 \\
a_3 & 0 & 0 & a_7 + a_5 + a_0 & 0 & 0 & 0 \\
a_4 & 0 & 0 & a_6 + a_5 + a_1 & 0 & 0 & 0 \\
a_5 & 0 & 0 & a_7 + a_6 + a_2 & 0 & 0 & 0 \\
a_6 & 0 & 0 & a_7 + a_3 & 0 & 0 & 0 \\
a_7 & 0 & 0 & a_4 & 0 & 0 & 0
\end{bmatrix} *
\begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}
$$

Multiplicador 09 en forma matricial
Figura 38

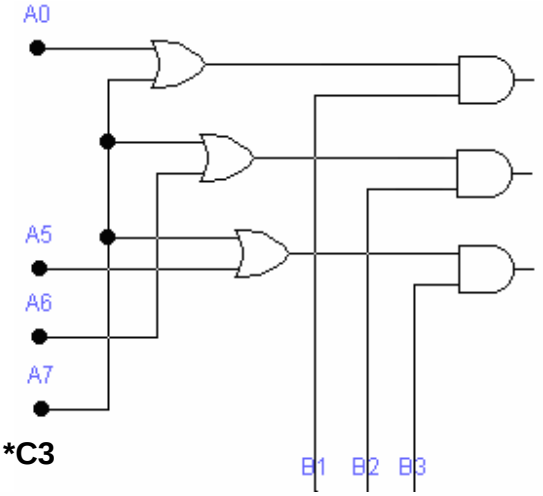
A continuación se presenta el modelo circuital equivalente de las matrices de multiplicación, en este caso para el multiplicador por OEh. En todos los casos los bit del operando b usados son iguales a 1, por lo cual se reduce la complejidad y uso de puertas lógicas solamente a las operaciones XOR. También se usa un bloque reductor a un bit, en este caso con puertas XOR en cascada.

MULTIPLICADOR 0E

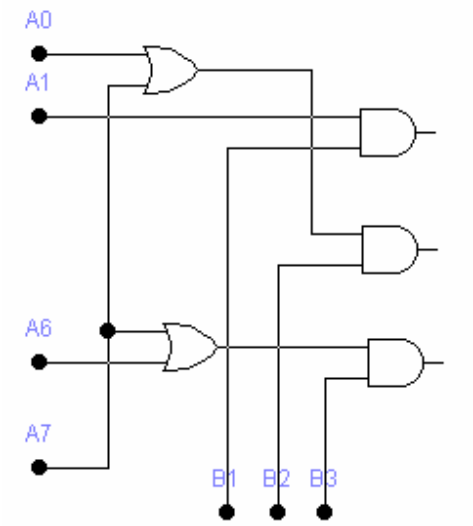
*** C0**



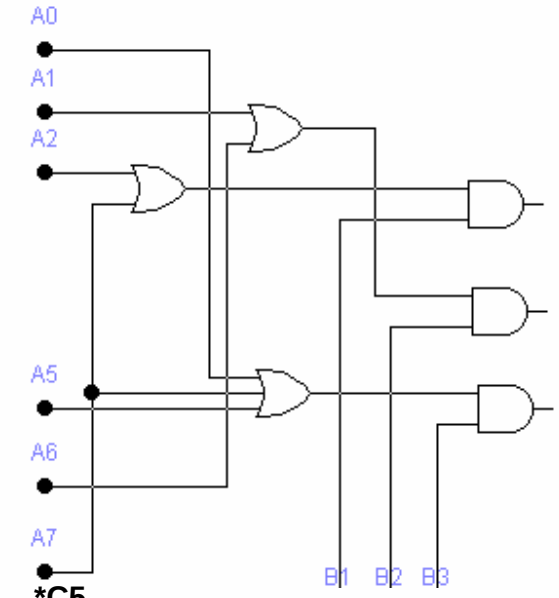
***C1**



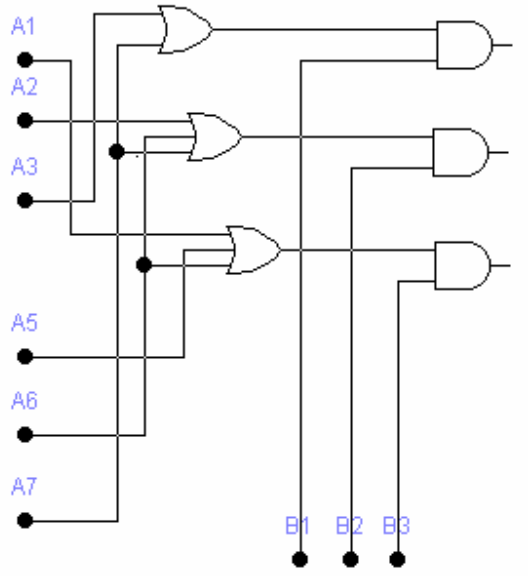
***C2**



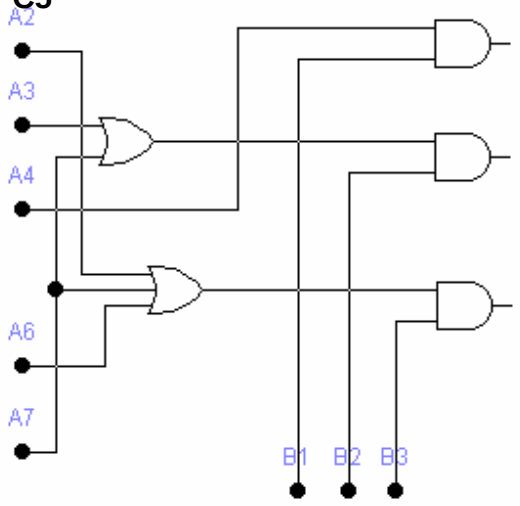
***C3**

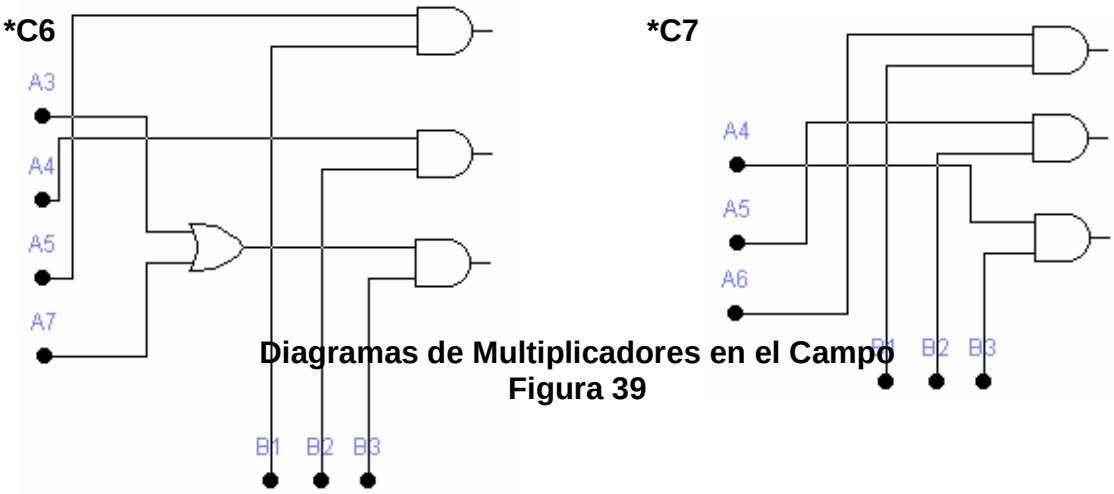


***C4**



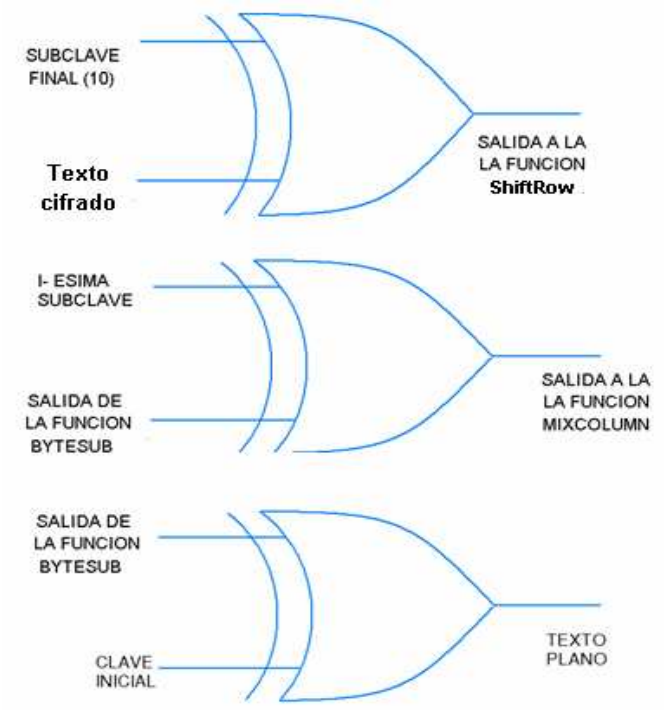
***C5**





Diagramas de Multiplicadores en el Campo
Figura 39

2.1.2.1.2.3 BLOQUE ADDROUNDKEY



Bloque Addroundkey Inverso
Figura 40

Esta función consiste en realizar una operación OR-Exclusiva entre el texto cifrado y la clave final (10); durante el proceso de decifrado también se realiza una operación OR-Exclusiva entre la matriz de Estado que proviene de la

transformación anterior (Función ByteSub) y la subclave generada a partir de la clave del sistema para esa ronda.

También efectúa una operación OR-Exclusiva entre la matriz de Estado que proviene de la transformación anterior (Función ByteSub) y la clave inicial, hallando dato de salida del algoritmo (texto plano).

2.1.3 Fase de implementación

2.1.3.1 Implementación del criptoprocesador

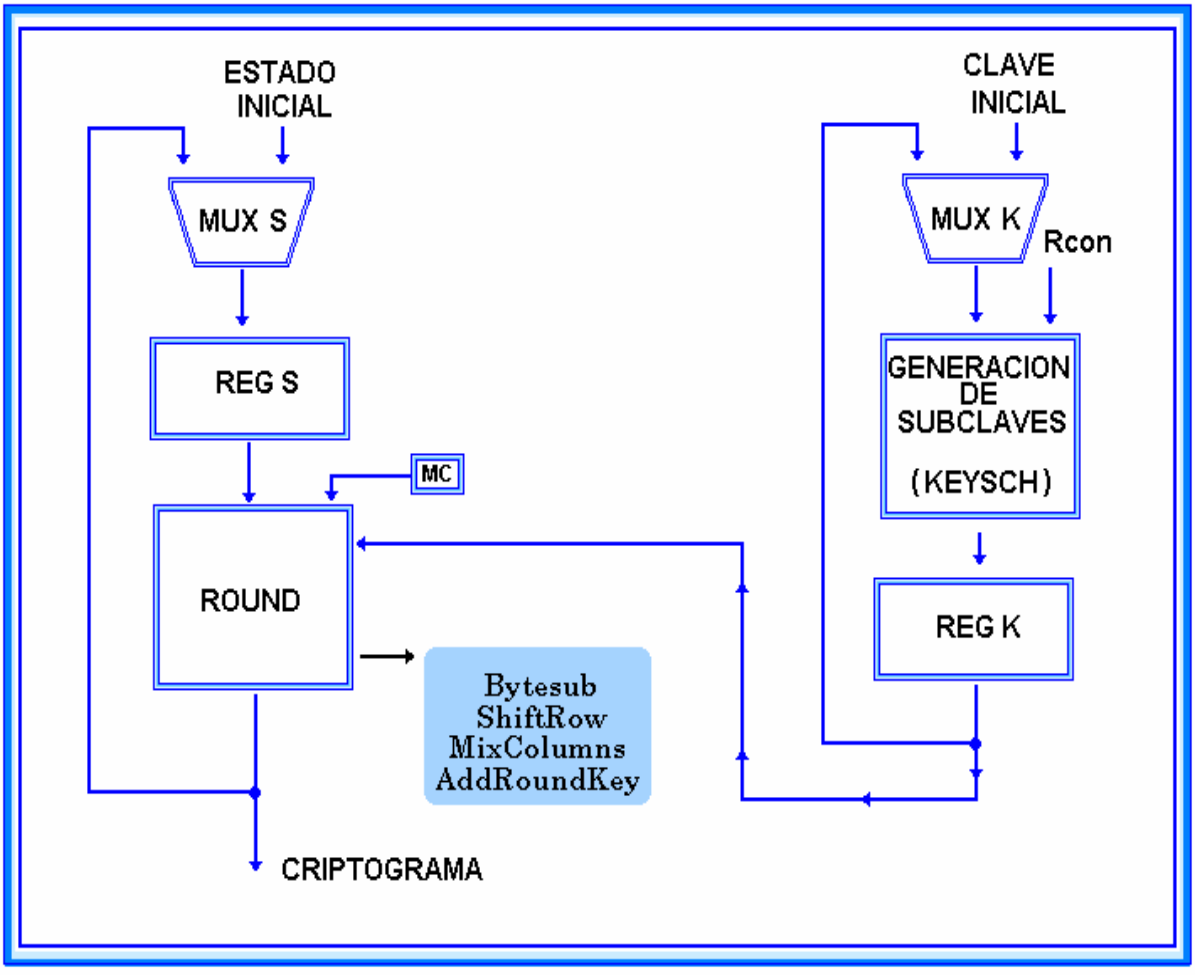


Diagrama en Bloques del proceso de cifrado
Figura 41

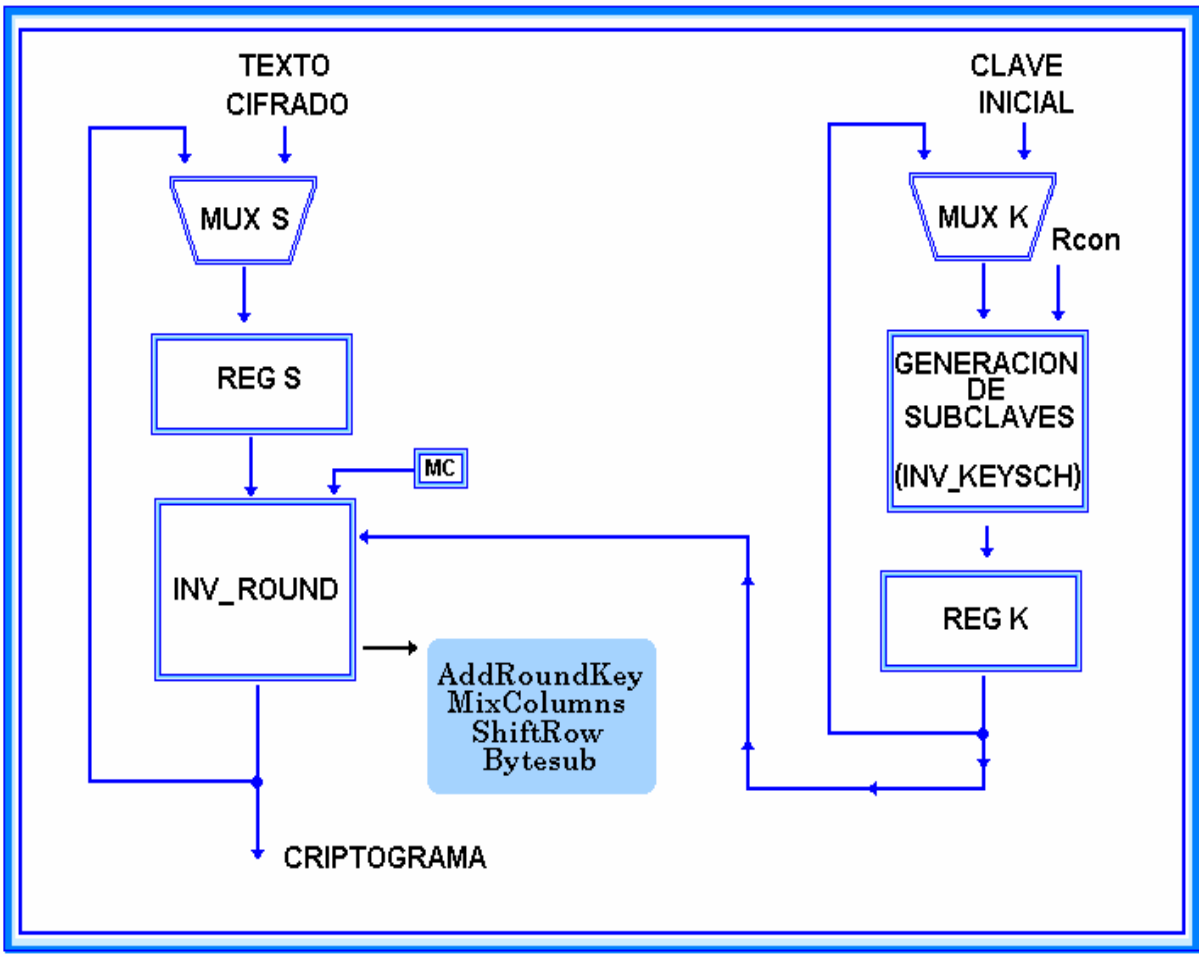


Diagrama en Bloques del proceso de Descifrado
Figura 42

2.1.3.1.2 CIFRADO

La implementación hardware del algoritmo AES-Rijndael corresponde al tipo *iterativo,* pues como puede observarse en la figura 41 se tiene un bloque Round único, el cual es usado para procesar iterativamente los datos obtenidos en cada una de las rondas. El funcionamiento comienza con el texto y la clave originales, luego un registro soporta el almacenamiento parcial, hasta llegar a completar las 10 iteraciones o rondas, contempladas para AES.

El boque Round consiste en la interconexión en cascada de los bloques q implementan cada una de las operaciones internas, de esta manera no se tiene ningún tipo de retardo para obtener la salida de la ronda.
También es necesario contar con la clave de ronda correspondiente, esto se logra con el bloque KeySch (Key Schedule).
El boque Round es modular, totalmente desarrollado en VHDL y es el objeto principal del trabajo, junto con el módulo InvRound, que implementa las rondas en el proceso de descifrado.
El boque Round recibe un bloque de 128 bits correspondiente al estado del proceso, junto con la clave de ronda correspondiente y un bit (mc) que indica la realización de la transformación MixColumns si está en un estado lógico 1 ó la ausencia de esta transformación en caso contrario.

El bloque KeySch se implementa completamente en paralelo, logrando una optimización significativa al no presentar retardos en el cálculo de la subclave correspondiente.
A este bloque ingresan en un principio la clave inicial
A este bloque ingresan la clave inicial ó la subclave anterior y el valor de Rcon correspondiente a la ronda, teniendo como salida la subclave que se lleva al bloque Round para su operación interna.

Los registros de almacenamiento de matriz estado y subclave se habilitan mediante una señal activa alta por flanco, aprovechando la capacidad de describir este tipo de componentes en VHDL. El proceso completo de operación de las 10 rondas se controla mediante una FSM, que se encarga de las siguientes operaciones:

- En el estado inicial coloca la entrada s de los multiplexores de estado y de subclave en 0, para habilitar las entrada de texto inicial y clave inicial, pone el enable del registro de estado en 0 y el de subclave en 1. Igualmente genera el valor de Rcon para el bloque KeySch y coloca el bit de

habilitación de la transformación MixColumns en 1, con la cual se efectúa la misma.

- En el estado siguiente coloca el enable de estado en 1 y el de subclave en 0. En este punto la subclave correspondiente queda almacenada en el registro.
- En el siguiente estado coloca ambos bit de enable en 0, terminando el ciclo correspondiente a una ronda.

Este proceso se repite para las 10 rondas, con una diferencia fundamental en la última ronda, en la cual coloca un 0 en el habilitador de MixColumns, con lo cual no se realiza esta transformación y activa la salida flag_crypt que indica la finalización del proceso de cifrado.

2.1.3.1.3 DESCIFRADO

Para el proceso de descifrado se diseñó una arquitectura semejante, con las siguientes consideraciones:

- La entrada del bloque es el texto cifrado y la clave de cifrado original. A partir de esta se calcula la 10° subclave, con la cual se inicia un proceso similar al del cifrado, hasta llegar a la clave inicial. Este proceso se implementa conectando en cascada 10 bloques KeySch y almacenando la salida en el registro correspondiente.
- El proceso interno del bloque Ronda en el caso del descifrado en inverso al de cifrado.
- La FSM de control es similar a la de cifrado, con excepción de la generación de la señal Rcon que va en orden inverso y el uso del bit de habilitación de MixColumns el cual también se maneja en orden inverso, ya que en la primera ronda no se realiza dicha operación.

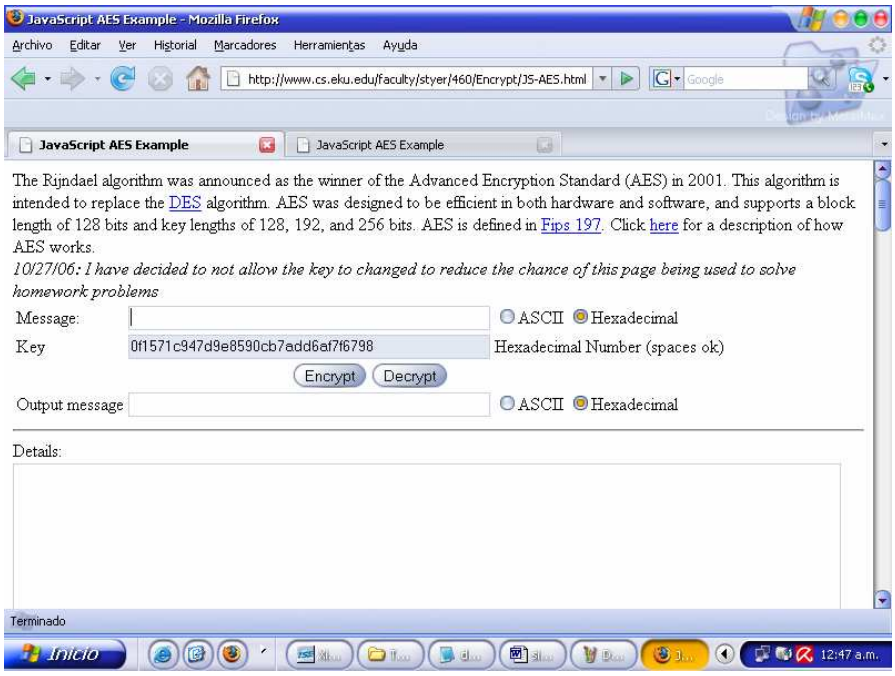
2.1.4 Fase de prueba y evaluación

Se implementó en el dispositivo FPGA un módulo UART usando el open core en VHDL desarrollado en (11). Este módulo se configuró para realizar comunicación serial con una tasa de transferencia de 115200 b/s, usando el conector DB9 incluido en la tarjeta de desarrollo Spartan 3E.

Para el control interno de comunicación y operación se implementó una FSM (Finite State Machine) que habilita la recepción de datos y su traspaso al bloque Cifrador, así mismo el traspaso de criptogramas al computador.

Como interfaz de prueba se usó el software HexTerminal desarrollado en la Universidad de los Andes (Colombia), para efectuar la comunicación entre el PC y la tarjeta de desarrollo.

Igualmente se usó un script desarrollado en JavaScript (Ver anexos), disponible y avalado por la comunidad científica a nivel mundial, para contrastar los resultados de cifrado y descifrado del prototipo, con respecto a valores de prueba estandarizados.



Script desarrollado en JavaScript
Figura 42

3. RESULTADOS

3.1 IMPLEMENTACIÓN DEL CIFRADOR

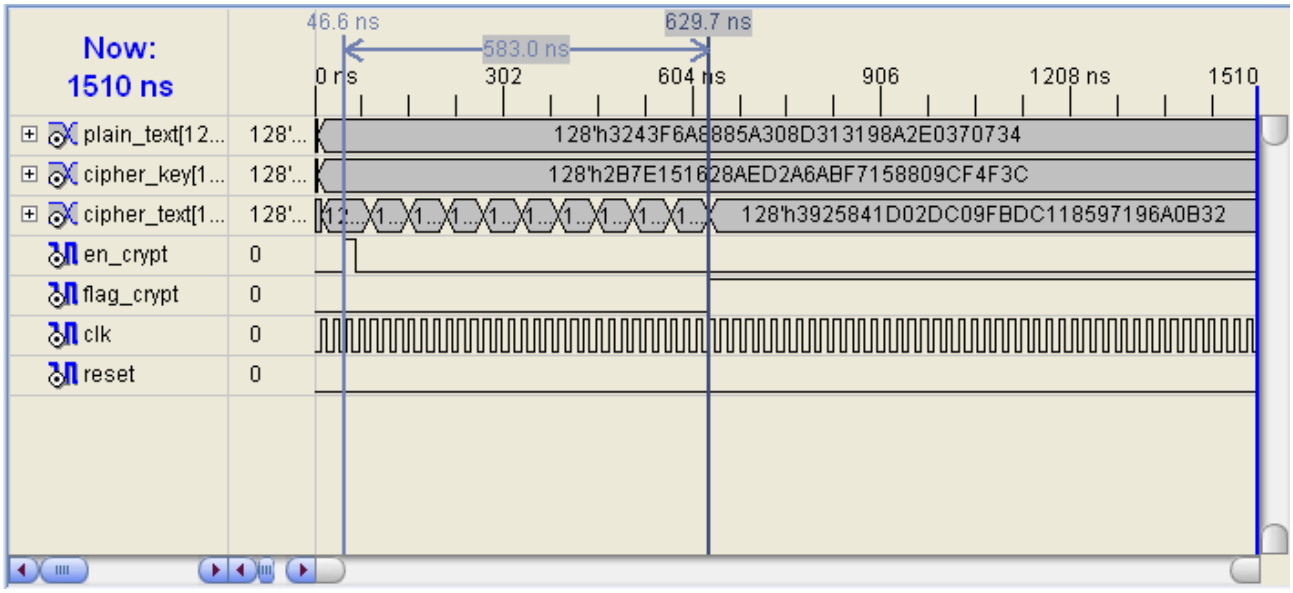
Se realizaron simulaciones funcionales usando la herramieta ISE Simulator integrada en el Foundation 8.2i.

- En esta caso se usó una frecuencia de reloj de 50 MHz, con un periodo T = 20 ns.

Para contrastar los resultados de operación del cifrador desarrollado, se usaron vectores de test reconocidos internacionalmente.

En este caso se usaron:

- Texto pleno(plain-text) **: 3243f6a8885a308d313198a2e0370734h**
- Clave : **2b7e151628aed2a6abf7158809cf4f3C h**



Simulación del Cifrador
Figura 43

En la simulación puede observarse el bit de inicio (en_crypt) y la bandera de fin (flan_crypt). El reporte de simulación indica un tiempo de cifrado de 583 ns. Puede obervarse también el flujo de datos de las 10 rondas, hasta llegar al valor deseado.

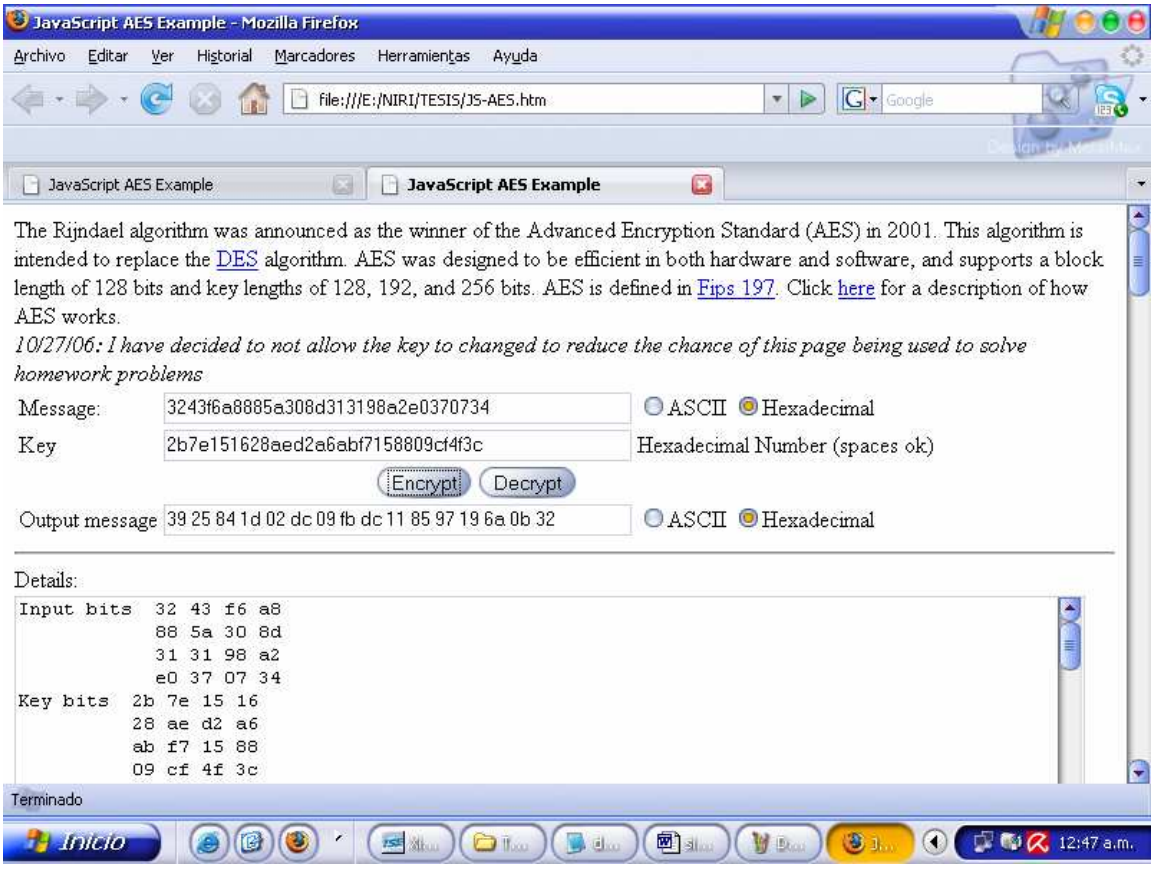
La medida del Throughput del cifrador se calcula:

Trhoughput = número de bits * frecuencia

- Para este caso la medida se da con respecto a 128 bits.
- En este caso la frecuencia es 1/T, por lo cual

F = 1/T = 1/583 ns = 1715265,866209262435677530017152 7 Hz

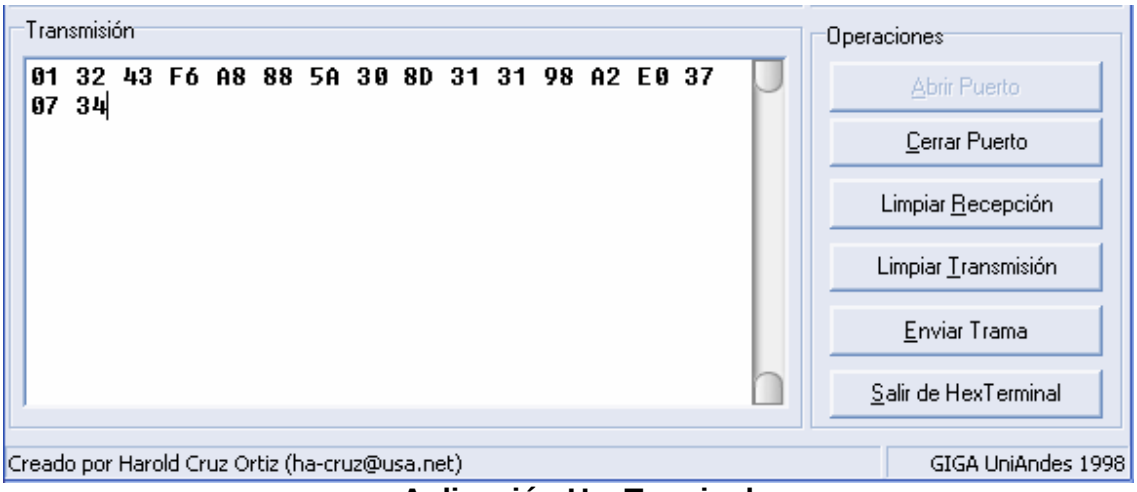
De esta manera se calcula el throughput, obteniendo el valor de 220 Mbps, que corresponde a la cantidad de bits que el cifrador puede procesar. Para validar se tuvo en cuenta la aplicación web mencionada.



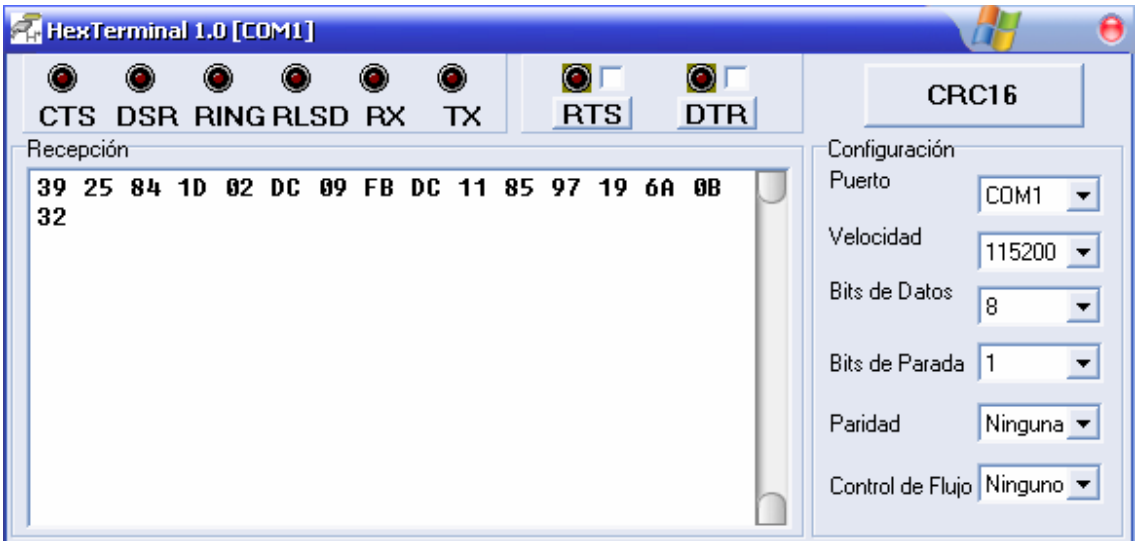
Aplicación Web

Figura 44

Iguales resultados válidos se obtuvieron al comunicar el PC con la tarjeta, usando la aplicación HexTerminal. En este caso el protocolo de comunicación incluye un 02h que corresponde a la orden de cifrado. Previamente se ha cargado la clave de cifrado y solamente se muestra la operación de cifrado.



Aplicación HexTerminal
Figura 45



Aplicación HexTerminal
Figura 46

La optimización de la síntesis se hizo con respecto al área, obteniéndose los siguientes resultados:

```
Device utilization summary:
---------------------------

Selected Device : 3s500eft256-4

 Number of Slices:                     1407  out of   4656    30%
 Number of Slice Flip Flops:            859  out of   9312     9%
 Number of 4 input LUTs:               2491  out of   9312    26%
 Number of IOs:                         388
 Number of bonded IOBs:                 388  out of    190   204%  (*)
 Number of BRAMs:                        16  out of     20    80%
 Number of GCLKs:                         1  out of     24     4%
```

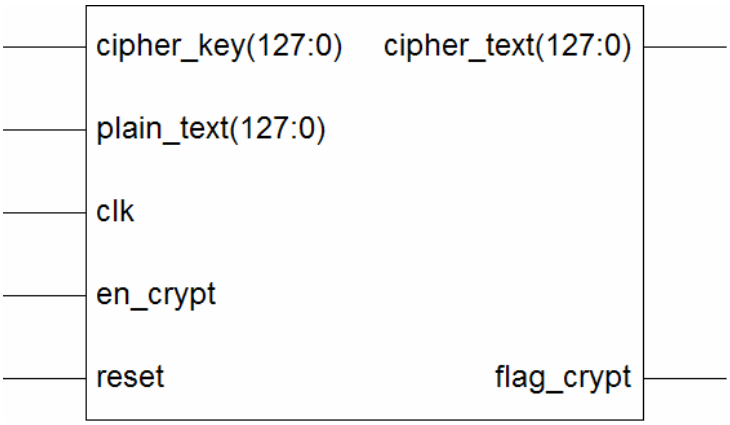
Resultados de la optimización de la síntesis
Tabla 10

Los resultados en cuanto al tiempo de ejecución son los siguientes:

```
Timing Summary:
---------------
Speed Grade: -4

   Minimum period: 7.927ns (Maximum Frequency: 126.147MHz)
   Minimum input arrival time before clock: 8.383ns
   Maximum output required time after clock: 5.033ns
   Maximum combinational path delay: No path found

Timing Detail:
--------------
All values displayed in nanoseconds (ns)

=========================================================================
Timing constraint: Default period analysis for Clock 'clk'
  Clock period: 7.927ns (frequency: 126.147MHz)
  Total number of paths / destination ports: 13527 / 1103
```

Los resultados en cuanto al tiempo
Tabla 11

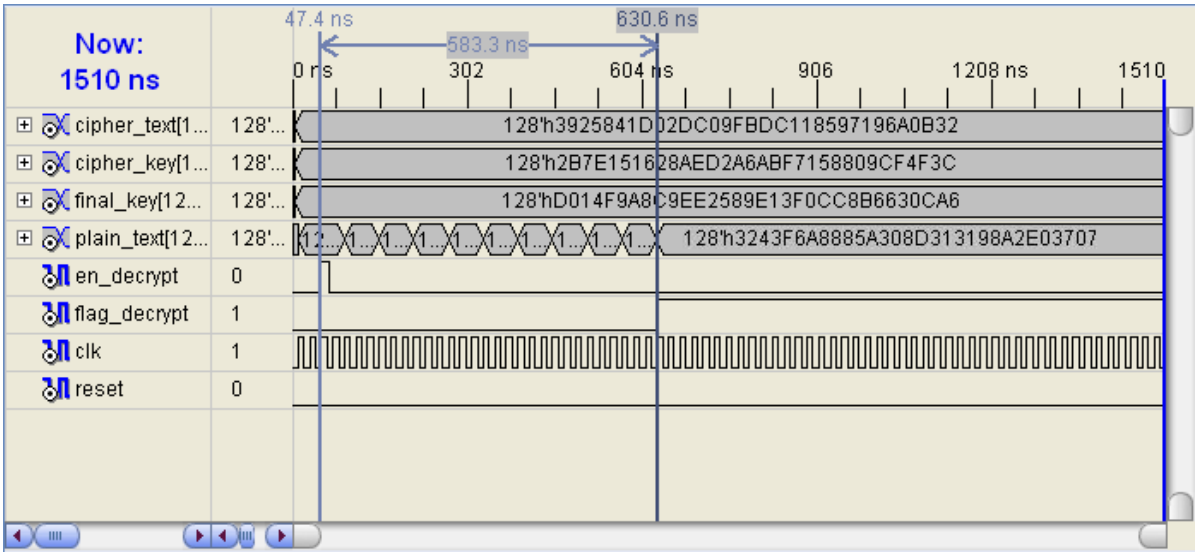
Finalmente se presenta el esquema RTL generado por la herramienta de desarrollo, correspondiente al bloque cifrador:



Esquemático RTL para el Bloque cifrador
Figura

3.2 IMPLEMENTACIÓN DEL DESCIFRADOR

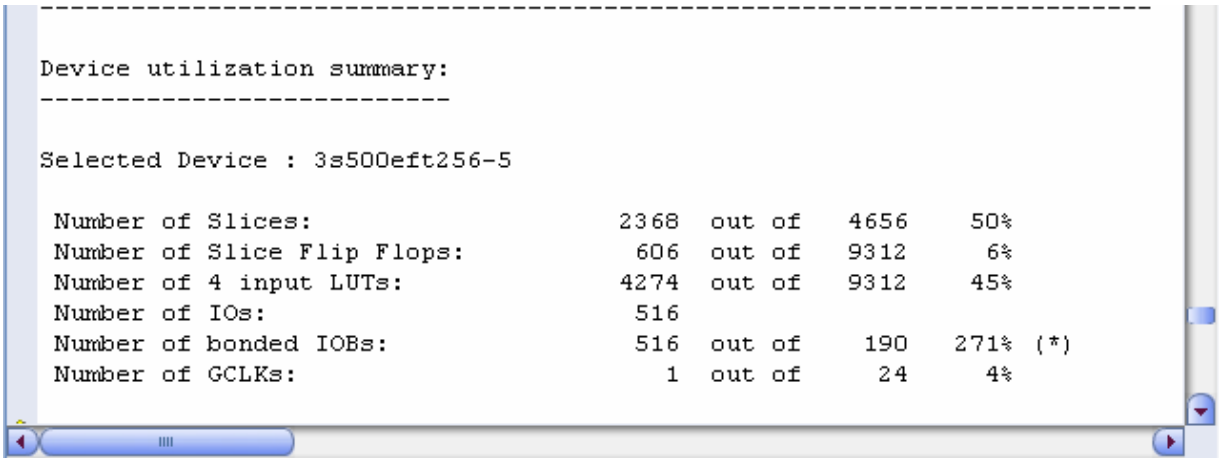
Se obtuvieron los siguientes resultados de simulación:



Resultados de la Implementación
Figura 47

Para el caso de bloque descifrador se tiene el mismo tiempo de operación, de 583.3 ns. Para el descifrador se obtienen los mismos resultados en cuanto al throughput.

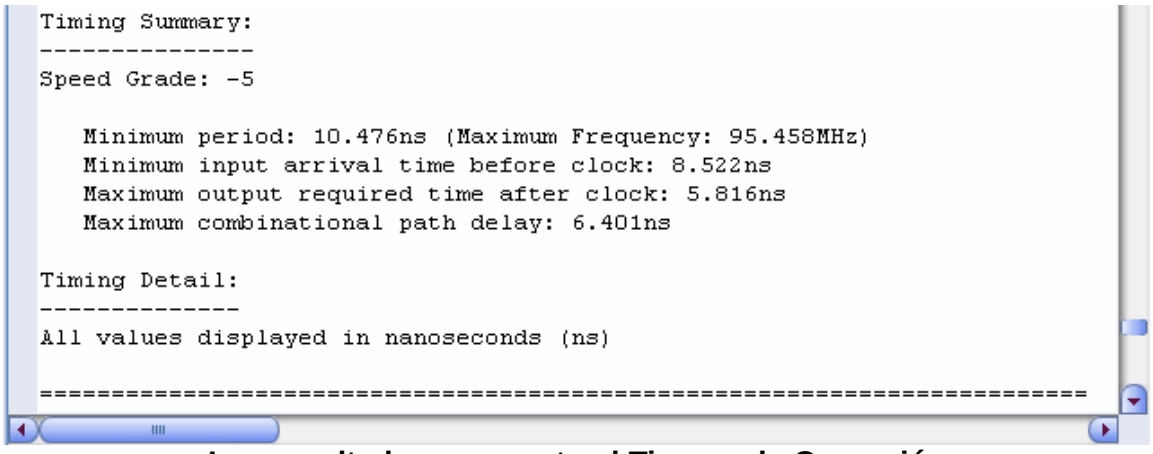
Se obtuvieron los siguientes resultados en ocupación de recursos de la FPGA:



```
--------------------------------------------------------------------------

Device utilization summary:
---------------------------

Selected Device : 3s500eft256-5

 Number of Slices:                     2368  out of   4656    50%
 Number of Slice Flip Flops:            606  out of   9312     6%
 Number of 4 input LUTs:               4274  out of   9312    45%
 Number of IOs:                         516
 Number of bonded IOBs:                 516  out of    190   271% (*)
 Number of GCLKs:                         1  out of     24     4%
```

Los resultados en cuanto al Recursos de la FPGA
Tabla 12

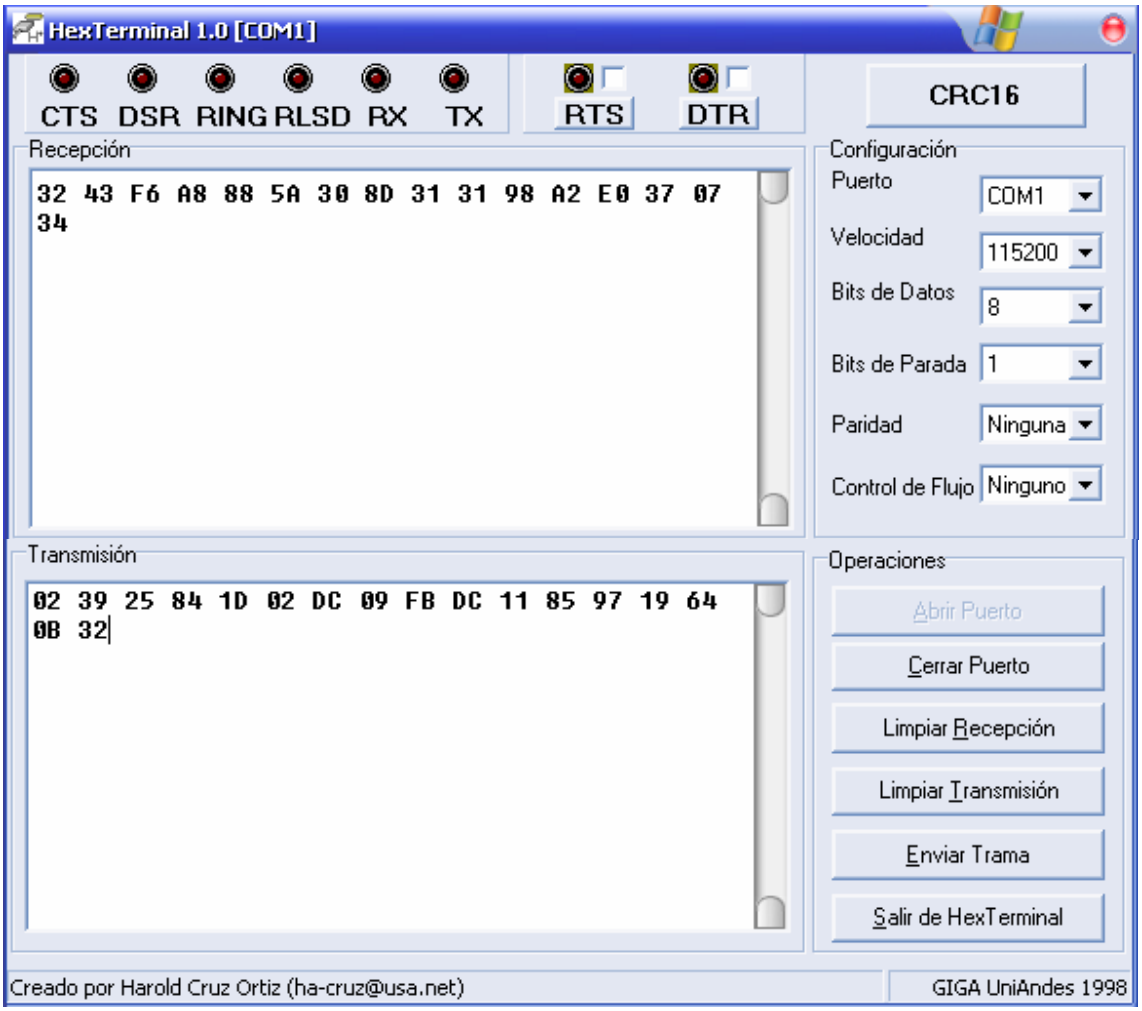
- La ocupación de área es superior para el proceso de descifrado.
- Los resultados en cuanto al tiempo de operación son:



```
Timing Summary:
---------------
Speed Grade: -5

   Minimum period: 10.476ns (Maximum Frequency: 95.458MHz)
   Minimum input arrival time before clock: 8.522ns
   Maximum output required time after clock: 5.816ns
   Maximum combinational path delay: 6.401ns

Timing Detail:
--------------
All values displayed in nanoseconds (ns)

=========================================================================
```

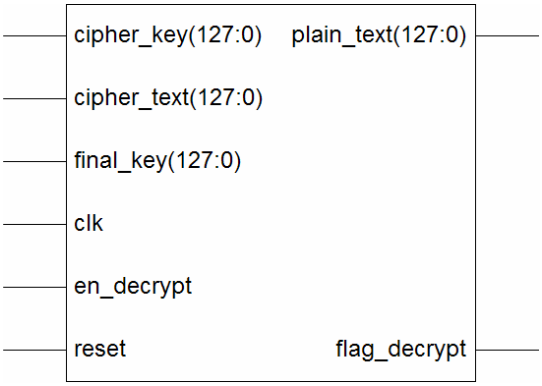
Los resultados en cuanto al Tiempo de Operación
Tabla 13

El proceso de descifrado de datos entre el PC y la FPGA se muestra a continuación:



Proceso de Descifrado
Figura 48

Se obtuvo el siguiente esquemático RTL para el bloque Descifrador:



Esquemático RTL para el Bloque Descifrador
Figura 49

En este caso se tiene en cuenta la clave de cifrado original y la última subclave (final_key), generada en el bloque KeySch en cascada.

En las siguientes tablas se presentan los resultados en forma condensada.

PARAMETRO	CIFRADO	DESCIFRADO
Celdas Logicas	1407	2368
I/O Pines	388	516
BRam	16	
Fmax	126.147 MHz	95.458 MHz
Clk	20ns	20ns
End Time	630 ns	630 ns
Throughput	220 Mbps	220 Mbps

Resultados de la Implementación del Cifrador
Tabla 14

4. ANÁLISIS DE RESULTADOS

Es posible considerar algunos factores que influyen en los resultados obetenidos.

Existen ventajas inherentes a los dipositivos FPGA, dada su arquitectura paralela y la posibilidad de procesamiento concurrente. Este hecho se evidenció en la implementación del prototipo, en el cual predominan los bloques paralelos y las conexiones masivas de elementos en cascada, logrando flujo de datos sin retardos significativos.

De igual manera, se debe tener en cuenta la optimización lograda en el bloque KeySch, el cual se implementó completamente en cascada (ver anexos), logrando la generación de la subclave directamente y sin retardos.

Otro punto importante de optimización, el cual es un aporte del presente trabajo, es el uso de multiplicadores paralelos en la transformación mixcolumns del descifrado, logrando también reducir el área ocupada por dicho bloque.

Si se analizan las implementaciones reportadas en la literatura del área, en otros trabajos, se evidencia la superioridad de las implementaciones en FPGA, sobre todo en cuanto al throughput, siendo este mas alto cuando la implementación hardware se hace usando arquitectura pipeline.

A continuación se presentan los principales trabajos reportados y sus parámetros de desempeño, contrastados con los logrados en el presente trabajo.

Autor	Encriptacion		Desencriptacion		Plataforma
	Throughput	CLs	Throughput	CLs	
48	637.24	1584	500.28	2506	Codigo Vhdl sobre una FPGA Xilinx Virtex XCV1000BG560-4 Iterativa.
48	9269.27	4205	5565.12	8243	Codigo Vhdl sobre una FPGAXilinx Altera EP2A70B724C7 pipeline
46	6.8Mbps				*II Version 7.21 Student Edition*, Altera Flex 10K.
49	294.2	3528			Codigo Vhdl sobre una FPGA Altera XCV1000BG560-4
50	268		248		Codigo Vhdl sobre una FPGA Altera Virtex XCV1000BG560-4
51	331.5	2902			Codigo Vhdl sobre un asic 0.5um
52	5745.06				Codigo Vhdl sobre un asic 0.5um
53	320	4805	256	5016	Codigo Vhdl sobre una FPGA Xilinx Altera EPF10K250AGC599-1
54	232.7	1585	211.5	2145	Codigo Vhdl sobre una FPGA Xilinx Altera EPF10K130EQC240-1
55	68.4		72.7		Codigo C. Celaron 200MHz
56	254		254		Codigo C. Celaron 850MHz
***	220Mbps	1407	220Mbps	2368	Codigo Vhdl sobre una FPGA Xilinx Spartan 3s500eft256-4

*** Resultados obtenidos en este trabajo.

Comparación con otras Implementaciones
Tabla 15

En Colombia se ha reportado solamente un trabajo en el área, desarrollado en la Universidad del Valle (48). En esta caso se trabajó con otra familia de dispositivos FPGA (Altera), por lo cual no son exactamente iguales las unidades funcionales de ocupación lógica, sin embargo la optimización es significativa.

Debido al tipo de arquitectura desarrollada se usan mas ciclos de reloj, por lo cual es tiempo de cifrado y descifrado es mayor que en algunas implementaciones, sin embargo el throughput obtenido en el presente trabajo es competitivo y cercano al obtenido por en otros trabajos. Además, es importante mencionar que la mayor parte de estos resultados corresponden a proyectos de Maestría.

5. BIBLIOGRAFIA

[1] NIST. Announcing the ADVANCED ENCRYPTION STANDARD (AES). Federal Information Standards Publication, Nov. 2001.

[2] Angel Angel Jose de Jesus, "AES -Advanced Encryption Standard", version 2005, dummies.

[3] Herstein I. N. "Álgebra Moderna", Biblioteca de Matemática Superior, Editorial Trillas.

[4] Nazar A. Saqib, "AES Algorithm Implementation-An devised for efficient implementations on reconfigurable devices", September 2003. IEEE Computer Society Press.

[5] Pons Matorell Manuel,"Criptología", Escuela universitaria politécnica de Mataró.

[6] Aguirre Jorge, "Libro Electrónico de Seguridad Informática y Criptografía" Versión 4.1Sexta edición de 1 de Marzo de 2006

[7] Lucena López Manuel José "Criptografía y Seguridad en Computadores" Tercera Edición (Versión 2.15). Marzo de 2004

[8] Carracedo Gallardo Justo. "Seguridad en redes telemáticas" 2004, Mc Graw Hill, Madrid España.

[9] Muñoz Muñoz Alfonso, "Criptosistema rijndael a fondo" Septiembre-2004. Madrid.

[10] Caballero Gil Pino. "Introducción a la criptografía", 2003, Alfaomega / Ra-Ma Madrid España.

[11] Castaño Javier, Velásquez Fabián “Diseño e implementación de un prototipo de transmisión de datos punto a punto con encriptación basada en curvas elípticas”, Universidad de los Llanos 2005.

[12] Daemen, Joan e Rijmen, Pawel. “AES Proposal: Rijndael“. (30 Jul. 2001)

[13]Guido Bertoni Marco Racchetti “Hardware implementation of the rijndael s-box: a case study”, Alari, Università della Svizzera Italiana Politecnico di Milano.

[14] Kai Schramm, Gregor Leander, Patrick Felke, Christof Para, “A Collision-Attack on AES Combining Side Channel- and Differential attack”, AES4 Horst Görtz Institute

[15] Ilia Toli, Alberto Zanoni, “An algebraic interpretation of AES-128”, AES04, Horst Görtz Institute.

[16] Yvo Desmedt, “Cyclic Properties of AES Round Functions”, AES4 Horst Görtz Institute.

[17] Galaviz Casas José, Magidin Arturo, “Introducción a la criptografía”, Universidad Nacional de México.

[18] A. Fuster, D. de la Guia, L. Hernández, F. Montoya, J. Muñoz, “Técnicas criptográficas de Protección de Datos”, Alfaomega 2001

[19] Lewand Robert Eduard, “Cryptological Mathemetics”, The Mathematical Asociation of America, 2000.

[20] Articulo 22 Resolución 016 de 2006 Facultad de Ciencias Básicas e ingeniería

[21] Rafael Villarroel Flores**,** “Introducción a la teoría de representaciones de grupos finitos“, 26 de marzo de 2004.

[22] GTA, SSITAD, “Seguridad en redes telemáticas Correo Electrónico y Servicios Internet”, Madrid, 8 de julio de 1996.

[23] Gustavus J. Simmons "Contemporary Cryptology. The Science of Information Integrity", IEEE

[24] Hwang O. Enoch, “Microprocessor Design, principles and practices with VHDL“, 2004, Brooks.

[25] Brown Stephen, Zvonko Vranesic, “Fundamentals of Digital Logic with VHDL” (McGraw Hill).

[26] Mark Balch, “Complete Digital Design A Comprehensive Guide to Digital Electronics and Computer System” McGRAW-HILL

[27] Perry Douglas, “VHDL Programming by Example”, McGraw Hill

[28] Volnei A. Pedroni, “Circuit Desing with VHDL”, MIT Press Cambridge Massachussets, London England

[29] Henk C.A Van Tilborg, “Fundamentals of Cryptology“, Kluver academia Publishers.

[30] Algredo Badillo Ignacio, Cumplido Parra René Armando,”Arquitectura Reconfigurable Para La Implementación De Algoritmos Estándares De Criptografía Aplicados A Comunicaciones” Coordinación de Ciencias Computacionales

[31] http://csrc.nist.gov/encryption/aes/index

[32] http://www.esat.kuleuven.ac.be/~rijmen/rijndael/

[33] http://www.actel.com

[34] http://www.altera.com

[35] http://www.atmel.com

[36] http://www.lattice.com

[37] http://www.quicklogic.com

[38] http://www.xilinx.com

[39]Stallings, W., "Cryptography and Network Security. Principles and Practices", Prentice Hall, EUA, 2003.

[40]Joseph Fernando, Dennis Dalessandro "Accelerated FPGA Based Encryption" Ohio Center – Springfield

[41] M. L. López Vallejo y J. L. Ayala Rodrigo, "FPGA: Nociones básicas e implementación"

[42] I. Bravo, A. Hernández, J.L. Lázaro, J. Ureña, M.Pérez "Estudio de FPGA's de Xilinx para reconfiguración total" Departamento de Electrónica. Universidad de Alcalá.

[43]Stephen D. Brown, Robert J. Francis, Jonathan Rose, and Zvonko G. Vranesic. "Field-Programmable Gate Arrays". Kluwer Academic Publishers, Norwell, Massachussets.

[44] NIST. "Overview of the AES Development Effort".

[45] Daemen, Joan e Rijmen, Pawel. "The block cipher Rijndael".

[46] Liberatori Mónica**, "**Desarrollo de Encriptado AES en FPGA**",** Maestria en Redes de Datos**,** Universidad de la Plata.

[47] López Trejo Emmanuel, “Implementación Eficiente en FPGA del Modo CCM usando AES”, Maestría en Ciencias del Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional.

[48] López – Hernandez, Julio, et. al “Implementación en Hardware del Algoritmo Rijndael” Grupo de Bioelectrónica y Nanoelectronica, Escuela IEEE, Universidad del Valle

[49]A. Elbirt, W. Yip, B. Chetwynd, C. Paar, "An FPGA-Based Performance Evaluation of the AES Block Cipher Candidate Algorithm Finalists", www.ece.wpi.edu/research/crypt/publications/do

[50] P. Mroczkowski, “Implementation of the block cipher Rijndael using Altera FPGA”,

[51] K. Gaj y P. Chodowiec, “Comparison of the hardware performance of the AES candidates using reconfigurable hardware”,

[52] NSA, “Hardware Performance Simulations of Round 2 Advanced Encryption Standard Algorithms”,

[53] M. Barcelos, A. Panato y R. Reis , “Um IP Criptografia padrao Rijndael para projetos em FPGA”.

[54] V. Fisher, “Realization of the Round 2 Candidates using Altera FPGA”,

[55] B. Gladman,, “AES Algorithm Efficiency”.

[56] w. Dai, “Speed Comparation of Popular CryptoAlgorithms”.

ANEXO 1
SCRIPT DE AES USADO PARA VERIFICACIÓN

AUTOR: Eugene Styer

http://www.cs.eku.edu/faculty/styer/index.html

```
<SCRIPT language="JavaScript">
// sample key to expand:
//      2b 7e 15 16 28 ae d2 a6 ab f7 15 88 09 cf 4f 3c
// sample data:
//      32 43 f6 a8 88 5a 30 8d 31 31 98 a2 e0 37 07 34
// output:
//      39 25 84 1d 02 dc 09 fb dc 11 85 97 19 6a 0b 32

// sample key/data:
// PLAINTEXT: 00112233445566778899aabbccddeeff
// KEY:       000102030405060708090a0b0c0d0e0f
// OUTPUT:    69c4e0d86a7b0430d8cdb78070b4c55a

// accumulate values to put into text area
var accumulated_output_info;

// add a labeled value to the text area
function accumulate_output( str )
{
   accumulated_output_info = accumulated_output_info + str + "\n";
}

// convert a 8-bit value to a string
function cvt_hex8( val )
{
   var vh = (val>>>4)&0x0f;
   return vh.toString(16) + (val&0x0f).toString(16);
}

// convert a 32-bit value to a 8-char hex string
function cvt_hex32( val )
{
   var str="";
   var i;
   var v;

   for( i=7; i>=0; i-- )
   {
      v = (val>>>(i*4))&0x0f;
      str += v.toString(16);
   }
   return str;
}

// convert a two-digit hex value to a number
function cvt_byte( str )
{
```

```
  // get the first hex digit
  var val1 = str.charCodeAt(0);

  // do some error checking
  if ( val1 >= 48 && val1 <= 57 )
      // have a valid digit 0-9
      val1 -= 48;
   else if ( val1 >= 65 && val1 <= 70 )
      // have a valid digit A-F
      val1 -= 55;
   else if ( val1 >= 97 && val1 <= 102 )
      // have a valid digit A-F
      val1 -= 87;
   else
   {
      // not 0-9 or A-F, complain
      window.alert( str.charAt(1)+" is not a valid hex digit" );
      return -1;
   }

  // get the second hex digit
  var val2 = str.charCodeAt(1);

  // do some error checking
  if ( val2 >= 48 && val2 <= 57 )
      // have a valid digit 0-9
      val2 -= 48;
   else if ( val2 >= 65 && val2 <= 70 )
      // have a valid digit A-F
      val2 -= 55;
   else if ( val2 >= 97 && val2 <= 102 )
      // have a valid digit A-F
      val2 -= 87;
   else
   {
      // not 0-9 or A-F, complain
      window.alert( str.charAt(2)+" is not a valid hex digit" );
      return -1;
   }

   // all is ok, return the value
   return val1*16 + val2;
}

// add a byte to the output
function accumulate_byte( label, val )
{
   accumulated_output_info += label + cvt_hex8(val) + "\n";
}

// add a word to the output
function accumulate_wordarray( label, ary )
{
   var i, j;
   accumulated_output_info += label + " ";

   // process the four elements in this word
```

```
   for( j=0; j<4; j++ )
      accumulated_output_info += " " + cvt_hex8( ary[j] );

   // mark the end of the word
   accumulated_output_info += "\n";
}

// add an array to the output
function accumulate_array( label, ary )
{
   var i, j;
   var spacer="";

   // build a set of spaces of equal length to the label
   while( spacer.length < label.length )
      spacer += " ";

   // build the table
   for( i=0; i<16; i+= 4 )
   {
      // add label/spaces
      if ( i== 0 )
         accumulated_output_info += label + " ";
      else
         accumulated_output_info += spacer + " ";

      // process the four elements in this "row"
      for( j=0; j<4; j++ )
         accumulated_output_info += " " + cvt_hex8( ary[i+j] );

      // mark the end of this row
      accumulated_output_info += "\n";
   }
}

// S-Box substitution table
var S_enc = new Array(
 0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5,
 0x30, 0x01, 0x67, 0x2b, 0xfe, 0xd7, 0xab, 0x76,
 0xca, 0x82, 0xc9, 0x7d, 0xfa, 0x59, 0x47, 0xf0,
 0xad, 0xd4, 0xa2, 0xaf, 0x9c, 0xa4, 0x72, 0xc0,
 0xb7, 0xfd, 0x93, 0x26, 0x36, 0x3f, 0xf7, 0xcc,
 0x34, 0xa5, 0xe5, 0xf1, 0x71, 0xd8, 0x31, 0x15,
 0x04, 0xc7, 0x23, 0xc3, 0x18, 0x96, 0x05, 0x9a,
 0x07, 0x12, 0x80, 0xe2, 0xeb, 0x27, 0xb2, 0x75,
 0x09, 0x83, 0x2c, 0x1a, 0x1b, 0x6e, 0x5a, 0xa0,
 0x52, 0x3b, 0xd6, 0xb3, 0x29, 0xe3, 0x2f, 0x84,
 0x53, 0xd1, 0x00, 0xed, 0x20, 0xfc, 0xb1, 0x5b,
 0x6a, 0xcb, 0xbe, 0x39, 0x4a, 0x4c, 0x58, 0xcf,
 0xd0, 0xef, 0xaa, 0xfb, 0x43, 0x4d, 0x33, 0x85,
 0x45, 0xf9, 0x02, 0x7f, 0x50, 0x3c, 0x9f, 0xa8,
 0x51, 0xa3, 0x40, 0x8f, 0x92, 0x9d, 0x38, 0xf5,
 0xbc, 0xb6, 0xda, 0x21, 0x10, 0xff, 0xf3, 0xd2,
 0xcd, 0x0c, 0x13, 0xec, 0x5f, 0x97, 0x44, 0x17,
 0xc4, 0xa7, 0x7e, 0x3d, 0x64, 0x5d, 0x19, 0x73,
 0x60, 0x81, 0x4f, 0xdc, 0x22, 0x2a, 0x90, 0x88,
 0x46, 0xee, 0xb8, 0x14, 0xde, 0x5e, 0x0b, 0xdb,
```

```
 0xe0, 0x32, 0x3a, 0x0a, 0x49, 0x06, 0x24, 0x5c,
 0xc2, 0xd3, 0xac, 0x62, 0x91, 0x95, 0xe4, 0x79,
 0xe7, 0xc8, 0x37, 0x6d, 0x8d, 0xd5, 0x4e, 0xa9,
 0x6c, 0x56, 0xf4, 0xea, 0x65, 0x7a, 0xae, 0x08,
 0xba, 0x78, 0x25, 0x2e, 0x1c, 0xa6, 0xb4, 0xc6,
 0xe8, 0xdd, 0x74, 0x1f, 0x4b, 0xbd, 0x8b, 0x8a,
 0x70, 0x3e, 0xb5, 0x66, 0x48, 0x03, 0xf6, 0x0e,
 0x61, 0x35, 0x57, 0xb9, 0x86, 0xc1, 0x1d, 0x9e,
 0xe1, 0xf8, 0x98, 0x11, 0x69, 0xd9, 0x8e, 0x94,
 0x9b, 0x1e, 0x87, 0xe9, 0xce, 0x55, 0x28, 0xdf,
 0x8c, 0xa1, 0x89, 0x0d, 0xbf, 0xe6, 0x42, 0x68,
 0x41, 0x99, 0x2d, 0x0f, 0xb0, 0x54, 0xbb, 0x16);

// inverse S-Box for decryptions
var S_dec = new Array(
 0x52, 0x09, 0x6a, 0xd5, 0x30, 0x36, 0xa5, 0x38,
 0xbf, 0x40, 0xa3, 0x9e, 0x81, 0xf3, 0xd7, 0xfb,
 0x7c, 0xe3, 0x39, 0x82, 0x9b, 0x2f, 0xff, 0x87,
 0x34, 0x8e, 0x43, 0x44, 0xc4, 0xde, 0xe9, 0xcb,
 0x54, 0x7b, 0x94, 0x32, 0xa6, 0xc2, 0x23, 0x3d,
 0xee, 0x4c, 0x95, 0x0b, 0x42, 0xfa, 0xc3, 0x4e,
 0x08, 0x2e, 0xa1, 0x66, 0x28, 0xd9, 0x24, 0xb2,
 0x76, 0x5b, 0xa2, 0x49, 0x6d, 0x8b, 0xd1, 0x25,
 0x72, 0xf8, 0xf6, 0x64, 0x86, 0x68, 0x98, 0x16,
 0xd4, 0xa4, 0x5c, 0xcc, 0x5d, 0x65, 0xb6, 0x92,
 0x6c, 0x70, 0x48, 0x50, 0xfd, 0xed, 0xb9, 0xda,
 0x5e, 0x15, 0x46, 0x57, 0xa7, 0x8d, 0x9d, 0x84,
 0x90, 0xd8, 0xab, 0x00, 0x8c, 0xbc, 0xd3, 0x0a,
 0xf7, 0xe4, 0x58, 0x05, 0xb8, 0xb3, 0x45, 0x06,
 0xd0, 0x2c, 0x1e, 0x8f, 0xca, 0x3f, 0x0f, 0x02,
 0xc1, 0xaf, 0xbd, 0x03, 0x01, 0x13, 0x8a, 0x6b,
 0x3a, 0x91, 0x11, 0x41, 0x4f, 0x67, 0xdc, 0xea,
 0x97, 0xf2, 0xcf, 0xce, 0xf0, 0xb4, 0xe6, 0x73,
 0x96, 0xac, 0x74, 0x22, 0xe7, 0xad, 0x35, 0x85,
 0xe2, 0xf9, 0x37, 0xe8, 0x1c, 0x75, 0xdf, 0x6e,
 0x47, 0xf1, 0x1a, 0x71, 0x1d, 0x29, 0xc5, 0x89,
 0x6f, 0xb7, 0x62, 0x0e, 0xaa, 0x18, 0xbe, 0x1b,
 0xfc, 0x56, 0x3e, 0x4b, 0xc6, 0xd2, 0x79, 0x20,
 0x9a, 0xdb, 0xc0, 0xfe, 0x78, 0xcd, 0x5a, 0xf4,
 0x1f, 0xdd, 0xa8, 0x33, 0x88, 0x07, 0xc7, 0x31,
 0xb1, 0x12, 0x10, 0x59, 0x27, 0x80, 0xec, 0x5f,
 0x60, 0x51, 0x7f, 0xa9, 0x19, 0xb5, 0x4a, 0x0d,
 0x2d, 0xe5, 0x7a, 0x9f, 0x93, 0xc9, 0x9c, 0xef,
 0xa0, 0xe0, 0x3b, 0x4d, 0xae, 0x2a, 0xf5, 0xb0,
 0xc8, 0xeb, 0xbb, 0x3c, 0x83, 0x53, 0x99, 0x61,
 0x17, 0x2b, 0x04, 0x7e, 0xba, 0x77, 0xd6, 0x26,
 0xe1, 0x69, 0x14, 0x63, 0x55, 0x21, 0x0c, 0x7d);

// convert two-dimensional indicies to one-dim array indices
var I00 = 0;
var I01 = 1;
var I02 = 2;
var I03 = 3;
var I10 = 4;
var I11 = 5;
var I12 = 6;
var I13 = 7;
```

```
var I20 = 8;
var I21 = 9;
var I22 = 10;
var I23 = 11;
var I30 = 12;
var I31 = 13;
var I32 = 14;
var I33 = 15;
// conversion function for non-constant subscripts
// assume subscript range 0..3
function I(x,y)
{ return (x*4) + y; }

// remove spaces from input
function remove_spaces( instr )
{
   var i;
   var outstr="";

   for( i=0; i<instr.length; i++ )
      if ( instr.charAt(i) != " " )
         // not a space, include it
         outstr += instr.charAt(i);

   return outstr;
}

// get the message to encrypt/decrypt or the key
// return as a 16-byte array
function get_value( lbl, str, isASCII )
{
   var dbyte = new Array(16);
   var i;
   var val;    // one hex digit

   if ( isASCII )
   {
// ??? pad with spaces/nulls if < 16 chars ???
      // check length of data
      if ( str.length != 16 )
      {
         window.alert( lbl + " length wrong: Is " + str.length +
               "characters, but must be 128 bits (16 ASCII characters)");
         dbyte[0] = -1;
         return dbyte;
      }

      // have ASCII data
      for( i=0; i<16; i++ )
      {
         dbyte[i] = str.charCodeAt(i);
      }
   }
   else
   {
      // have hex data - remove any spaces they used, then convert
      str = remove_spaces(str);
```

```
      // check length of data
      if ( str.length != 32 )
      {
         window.alert( lbl + " length wrong: Is " + str.length +
               " hex digits, but must be 128 bits (32 hex digits)");
         dbyte[0] = -1;
         return dbyte;
      }

      for( i=0; i<16; i++ )
      {
         // isolate and convert this substring
         dbyte[i] = cvt_byte( str.substr(i*2,2) );
         if( dbyte[i] < 0 )
         {
            // have an error
            dbyte[0] = -1;
            return dbyte;
         }
      } // for i
   } // if isASCII

   // return successful conversion
   return dbyte;
} // get_value

//do the AES GF(2**8) multiplication
// do this by the shift-and-"add" approach
function aes_mul( a, b )
{
   var res = 0;

   while( a > 0 )
   {
      if ( a&1 )
         res = res ^ b;               // "add" to the result
      a >>>= 1;               // shift a to get next higher-order bit
      b <<= 1;                // shift multiplier also
   }

   // now reduce it modulo x**8 + x**4 + x**3 + x + 1
   var hbit = 0x10000;        // bit to test if we need to take action
   var modulus = 0x11b00;     // modulus - XOR by this to change value
   while( hbit >= 0x100 )
   {
      if ( res & hbit )               // if the high-order bit is set
         res ^= modulus;      // XOR with the modulus

      // prepare for the next loop
      hbit >>= 1;
      modulus >>= 1;
   }

   return res;
}
```

```
// apply the S-box substitution to the key expansion
function SubWord( word_ary )
{
   var i;

   for( i=0; i<16; i++ )
      word_ary[i] = S_enc[ word_ary[i] ];

   return word_ary;
}

// rotate the bytes in a word
function RotWord( word_ary )
{
   return new Array( word_ary[1], word_ary[2], word_ary[3], word_ary[0]
);
}

// calculate the first item Rcon[i] = { x^(i-1), 0, 0, 0 }
// note we only return the first item
function Rcon( exp )
{
   var val = 2;
   var result = 1;

   // remember to calculate x^(exp-1)
   exp--;

   // process the exponent using normal shift and multiply
   while ( exp > 0 )
   {
      if ( exp & 1 )
         result = aes_mul( result, val );

      // square the value
      val = aes_mul( val, val );

      // move to the next bit
      exp >>= 1;
   }

   return result;
}

// round key generation
// return a byte array with the expanded key information
function key_expand( key )
{
   var temp = new Array(4);
   var i, j;
   var w = new Array( 4*11 );

   // copy initial key stuff
   for( i=0; i<16; i++ )
   {
      w[i] = key[i];
   }
```

```
   accumulate_wordarray( "w[0] = ", w.slice(0,4) );
   accumulate_wordarray( "w[1] = ", w.slice(4,8) );
   accumulate_wordarray( "w[2] = ", w.slice(8,12) );
   accumulate_wordarray( "w[3] = ", w.slice(12,16) );

   // generate rest of key schedule using 32-bit words
   i = 4;
   while ( i < 44 )           // blocksize * ( rounds + 1 )
   {
      // copy word W[i-1] to temp
      for( j=0; j<4; j++ )
         temp[j] = w[(i-1)*4+j];

      if ( i % 4 == 0)
      {
         // temp = SubWord(RotWord(temp)) ^ Rcon[i/4];
         temp = RotWord( temp );
         accumulate_wordarray( "RotWord()=", temp );
         temp = SubWord( temp );
         accumulate_wordarray( "SubWord()=", temp );
         temp[0] ^= Rcon( i>>>2 );
         accumulate_wordarray( " ^ Rcon()=", temp );
      }

      // word = word ^ temp
      for( j=0; j<4; j++ )
         w[i*4+j] = w[(i-4)*4+j] ^ temp[j];
      accumulate_wordarray( "w["+i+"] = ", w.slice( i*4, i*4+4 ) );

      i++;
   }

   return w;
}

// do S-Box substitution
function SubBytes(state, Sbox)
{
   var i;

   for( i=0; i<16; i++ )
      state[i] = Sbox[ state[i] ];

   return state;
}

// shift each row as appropriate
function ShiftRows(state)
{
   var t0, t1, t2, t3;

   // top row (row 0) isn't shifted

   // next row (row 1) rotated left 1 place
   t0 = state[I10];
   t1 = state[I11];
   t2 = state[I12];
```

```
   t3 = state[I13];
   state[I10] = t1;
   state[I11] = t2;
   state[I12] = t3;
   state[I13] = t0;

   // next row (row 2) rotated left 2 places
   t0 = state[I20];
   t1 = state[I21];
   t2 = state[I22];
   t3 = state[I23];
   state[I20] = t2;
   state[I21] = t3;
   state[I22] = t0;
   state[I23] = t1;

   // bottom row (row 3) rotated left 3 places
   t0 = state[I30];
   t1 = state[I31];
   t2 = state[I32];
   t3 = state[I33];
   state[I30] = t3;
   state[I31] = t0;
   state[I32] = t1;
   state[I33] = t2;

   return state;
}

// inverset shift each row as appropriate
function InvShiftRows(state)
{
   var t0, t1, t2, t3;

   // top row (row 0) isn't shifted

   // next row (row 1) rotated left 1 place
   t0 = state[I10];
   t1 = state[I11];
   t2 = state[I12];
   t3 = state[I13];
   state[I10] = t3;
   state[I11] = t0;
   state[I12] = t1;
   state[I13] = t2;

   // next row (row 2) rotated left 2 places
   t0 = state[I20];
   t1 = state[I21];
   t2 = state[I22];
   t3 = state[I23];
   state[I20] = t2;
   state[I21] = t3;
   state[I22] = t0;
   state[I23] = t1;

   // bottom row (row 3) rotated left 3 places
```

```
   t0 = state[I30];
   t1 = state[I31];
   t2 = state[I32];
   t3 = state[I33];
   state[I30] = t1;
   state[I31] = t2;
   state[I32] = t3;
   state[I33] = t0;

   return state;
}

// process column info
function MixColumns(state)
{
   var col;
   var c0, c1, c2, c3;

   for( col=0; col<4; col++ )
   {
      c0 = state[I(0,col)];
      c1 = state[I(1,col)];
      c2 = state[I(2,col)];
      c3 = state[I(3,col)];

      // do mixing, and put back into array
      state[I(0,col)] = aes_mul(2,c0) ^ aes_mul(3,c1) ^ c2 ^ c3;
      state[I(1,col)] = c0 ^ aes_mul(2,c1) ^ aes_mul(3,c2) ^ c3;
      state[I(2,col)] = c0 ^ c1 ^ aes_mul(2,c2) ^ aes_mul(3,c3);
      state[I(3,col)] = aes_mul(3,c0) ^ c1 ^ c2 ^ aes_mul(2,c3);
   }

   return state;
}

// inverse process column info
function InvMixColumns(state)
{
   var col;
   var c0, c1, c2, c3;

   for( col=0; col<4; col++ )
   {
      c0 = state[I(0,col)];
      c1 = state[I(1,col)];
      c2 = state[I(2,col)];
      c3 = state[I(3,col)];

      // do inverse mixing, and put back into array
      state[I(0,col)] = aes_mul(0x0e,c0) ^ aes_mul(0x0b,c1)
                        ^ aes_mul(0x0d,c2) ^ aes_mul(0x09,c3);
      state[I(1,col)] = aes_mul(0x09,c0) ^ aes_mul(0x0e,c1)
                        ^ aes_mul(0x0b,c2) ^ aes_mul(0x0d,c3);
      state[I(2,col)] = aes_mul(0x0d,c0) ^ aes_mul(0x09,c1)
                        ^ aes_mul(0x0e,c2) ^ aes_mul(0x0b,c3);
      state[I(3,col)] = aes_mul(0x0b,c0) ^ aes_mul(0x0d,c1)
                        ^ aes_mul(0x09,c2) ^ aes_mul(0x0e,c3);
```

```
   }

   return state;
}

// insert subkey information
function AddRoundKey( state, w, base )
{
   var col;

   for( col=0; col<4; col++ )
   {
      state[I(0,col)] ^= w[base+col*4];
      state[I(1,col)] ^= w[base+col*4+1];
      state[I(2,col)] ^= w[base+col*4+2];
      state[I(3,col)] ^= w[base+col*4+3];
   }

   return state;
}

// return a transposed array
function transpose( msg )
{
   var row, col;
   var state = new Array( 16 );

   for( row=0; row<4; row++ )
      for( col=0; col<4; col++ )
         state[I(row,col)] = msg[I(col,row)];

   return state;
}

// final AES state
var AES_output = new Array(16);

// format AES output
// -- uses the global array DES_output
function format_AES_output()
{
   var i;
   var bits;
   var str="";

   // what type of data do we have to work with?
   if ( document.stuff.outtype[0].checked )
   {
      // convert each set of bits back to ASCII
      for( i=0; i<16; i++ )
         str += String.fromCharCode( AES_output[i] );
   }
   else
   {
      // output hexdecimal data (insert spaces)
      str = cvt_hex8( AES_output[0] );
      for( i=1; i<16; i++ )
```

```
      {
         str += " " + cvt_hex8( AES_output[i] );
      }
   }

   // copy to textbox
   document.stuff.outdata.value = str;
}

// do encrytion
function aes_encrypt()
{
   var w = new Array( 44 );                    // subkey information
   var state = new Array( 16 );                        // working state
   var round;

   accumulated_output_info = "";

   // get the message from the user
   // also check if it is ASCII or hex
   var msg = get_value( "Message", document.stuff.indata.value,
               document.stuff.intype[0].checked );

   // problems??
   if ( msg[0] < 0 )
   {
      document.stuff.details.value = accumulated_output_info;
      return;
   }
   accumulate_array( "Input bits", msg );

   // get the key from the user
   var key = get_value( "Key", document.stuff.key.value, false );
   // problems??
   if ( key[0] < 0 )
   {
      document.stuff.details.value = accumulated_output_info;
      return;
   }
   accumulate_array( "Key bits", key );

   // expand the key
   w = key_expand( key );

   // initial state = message in columns (transposed from what we input)
   state = transpose( msg );

   accumulate_array( "Initial state", state );
   // display the round key - Transpose due to the way it is stored/used
   accumulate_array( "Round Key", transpose(w.slice( 0, 16 )) );
   state = AddRoundKey(state, w, 0);

   for( round=1; round<10; round++ )
   {
      accumulate_array( "Round " + round, state );
      state = SubBytes(state, S_enc);
      accumulate_array( "After SubBytes", state );
```

```
      state = ShiftRows(state);
      accumulate_array( "After ShiftRows", state );
      state = MixColumns(state);
      accumulate_array( "After MixColumns", state );
      // display the round key - Transpose due to the way it is
stored/used
      accumulate_array( "Round Key", transpose(w.slice( round*4*4,
round*16+16 )) );
      // note here the spec uses 32-bit words, we are using bytes, so an
extra *4
      state = AddRoundKey(state, w, round*4*4);
   }

   SubBytes(state, S_enc);
   accumulate_array( "After SubBytes", state );
   ShiftRows(state);
   accumulate_array( "After ShiftRows", state );
   AddRoundKey(state, w, 10*4*4);
   accumulate_array( "Output", state );

   // process output
   AES_output = transpose( state );
   format_AES_output( );
   document.stuff.details.value = accumulated_output_info;
}

// do decryption
function aes_decrypt()
{
   var w = new Array( 44 );                  // subkey information
   var state = new Array( 16 );                      // working state
   var round;

   accumulated_output_info = "";

   // get the message from the user
   // also check if it is ASCII or hex
   var msg = get_value( "Message", document.stuff.indata.value,
               document.stuff.intype[0].checked );

   // problems??
   if ( msg[0] < 0 )
   {
      document.stuff.details.value = accumulated_output_info;
      return;
   }
   accumulate_array( "Input bits", msg );

   // get the key from the user
   var key = get_value( "Key", document.stuff.key.value, false );
   // problems??
   if ( key[0] < 0 )
   {
      document.stuff.details.value = accumulated_output_info;
      return;
   }
   accumulate_array( "Key bits", key );
```

```
   // expand the key
   w = key_expand( key );

   // initial state = message
   state = transpose( msg );

   accumulate_array( "Initial state", state );
   // display the round key - Transpose due to the way it is stored/used
   accumulate_array( "Round Key", transpose(w.slice( 10*4*4, 10*4*4+16 ))
);
   state = AddRoundKey(state, w, 10*4*4);

   for( round=9; round>=1; round-- )
   {
      accumulate_array( "Round " + round, state );
      state = InvShiftRows(state);
      accumulate_array( "After InvShiftRows", state );
      state = SubBytes(state, S_dec);
      accumulate_array( "After SubBytes", state );
      // display the round key - Transpose due to the way it is
stored/used
      accumulate_array( "Round Key", transpose(w.slice( round*4*4,
round*16+16 )) );
      // note here the spec uses 32-bit words, we are using bytes, so an
extra *4
      state = AddRoundKey(state, w, round*4*4);
      accumulate_array( "After AddRoundKey", state );
      state = InvMixColumns(state);
   }

   InvShiftRows(state);
   accumulate_array( "After InvShiftRows", state );
   SubBytes(state, S_dec);
   accumulate_array( "After SubBytes", state );
   AddRoundKey(state, w, 0);
   accumulate_array( "Output", state );

   // process output
   AES_output = transpose( state );
   format_AES_output( );
   document.stuff.details.value = accumulated_output_info;
}
```

ANEXO 2

VECTORES DE PRUEBA

Available: http://fp.gladman.plus.com/AES/

These test have been generated by Dr Brian Gladman using the
program aes_vec.cpp 24th May 2001.

LEGEND FOR CIPHER ENCRYPT - round r = 0 to R, R = 10..14
input: cipher input
start: state at start of round[r]
s_box: state after s_box substitution
s_row: state after shift row transformation
m_col: state after mix column transformation
k_sch: key schedule value for round[r]
output: cipher output

LEGEND FOR CIPHER DECRYPT - round r = 0 to R, R = 10..14
KEY SCHEDULE FOR KEY XOR FOLLOWED BY INVERSE MIX COLUMN
iinput: inverse cipher input
istart: state at start of round[r]
is_box: state after inverse s_box substitution
is_row: state after inverse shift row transformation
ik_sch: key schedule value for round[r]
ik_add: state after key addition
ioutput: cipher output

LEGEND FOR CIPHER DECRYPT (MOD) - round r = 0 to R, R = 10..14
KEY SCHEDULE FOR INVERSE MIX COLUMN FOLLOWED BY KEY XOR
iinput: inverse cipher input
istart: state at start of round[r]
is_box: state after inverse s_box substitution
is_row: state after inverse shift row transformation
im_col: state after inverse mix column transformation
ik_sch: key schedule value for round[r]
ioutput: cipher output

PLAINTEXT: 3243f6a8885a308d313198a2e0370734
KEY: 2b7e151628aed2a6abf7158809cf4f3c
ENCRYPT 16 byte block, 16 byte key
R[00].input 3243f6a8885a308d313198a2e0370734
R[00].k_sch 2b7e151628aed2a6abf7158809cf4f3c
R[01].start 193de3bea0f4e22b9ac68d2ae9f84808
R[01].s_box d42711aee0bf98f1b8b45de51e415230
R[01].s_row d4bf5d30e0b452aeb84111f11e2798e5
R[01].m_col 046681e5e0cb199a48f8d37a2806264c

R[01].k_sch a0fafe1788542cb123a339392a6c7605
R[02].start a49c7ff2689f352b6b5bea43026a5049
R[02].s_box 49ded28945db96f17f39871a7702533b
R[02].s_row 49db873b453953897f02d2f177de961a
R[02].m_col 584dcaf11b4b5aacdbe7caa81b6bb0e5
R[02].k_sch f2c295f27a96b9435935807a7359f67f
R[03].start aa8f5f0361dde3ef82d24ad26832469a
R[03].s_box ac73cf7befc111df13b5d6b545235ab8
R[03].s_row acc1d6b8efb55a7b1323cfdf457311b5
R[03].m_col 75ec0993200b633353c0cf7cbb25d0dc
R[03].k_sch 3d80477d4716fe3e1e237e446d7a883b
R[04].start 486c4eee671d9d0d4de3b138d65f58e7
R[04].s_box 52502f2885a45ed7e311c807f6cf6a94
R[04].s_row 52a4c89485116a28e3cf2fd7f6505e07
R[04].m_col 0fd6daa9603138bf6fc0106b5eb31301
R[04].k_sch ef44a541a8525b7fb671253bdb0bad00
R[05].start e0927fe8c86363c0d9b1355085b8be01
R[05].s_box e14fd29be8fbfbba35c89653976cae7c
R[05].s_row e1fb967ce8c8ae9b356cd2ba974ffb53
R[05].m_col 25d1a9adbd11d168b63a338e4c4cc0b0
R[05].k_sch d4d1c6f87c839d87caf2b8bc11f915bc
R[06].start f1006f55c1924cef7cc88b325db5d50c
R[06].s_box a163a8fc784f29df10e83d234cd503fe
R[06].s_row a14f3dfe78e803fc10d5a8df4c632923
R[06].m_col 4b868d6d2c4a8980339df4e837d218d8
R[06].k_sch 6d88a37a110b3efddbf98641ca0093fd
R[07].start 260e2e173d41b77de86472a9fdd28b25
R[07].s_box f7ab31f02783a9ff9b4340d354b53d3f
R[07].s_row f783403f27433df09bb531ff54aba9d3
R[07].m_col 1415b5bf461615ec274656d7342ad843
R[07].k_sch 4e54f70e5f5fc9f384a64fb24ea6dc4f
R[08].start 5a4142b11949dc1fa3e019657a8c040c
R[08].s_box be832cc8d43b86c00ae1d44dda64f2fe
R[08].s_row be3bd4fed4e1f2c80a642cc0da83864d
R[08].m_col 00512fd1b1c889ff54766dcdfa1b99ea
R[08].k_sch ead27321b58dbad2312bf5607f8d292f
R[09].start ea835cf00445332d655d98ad8596b0c5
R[09].s_box 87ec4a8cf26ec3d84d4c46959790e7a6
R[09].s_row 876e46a6f24ce78c4d904ad897ecc395
R[09].m_col 473794ed40d4e4a5a3703aa64c9f42bc
R[09].k_sch ac7766f319fadc2128d12941575c006e
R[10].start eb40f21e592e38848ba113e71bc342d2
R[10].s_box e9098972cb31075f3d327d94af2e2cb5
R[10].s_row e9317db5cb322c723d2e895faf090794
R[10].k_sch d014f9a8c9ee2589e13f0cc8b6630ca6
R[10].output 3925841d02dc09fbdc118597196a0b32

ANEXO 3

CÓDIGO EN VHDL DE ALGUNOS BLOQUES SIGNIFICATIVOS

KEY SCHEDULE CIFRADOR

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use work.aespackage.all;
use work.my_components.all;

entity keysch is
 port (
   key_in   : in  s_vector;
   key_out  : out s_vector;
         rcon       : in  std_logic_vector(7 downto 0);
         clk   : in std_logic;
   reset : in std_logic
   );
end keysch;

architecture Behavioral of keysch is

signal inkey,round_key : s_vector;
signal ci3,ci2,ci1,ci0: std_logic_vector(31 downto 0);
signal co3,co2,co1,co0: std_logic_vector(31 downto 0);
signal rotword,srotword: std_logic_vector(31 downto 0);

begin

---generacion de la subclave 128 bits
-----------------------------------------------------
ci3<= key_in(0)       & key_in(4)   & key_in(8)   & key_in(12);
ci2<=  key_in(1)      & key_in(5)   & key_in(9)   & key_in(13);
ci1<=  key_in(2)      & key_in(6)   & key_in(10)  & key_in(14);
ci0<=  key_in(3)      & key_in(7)   & key_in(11)  & key_in(15);
-------------------------------------------------------------
rotword<= ci0(23 downto 0) & ci0(31 downto 24);

co3<=((rcon xor srotword(31 downto 24)) & srotword(23 downto 0)) xor ci3;
co2<=co3 xor ci2;
co1<=co2 xor ci1;
co0<=co1 xor ci0;
```

```
key_out(0) <= co3(31 downto 24);
key_out(4) <= co3(23 downto 16);
key_out(8) <= co3(15 downto 8);
key_out(12) <= co3(7 downto 0);
key_out(1) <= co2(31 downto 24);
key_out(5) <= co2(23 downto 16);
key_out(9) <= co2(15 downto 8);
key_out(13) <= co2(7 downto 0);
key_out(2) <= co1(31 downto 24);
key_out(6) <= co1(23 downto 16);
key_out(10) <= co1(15 downto 8);
key_out(14) <= co1(7 downto 0);
key_out(3) <= co0(31 downto 24);
key_out(7) <= co0(23 downto 16);
key_out(11) <= co0(15 downto 8);
key_out(15) <= co0(7 downto 0);

-------------------------------------------------------
--genera las 4 sbox de key schedule

   GEN_SBOX: for i in 0 to 3 generate
     SBOX_i: sbox
       port map (
          clk   => clk,
          reset => reset,
          addra => rotword((8*i)+7 downto (8*i)),
          douta => srotword((8*i)+7 downto (8*i)));
   end generate GEN_SBOX;

-------------------------------------------------------------

end Behavioral;
```

KEY SCHEDULE DESCIFRADOR

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use work.aespackage.all;
use work.my_components.all;

entity keysch is
 port (
   key_in   : in  s_vector;
   key_out  : out s_vector;
```

```
        rcon       : in  std_logic_vector(7 downto 0);
        clk   : in std_logic;
   reset : in std_logic
   );
end keysch;

architecture Behavioral of keysch is

signal ci3,ci2,ci1,ci0: std_logic_vector(31 downto 0);
signal co3,co2,co1,co0,col: std_logic_vector(31 downto 0);
signal rotword,srotword: std_logic_vector(31 downto 0);

begin

---generacion de la subclave 128 bits
-------------------------------------------------------
ci3<= key_in(0)& key_in(4)  & key_in(8)    & key_in(12);
ci2<=  key_in(1)& key_in(5)& key_in(9)    & key_in(13);
ci1<=  key_in(2)& key_in(6)& key_in(10)& key_in(14);
ci0<=  key_in(3)& key_in(7)& key_in(11)& key_in(15);
-------------------------------------------------------------

-----------------------------------------------

col<= (ci3(31 downto 24) xor rcon) & ci3(23 downto 0);
rotword<=co0(23 downto 0) & co0(31 downto 24);

co3<=col xor srotword;
co2<=ci3 xor ci2;
co1<=ci2 xor ci1;
co0<=ci1 xor ci0;

key_out(0) <= co3(31 downto 24);
key_out(4) <= co3(23 downto 16);
key_out(8) <= co3(15 downto 8);
key_out(12) <=co3(7 downto 0);
key_out(1) <= co2(31 downto 24);
key_out(5) <= co2(23 downto 16);
key_out(9) <= co2(15 downto 8);
key_out(13) <= co2(7 downto 0);
key_out(2) <= co1(31 downto 24);
key_out(6) <= co1(23 downto 16);
key_out(10) <= co1(15 downto 8);
key_out(14) <= co1(7 downto 0);
key_out(3) <= co0(31 downto 24);
key_out(7) <= co0(23 downto 16);
key_out(11) <= co0(15 downto 8);
```

```
key_out(15) <= co0(7 downto 0);

-------------------------------------------------
--genera las 4 sbox de key schedule

   GEN_SBOX: for i in 0 to 3 generate
     SBOX_i: sbox
       port map (
          clk   => clk,
          reset => reset,
          addra => rotword((8*i)+7 downto (8*i)),
          douta => srotword((8*i)+7 downto (8*i)));
   end generate GEN_SBOX;

-----------------------------------------------------------

end Behavioral;
```

MIXCOLUMNS INVERSA (DESCIFRADO)

```
library ieee;
use ieee.std_logic_1164.all;
use work.my_components.all;
use work.aespackage.all;

entity inv_mixcolumns is
  port (
    bytes_in  : in  s_vector;
    bytes_out : out s_vector;
           clk   : in std_logic;
    reset : in std_logic
    );
end inv_mixcolumns;

architecture rtl of inv_mixcolumns is
  signal reg : s_vector;

-------------------------------
 function transform(p : std_logic_vector(31 downto 0) ) return std_logic_vector is
    variable result   : std_logic_vector(7 downto 0);
    variable b1,b2,b3,b4 : std_logic_vector(7 downto 0);
           variable M0E,M0B,M0D,M09 : std_logic_vector(7 downto 0);
    variable vE1,vE2,vE3: std_logic;
           variable vD1,vD2,vD3,vD4: std_logic;
           variable vB1,vB2,vB3: std_logic;
```

```
        variable v1,v2: std_logic;

 begin

  b1:= p(31 downto 24);
  b2:= p(23 downto 16);
  b3:= p(15 downto 8);
  b4:= p(7 downto 0);

  vE1:=b1(6) xor b1(5);
  vE2:=b1(2) xor b1(1);
  vE3:=b1(5) xor b1(3);
  M0E:= (vE1 xor b1(4)) & (b1(7) xor b1(4) xor vE3) & (b1(6) xor b1(4) xor b1(3) xor
b1(2)) & (vE3 xor vE2) & (vE1 xor vE2 xor b1(0)) & (b1(6) xor b1(1) xor b1(0)) & (b1(5)
xor b1(0)) & (b1(7) xor vE1);

  vB1:=b2(7) xor b2(6);
  vB2:=b2(5) xor b2(3);
  vB3:=b2(5) xor b2(4);
  M0B:= (vB1 xor b2(4)) & (vB1 xor vB2) & (vB1 xor vB3 xor b2(2)) & (vB1 xor vB3 xor
b2(3) xor b2(1)) & (vB2 xor b2(2) xor b2(0)) & (vB1 xor b2(2) xor b2(1)) & (vB1 xor
b2(5) xor b2(1) xor b2(0)) & (b2(7) xor b2(5) xor b2(0));

  vD1:=b3(6) xor b3(5);
  vD2:=b3(5) xor b3(4);
  vD3:=b3(3) xor b3(2);
  vD4:=b3(7) xor b3(1);
  M0D:= (b3(7) xor vD2) & (b3(7) xor b3(6) xor b3(4) xor b3(3)) & (vD1 xor vD3) & (vD2
xor vD4 xor b3(2)) & (vD1 xor b3(7) xor b3(3) xor b3(1) xor b3(0)) & (b3(6) xor b3(2) xor
b3(0)) & (vD4 xor b3(5)) & (vD1 xor b3(0));

  v1:=b4(6) xor b4(5);
  v2:=b4(7) xor b4(6);
  M09:= (b4(7) xor b4(4)) & (v2 xor b4(3)) & (b4(7) xor b4(2) xor v1) & (v1 xor b4(4) xor
b4(1)) & (b4(7) xor b4(5) xor b4(3) xor b4(0)) & (v2 xor b4(2)) & (v1 xor b4(1)) & (b4(5)
xor b4(0));

  result := M0E xor M0B xor M0D xor M09;
    return result;
  end function transform;
-------------------------------------------
BEGIN

  REGGEN: process (clk, reset)
  begin
    if reset = '1' then
      reg <= (others => (others => '0'));
```

```
    elsif clk'event and clk = '1' then
      reg <= bytes_in;
    end if;
  end process REGGEN;

-----------------------------------------------------
  bytes_out(0) <= transform(reg(0) & reg(4) & reg(8) & reg(12));
  bytes_out(1) <= transform(reg(1) & reg(5) & reg(9) & reg(13));
  bytes_out(2) <= transform(reg(2) & reg(6) & reg(10) & reg(14));
  bytes_out(3) <= transform(reg(3) & reg(7) & reg(11) & reg(15));

  bytes_out(4) <= transform(reg(4) & reg(8) & reg(12) & reg(0));
  bytes_out(5) <= transform(reg(5) & reg(9) & reg(13) & reg(1));
  bytes_out(6) <= transform(reg(6) & reg(10) & reg(14) & reg(2));
  bytes_out(7) <= transform(reg(7) & reg(11) & reg(15) & reg(3));

  bytes_out(8) <= transform(reg(8) & reg(12) & reg(0) & reg(4));
  bytes_out(9) <= transform(reg(9) & reg(13) & reg(1) & reg(5));
  bytes_out(10) <= transform(reg(10) & reg(14) & reg(2) & reg(6));
  bytes_out(11) <= transform(reg(11) & reg(15) & reg(3) & reg(7));

  bytes_out(12) <= transform(reg(12) & reg(0) & reg(4) & reg(8));
  bytes_out(13) <= transform(reg(13) & reg(1) & reg(5) & reg(9));
  bytes_out(14) <= transform(reg(14) & reg(2) & reg(6) & reg(10));
  bytes_out(15) <= transform(reg(15) & reg(3) & reg(7) & reg(11));

end rtl;
```

BLOQUE DE CIFRADO

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use work.my_components.all;
use work.aespackage.all;

entity aes_unit is
  port (
      plain_text: in std_logic_vector(127 downto 0);
      cipher_key: in std_logic_vector(127 downto 0);
      cipher_text : out std_logic_vector(127 downto 0);
```

```
      en_crypt   : in std_logic;
      flag_crypt : out std_logic;
      clk   : in std_logic;
    reset : in std_logic
    );
end aes_unit;

architecture rtl of aes_unit is

type state_values is (inicio,r0,r1,r2,r3,r4,r5,r6,r7,r8,r9,
                                          delay0,delay1,delay2,delay3,delay4,

                                          delay5,delay6,delay7,delay8,delay9,
                                          d0,d1,d2,d3,d4,d5,d6,d7,d8,d9
                                          );

signal actual, futuro: state_values;

signal round_out,round_out_r: s_vector;--in,out reg
signal round_key,round_key_r: s_vector;--in, out reg

signal round_in,key_in: s_vector;
signal state_ini,key_ini: s_vector;--valores ronda inicial

signal rcon: std_logic_vector(7 downto 0);
signal round_ini: std_logic_vector(127 downto 0);

signal enrk,enrs,muxkey,muxstate: std_logic;
signal mc: std_logic;
signal round_counter : std_logic_vector(9 downto 0);
```

```
begin
-------------------------------------------------------------
round_ini<=plain_text xor cipher_key;

state_ini(0)  <= round_ini(127 downto 120);
state_ini(4)  <= round_ini(119 downto 112);
state_ini(8)  <= round_ini(111 downto 104);
state_ini(12) <= round_ini(103 downto 96);
state_ini(1)  <= round_ini(95 downto 88);
state_ini(5)  <= round_ini(87 downto 80);
state_ini(9)  <= round_ini(79 downto 72);
state_ini(13) <= round_ini(71 downto 64);
state_ini(2)  <= round_ini(63 downto 56);
state_ini(6)  <= round_ini(55 downto 48);
state_ini(10) <= round_ini(47 downto 40);
state_ini(14) <= round_ini(39 downto 32);
state_ini(3)  <= round_ini(31 downto 24);
state_ini(7)  <= round_ini(23 downto 16);
state_ini(11) <= round_ini(15 downto 8);
state_ini(15) <= round_ini(7 downto 0);

key_ini(0)  <= cipher_key(127 downto 120);
key_ini(4)  <= cipher_key(119 downto 112);
key_ini(8)  <= cipher_key(111 downto 104);
key_ini(12) <= cipher_key(103 downto 96);
key_ini(1)  <= cipher_key(95 downto 88);
key_ini(5)  <= cipher_key(87 downto 80);
key_ini(9)  <= cipher_key(79 downto 72);
key_ini(13) <= cipher_key(71 downto 64);
key_ini(2)  <= cipher_key(63 downto 56);
key_ini(6)  <= cipher_key(55 downto 48);
```

```
key_ini(10) <= cipher_key(47 downto 40);
key_ini(14) <= cipher_key(39 downto 32);
key_ini(3)  <= cipher_key(31 downto 24);
key_ini(7)  <= cipher_key(23 downto 16);
key_ini(11) <= cipher_key(15 downto 8);
key_ini(15) <= cipher_key(7 downto 0);
-------------------------------------------------------------
--SALIDA TEXTO PLANO
cipher_text(127 downto 120)<=round_out_r(0);
cipher_text(119 downto 112)<=round_out_r(4) ;
cipher_text(111 downto 104)<=round_out_r(8) ;
cipher_text(103 downto 96)<=round_out_r(12) ;
cipher_text(95 downto 88)<=round_out_r(1) ;
cipher_text(87 downto 80)<=round_out_r(5) ;
cipher_text(79 downto 72)<=round_out_r(9) ;
cipher_text(71 downto 64)<=round_out_r(13) ;
cipher_text(63 downto 56)<=round_out_r(2) ;
cipher_text(55 downto 48)<=round_out_r(6) ;
cipher_text(47 downto 40)<=round_out_r(10) ;
cipher_text(39 downto 32)<=round_out_r(14) ;
cipher_text(31 downto 24)<=round_out_r(3) ;
cipher_text(23 downto 16)<=round_out_r(7) ;
cipher_text(15 downto 8)<=round_out_r(11) ;
cipher_text(7 downto 0)<=round_out_r(15) ;

-------------------------------------------------------------
-----------------------------------------------------
with round_counter select
    rcon <=
    X"02" when "0000000010",
    X"04" when "0000000100",
```

```
    X"08" when "0000001000",
    X"10" when "0000010000",
    X"20" when "0000100000",
    X"40" when "0001000000",
    X"80" when "0010000000",
    X"1b" when "0100000000",
    X"36" when "1000000000",
    X"01" when others;
------------------------------------------------------------
--FSM PARA EL CONTROL DEL CIFRADO
fsmcrypt : process(actual,en_crypt)

        begin

                case actual is

                when inicio =>

                        round_counter <= "0000000001";
                        enrk<='0';enrs<='1';
                        muxkey<='0';muxstate<='0';
                        mc<='1';
                        flag_crypt<='0';

                        if en_crypt = '1' then
                                futuro <= r0;
                        else
                                futuro <= inicio;
                        end if;

                 when r0 =>
```

```
                enrk<='1';enrs<='0';
                muxkey<='0';muxstate<='0';
                mc<='1';
                flag_crypt<='0';
                futuro <= delay0;

when delay0 =>

                round_counter <= "0000000010";
                enrk<='0';enrs<='1';
                muxkey<='1';muxstate<='1';
                mc<='1';
                flag_crypt<='0';
                futuro <= d0;

when d0 =>

                enrk<='0';enrs<='0';
                muxkey<='1';muxstate<='1';
                mc<='1';
                flag_crypt<='0';
                futuro <= r1;

when r1 =>

                enrk<='1';enrs<='0';
                muxkey<='1';muxstate<='1';
                mc<='1';
                flag_crypt<='0';
                futuro <= delay1;
```

```
when delay1 =>

                round_counter <= round_counter(8 downto 0) & "0";


                enrk<='0';enrs<='1';
                muxkey<='1';muxstate<='1';
                mc<='1';
                flag_crypt<='0';
                futuro <= d1;

when d1 =>


                enrk<='0';enrs<='0';
                muxkey<='1';muxstate<='1';
                mc<='1';
                flag_crypt<='0';
                futuro <= r2;

when r2 =>

                enrk<='1';enrs<='0';
                muxkey<='1';muxstate<='1';
                mc<='1';
                flag_crypt<='0';
                futuro <= delay2;

when delay2 =>
```

```
                round_counter <= round_counter(8 downto 0) & "0";


                enrk<='0';enrs<='1';
                muxkey<='1';muxstate<='1';
                mc<='1';
                flag_crypt<='0';
                futuro <= d2;

when d2 =>

                enrk<='0';enrs<='0';
                muxkey<='1';muxstate<='1';
                mc<='1';
                flag_crypt<='0';
                futuro <= r3;

 when r3 =>

                enrk<='1';enrs<='0';
                muxkey<='1';muxstate<='1';
                mc<='1';
                flag_crypt<='0';
                futuro <= delay3;

 when delay3 =>

                round_counter <= round_counter(8 downto 0) & "0";

                enrk<='0';enrs<='1';
                muxkey<='1';muxstate<='1';
```

```
                mc<='1';
                flag_crypt<='0';
                futuro <= d3;

when d3 =>

                enrk<='0';enrs<='0';
                muxkey<='1';muxstate<='1';
                mc<='1';
                flag_crypt<='0';
                futuro <= r4;

when r4 =>

                enrk<='1';enrs<='0';
                muxkey<='1';muxstate<='1';
                mc<='1';
                flag_crypt<='0';
                futuro <= delay4;

 when delay4 =>

                round_counter <= round_counter(8 downto 0) & "0";


                enrk<='0';enrs<='1';
                muxkey<='1';muxstate<='1';
                mc<='1';
                flag_crypt<='0';
                futuro <= d4;
```

```
when d4 =>

                enrk<='0';enrs<='0';
                muxkey<='1';muxstate<='1';
                mc<='1';
                flag_crypt<='0';
                futuro <= r5;

when r5 =>

                enrk<='1';enrs<='0';
                muxkey<='1';muxstate<='1';
                mc<='1';
                flag_crypt<='0';
                futuro <= delay5;

when delay5 =>

                round_counter <= round_counter(8 downto 0) & "0";


                enrk<='0';enrs<='1';
                muxkey<='1';muxstate<='1';
                mc<='1';
                flag_crypt<='0';
                futuro <= d5;

when d5 =>

                enrk<='0';enrs<='0';
                muxkey<='1';muxstate<='1';
```

```
                mc<='1';
                flag_crypt<='0';
                futuro <= r6;

when r6 =>

                enrk<='1';enrs<='0';
                muxkey<='1';muxstate<='1';
                mc<='1';
                flag_crypt<='0';
                futuro <= delay6;

when delay6 =>

                round_counter <= round_counter(8 downto 0) & "0";


                enrk<='0';enrs<='1';
                muxkey<='1';muxstate<='1';
                mc<='1';
                flag_crypt<='0';
                futuro <= d6;

when d6 =>

                enrk<='0';enrs<='0';
                muxkey<='1';muxstate<='1';
                mc<='1';
                flag_crypt<='0';
                futuro <= r7;
```

```
when r7 =>

                enrk<='1';enrs<='0';
                muxkey<='1';muxstate<='1';
                mc<='1';
                flag_crypt<='0';
                futuro <= delay7;

when delay7 =>

                round_counter <= round_counter(8 downto 0) & "0";


                enrk<='0';enrs<='1';
                muxkey<='1';muxstate<='1';
                mc<='1';
                flag_crypt<='0';
                futuro <= d7;

when d7 =>

                enrk<='0';enrs<='0';
                muxkey<='1';muxstate<='1';
                mc<='1';
                flag_crypt<='0';
                futuro <= r8;

when r8 =>

                enrk<='1';enrs<='0';
                muxkey<='1';muxstate<='1';
```

```
                mc<='1';
                flag_crypt<='0';
                futuro <= delay8;

when delay8 =>

            round_counter <= round_counter(8 downto 0) & "0";


                enrk<='0';enrs<='1';
                muxkey<='1';muxstate<='1';
                mc<='1';
                flag_crypt<='0';
                futuro <= d8;

 when d8 =>

                enrk<='0';enrs<='0';
                muxkey<='1';muxstate<='1';
                mc<='1';
                flag_crypt<='0';
                futuro <= r9;

 when r9 =>

                enrk<='1';enrs<='0';
                muxkey<='1';muxstate<='1';
                mc<='0';
                flag_crypt<='0';
                futuro <= delay9;
```

```
                when delay9 =>

                        enrk<='0';enrs<='1';
                        muxkey<='1';muxstate<='1';
                        mc<='0';
                        flag_crypt<='0';
                        futuro <= d9;

                when d9 =>

                        enrk<='0';enrs<='0';
                        muxkey<='1';muxstate<='1';
                        mc<='0';
                        flag_crypt<='1';
                        futuro <= d9;

                end case;

end process fsmcrypt;

sincronismo : process(clk,futuro,reset)

begin
    if(reset = '1') then
                actual <= inicio;
        elsif(clk'event and clk ='1') then
                actual <= futuro;
      end if;
end process sincronismo;
-------------------------------------------------------------
--CONEXIÓN UNIDADES FUNCIONALES
```

```
muxkgen: muxk
          port map (
          w0  => key_ini,--clave inicial
       w1 => round_key_r,--
       f => key_in,--
          s  => muxkey
          );

muxsgen: muxs
          port map (
          w0  => state_ini,--state ronda inicial
       w1 => round_out,--
       f => round_in,--
          s  => muxstate
          );

roundkeygen: keysch
          port map (
      key_in  => key_in,--
      key_out => round_key,--
      rcon => rcon,
          clk  => clk,
      reset  => reset
          );

regkey : reg128
    port map (
      R  => round_key,
      Q => round_key_r,
      clk  => clk,
```

```
      Rin  => enrk
          );

regstate : reg128
    port map (
      R  => round_in,
      Q => round_out_r,
          Rin  => enrs,
      clk  => clk
      );

roundgen : round
    port map (
          mc => mc,
      bytes_in => round_out_r,--
      key_in => round_key_r,--
          bytes_out => round_out,
          reset      => reset,
      clk       => clk
          );

end rtl;
```

ANEXO 4

TARJETA DE DESARROLLO SPARTAN 3E