ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

ESTUDIO, SIMULACIÓN E IMPLEMENTACIÓN DE ESTÁNDARES DE CIFRADO SIMÉTRICO, UTILIZANDO MATLAB, VHDL Y FPGA

PROYECTO PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN ELECTRÓNICA Y REDES DE INFORMACIÓN

GRANDA GALARZA MARÍA FERNANDA
ferchapisca12@gmail.com

REINA PÉREZ VERLAINE LEANDRO
leopiscofer@gmail.com

DIRECTOR: DR. ROBIN ÁLVAREZ RUEDA.
robin.alvarez@epn.edu.ec

Quito, Diciembre 2012

DECLARACIÓN

Nosotros, María Fernanda Granda Galarza y Verlaine Leandro Reina Pérez, declaramos bajo juramento que el trabajo aquí descrito es de nuestra autoría; que no ha sido previamente presentada para ningún grado o calificación profesional; y, que hemos consultado las referencias bibliográficas que se incluyen en este documento.

A través de la presente declaración cedemos nuestros derechos de propiedad intelectual correspondientes a este trabajo, a la Escuela Politécnica Nacional, según lo establecido por la Ley de Propiedad Intelectual, por su Reglamento y por la normatividad institucional vigente.

_____	_____
María Fernanda Granda Galarza	Verlaine Leandro Reina Pérez

CERTIFICACIÓN

Certifico que el presente trabajo fue desarrollado por María Fernanda Granda Galarza y Verlaine Leandro Reina Pérez, bajo mi supervisión.

_____

Ing. Robin Álvarez Rueda, MSc, PhD
DIRECTOR DEL PROYECTO

AGRADECIMIENTO

A mis padres, hermanos y sobrino por el apoyo incondicional en cada momento de mi vida.

A mi compañero de Tesis, por todo este tiempo juntos.

A nuestro Director de Proyecto: Dr. Robín Álvarez, por creer en este trabajo.

María Fernanda.

AGRADECIMIENTO

A mi madre, mis abuelitos y tíos, por estar siempre en los momentos más difíciles.

A mi compañera de tesis, por toda la dedicación y empeño para sacar adelante el proyecto.

Al PhD Robín Álvarez, por brindarnos su apoyo incondicional.

A mis amigos, por todos los momentos compartidos en la Escuela Politécnica Nacional.

Leandro.

DEDICATORIA

A mis padres, hermanos y amigos por todo el apoyo.

María Fernanda.

DEDICATORIA

Por tu ternura y valentía.
A mi madre.

Leandro.

ÍNDICE DE CONTENIDOS

CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

RESUMEN

La necesidad de comunicarnos de forma segura por medios digitales, requiere del desarrollo de aplicaciones computacionalmente seguras, las mismas que están basadas en estándares y algoritmos matemáticos. De todos estos, el estándar AES es quizá uno de los más seguros y más usados en una amplia gama de aplicaciones.

Si bien la utilización de dichos algoritmos se lo hace diariamente, este trabajo aporta con un estudio detallado de los estándares de cifrado simétrico, DES y AES y su simulación en MatLab, exponiéndolos de forma didáctica. Adicionalmente se realiza la implementación en hardware del cifrado S-DES y AES: la programación de los algoritmos se la realiza empleando el lenguaje de descripción de hardware VHDL dentro del software de desarrollo ISE Project Navigator 13.1, y su implementación en una tarjeta de entrenamiento XUPV5-LX110T, que posee un chip FPGA Xilinx Virtex-5 XC5VLX110T de Xilinx. El estándar AES es implementado con una llave y texto plano, ambos de 128 bits. Estos datos son enviados a la tarjeta de entrenamiento, desde una interfaz gráfica usando MatLab a través de la interfaz serial. Los resultados obtenidos tanto de la simulación en MatLab como de su implementación en el FPGA han sido verificados empleando aplicaciones comerciales.

PRESENTACIÓN

Este proyecto presenta el estudio, simulación de S-DES, DES y AES, usando MatLab, así como los procesos de descripción VHDL, síntesis e implementación en un FPGA, del cifrado S-DES y el estándar AES.

El desarrollo de este proyecto se divide en cinco capítulos:

Capítulo 1: Introducción. Este capítulo comienza presentando una breve descripción de los estándares de cifrado desde los tradicionales hasta los modernos. El estudio se realiza en orden cronológico al desarrollo y evolución de los estándares de cifrado, así como una descripción didáctica de los estándares más representativos.

Capítulo 2: S-DES y DES. Se expone inicialmente el cifrado S-DES, de una forma introductoria, realizando cada una de las etapas de cifrado y desarrollando paso a paso un ejemplo, adicionalmente se realiza el diseño e implementación de este cifrado en el dispositivo FPGA. Continuando con el desarrollo del capítulo, se realiza el estudio del estándar DES y se detalla el proceso para la implementación en MatLab.

Capítulo 3: AES. El capítulo inicia con el estudio del estándar AES y su desarrollo en MatLab, es importante indicar que el fundamento matemático para este estudio se detalla en los anexos 9 y 10; a continuación se explica el diseño, para la implementación en hardware del estándar, así como el proceso de síntesis. Por último se realizan las pruebas de comprobación de resultados.

Capítulo 4: PRUEBAS Y RESULTADOS. Se expone inicialmente el resultado de un ejemplo práctico para la implementación del cifrado S-DES en MatLab y luego se desarrolla el mismo ejemplo para la implementación en el dispositivo FPGA; a continuación se expone un ejemplo para el cifrado DES en MatLab y la respectiva comprobación de resultados. Por último se realiza el mismo procedimiento para el estándar AES en MatLab y VHDL.

Capítulo 5: Conclusiones y recomendaciones. Se expone las conclusiones y recomendaciones obtenidas durante el desarrollo del proyecto.

Como un soporte adicional, al final del documento, se presentan varios anexos: estos son esenciales para facilitar la comprensión de los capítulos tratados anteriormente.

CAPÍTULO 1
INTRODUCCIÓN

1.1 CONCEPTOS

Esta sección define los conceptos básicos de cifrado:

- **Texto plano:** es el mensaje o los datos originales.
- **Texto cifrado:** es el mensaje ilegible, depende del texto plano y la llave.
- **Criptografía:** la palabra criptografía, se deriva de dos vocablos griegos: "kryptos" (ocultar) y "graphia" (escribir). Esta técnica consiste en transformar un texto original, dato o texto plano, en una forma de texto ilegible o texto cifrado, con el fin de proteger el contenido del texto plano, de manera que solo las partes involucradas poseedoras de la llave, conozcan el contenido original.
- **Criptoanálisis:** es la técnica, mediante la cual se trata de descubrir el texto plano, sin conocer la llave utilizada en el proceso de cifrado, esto se consigue analizando las vulnerabilidades en el método de cifrado.

1.2 CLASIFICACIÓN DE CRIPTOSISTEMAS

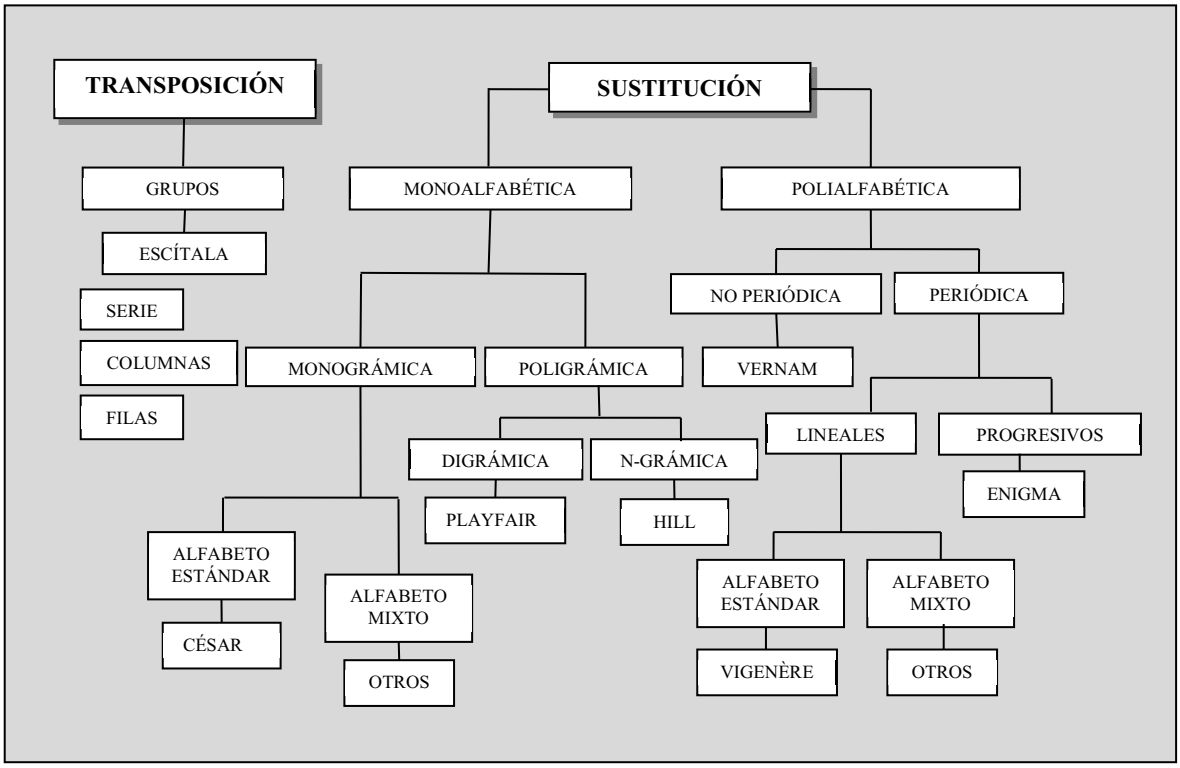
Existen varias formas de clasificar los criptosistemas, entre ellas por el tratamiento de la información o por el tipo de llave, para este estudio lo realizaremos con relación a la historia de la criptografía en:

- Sistemas de cifrado clásico.
- Sistemas de cifrado moderno.

Se realizará una descripción breve de los algoritmos más representativos.

1.2.1 SISTEMAS DE CIFRADO CLÁSICO

Los métodos de cifrado clásico, utilizan técnicas como: la sustitución, permutación o desplazamiento de símbolos. En la figura 1-1, se muestra una clasificación detallada de esta técnica de cifrado.



Fuente [1.1]

Figura 1-1. Sistemas de cifrado clásico.

Las técnicas de cifrado clásico fueron descartadas ya que se encontraron vulnerabilidades por medio del criptoanálisis.

1.2.1.1 Transposición

La transposición consiste en cambiar el orden de los datos del texto plano. Los cifradores por transposición se desarrollaron en la antigüedad, siendo el más conocido el desarrollado por los espartanos:

- Escítala.

1.2.1.2 Sustitución

En los cifradores por sustitución los elementos del texto plano, son cambiados por otros elementos. De la clasificación que se observa en la figura 1-1 para el método de sustitución se explicará el funcionamiento los siguientes tipos de cifrado:

- Cifrado monoalfabético → César
- Cifrado polialfabético → Vernam y Vigenère

1.2.1.2.1 Cifrado monoalfabético

Realiza permutaciones de los componentes del alfabeto, por lo que a un componente del alfabeto le corresponderá uno y solo un componente del mismo alfabeto esto según la regla de desplazamiento empleada, para el caso del castellano se emplearán las 27 letras del mismo. Uno de los más comunes es el cifrado César.

La debilidad de este tipo de cifrados, es la frecuencia con la que ciertas letras de distintos alfabetos se repiten en un texto. Para el caso del español las letras más frecuentes son la E y A como se observa en la figura 1-2, mientras que en el inglés son las E y la T indicado en la figura 1-3.

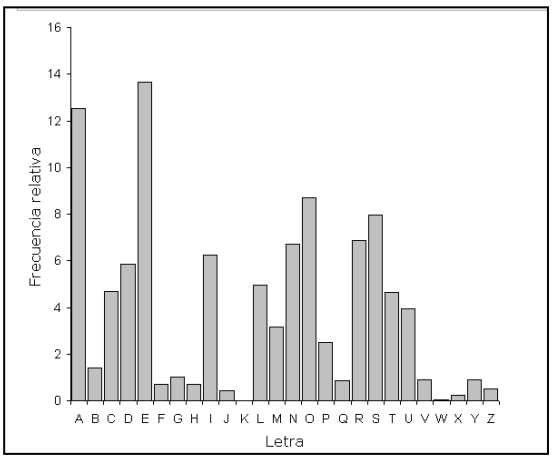


Figura 1-2. Distribución de las letras en un texto común en español. Fuente [1.2]

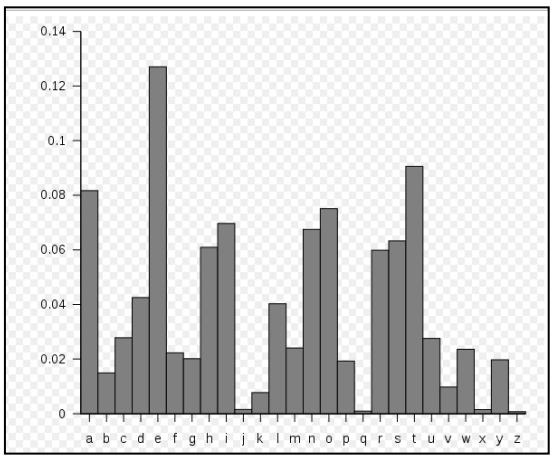


Figura 1-3. Distribución de las letras en un texto común en inglés. Fuente [1.2]

Ejemplo de Distribución de letras de un texto en español

Se selecciona un texto en español, cuyas características se especifican en la tabla 1-1.

Características de texto en español	
Nombre del libro	El mundo y sus demonios
Autor	Carl Sagan
Página	43-44
Total de letras	2402

Tabla 1-1. Tabla de datos: texto español.

El texto a ser analizado se muestra en la figura1-4. Las letras mayúsculas y minúsculas serán tomadas en cuenta como una misma letra, como ejemplo si se tiene: “A” y “a”, se contará como 2 valores a la letra “a”.

Yo fui niño en una época de esperanza. Quise ser científico desde mis primeros días de escuela. El momento en que cristalizó mi deseo llegó cuando capté por primera vez que las estrellas eran soles poderosos, cuando constaté lo increíblemente lejos que debían de estar para aparecer como simples puntos de luz en el cielo. No estoy seguro de que entonces supiera siquiera el significado de la palabra «ciencia», pero de alguna manera quería sumergirme en toda su grandeza. Me llamaba la atención el esplendor del universo, me fascinaba la perspectiva de comprender cómo funcionan realmente las cosas, de ayudar a descubrir misterios profundos, de explorar nuevos mundos... quizá incluso literalmente. He tenido la suerte de haber podido realizar este sueño al menos en parte. Para mí, el romanticismo de la ciencia sigue siendo tan atractivo y nuevo como lo fuera aquel día, hace más de medio siglo, que me enseñaron las maravillas de la Feria Mundial de 1939.
Popularizar la ciencia —intentar hacer accesibles sus métodos y descubrimientos a los no científicos— es algo que viene a continuación, de manera natural e inmediata. No explicar la ciencia me parece perverso. Cuando uno se enamora, quiere contarlo al mundo. Este libro es una declaración personal que refleja mi relación de amor de toda la vida con la ciencia.

Pero hay otra razón: la ciencia es más que un cuerpo de conocimiento, es una manera de pensar. Preveo cómo será la América de la época de mis hijos o nietos: Estados Unidos será una economía de servicio e información; casi todas las industrias manufactureras clave se habrán desplazado a otros países; los temibles poderes tecnológicos estarán en manos de unos pocos y nadie que represente el interés público se podrá acercar siquiera a los asuntos importantes; la gente habrá perdido la capacidad de establecer sus prioridades o de cuestionar con conocimiento a los que ejercen la autoridad; nosotros, aferrados a nuestros cristales y consultando nerviosos nuestros horóscopos, con las facultades críticas en declive, incapaces de discernir entre lo que nos hace sentir bien y lo que es cierto, nos iremos deslizando, casi sin darnos cuenta, en la superstición y la oscuridad.

La caída en la estupidez de Norteamérica se hace evidente principalmente en la lenta decadencia del contenido de los medios de comunicación, de enorme influencia, las cuñas de sonido de treinta segundos (ahora reducidas a diez o menos), la programación de nivel ínfimo, las crédulas presentaciones de pseudociencia y superstición, pero sobre todo en una especie de celebración de la ignorancia. En estos momentos, la película en vídeo que más se alquila en Estados Unidos es Dumb and Dumber. Beavis y Buttheadi siguen siendo populares (e influyentes) entre los jóvenes espectadores de televisión. La moraleja más clara es que el estudio y el conocimiento —no sólo de la ciencia, sino de cualquier cosa— son prescindibles, incluso indeseables.

Fuente [1.3]

Figura 1- 4. Texto en español. Libro: El mundo y sus demonios, pags: 43-44.

Los resultados de la distribución de letras del texto en español se muestran en la figura 1-5.

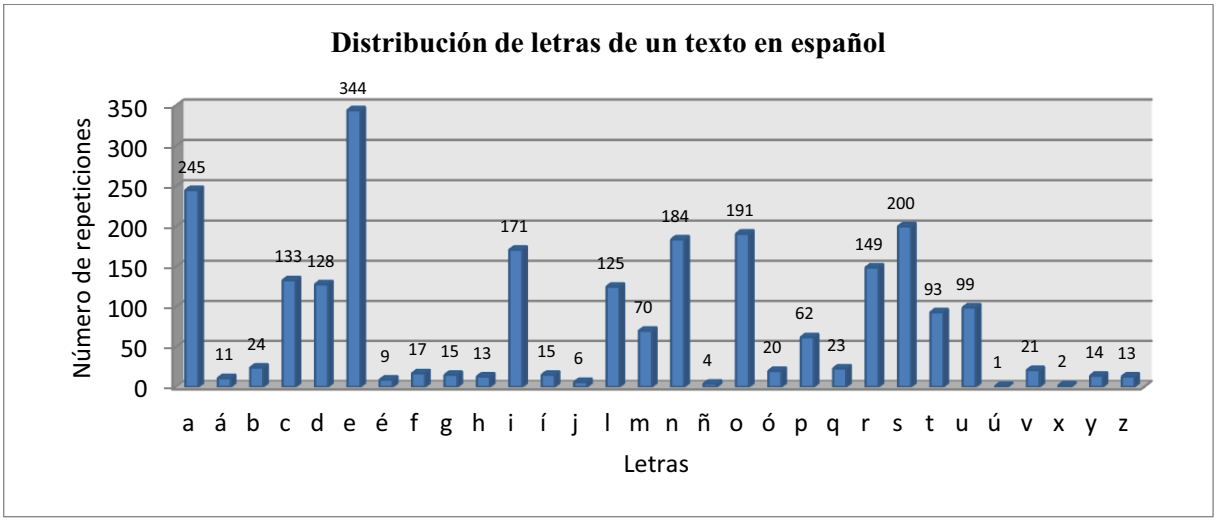


Figura 1-5. Resultado de distribución de letras de un texto en español.

Ejemplo de distribución de letras de un texto en inglés

Se selecciona un texto en inglés, cuyas características se especifican en la tabla 1-2.

Características del texto en inglés	
Nombre del libro	The Demon-Haunted Worls
Autor	Carl Sagan
Página	27-28
Total de letras	2070

Tabla 1-2. Tabla de datos de un texto en inglés.

El texto a ser analizado se observa en la figura1-6.

I was a child in a time of hope. I wanted to be a scientist from my earliest school days. The crystallizing moment came when I first caught on that the stars are mighty suns, when it first dawned on me how staggeringly far away they must be to appear as mere points of light in the sky. I'm not sure I even knew the meaning of the word 'science' then, but I wanted somehow to immerse myself in all that grandeur. I was gripped by the splendour of the Universe, transfixed by the prospect of understanding how things really work, of helping to uncover deep mysteries, of exploring new worlds - maybe even literally. It has been my good fortune to have had that dream in part fulfilled. For me, the romance of science remains as appealing and new as it was on that day, more than half a century ago, when I was shown the wonders of the 1939 World's Fair. Popularizing science - trying to make its methods and findings accessible to non-scientists - then follows naturally andimmediately. Not explaining science seems to me perverse. When you're in love, you want to tell the world. This book is a personal statement, reflecting my lifelong love affair with science. But there's another reason: science is more than a body of knowledge; it is a way of thinking. I have a foreboding of an America in my children's or grandchildren's time - when the United States is a service and information economy; when nearly all the key manufacturing industries have slipped away to other countries; when awesome technological powers are in the hands of a very few, and no one representing the public interest can even grasp the issues; when the people have lost the ability to set their own agendas or knowledgeably question those in authority; when, clutching our crystals and nervously consulting our horoscopes, our critical faculties in decline, unable to distinguish between what feels good and what's true, we slide, almost without noticing, back into superstition and darkness. The dumbing down of America is most evident in the slow decay of substantive content in the enormously influential media, the 30-second sound bites (now down to 10 seconds or less), lowest common denominator programming, credulous presentations on pseudoscience and superstition, but especially a kind of celebration of ignorance. As I write, the number one video cassette rental in America is the movie Dumb and Dumber. Beavis and Butthead remains popular (and influential) with young TV viewers. The plain lesson is that study and learning - not just of science, but of anything - are avoidable, even undesirable.

Fuente [1.4]

Figura 1-6. Texto en inglés. Libro: The Demon-Haunted Worls, pags: 27-28.

Los resultados de distribución de letras en inglés, se indican en la figura 1-7.

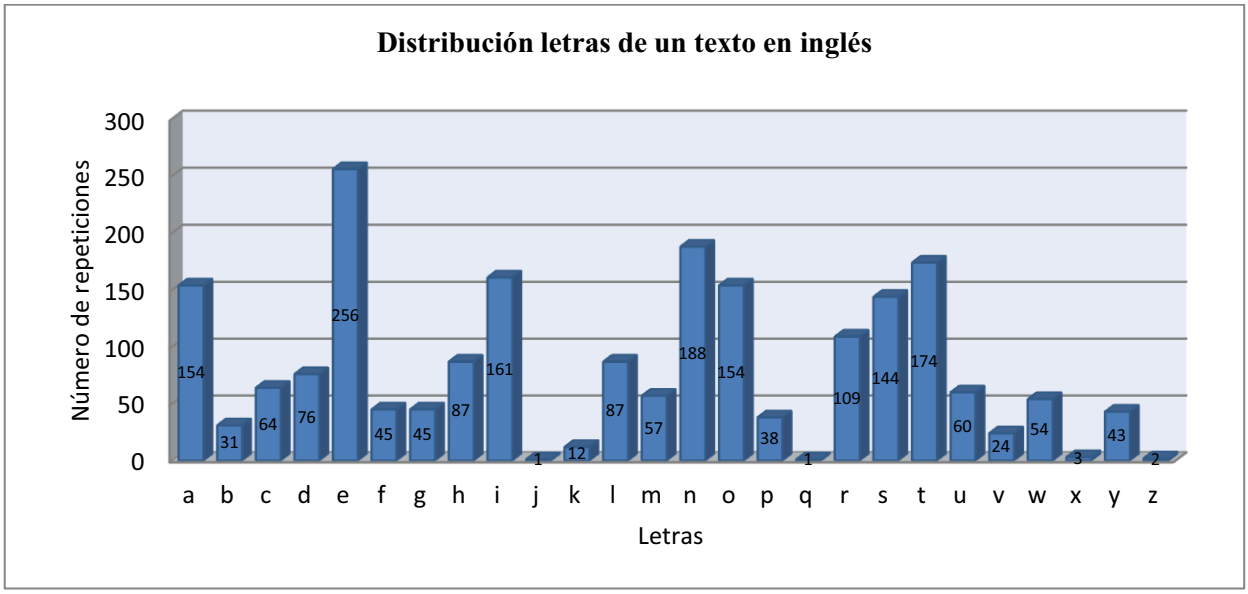


Figura 1-7. Resultado de distribución de letras en inglés.

CIFRADO CÉSAR

Consiste en desplazar las letras del alfabeto tres posiciones hacia adelante como se puede observar en la figura 1-8, a la letra **F,** le corresponderá únicamente la letra **I**, a la **G** la letra **J**, así sucesivamente.

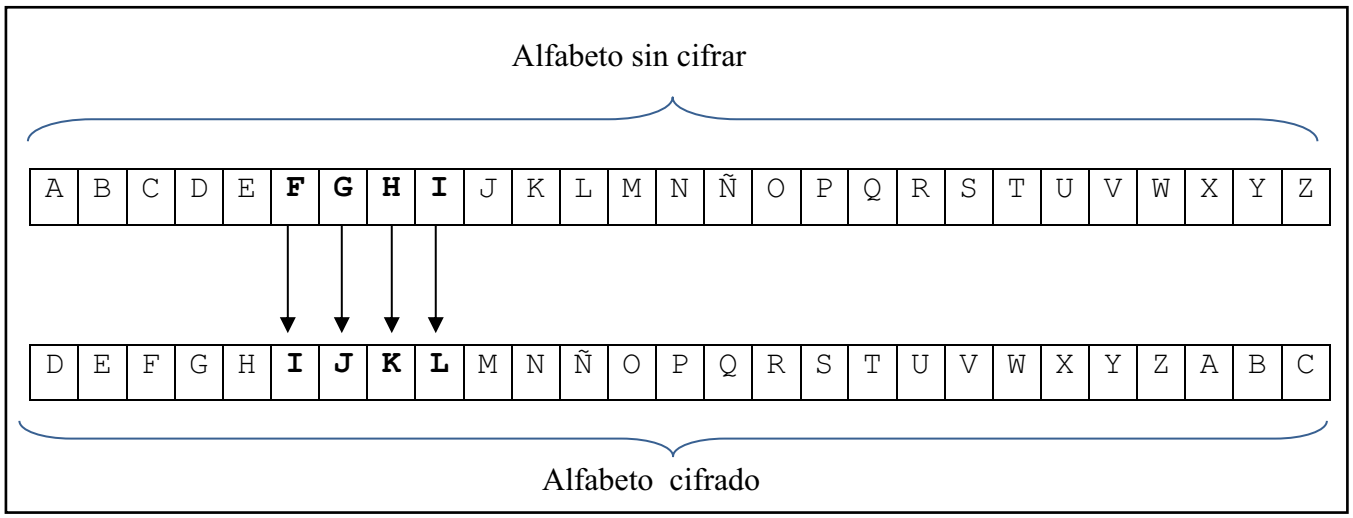


Figura 1-8. Alfabeto cifrado César.

Criptoanálisis cifrado César.

- Utilizando la figura 1-9, podemos cifrar la siguiente frase: “HOLA MUNDO”.

Texto plano	H	O	L	A	M	U	N	D	O
Llave	3 desplazamientos a la derecha								
Texto cifrado	K	R	Ñ	D	O	X	P	G	R

Figura 1-9. Ejemplo cifrado César.

- Si se considera al texto cifrado como C, al texto plano como P, al proceso de cifrado como E(P) y al proceso de descifrado como D(C), se tiene:

$$\text{Texto cifrado } C = E(P) = (P + 3) \bmod 27$$

Donde mod 27 hace referencia al alfabeto castellano de 27 letras.

- El proceso de descifrado sería simplemente desplazar esas tres posiciones hacia la izquierda:

$$\text{Texto plano } P = D(C) = (C - 3) \bmod 27$$

- Generalizando se tiene:

$$C = E(P) = (P + k) \bmod 27$$
$$P = D(C) = (C - k) \bmod 27$$

Donde **k** representa la llave del descifrado es decir el número de posiciones desplazadas que para el ejemplo se puede tomar valores entre 1 y 26. Por lo tanto para conocer el texto plano sin conocer la llave **k** lo que se debe hacer es ensayar de 1 a 26 posibilidades que tiene la llave **k**, como se observa en la figura 1-10.

Texto cifrado	K	R	Ñ	D	O	X	P	G	R
k = 1	J	Q	N	C	Ñ	W	O	F	Q
k = 2	I	P	M	B	N	V	Ñ	E	P
k = 3	H	O	L	A	M	U	N	D	O

Figura 1-10. Resultado criptoanálisis cifrado César.

A este tipo de ataques se conoce como fuerza bruta, se ensaya posibilidades de llave hasta dar con el texto plano.

1.2.1.2.2 Cifrado polialfabético

A diferencia del cifrado monoalfabético, en el cifrado polialfabético se pueden usar para cifrar a más del alfabeto, números o símbolos. Una letra del texto plano puede ser sustituida por diferentes letras, números o símbolos. Como se puede observar en la figura 1-11, la letra **F**, puede ser sustituida por la letra **X**, el número **6**, o el símbolo **π**, dependiendo de la técnica de cifrado usada.

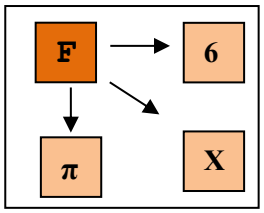


Figura 1-11. Cifrado polialfabético.

CIFRADO VERNAM

Desarrollado por Gilbert Vernam para AT&T en 1917, consiste en realizar la operación XOR, entre el texto plano y la llave. La representación del texto plano y la llave se toma en base al código Baudot, cuyos valores se indican en la tabla 1-3.

Los códigos Baudot representan los carácteres del lenguaje con cinco elementos que pueden ser el espacio o la marca (el cero y el uno) diseñado para transmisiones telegráficas. [1.1]

Código Binario	Carácter	Código Binario	Carácter	Código Binario	Carácter	Código Binario	Carácter
00000	Blanco	01000	<	10000	T	11000	O
00001	E	01001	D	10001	Z	11001	B
00010	=	01010	R	10010	L	11010	G
00011	A	01011	J	10011	W	11011	↑
00100	Espacio	01100	N	10100	H	11100	M
00101	S	01101	F	10101	Y	11101	X
00110	I	01110	C	10110	P	11110	V
00111	U	01111	K	10111	Q	11111	↓

Tabla 1-3. Código Baudot. [1.1]

Cifrado

En primer lugar se debe sustituir cada carácter del texto plano y la llave, como una representación del código baudot, es decir cada carácter del texto plano y la llave, serán representados por cinco bits.

A cada carácter del texto plano, sustituído por su equivalente en el código Baudot, se denominará **Mi**.

A cada carácter de la llave, sustituído por su equivalente en el código Baudot, se denominará **Ki.**

Si se considera al texto plano como P, al texto cifrado como C, al proceso de cifrado E(P) y al proceso de descifrado como D(C), se tiene:

$$\text{Texto cifrado } C = E(P) = (Mi + Ki) \bmod 2 \quad \text{para } i = 1,2\ldots..N$$

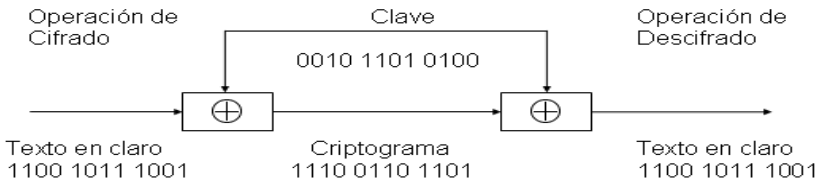
$$\therefore \text{Texto cifrado } C = Mi \oplus Ki$$

Descifrado

El proceso de descifrado, se realiza con la operación XOR, entre el texto cifrado y la llave:

$$\text{Texto plano } P = D(C) = C_i \oplus K_i = (M_i \oplus Ki) \oplus Ki$$

Una representación gráfica se ilustra en la figura 1-12.



[1.1]

Figura 1-12. Representación gráfica del cifrado Vernam.

El cifrado Vernam es la base fundamental para los algoritmos simétricos, ya que estos dependen del conocimiento de una llave y solo aquel que la conozca, podrá descifrar el mensaje.

Ejemplo de cifrado Vernam

- Texto plano: HOLA MUNDO
- Llave: SOL
- El proceso de cifrado se observa en la tabla 1-4.

Texto plano ⊕ Llave	Operación Xor	Resultado Xor	Texto cifrado
H ⊕ S	10100 ⊕ 00101	10001	Z
O ⊕ O	11000 ⊕ 11000	00000	Blanco
L ⊕ L	10010 ⊕ 10010	00000	Blanco
A ⊕ S	00011 ⊕ 00101	00110	I
Espacio⊕O	00100 ⊕ 11000	11100	M
M ⊕ L	11100 ⊕ 10010	01110	N
U ⊕ S	00111 ⊕ 00101	00010	=
N ⊕ O	01100 ⊕ 11000	10100	H
D ⊕ L	01001 ⊕ 10010	11011	↑
O ⊕ S	11000 ⊕ 00101	11101	X

Tabla 1-4. Ejemplo cifrado Vernam.

Ejemplo descifrado Vernam

- Texto cifrado: Z I M N=H↑X

- Llave: SOL
- El proceso de descifrado se observa en la tabla 1-5.

Texto cifrado ⊕ llave	Operación Xor	Resultado Xor	Texto plano
Z ⊕ S	10001 ⊕ 00101	10100	H
Blanco ⊕ O	00000 ⊕ 11000	11000	O
Blanco⊕ L	00000 ⊕ 10010	10010	L
I ⊕ S	00110 ⊕ 00101	00011	A
M ⊕ O	11100 ⊕ 11000	00100	Espacio
N ⊕ L	01110 ⊕ 10010	11100	M
= ⊕ S	00010 ⊕ 00101	00111	U
H ⊕ O	10100 ⊕ 11000	01100	N
↑ ⊕ L	11011 ⊕ 10010	01001	D
X ⊕ S	11101 ⊕ 00101	11000	O

Tabla 1-5. Ejemplo descifrado Vernam.

CIFRADO VIGENÈRE

El cifrado Vigenère, es un ejemplo de cifrado polialfabético de sustitución, está basado en la tabla 1-6.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

Tabla 1-6. Tabla de Vigenère. [1.5]

Funcionamiento del cifrado Vigenère

- La figura 1-13, está formada por una serie de desplazamientos tipo cifrado César.
- Se toma un texto plano, por ejemplo: HOLA MUNDO.
- Y una llave: SOL.
- Se hace corresponder a cada letra del texto plano, las letras de la llave, las veces que fueran necesarias, hasta completar el texto plano como se observa a continuación:

Texto plano	H	O	L	A	M	U	N	D	O
Llave	S	O	L	S	O	L	S	O	L

Figura 1-13. Ejemplo cifrado Vigenère.

- Las letras correspondientes a la llave corresponden a las filas de la tabla 1-6, y las letras del texto plano corresponden a las columnas de la misma tabla. Para cifrar el texto hay que encontrar la letra que corresponde tanto a la llave como al texto plano, como indica la figura 1-14.

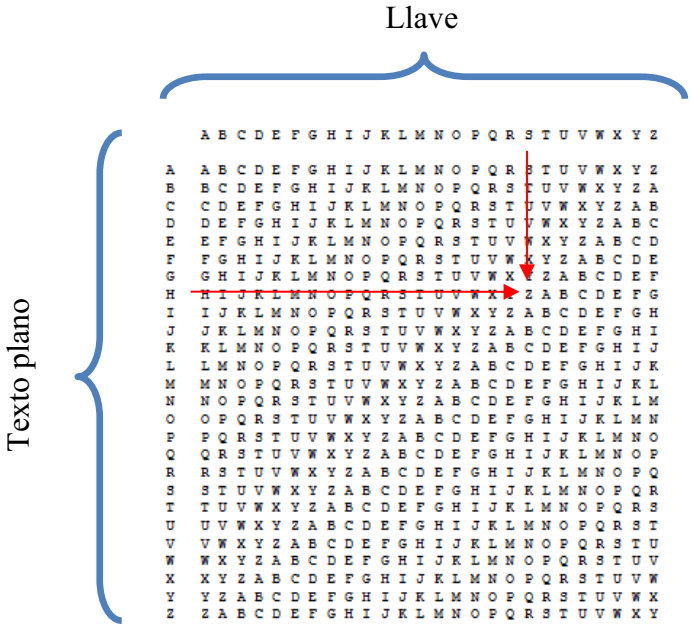


Figura 1-14. Funcionamiento cifrado Vigenère.

- En el ejemplo el resultado sería el siguiente:

Texto plano	H	**O**	L	A	M	U	N	D	**O**
Llave	S	O	L	S	O	L	S	O	L
Texto cifrado	Z	**C**	W	S	A	F	F	R	**Z**

Figura 1-15. Resultado cifrado Vigenère.

- Como se observa en el resultado del texto cifrado, a la letra **O,** del texto plano, se le asignó la letra **C** y luego la **Z,** característica de los cifrados polialfabéticos.
- Para el proceso de descifrado se recurre al mismo método de cifrado, la llave es la misma y ahora el texto cifrado es el texto plano, representado en la figura 1-16.

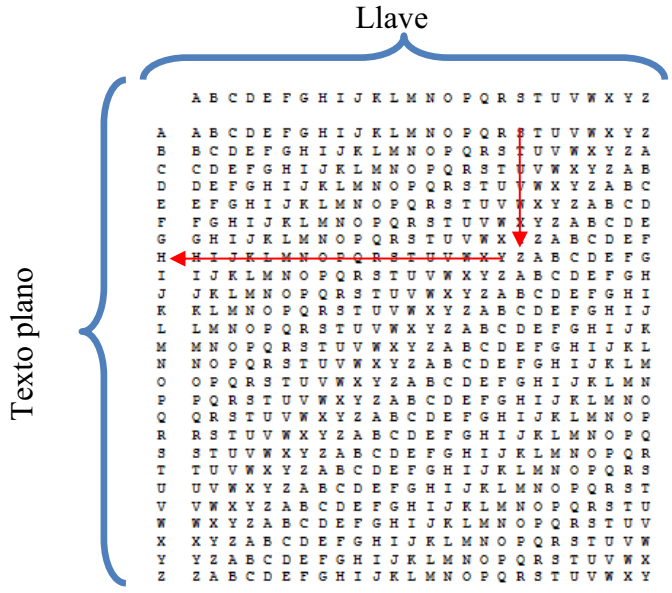


Figura 1-16. Funcionamiento descifrado Vigenère.

- El resultado se muestra a continuación:

Texto plano	Z	C	W	S	A	F	F	R	Z
Llave	S	O	L	S	O	L	S	O	L
Texto cifrado	H	O	L	A	M	U	N	D	O

Figura 1-17. Ejemplo de descifrado Vigenère.

1.2.2 SISTEMAS DE CIFRADO MODERNO

Ya en épocas modernas se desarrollaron máquinas de cifrado, mucho más eficientes, pero con el desarrollo de dichas máquinas también aparecieron criptoanalistas, (personas que practican el criptoanálisis), capaces de leer el texto claro sin necesidad de conocer la llave. Por el año de 1948 Claude Shannon es el que estableció las bases teóricas para la criptología, que es el conjunto de la criptografía y el criptoanálisis.

Con las bases, desarrolladas por Shannon y con el desarrollo de las computadoras y el mundo digital, los gobiernos, empresas importantes vieron la necesidad de desarrollar algoritmos de cifrado más confiables, creando el estándar DES para cifrar mensajes de texto, archivos de un disco duro, registros de una base de datos y en general para cifrar grandes cantidades de datos.

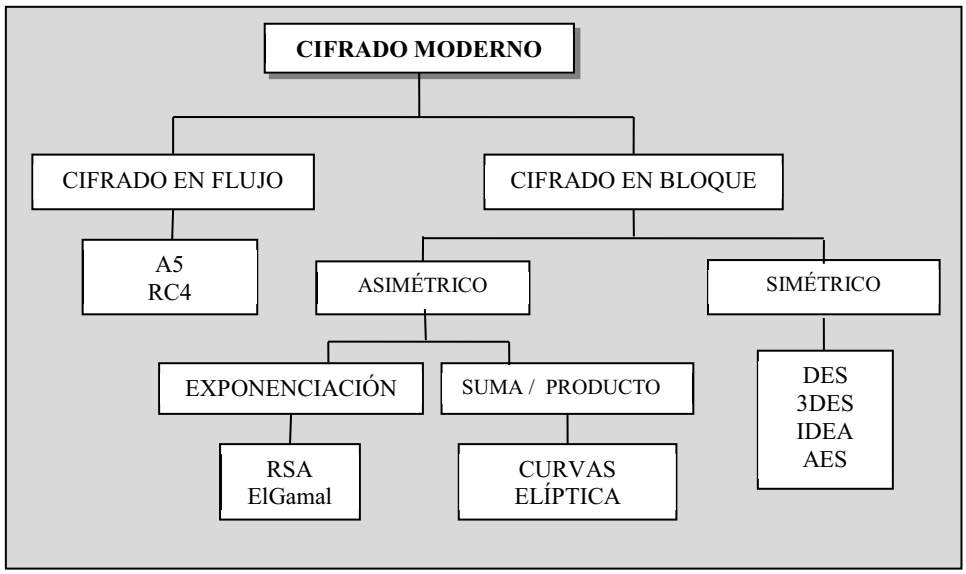
El estudio del cifrado moderno se realizará en base a la figura 1-18, y haciendo referencia a los métodos más utilizados.

1.2.2.1 Cifrado en flujo

El cifrado en flujo es un algoritmo, que realiza el cifrado incrementalmente, convirtiendo el texto plano en un texto ilegible. Esto se logra construyendo un generador de flujo de llave.

Un flujo de llave es una secuencia de bits de tamaño arbitrario que puede emplearse para oscurecer los contenidos de un conjunto de datos, combinando el flujo de llave y de datos mediante la función XOR. Si el flujo de llave es seguro, el de datos cifrados también lo será. [1.7]

En este trabajo no se dará más detalles ya que este estudio se basará en el cifrado en bloque.



Fuente [1.1]

Figura 1-18. Sistemas de cifrado moderno.

1.2.2.2 Cifrado en bloque

Un cifrado en bloque, procesa el texto plano en bloques. La longitud del bloque depende de la técnica de cifrado que se utilice.

CIFRADO SIMÉTRICO

En el cifrado simétrico, se utiliza un algoritmo de cifrado con una llave para cifrar el texto plano. Para descifrar el texto cifrado se usa la misma llave de cifrado. A este tipo de cifrado también se lo conoce como cifrado de llave secreta. La seguridad del cifrado simétrico, depende de la privacidad de la llave, mas no de la privacidad del algoritmo de cifrado, el cual es público. Una representación gráfica se indica en la figura 1-19.

Algunos ejemplos de cifrado simétrico son:

- DES (Data Encryption Standard).
- 3DES (Triple Data Encryption Standard).

- AES (Advanced Encryption Standard).

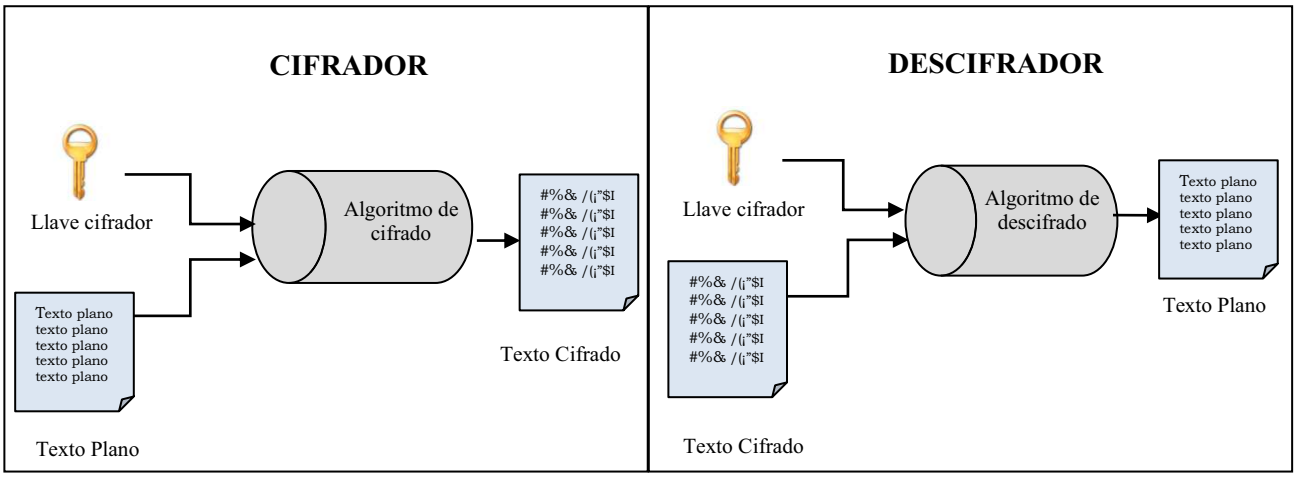


Figura 1-19. Cifrado simétrico.

La mayoría de los algoritmos de cifrado de bloques simétricos se basan en una estructura de cifrado por bloques de Feistel.

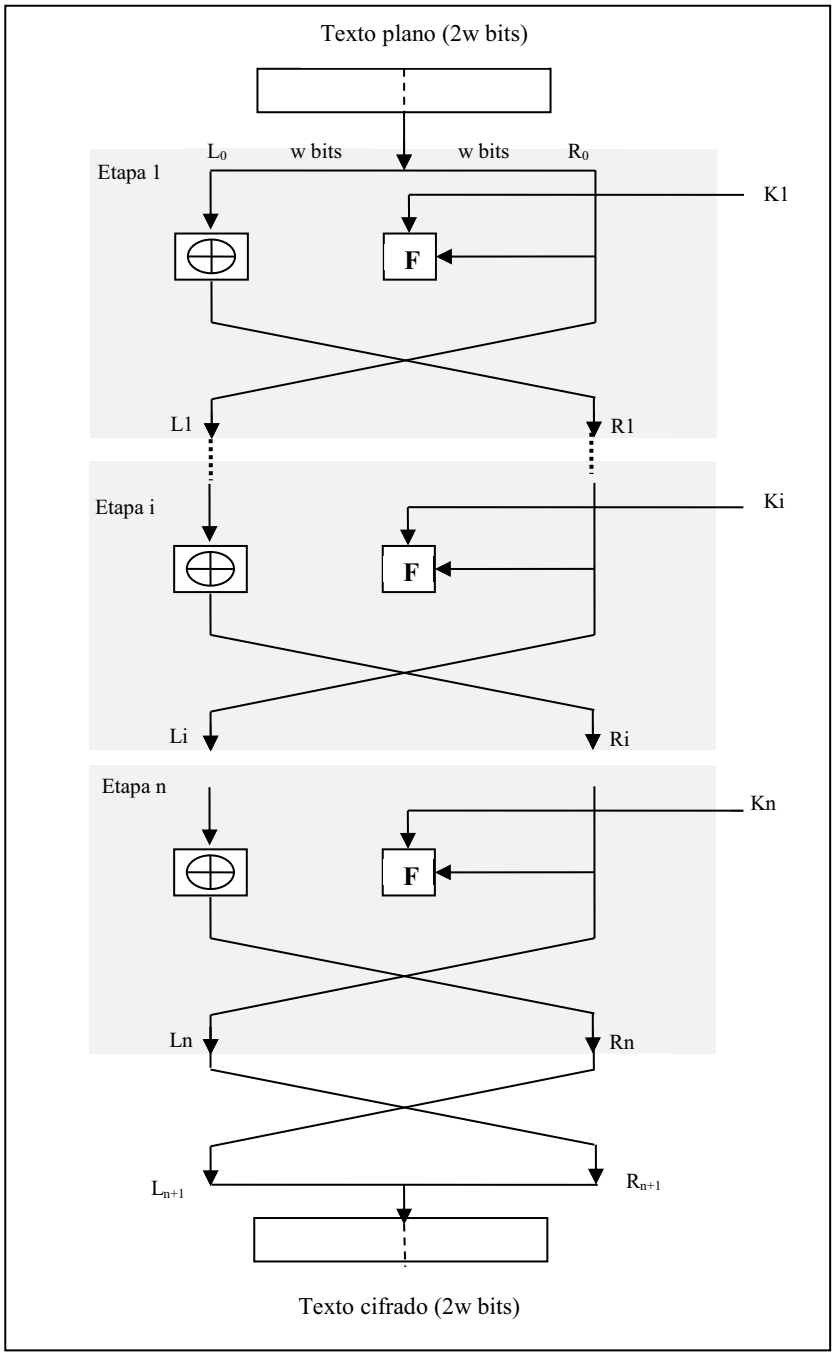
ESTRUCTURA DE FEISTEL

La representación gráfica del cifrado Feistel se muestra en la figura 1-20. Las entradas para el algoritmo de cifrado son:

- Un bloque de texto plano de longitud 2w bits, dividido en dos mitades L_0 y R_0.
- Una llave **K**.

Las dos mitades del texto plano (L_0 y R_0), pasan a través de **n** etapas de procesamiento y, a continuación, se combinan para producir el bloque de texto cifrado.

Cada etapa tiene como entradas Li-1 y Ri-1, derivadas de la etapa anterior, así como una subllave **Ki**, que se deriva de **K**. En general, las subllaves **Ki** son diferentes de **K**.



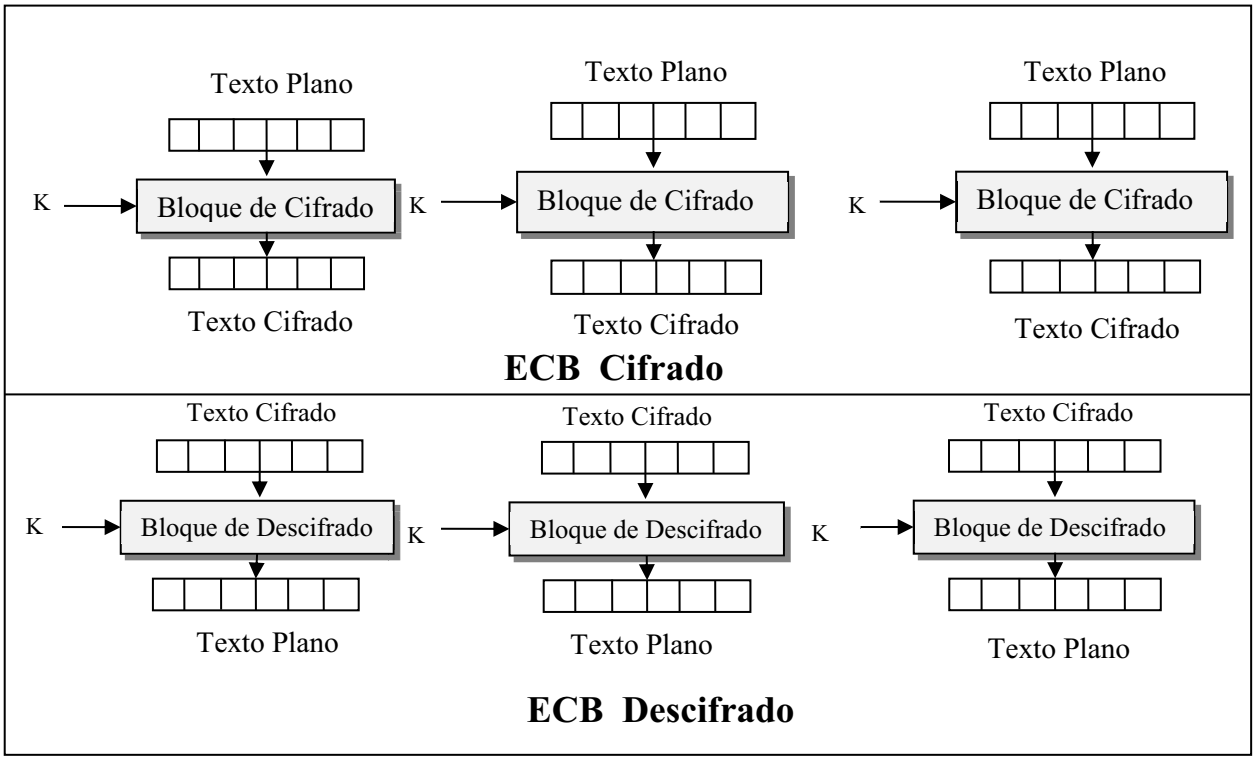
[1.7]

Figura 1-20. Red clásica de Feistel.

Modos de Operación del Cifrado en Bloque

Un cifrado en bloque, procesa un bloque de datos a la vez. En el caso de DES y el 3DES, la longitud del bloque es de 64 bits. Para cantidades mayores de texto plano, es necesario dividir en bloques de 64 bits (rellenando el último bloque si fuera necesario). [1.7]

- **Modo EBC (*Electronic Codebook*):** en el modo EBC los mensajes se dividen en bloques y cada uno de ellos es cifrado por separado utilizando la misma llave K, como se ilustra en la figura 1-21.

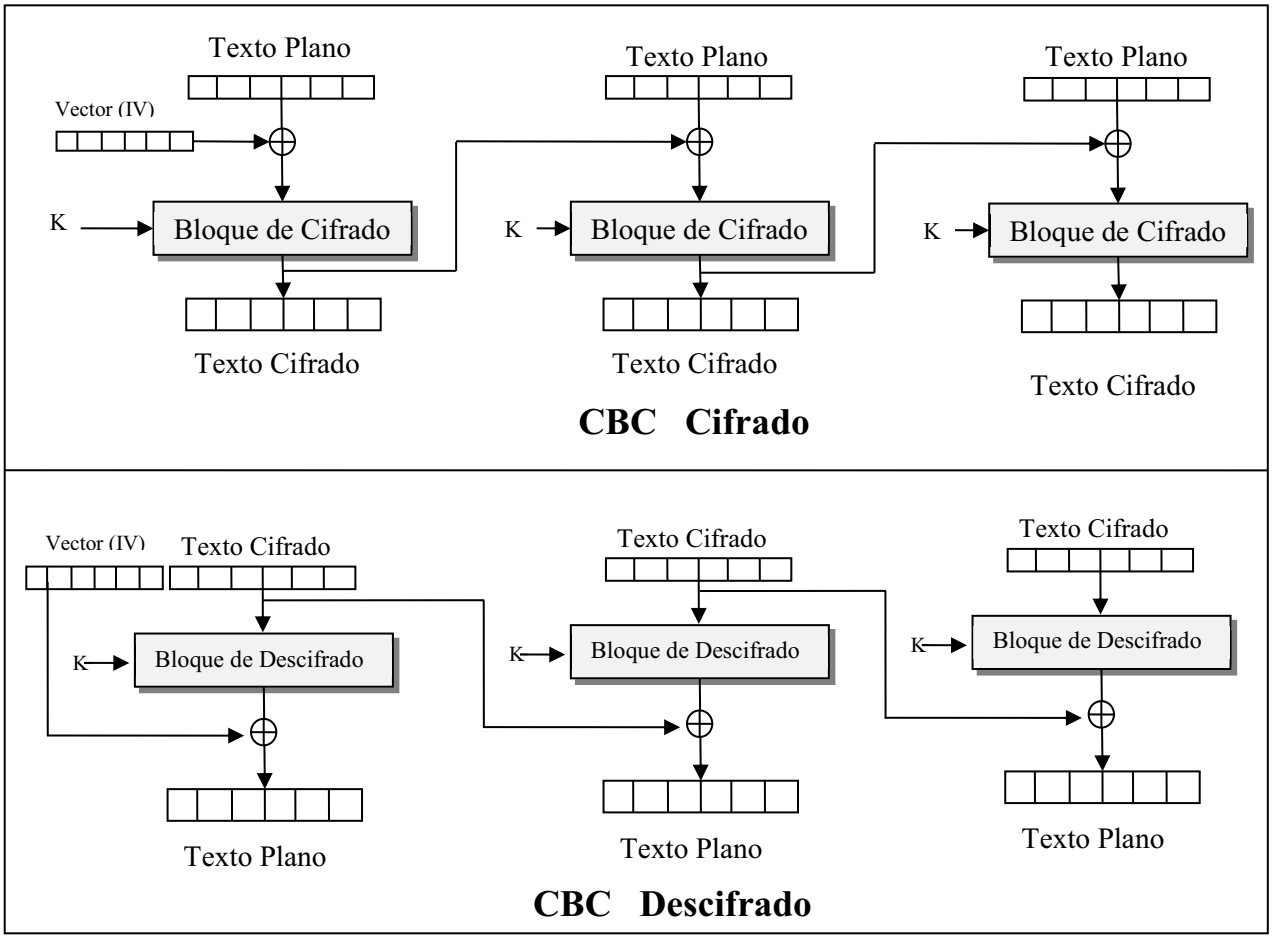


Fuente [1.8]

Figura 1-21. Modo EBC.

- **Modo CBC (*Cipher Block Chaining*):** a diferencia del anterior, en el modo CBC, la entrada al algoritmo de cifrado es el XOR entre el bloque de texto plano actual y el bloque de texto cifrado anterior, se usa la misma clave para cada bloque, como se indica en la figura 1-22. Para descifrar cada bloque de cifrado se pasa por el algoritmo de descifrado. Con el resultado y el bloque de texto cifrado precedente se hace un XOR, para obtener el bloque de texto plano. [1.7]

  El valor del vector IV, debe ser conocido por el emisor y el receptor. Para mayor seguridad este vector debería ser protegido, al igual que la llave. [1.7]



Fuente [1.8]

Figura 1-22. Modo CBC.

Entre otros modos de operación de Cifrado en bloque se puede nombrar:

- Modo CBF (*Cipher feedback*).
- Modo OFB (*Output feedback*).

CIFRADO ASIMÉTRICO

En el cifrado asimétrico se usa un algoritmo de cifrado, y dos llaves, una pública para el proceso de cifrado, y una llave privada para el proceso de descifrado. La llave pública es conocida por todo el mundo, mientras que la privada es conocida solo por el usuario. Una representación gráfica se indica en la figura 1-23.

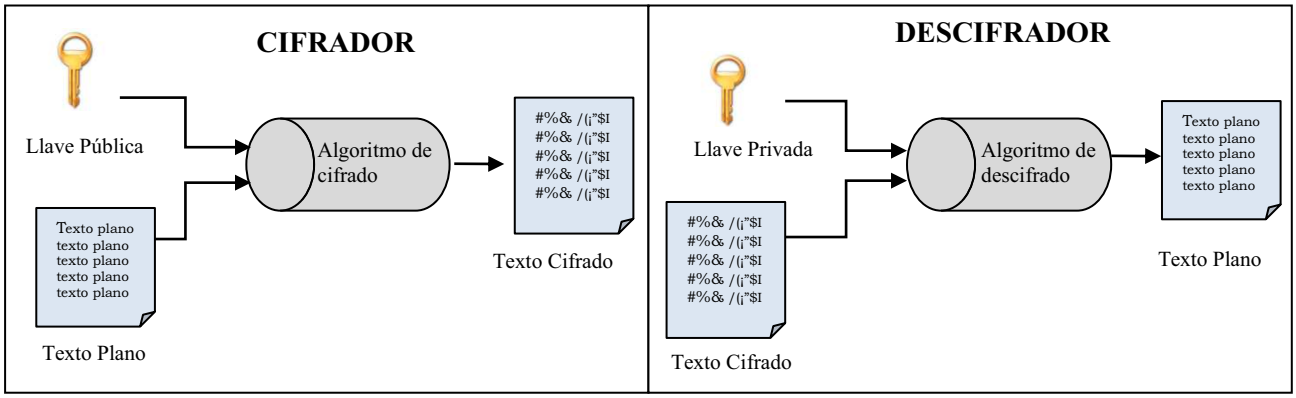


Figura 1-23. Cifrado asimétrico.

REFERENCIAS

[1.1] (2011) Libro Electrónico Seguridad en Redes. [Online]. Disponible: http://www.lpsi.eui.upm.es/SInformatica/diapositivas.htm

[1.2] (2011) Análisis de Frecuencias. [Online]. Disponible: http://es.wikipedia.org/wiki/An%C3%A1lisis_de_frecuencia

[1.3] C. Sagan, *El mundo y sus demonios*. Segunda Edición, 1997.

[1.4] C. Sagan, *The Demon-Haunted Word*. Primera Edición, 1996.

[1.5] (2011) The Vigenère Chiper. [Online]. Disponible: http://www.cs.trincoll.edu/~crypto/historical/vigenere.html

[1.6] (2011) Cifrado en Flujo. [Online]. Disponible: http://es.wikipedia.org/wiki/Cifrador_de_flujo

[1.7] W. Stallings, *Fundamentos de Seguridad en Redes, Aplicaciones y Estándares*. Segunda Edición, Madrid, 2004.

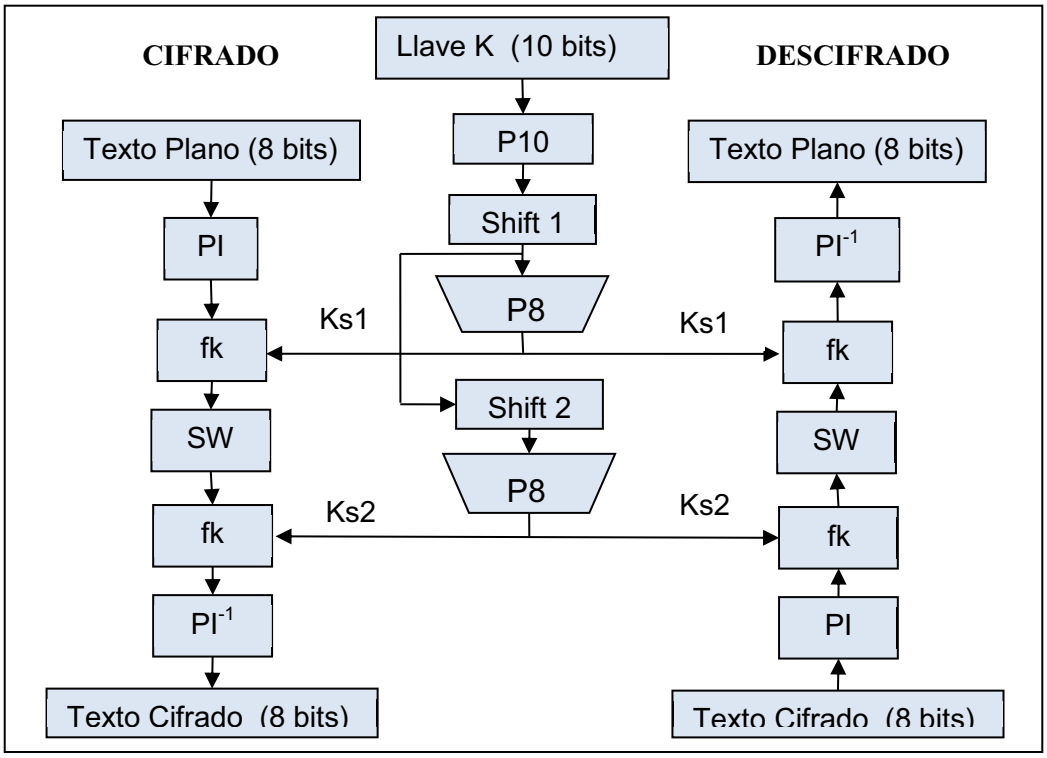
[1.8] (2011) Modos de operación de una unidad de cifrado por bloques. [Online]. Disponible: http://es.wikipedia.org/wiki/Modos_de_operaci%C3%B3n_de_una_unidad_de_cifrado_por_bloques

CAPÍTULO 2

ESTÁNDAR DE CIFRADO DE DATOS (DES) Y ESTÁNDAR SIMPLE DE CIFRADO DE DATOS (S-DES)

2.1 S-DES

S-DES (*Simplified Data Encryption Standard*), es una versión reducida del estándar DES (*Data Encryption Standard*). Tiene propiedades similares a DES pero este abarca un texto plano más reducido. El proceso de cifrado, generación de subllaves y descifrado siguen el esquema de la figura 2-1.



Fuente [2.1]

Figura 2-1. Representación gráfica del cifrado S-DES.

2.1.1 GENERACIÓN DE SUBLLAVES Ks1 Y Ks2

Como se indica en la figura 2-1, la etapa de generación de las subllaves consiste en generar a partir de la llave inicial **K,** dos subllaves. En esta etapa se realizan

operaciones de permutación y desplazamiento. Las operaciones generadoras de la subllaves se las enumeran a continuación:

- Permutación P10
- Shift 1
- Permutación P8
- Shift 2

Si se considera a la llave de 10 bits como **K**, cada bit correspondiente a **K** se representa como:

$$K = K_1, K_2, K_3, K_4, K_5, K_6, K_7, K_8, K_9, K_{10}$$

Ejemplo:

K = 0 1 0 0 0 0 0 1 0 1

2.1.1.1 Permutación P10

Esta permutación debe ser leída de izquierda a derecha, la tabla 2-1, genera las posiciones de salida, así por ejemplo la primera posición de P10 está formada por el tercer bit de la llave.

P10									
3	5	2	7	4	10	1	9	8	6

Tabla 2-1. Permutación P10. [2.1]

La llave **K** se permuta con P10 como se indica a continuación:

Llave permutada = P10 (K)

Llave permutada = P10 ($K_1, K_2, K_3, K_4, K_5, K_6, K_7, K_8, K_9, K_{10}$)

Llave permutada = $K_3, K_5, K_2, K_7, K_4, K_{10}, K_1, K_9, K_8, K_6$

Ejemplo:

Llave permutada = P10 (K)

Llave permutada = P10 (0 1 0 0 0 0 0 1 0 1)

Llave permutada = 0 0 1 0 0 1 0 0 1 0

Diagrama de flujo de la permutación P10

El diagrama de flujo correspondiente a la permutación P10 se muestra en la figura 2-2.

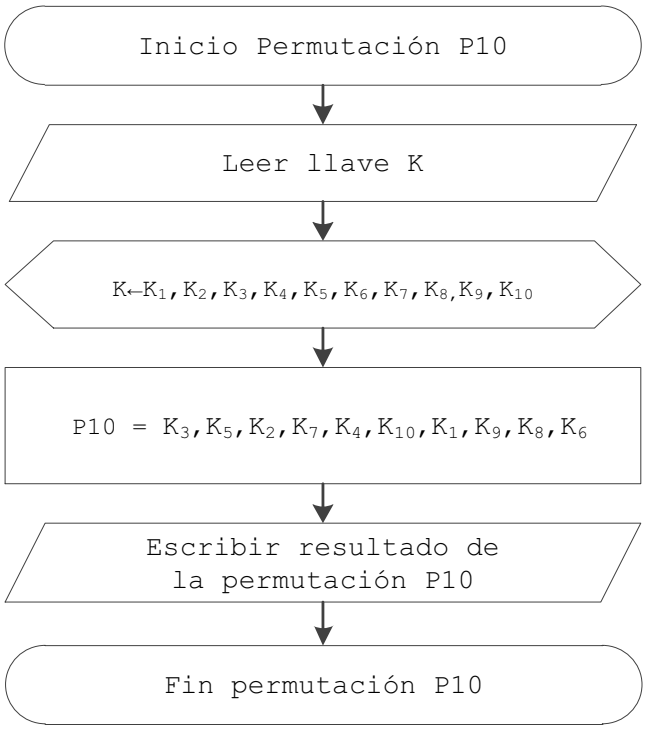


Figura 2-2. Diagrama de flujo de la permutación P10.

2.1.1.2 Shift 1

El valor resultante de la permutación P10, se divide en dos partes iguales, cada una de 5 bits denominados LS11 y LS12 respectivamente, a las cuales se les realiza un desplazamiento a la izquierda como se indica en la figura 2-3. El resultado de la permutación P10, se desplaza como se indica a continuación:

Shift1 = Shift (P10)

Shift1 = Shift (P_1, P_2, P_3, P_4, P_5, P_6, P_7, P_8, P_9, P_{10})

LS11 = P_2, P_3, P_4, P_5, P_1

LS12 = P_7, P_8, P_9, $P_{10,}$ P_6

Shif1 = (LS11, LS12)

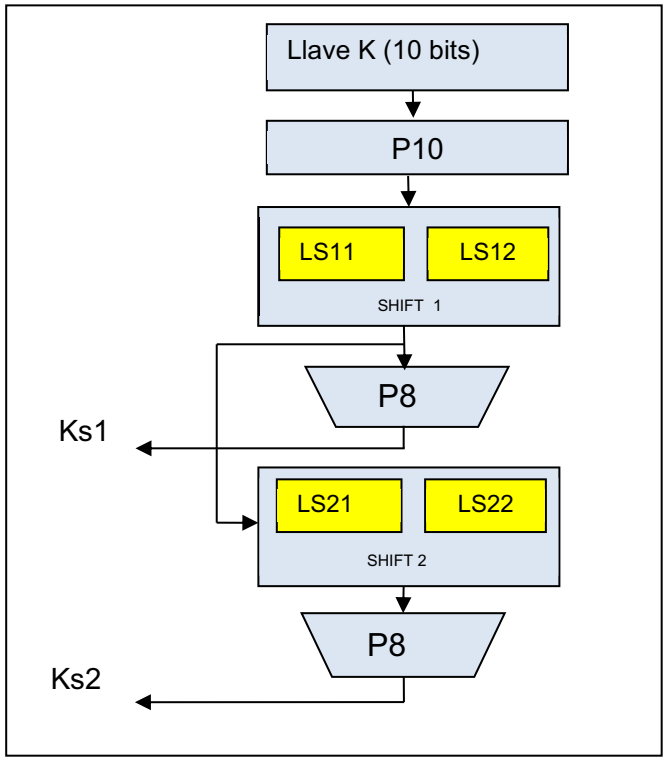


Figura 2-3. Diagrama del desplazamiento shift1 y shift2.

Ejemplo:

Shift1 = Shift (P10)

Shift1 = Shift (0 0 1 0 0 1 0 0 1 0)

LS11 = 0 1 0 0 0

LS12 = 0 0 1 0 1

Shift1 = 0 1 0 0 0 0 0 1 0 1

Diagrama de flujo del desplazamiento shift 1

El diagrama de flujo correspondiente al desplazamiento shift1 se muestra en la figura 2-4.

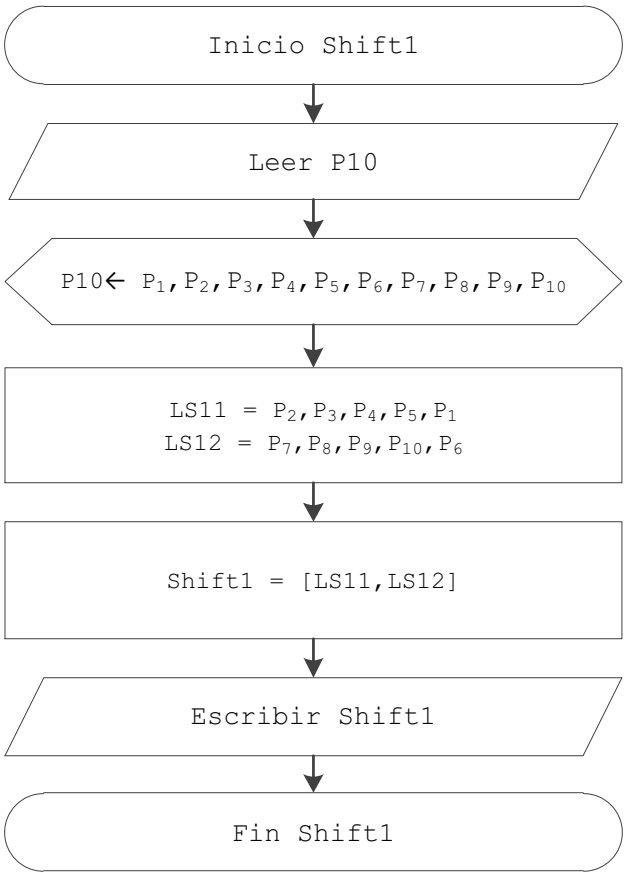


Figura 2-4. Diagrama de flujo del desplazamiento shift1.

2.1.1.3 Permutación P8

Siguiendo la figura 2-1, se realiza la permutación P8, a los diez bits resultantes de shift1, según la regla del algoritmo definida en la tabla 2-2. La subllave Ks1, se obtiene después de realizar la permutación P8 al resultado de shitf1.

P8							
6	3	7	4	8	5	10	9

Tabla 2-2. Permutación P8. [2.1]

El desplazamiento **shift1** se permuta con P8 como se indica a continuación:

Shift permutado = P8 (shift1)

Shift permutado = P8 (s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8, s_9, s_{10})

Shift permutado = s_6, s_3, s_7, s_4, s_8, s_5, s_{10}, s_9

La subllave Ks1 se obtiene del valor de la permutación P8:

Ks1 = Shift permutado

Ejemplo:

Shift permutado = P8 (0 1 0 0 0 0 0 1 0 1)

Shift permutado = P8 (s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8, s_9, s_{10})

Shift permutado = 0 0 0 0 1 0 1 0

Ks1 = 0 0 0 0 1 0 1 0

Diagrama de flujo de la permutación P8

El diagrama de flujo correspondiente a la permutación P8 se muestra en la figura 2-5.

2.1.1.4 Shift 2

Como se indica en la figura 2-3, el resultado de Shift 1, se divide en dos partes, cada una de 5 bits denominadas LS21 y LS22 respectivamente, a las cuales se les realiza dos desplazamiento a la izquierda.

El resultado del desplazamiento shift1, se desplaza como se como se indica a continuación:

Shift2 = Shift (Shift1)

Shift2 = Shift (s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8, s_9, s_{10})

LS21 = s_3, s_4, s_5, s_1, s_2

LS22 = s_8, s_9, $s_{10,}$ $s_{6,}$ s_7

Shif2 = (LS21, LS22)

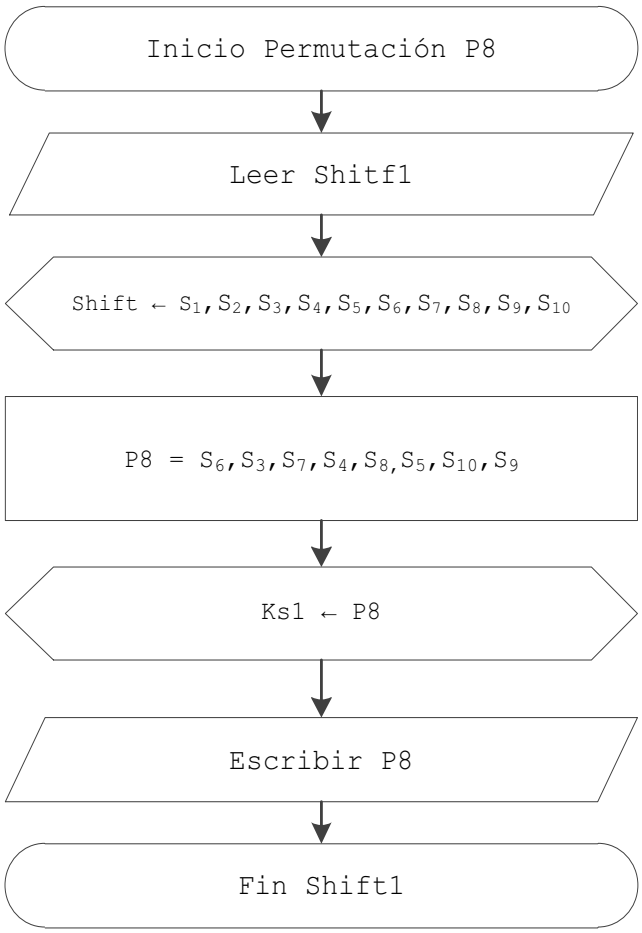


Figura 2-5. Diagrama de la permutación P8, para obtener la subllave Ks1.

Ejemplo:

Shift2 = Shift (Shift1)

Shift2 = Shift (0 1 0 0 0 0 0 1 0 1)

LS21 = 0 0 0 0 1

LS22 = 1 0 1 0 0

Shift2 = 0 0 0 0 1 1 0 1 0 0

Diagrama de flujo del desplazamiento shift 2

El diagrama de flujo correspondiente al desplazamiento Shift2 se muestra en la figura 2-6.

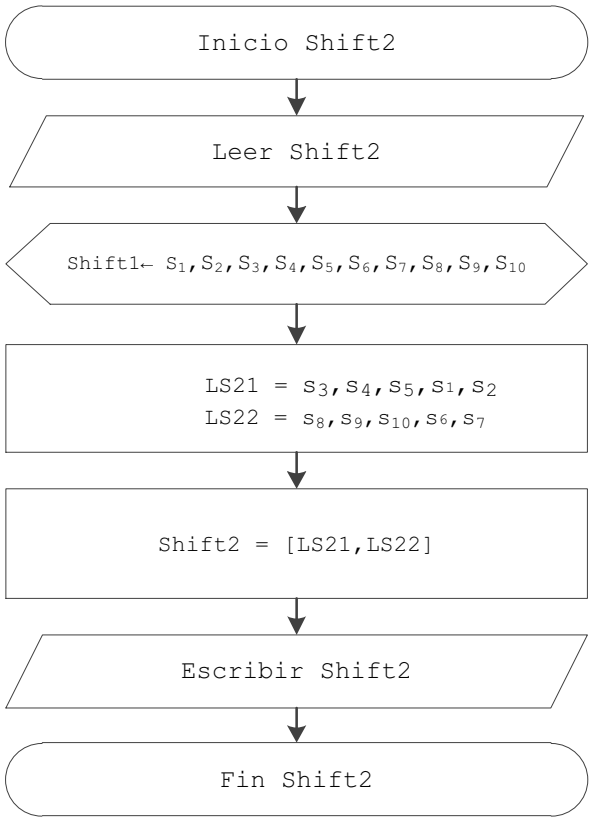


Figura 2-6. Diagrama de flujo shift2.

Al igual que la subllave Ks1, para obtener la subllave Ks2, se realiza la permutación P8 definida en la tabla 2-2 al resultado de shift2. El desplazamiento **shift2** se permuta con P8 como se indica a continuación:

Shift permutado = P8 (shift2)

Shift permutado = P8 (s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8, s_9, s_{10})

Shift permutado = s_6, s_3, s_7, s_4, s_8, s_5, s_{10}, s_9

La subllave Ks2 se obtiene del valor de la permutación P8:

Ks2 = Shift permutado

Ejemplo:

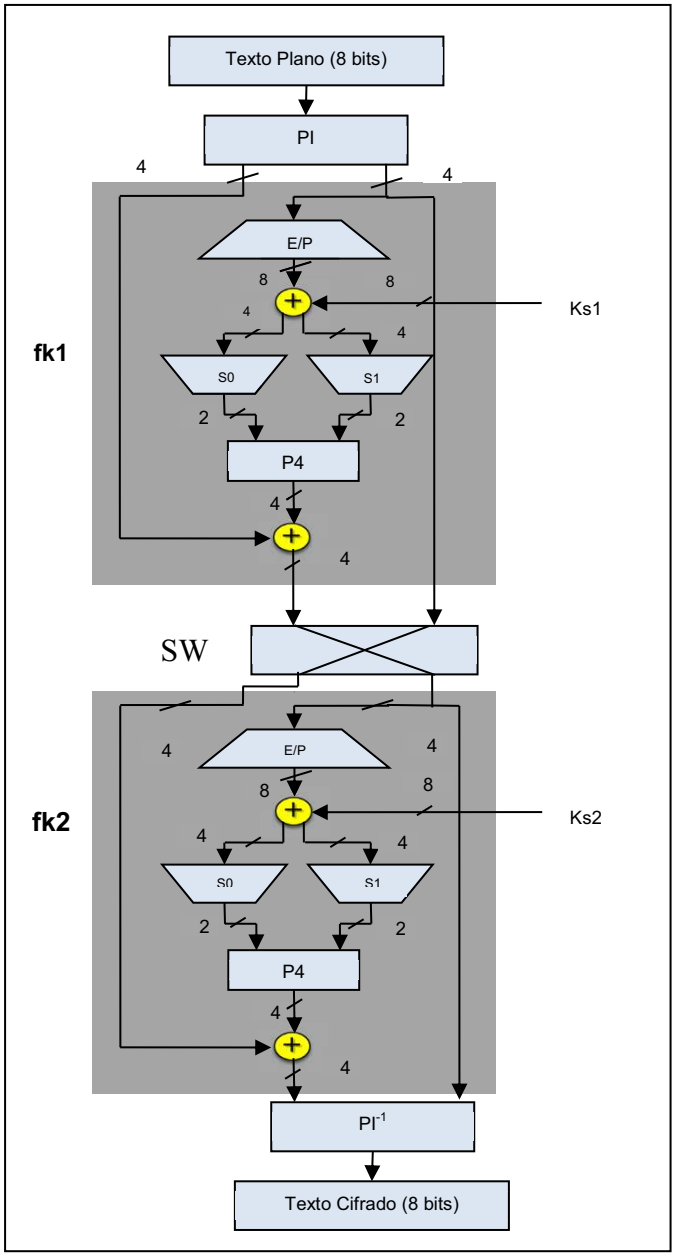
Shift permutado = P8 (0 0 0 0 1 1 0 1 0 0)

Shift permutado = P8 (s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8, s_9, s_{10}) = 1 0 0 0 1 1 0 0

Ks2 = 1 0 0 0 1 1 0 0

2.1.2 CIFRADO S-DES

Siguiendo la figura 2-1, una vez halladas las subllaves se continúa con el proceso de cifrado, esto se detalla en la figura 2-7.



Fuente [2.1]

Figura 2-7. Representación detallada del proceso de cifrado S-DES.

Si se considera al texto plano de 8 bits como **TP**, cada bit correspondiente a **TP** se representa como:

$$TP = TP_1, TP_2, TP_3, TP_4, TP_5, TP_6, TP_7, TP_8$$

Ejemplo:

$$TP = 0\ 1\ 0\ 0\ 0\ 0\ 0\ 1$$

Las etapas correspondientes al cifrado se las enumera a continuación:

- Permutación inicial PI
- Función fk1
- SW
- Función fk2
- Permutación IP^{-1}

Cada una estas etapas se detalla a continuación:

2.1.2.1 Permutación inicial PI

Como se indica en la figura 2-7, el proceso de cifrado empieza con la permutación inicial. Se realiza la permutación de los ocho bits del texto plano, según la regla definida por el algoritmo, como indica la tabla 2-3.

PI							
2	6	3	1	4	8	5	7

Tabla 2-3. Permutación inicial PI. [2.1]

El texto plano **TP** se permuta con PI como se indica a continuación:

$$\text{Texto plano permutado} = PI\ (TP)$$
$$\text{Texto plano permutado} = PI\ (TP_1, TP_2, TP_3, TP_4, TP_5, TP_6, TP_7, TP_8)$$
$$\text{Texto plano permutado} = TP_2, TP_6, TP_3, TP_1, TP_4, TP_8, TP_5, TP_7$$

Ejemplo:

$$\text{Texto plano permutado} = PI\ (TP)$$

Texto plano permutado = PI (0 1 0 0 0 0 0 1) **=** 1 0 0 0 0 1 0 0

Diagrama de flujo de la permutación inicial PI

El diagrama de flujo correspondiente a la permutación inicial PI se indica en la figura 2-8.

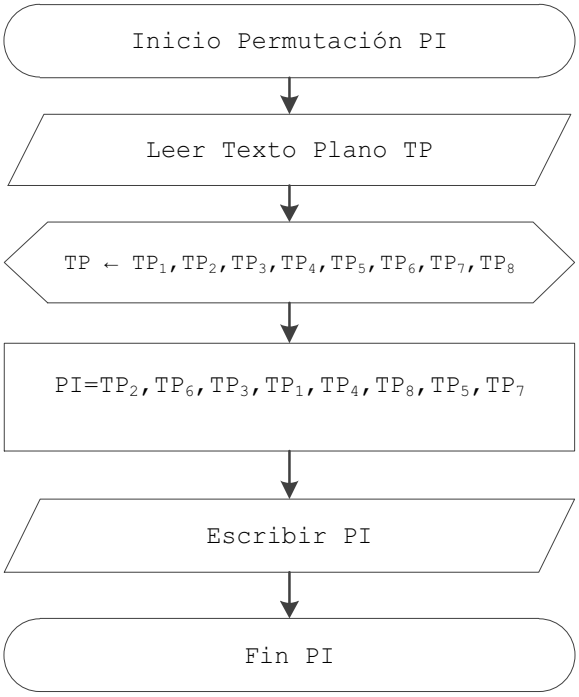


Figura 2-8. Diagrama de flujo de la permutación inicial PI.

2.1.2.2 Función FK1

Como se observa en la figura 2-7, la componente de mayor complejidad es la función fk, la cual está compuesta por distintas operaciones como:

A. Permutación E/P
B. Operación xor 1
C. Sustitución S0 y S1
D. Permutación P4
E. Operación xor 2

A. Permutación E/P

La permutación E/P, se realiza con 4 bits de entrada, correspondientes a los bits: PI_5, PI_6, PI_7, PI_8 correspondientes al resultado de la permutación inicial, según la regla definida por el algoritmo, como indica la tabla 2-4.

E/P							
4	1	2	3	2	3	4	1

Tabla 2-4. Permutación E/P. [2.1]

Donde:

$PI_5 = Pi_1$

$PI_6 = Pi_2$

$PI_7 = Pi_3$

$PI_8 = Pi_4$

El resultado de la permutación inicial **PI,** se permuta con **E/P** como se indica a continuación:

PI permutado = E/P (Pi)

PI permutado = E/P (Pi_1, Pi_2, Pi_3, Pi_4)

PI permutado = Pi_4, Pi_1, Pi_2, Pi_3, Pi_2, Pi_3, Pi_4, Pi_1

Ejemplo:

PI permutado = E/P (Pi)

PI permutado = E/P (0 1 0 0)

PI permutado = 0 0 1 0 1 0 0 0

Diagrama de la permutación E/P

El diagrama de flujo correspondiente a la permutación E/P se indica en la figura 2-9.

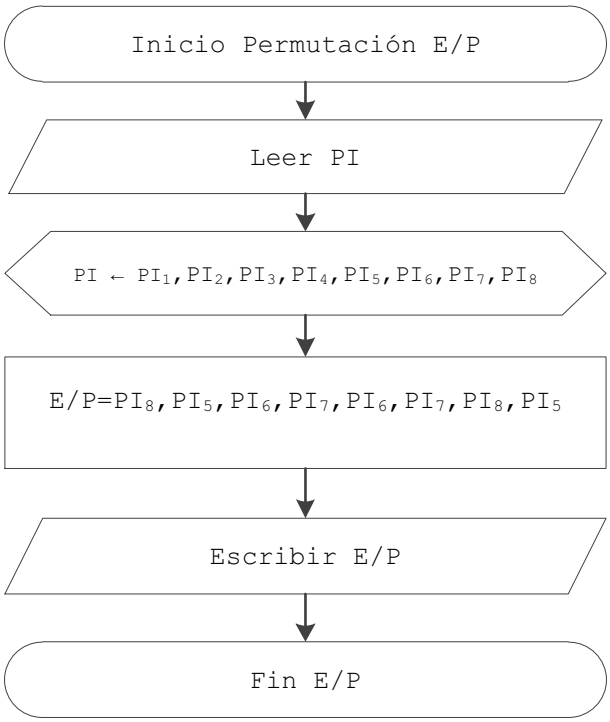


Figura 2-9. Diagrama de flujo de la permutación E/P.

B. Operación XOR1

Como indica la figura 2-7, el siguiente paso es realizar la operación xor entre el resultado obtenido de la permutación E/P y la subllave Ks1. Donde el resultado de la permutación E/P se representa como indica la figura 2-10.

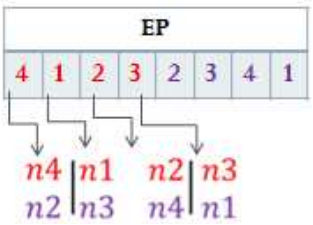


Figura 2-10. Valores n1, n2, n3, n4, correspondientes a la permutación E/P.

Los 8 bits de la subllave **Ks1** se representan como:

$$Ks1 = k_{11}, k_{12}, k_{13}, k_{14}, k_{15}, k_{16}, k_{17}, k_{18}$$

La operación xor, se realiza como indica la figura 2-11.

$$\begin{array}{c|c|c|c} n_4 \oplus k_{11} & n_1 \oplus k_{12} & n_2 \oplus k_{13} & n_3 \oplus k_{14} \\ n_2 \oplus k_{15} & n_3 \oplus k_{16} & n_4 \oplus k_{17} & n_1 \oplus k_{18} \end{array}$$

Fuente [2.1]

Figura 2-11. Operación xor.

Renombrando el resultado anterior se obtiene:

$$\begin{array}{c|c|c|c} p_{0,0} & p_{0,1} & p_{0,2} & p_{0,3} \\ p_{1,0} & p_{1,1} & p_{1,2} & p_{1,3} \end{array}$$

Fuente [2.1]

Ejemplo:

E/P = 0 0 1 0 1 0 0 0

Ks1 = 0 0 0 0 1 0 1 0

$$\begin{array}{c|c|c} 0 \oplus 0 & 0 \oplus 0 \quad 1 \oplus 0 & 0 \oplus 0 \\ 1 \oplus 1 & 0 \oplus 0 \quad 0 \oplus 1 & 0 \oplus 0 \end{array}$$

$P_{00} = 0 \quad P_{10} = 0$

$P_{01} = 0 \quad P_{11} = 0$

$P_{02} = 1 \quad P_{12} = 1$

$P_{03} = 0 \quad P_{13} = 0$

C. Sustitución S0 y S1

Siguiendo la figura 2-7, S0 y S1 son una versión reducida de las cajas S, del algoritmo de cifrado DES, y sus valores constantes se indican a continuación:

$$S0 = \begin{array}{c} 0 \\ 1 \\ 2 \\ 3 \end{array} \begin{bmatrix} 1 & 0 & 3 & 2 \\ 3 & 2 & 1 & 0 \\ 0 & 2 & 1 & 3 \\ 3 & 1 & 3 & 2 \end{bmatrix} \quad \begin{array}{c} 0 \quad 1 \quad 2 \quad 3 \end{array}$$

$$S1 = \begin{array}{c} 0 \\ 1 \\ 2 \\ 3 \end{array} \begin{bmatrix} 0 & 1 & 2 & 3 \\ 2 & 0 & 1 & 3 \\ 3 & 0 & 1 & 0 \\ 2 & 1 & 0 & 3 \end{bmatrix} \quad \begin{array}{c} 0 \quad 1 \quad 2 \quad 3 \end{array}$$

Esta etapa de cifrado consiste en sustituir los valores obtenidos en la operación **xor,** realizada en el paso anterior por los valores de las cajas S0 y S1. El resultado de cada caja es 2 bits respectivamente. Este proceso se indica en la figura 2-12.

Ejemplo:

Datos de entrada:

$P_{00} = 0 \quad P_{03} = 0$

$P_{01} = 0 \quad P_{02} = 1$

$P_{10} = 0 \quad P_{13} = 0$

$P_{11} = 0 \quad P_{12} = 1$

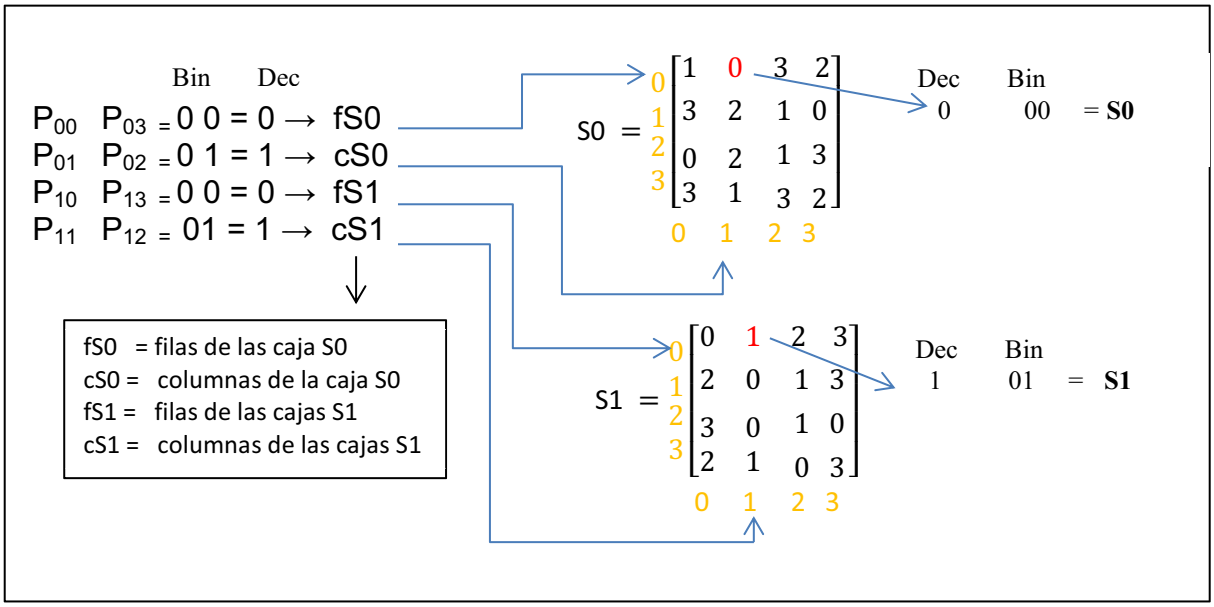


Figura 2-12. Asignación de valores de las cajas S0 y S1.

Del proceso realizado en la figura 2-12, se obtiene los valores de las cajas S0 y S1 como se indica a continuación:

S0 = 00

S1 = 01

Diagrama de flujo de la sustitución S0 y S1

El diagrama de flujo correspondiente a la sustitución de las cajas S0 y S1 se representa en la figura 2-13.

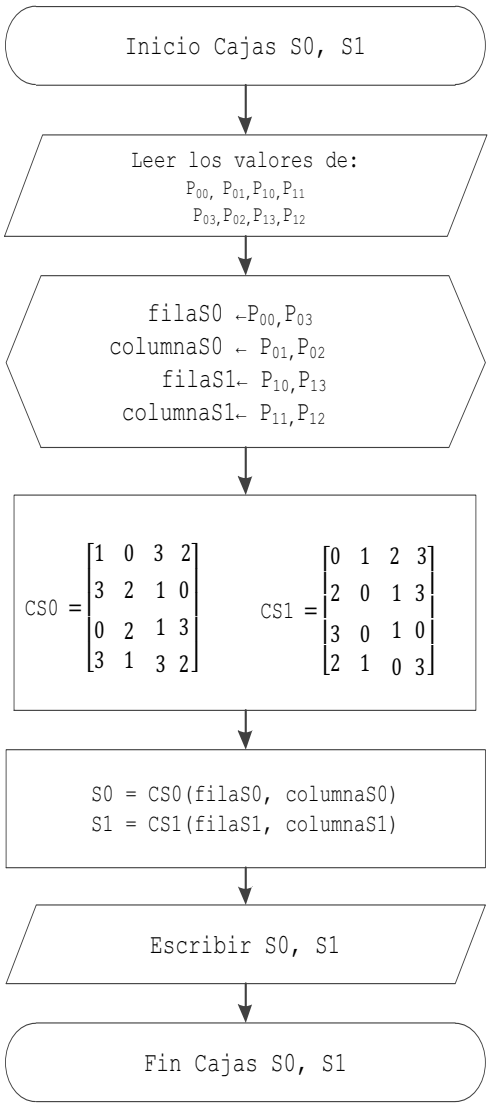


Figura 2-13. Diagrama de flujo de la sustitución S0 yS1.

D. Permutación P4

Siguiendo la figura 2-7, la siguiente etapa consiste en realizar la permutación P4, con los 4 bits de entrada correspondientes al resultado de S0 y S1.La regla está definida por el algoritmo, como indica la tabla 2-5.

P4			
2	4	3	1

Tabla 2-5. Permutación P4. [2.1]

El resultado de **S0** y **S1,** se permuta con **P4** como se indica a continuación:

$S0 = S0_1, S0_2$

$S1 = S1_1, S1_2$

$Si = S0_1, S0_2, S1_1, S1_2$

Si permutado = P4 (Si)

Si permutado = P4 (Si_1, Si_2, Si_3, Si_4)

Si permutado = Si_2, Si_4, Si_3, Si_1

Ejemplo:

Si permutado = P4 (Si)

Si permutado = P4 (0 0 0 1) = 0 1 0 0

Diagrama de flujo de la permutación P4

El diagrama de flujo correspondiente a la permutación P4 se representa en la figura 2-14.

E. Operación XOR2

Como indica la figura 2-7, el siguiente paso es realizar la operación xor entre los primeros cuatro bits del resultado de la permutación PI y el resultado de la permutación P4.

Ejemplo:

$PI = PI_1, PI_2, PI_3, PI_4 = 1\ 0\ 0\ 0$

$P4 = P4_1, P4_2, P4_3, P4_4 = 0\ 1\ 0\ 0$

$Xor2 = PI \oplus P4 = 1\ 0\ 0\ 0 \oplus 0\ 1\ 0\ 0$

$Xor2 = 1\ 1\ 0\ 0$

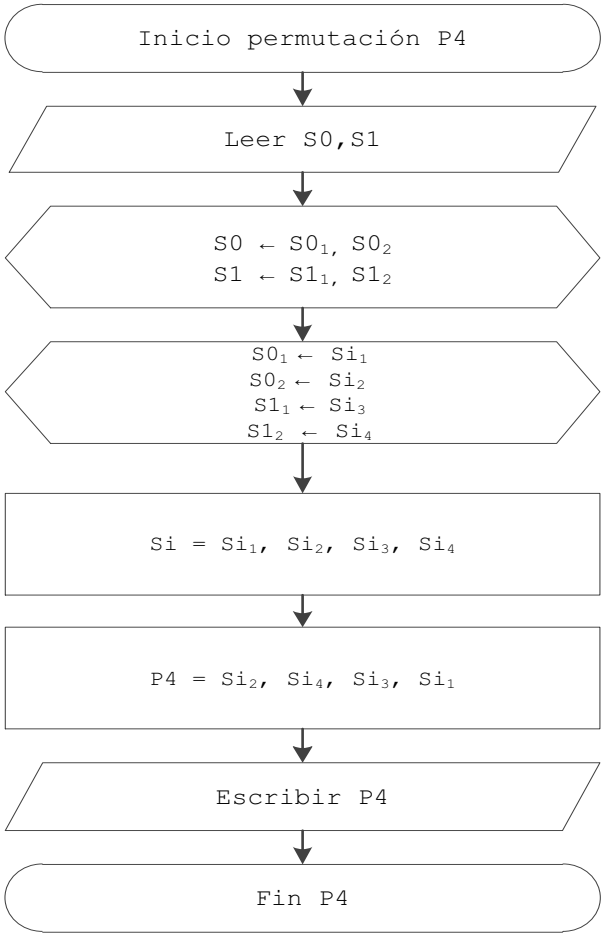


Figura 2-14. Diagrama de flujo correspondiente a la permutación P4.

2.1.2.3 SW

Como indica la figura 2-7, una vez finalizada la etapa correspondiente a **fk1,** se debe realizar la operación **SW.** Esta operación consiste en intercambiar el orden de los resultados de la operación xor2 y el segundo bloque de 4 bits de la permutación PI como se indica a continuación:

$Xor2 = X_1, X_2, X_3, X_4$

$PI = PI_5, PI_6, PI_7, PI_8$

$SW = PI_5, PI_6, PI_7, PI_8, X_1, X_2, X_3, X_4$

Ejemplo:

Xor2 = 1 1 0 0

PI = 0 1 0 0

SW = 0 1 0 0 1 1 0 0

2.1.2.4 Función FK2

Al resultado de SW, se le debe realizar nuevamente la operación de la función **fk1,** como indica la figura 2-7. El resultado de realizar nuevamente las operaciones correspondientes a **fk2** se indican en el siguiente ejemplo:

Ejemplo:

SW = 0 1 0 0 1 1 0 0

E/P = 01101001

Xor1= E/P $\oplus$ Ks2

Xor1= E/P $\oplus$ Ks2 = 01101001 $\oplus$ 1 0 0 0 1 1 0 0

Xor1= E/P $\oplus$ Ks2 = 1 1 1 0 0 1 0 1

S0 = 11

S1 = 01

P4 = 1101

Xor2 = 1001

fk2 = 10011100

2.1.2.4 Permutación PI^{-1}

Para finalizar se realiza la permutación PI^{-1}, que se define en la tabla 2-6. El resultado de esta permutación PI^{-1} nos da el texto cifrado.

El resultado **fk2,** se permuta con **PI^{-1},** como se indica a continuación:

fk2 permutado = PI^{-1} (fk2)

fk2 permutado = PI^{-1} (fk_1, fk_2, fk_3, fk_4, fk_5, fk_6, fk_7, fk_8)

fk2 permutado = fk_4, fk_1, fk_3, fk_5, fk_7, fk_2, fk_8, fk_6

Texto Cifrado = TC

TC = fk_4, fk_1, fk_3, fk_5, fk_7, fk_2, fk_8, fk_6

IP^{-1}							
4	1	3	5	7	2	8	6

Tabla 2-6. Permutación PI-1. [2.1]

Ejemplo:

fk2 permutado = PI^{-1} (fk2)

fk2 permutado = PI^{-1} (10011100)

fk2 permutado = 1 1 0 1 0 0 0 1

TC = 1 1 0 1 0 0 0 1

Diagrama de flujo de la permutación PI^{-1}

El diagrama de flujo correspondiente a la permutación PI^{-1} se indica en la figura 2-15.

2.1.3 DESCRIPCIÓN DE LA INTERFAZ GRÁFICA DE MATLAB PARA S-DES

La interfaz gráfica, se desarrolló en MatLab 7.6.0 (R2008a). El usuario deberá ingresar el texto plano de 8 bit y la llave de 10 bits. El proceso de cifrado se ejecutara con el botón **ENTER**. La interfaz gráfica se observa en la figura 2-16. El código fuente correspondiente a cada una de las etapas de cifrado e interfaz gráfica se ilustra en el anexo 1.

El respectivo diagrama de flujo, para el funcionamiento de la interfaz gráfica de S-DES, se muestra en la figura 2-17.

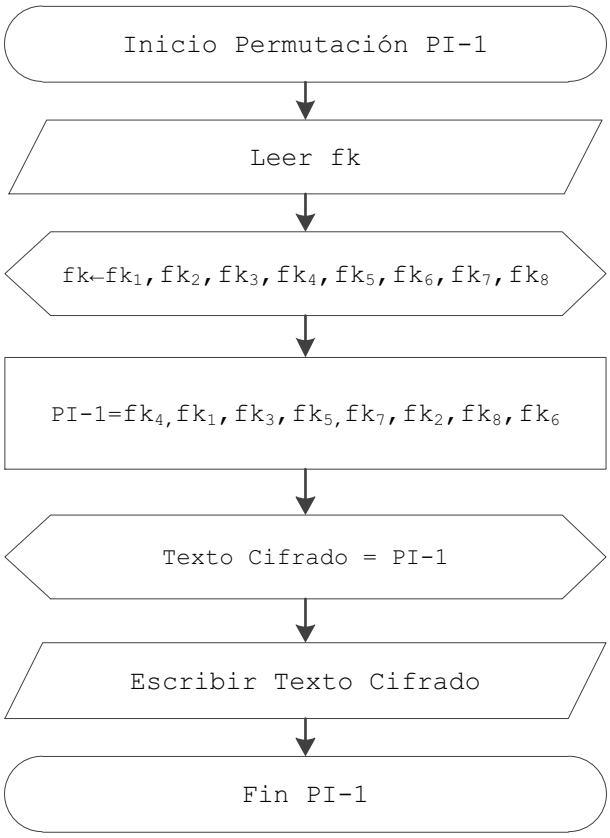


Figura 2-15. Diagrama de flujo de la permutación PI^{-1}.

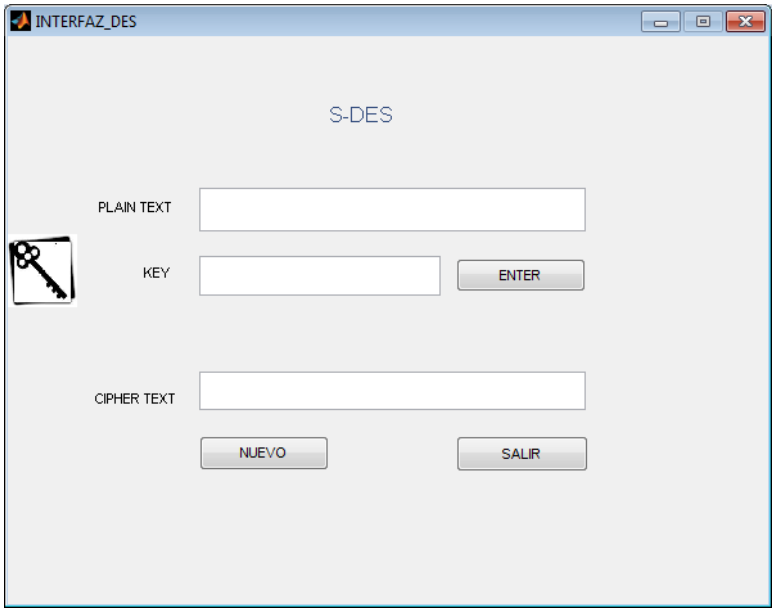


Figura 2-16. Interfaz gráfica S-DES.

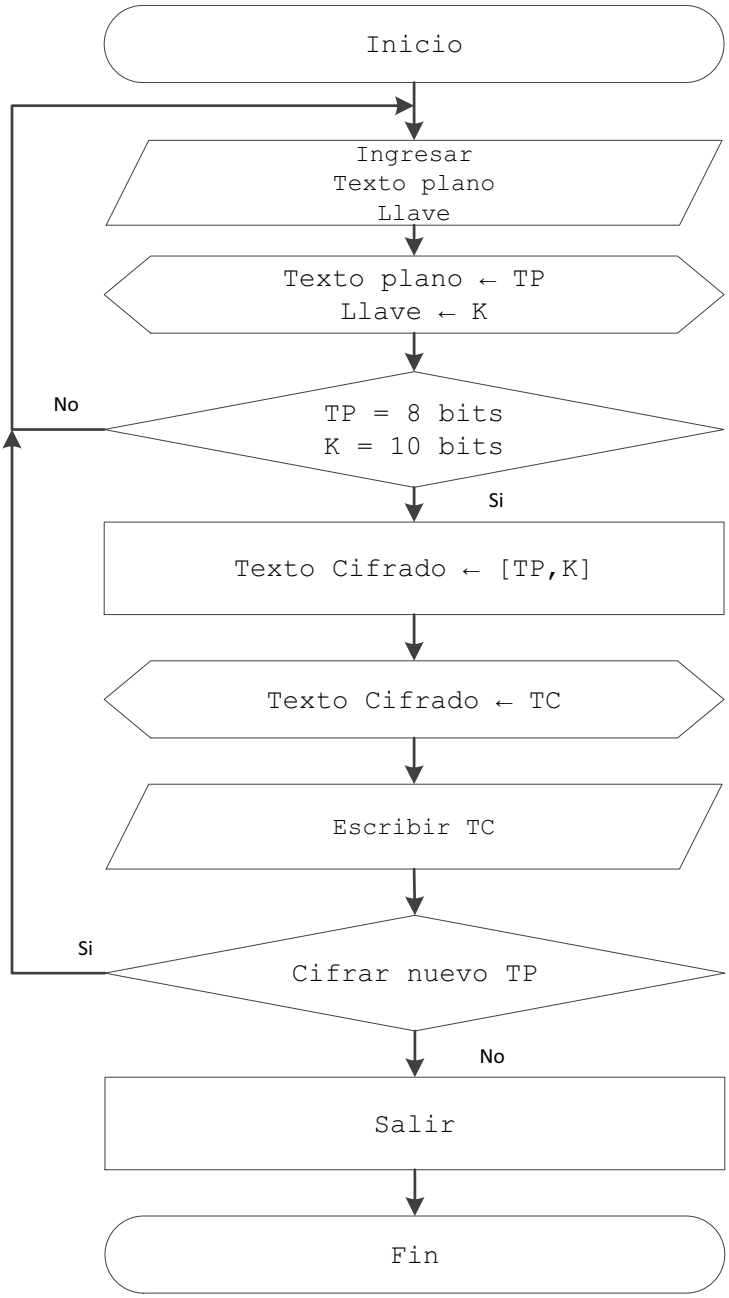


Figura 2-17. Diagrama de flujo Interfaz gráfica S-DES.

2.1.4 DISEÑO DEL SISTEMA DE CIFRADO S-DES USANDO UN FPGA

Es importante recalcar que para la realización del cifrado S-DES usando un dispositivo FPGA, se debe tener un conocimiento previo de las herramientas necesarias para la implementación de este diseño, motivo por el cual se han desarrollado anexos de cada uno de los elementos necesarios:

- Anexo 2: Arreglo de compuertas programables (FPGA).
- Anexo 3: Introducción a VHDL.
- Anexo 4: Mi primer proyecto en Ise Project Navigator 13.1.
- Anexo 5: Simulación en Ise Project Navigator 10.1.
- Anexo 6: Diagrama RTL.

La información que contiene cada uno de los anexos, debe ser comprendida antes de continuar con la descripción de cada una de las etapas del diseño.

En un diseño inicial para la implementación de S-DES, se intentó usar el código realizado en MatLab, asimilarlo y convertirlo en VHDL, para lo cual se usa subprogramas basados en funciones (*function)* y procedimientos (*procedure)*. Este método no resultó efectivo, ya que se creó código con la sintaxis correcta, pero no sintetizable.

Se debe entender que lo que se intenta hacer en VHDL, no es programar en un nuevo lenguaje, sino modelar el algoritmo S-DES en hardware. Una vez comprendido el objetivo de VHDL, se usa una descripción estructural del algoritmo, creando un circuito basado en componentes más pequeños, especificando cada una de sus partes y conexiones. La unión de cada una de estas partes denominadas entidades se realiza mediante la técnica de componentes (*components)*. La implementación de este sistema de cifrado, se usará como un ejemplo básico introductorio al uso de VHDL y FPGA. El diseño de bloques se indica en la figura 2-18.

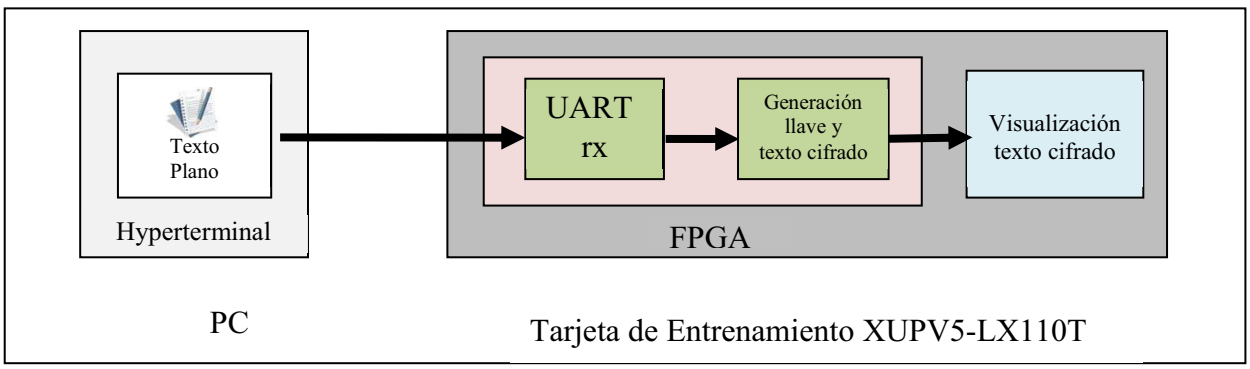


Figura 2-18. Diseño de un sistema de cifrado usando S-DES.

- Bloque Texto Plano: La transmisión de datos desde la PC, hacia la tarjeta será realizado mediante el puerto serial. Se envía un carácter ASCII, usando el hyperterminal, con los siguientes parámetros:

    - Bits por segundo 19200
    - Bit de datos: 8
    - Paridad: ninguno
    - Bits de parada: 1
    - Control de flujo: ninguno

- Bloque UART rx: Este bloque se usa para la recepción de los datos en el dispositivo FPGA.
- Bloque Generación de llave y texto cifrado: Este bloque recibirá el texto plano de 8 bits y generará la llave de 10 bits en base al texto plano ingresado por el usuario.
- Bloque Visualización texto cifrado: El texto cifrado, se indicará directamente en la tarjeta de entrenamiento usando los leds de la misma.

2.1.4.1 Bloque de generación de llave y texto cifrado

Se creó una entidad denominada SDES, la cual contendrá cada una de la etapas del cifrador descritas en la figura 2-1. La representación conceptual de esta entidad se observa en la figura 2-19.

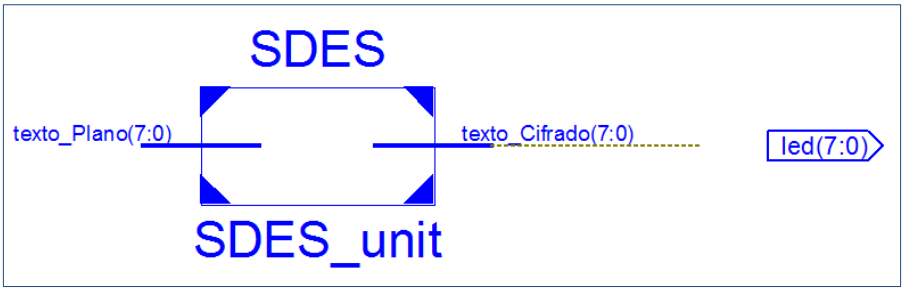


Figura 2-19. Esquema RTL de la entidad SDES.

Como se indica en la figura 2-19, la entidad SDES, contiene los siguientes puertos de entrada y salida:

- Texto plano denominado → texto_Plano : in STD_LOGIC_VECTOR (7 downto 0);
- Texto cifrado denominado → texto_Cifrado : out STD_LOGIC_VECTOR (7 downto 0);

Internamente está compuesta por dos entidades denominadas: llave_Expansion y cifrador, como se observa en la figura 2-20. Cada bloque se describirá de forma detallada.

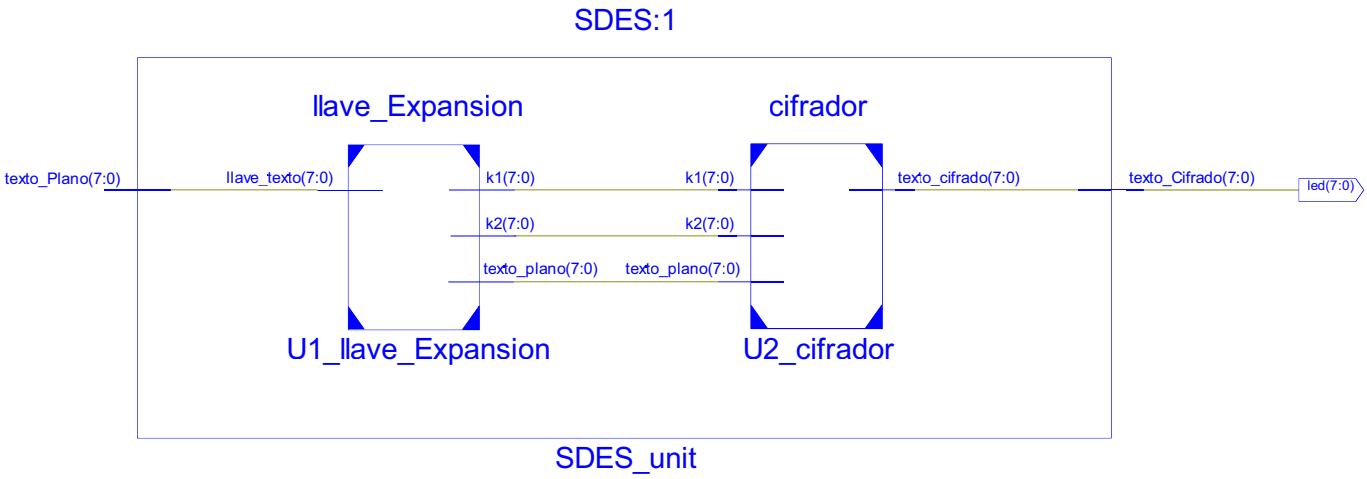


Figura 2-20. Esquema RTL interno de la entidad SDES.

Estos dos bloques se unen entre si mediante ***components***, dentro de la entidad SDES, y de esta forma se ralizarán todas las conexiones entre entidades. El código VHDL se indica a continuación:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity SDES is

 Port ( texto_Plano : in STD_LOGIC_VECTOR (7 downto 0);
        texto_Cifrado : out STD_LOGIC_VECTOR (7 downto 0));

end SDES;
```

```
architecture arch of SDES is
        signal salida_texto_plano : std_logic_vector(7 downto 0);
        signal salida_k1 : std_logic_vector(7 downto 0);
        signal salida_k2 : std_logic_vector(7 downto 0);
begin
        U1_llave_Expansion : entity work.llave_Expansion(arch)
                port map ( llave_texto => texto_plano,
                           texto_plano => salida_texto_plano,
                           k1 => salida_k1,
                           k2 => salida_k2);
        U2_cifrador : entity work.cifrador(arch)
                port map ( texto_plano => salida_texto_plano,
                           k1 => salida_k1,
                           k2 => salida_k2,
                           texto_cifrado => texto_Cifrado);
end arch;
```

2.1.4.1.1 Entidad llave_Expansion

Realiza la generación de la llave en base al texto plano entrante, y las dos subllaves k1 y k2. Los datos obtenidos corresponderán a los puertos de entrada de la entidad cifrador. A continuación se especifican los puertos de entrada y salida.

llave_texto : in STD_LOGIC_VECTOR (7 downto 0);
texto_plano : out STD_LOGIC_VECTOR (7 downto 0);
k1 : out STD_LOGIC_VECTOR (7 downto 0);
k2 : out STD_LOGIC_VECTOR (7 downto 0);

Internamente esta entidad esta compuesta, por cada una de las permutaciones y funciones determinadas en el algoritmo S-DES, para la generación de las subllaves. El esquema RTL se indica en la figura 2-21.

La llave que se genera se forma con los 8 bits del texto plano y se le añade los dos últimos bits del mismo texto plano para completar los 10 bits de la llave, y con este valor de llave se generan las subllaves k1 y k2.

Ejemplo:

Texto plano → 01000001
Llave →0100000101

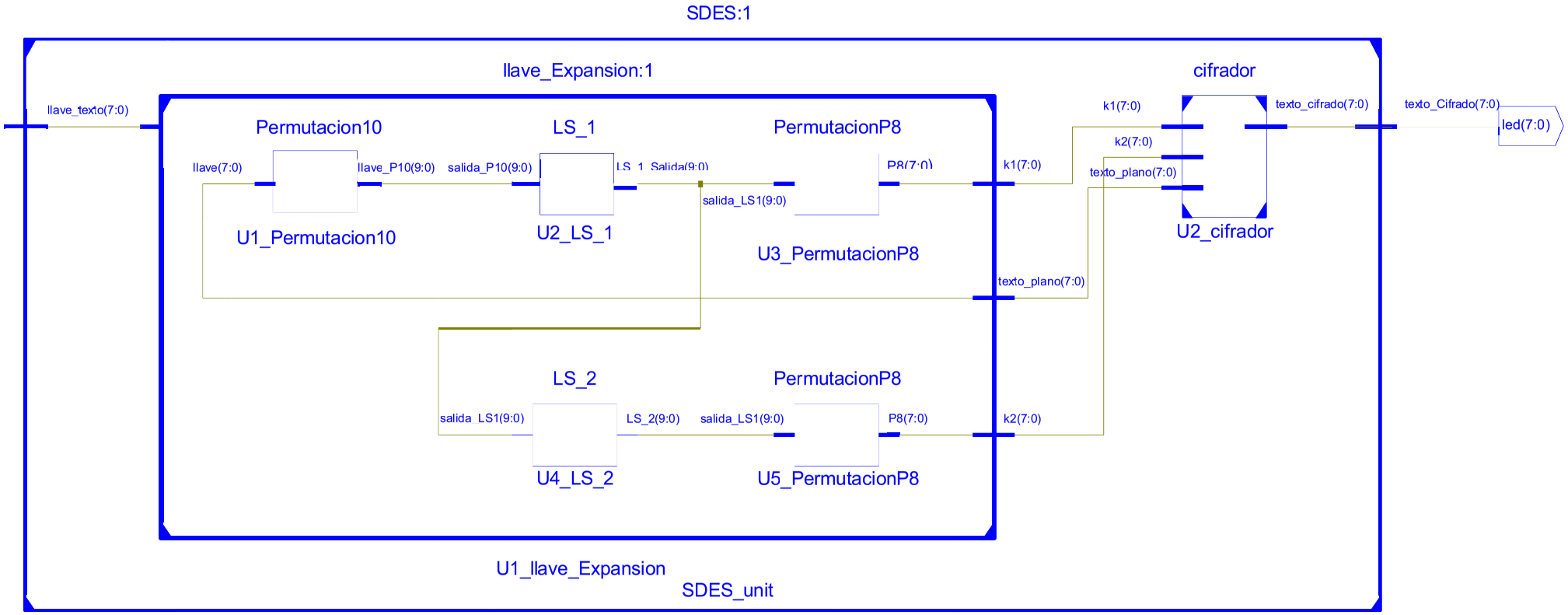


Figura 2-21. Esquema RTL interno de la entidad llave_Expansion.

Las entidades internas son las siguientes:

- Entidad → Permutacion10, realiza la permutación correspondiente a la tabla 2-1. Los puertos de entrada y salida son:

  llave : in STD_LOGIC_VECTOR (7 downto 0);
  llave_P10 : out STD_LOGIC_VECTOR (9 downto 0);

- Entidad → LS_1, ejecuta la operación correspondiente a Shift1 como se ilustra en la figura 2-3, y los puertos de entrada y salida son:

  salida_P10 : in STD_LOGIC_VECTOR (9 downto 0);
  LS_1_Salida : out STD_LOGIC_VECTOR (9 downto 0);

- Entidad →PermutacionP8, ejecuta la permutación de la tabla 2-2, y los puertos de entrada y salida son:

  salida_LS1 : in STD_LOGIC_VECTOR (9 downto 0);
  P8 : out STD_LOGIC_VECTOR (7 downto 0);

- Entidad → LS2: ejecuta la operación de Shift2 como indica la figura 2-3, y los puertos de entrada y salida son:

  salida_LS1 : in STD_LOGIC_VECTOR (9 downto 0);
  LS_2 : out STD_LOGIC_VECTOR (9 downto 0);

Ejemplo simulación entidad llave_Expansion

Sólo para el proceso de simulación se usará el software de Xilinx ISE 10.1, en esta versión se encuentra la herramienta *Test Bench WaveFor*, que permite realizar la simulación de forma gráfica.

Los datos de entrada en bits, para la simulación son los siguientes:

texto_Plano → 01000001

Llave →0100000101

El resultado se observa en la figura 2-22.

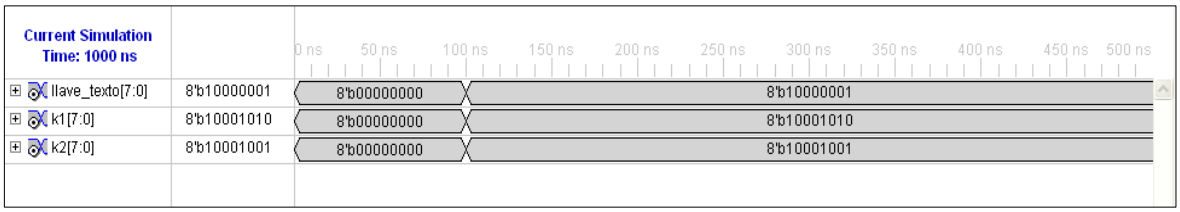


Figura 2-22. Simulación entidad *llave_Expansion.*

2.1.4.1.2 Entidad cifrador

Cifra el texto plano. Los puertos de entrada y salida se describen a continuación.

texto_plano : in STD_LOGIC_VECTOR (7 downto 0);
k1 : in STD_LOGIC_VECTOR (7 downto 0);
k2 : in STD_LOGIC_VECTOR (7 downto 0);
texto_cifrado : out STD_LOGIC_VECTOR (7 downto 0);

Internamente contiene las permutaciones y funciones determinadas en el algoritmo S-DES, para el proceso de cifrado descrito en la figura 2-7. El esquema RTL se indica en la figura 2-23.

Las entidades internas son las siguientes:

- Entidad → IP, realiza la permutación correspondiente a la tabla 2-3. Los puertos de entrada y salida son:

  texto_plano : in STD_LOGIC_VECTOR (7 downto 0);
  izquierda_ip : out STD_LOGIC_VECTOR (3 downto 0);
  derecha_ip : out STD_LOGIC_VECTOR (3 downto 0);

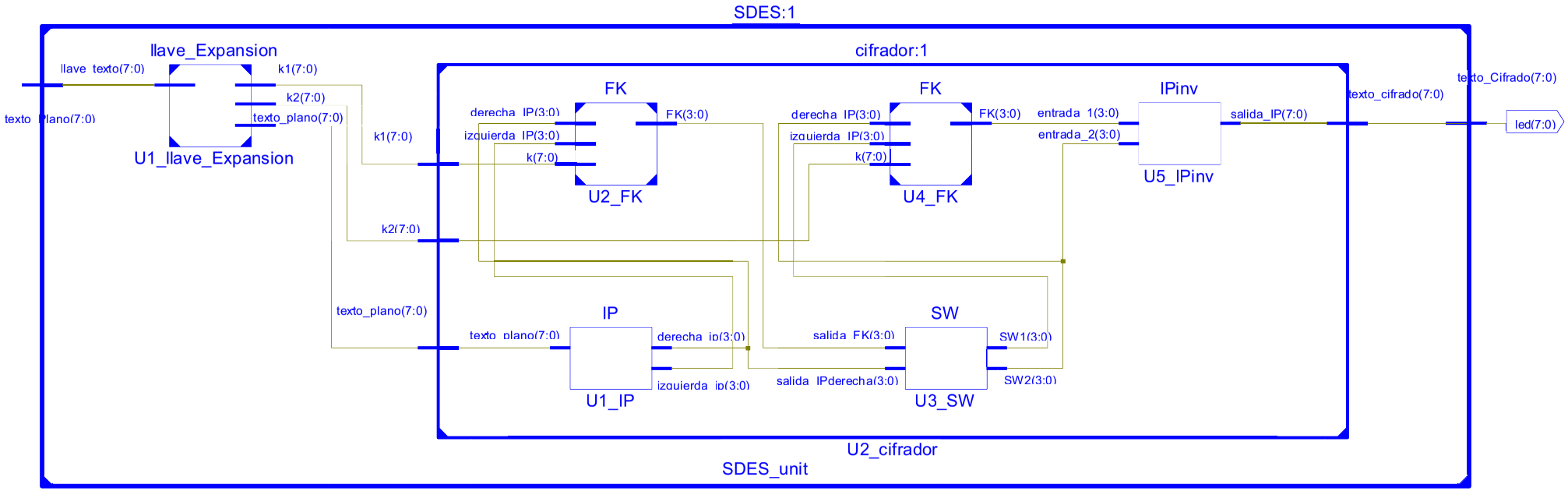


Figura 2-23. Esquema RTL interno de la entidad *cifrador.*

- Entidad → FK, corresponde a la función fk, detallada en la figura 2-7. Los puertos de entrada y salida son:

  izquierda_IP : in STD_LOGIC_VECTOR (3 downto 0);
  derecha_IP : in STD_LOGIC_VECTOR (3 downto 0);
  k : in STD_LOGIC_VECTOR (7 downto 0);
  FK : out STD_LOGIC_VECTOR (3 downto 0);

  El diagrama interno de la entidad FK, corresponde a la figura 2-24.

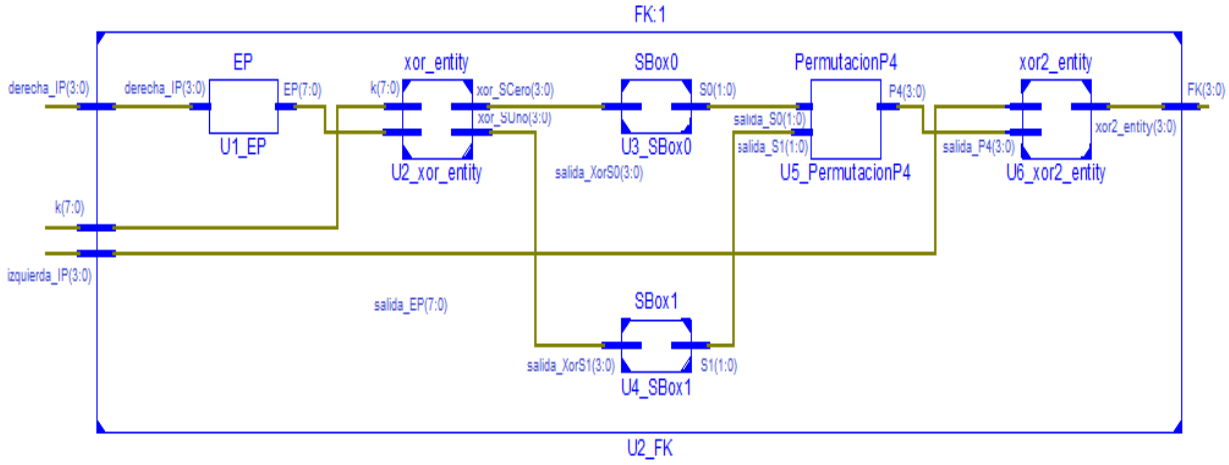


Figura 2-24. Esquema RTL interno de la entidad FK.

- Entidad → SW, corresponde a la operación SW. Los puertos de entrada y salida son:

  salida_FK : in STD_LOGIC_VECTOR (3 downto 0);
  salida_IPderecha : in STD_LOGIC_VECTOR (3 downto 0);
  SW1 : out STD_LOGIC_VECTOR (3 downto 0);
  SW2 : out STD_LOGIC_VECTOR (3 downto 0));

- Entidad → IPinv, corresponde a la permutación de la tabla 2-6. Los puertos de entrada y salida son:

  entrada_1 : in STD_LOGIC_VECTOR (3 downto 0);

entrada_2 : in STD_LOGIC_VECTOR (3 downto 0);
salida_IP : out STD_LOGIC_VECTOR (7 downto 0));

Ejemplo simulación entidad cifrador

Los datos de entrada son:

texto_plano → 01000001
k1 →00001010
k2 →10001100

El resultado se indica en la figura 2-25.

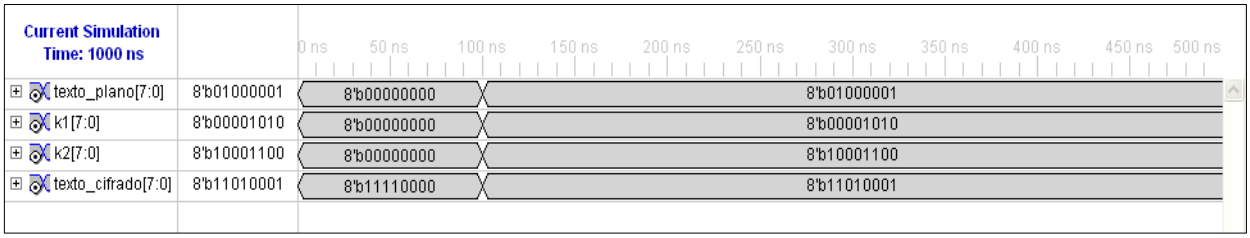


Figura 2-25. Simulación entidad *cifrador.*

Resultado:

texto_cifrado →11010001

Una vez que se a explicado cada componente de la entidad SDES, esta debe unirse al bloque UART rx, para recibir los datos de la PC y proceder a realizar el cifrado del texto plano. El esquema RTL de esta conexión se indica en la figura 2-26.

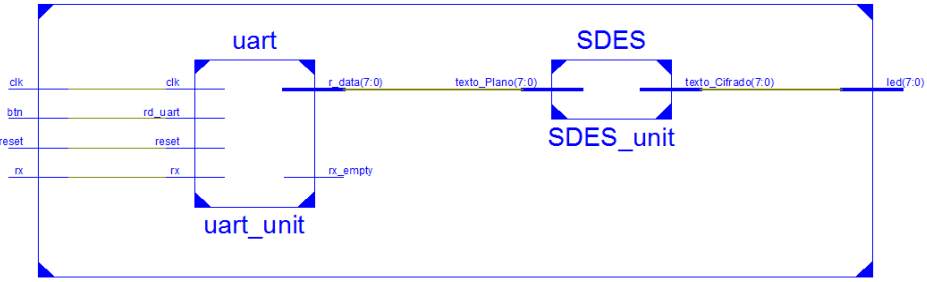


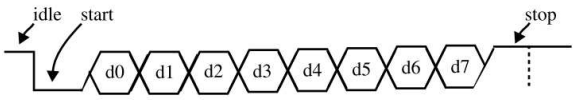
Figura 2-26. Esquema RTL del sistema de cifrado S-DES.

2.1.4.2 Bloque UART

Una UART (*Universal asynchronous receiver and transmitter*) es una unidad de comunicación en serie, que envía datos a través de una línea serial, es típicamente usada con el estándar RS-232. [2.2]

Especificaciones de la UART

La transmisión inicia cuando el bit de inicio se *setea* en cero (0L), seguido por los 8 bits de datos, un bit opcional de paridad y un bit de parada.



Fuente [2.2]

Figura 2-27. Transmisión de un byte.

La señal de reloj no es enviada en la línea serial, por lo tanto este tipo de transmisión requiere de un previo acuerdo de los parámetros de transmisión, entre el transmisor y el receptor: la velocidad de señal (*baud rate*), el número de bits de datos, bits de parada y paridad en caso de que ésta se esté usando. Las velocidades de señal comunes son: 2400, 4800, 9600, 19200. [2.2]. Los puertos asignados a esta entidad ***uart*** se observan en la figura 2-28.

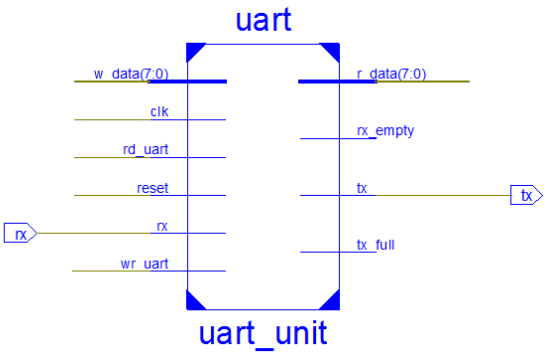


Figura 2-28. Esquema RTL de la entidad *uart.*

Internamente el bloque **UART** se describte en la figura 2-29.

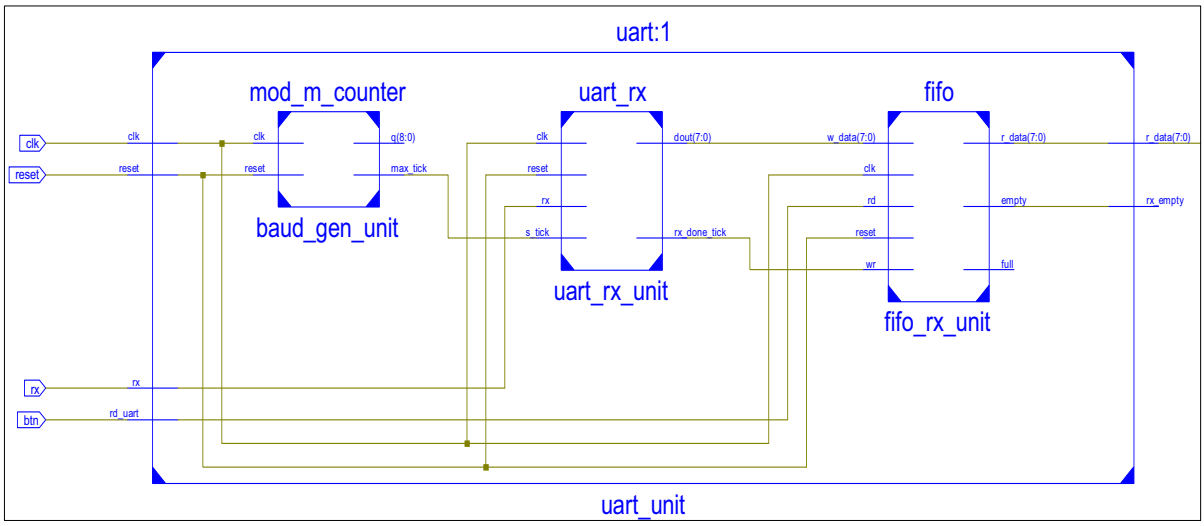


Figura 2-29. Diagrama RTL interno de la entidad *uart.*

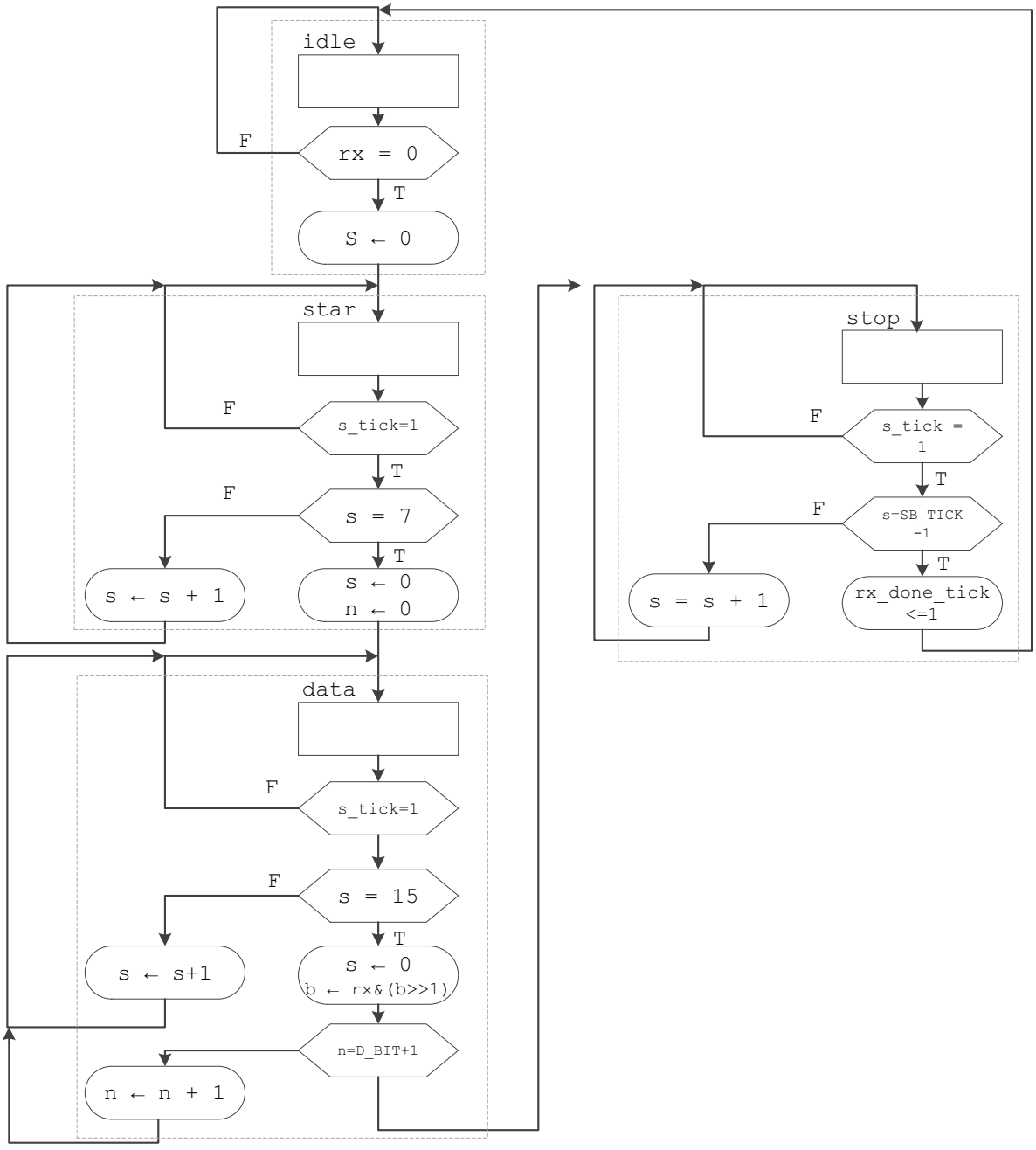
Las entidades internas son las siguientes:

- Entidad → uart_rx, recibe los datos, enviados por la interfaz gráfica en MatLab y los envia a la fifo de recepción:

```
clk : in  STD_LOGIC;
reset : in  STD_LOGIC;
rx : in  STD_LOGIC;
s_tick : in  STD_LOGIC;
rx_done_tick : out  STD_LOGIC;
dout : out  STD_LOGIC_VECTOR (7 downto 0);
```

  La entidad uart_rx, se basa en un esquema de sobremuestreo, comúnmente utilizado para localizar la posición media de cada bit transmitido cuando se tome la muestra real. La frecuencia de muestreo más común es de 16 veces la frecuencia de la señal. Por lo tanto cada bit se muestrea 16 veces, pero solo una muestra se guarda. [2.2]

El esquema de sobremuestreo usa N bits de datos y M bits de parada. El diagrama ASMD[1] (*algorithmic state machine with data path*) de este bloque se describe en la figura 2-30. [2.2]



Fuente [2.2]

Figura 2-30. Diagrama ASMD de la máquina de estados del receptor.

[1]ASMD (Algorithmic StateMachine with Data path). Se trata de un tipo de representación de una máquina de estados, que funciona con unidad de control y rutas de datos.

Donde:

- Se espera hasta que la señal de entrada se convierte en "0" (el bit de inicio) y luego inicia la toma de muestras (*tick*).
- Cuando el contador llega a 7, la señal de entrada alcanza la posición media del bit de inicio. Desactiva el contador y reinicia.
- Cuando el contador llega a 15, se encuentra en el centro del primer bit de datos, se recupera y almacena en un registro. Luego se reinicia el contador.
- Repite el paso anterior N-1 veces para recuperar el resto de los bits de datos.
- Si el bit opcional de paridad se utiliza, repita este paso una vez más.
- Repita este paso M veces más para obtener los bits de parada.

El esquema de sobremuestreo reemplaza la función del reloj. En vez de usar el flanco ascendente para muestrear, se usan las marcas de muestreo (*ticks*) para estimar la posición central de cada bit. [2.2]

Cabe recalcar que el sistema de reloj debe ser más rápido que la velocidad de señal para que el sobremuestreo sea posible.

- Entidad → fifoRecepcion, almacena los datos correspondientes al texto plano y la llave. Posteriormente son enviados a la entidad cifrador. Los puertos de entrada y salida son:

```
clk : in  STD_LOGIC;
reset : in  STD_LOGIC;
rd : in  STD_LOGIC;
wr : in  STD_LOGIC;
w_data : in  STD_LOGIC_VECTOR (B-1 downto 0);
empty : out  STD_LOGIC;
full : out  STD_LOGIC;
r_data : out  STD_LOGIC_VECTOR (255 downto 0);
```

Donde los valores de B corresponden a un tipo de dato *generic* denotado como:

```
generic( B: natural := 8);
```

La fifo de recepción está diseñada para almacenar 32 bits. Estos datos son enviados a la entidad ***SDES***, y es aquí donde se realizan cada una de las operaciones para cifrar.

La fifo se implementa con punteros de lectura y escritura. Las banderas se usan para identificar si está llena (*full*) o vacía (*empty*). [2.2]

Puntero de lectura → registro de dirección de lectura
Puntero de escritura →registro de dirección de escritura

- FIFO Vacía → Si el registro de dirección de lectura alcanza al registro de dirección de escritura, envía la bandera *empty*. [2.2]

- FIFO Llena → Si el registro de dirección de escritura alcanza al registro de dirección de lectura, envía la bandera *full*. [2.2]

- Entidad → mod_m_counter, genera una señal de muestreo. Los puertos de entrada y salida son:

```
clk : in  STD_LOGIC;
reset : in  STD_LOGIC;
max_tick : out  STD_LOGIC;
q : out  STD_LOGIC_VECTOR(N-1 downto 0);
```

Donde los valores de N corresponden a un tipo de dato *generic* denotado como:

```
generic( N: integer := 9);
```

En esta entidad se genera una señal de muestreo cuya frecuencia es exactamente 16 veces la frecuencia de la velocidad de transmisión designada de la UART. Para evitar la creación de un nuevo dominio de reloj, la salida del circuito generador de la velocidad de baudios podría servir para habilitar *ticks*. En lugar de eso, todo el sistema tendrá un único reloj, para garantizar confiabilidad en el funcionamiento. [2.2]

Para una velocidad de señal de 19200 baudios por segundo, la velocidad de muestreo es 307200 (19200 * 16) *ticks* por segundo.
Con un sistema de reloj a 100MHz, el generador de baudios necesita un contador módulo 326 (100M/307200 = 326). [2.2]

El código fuente de los módulos VHDL, de cada una de las etapas descritas se muestran en el anexo 7.

2.2 DES

A finales de la década de los sesenta, IBM creó un proyecto de investigación en criptografía para computadoras liderado por Horst Feistel. El proyecto concluyó en 1971 con el desarrollo de un algoritmo denominado LUCIFER, que posteriormente fue vendido a Lloyd de Londres para ser usado en un sistema que entregaba dinero. LUCIFER es un cifrado de bloques de Feistel, que opera con bloques de 64 bits, usando una llave de tamaño de 128 bits.

Debido a los resultados favorables que produjo el proyecto LUCIFER, IBM se embarcó en el desarrollo de un producto comercial de cifrado, dicho producto podría ser implementado en un solo chip. El resultado de este proyecto fue una versión refinada del algoritmo LUCIFER que era más resistente al criptoanálisis pero que también reducía el tamaño de la llave a 56 bits para adaptarse a un solo chip.

Por el año de 1973 la Oficina Nacional de Estándares (*NBS National Bureau Standards*) emitió una solicitud de propuestas para un estándar de cifrado nacional. IBM presentó el proyecto que había desarrollado y este fue el mejor algoritmo propuesto y luego fue adoptado en el año de 1977 como el Estándar de Cifrado de Datos DES.

Antes de que el algoritmo desarrollado por IBM fuese incluido como el estándar, este fue sujeto a intensas críticas principalmente en dos áreas:

La primera es que la longitud de la llave en el algoritmo original de IBM denominado LUCIFER era de 128 bits y la del algoritmo DES es únicamente de 56 bits, presentándose una enorme reducción de 72 bits, la preocupación surgía debido a la longitud muy corta de la llave que no era lo suficientemente fuerte para soportar un ataque de fuerza bruta.

La segunda concierne al criterio de diseño en la estructura interna de DES, las cajas S fueron de naturaleza secreta. Así que los usuarios no estaban seguros de que la estructura interna de DES estuviese libre de cualquier punto de debilidad que luego la NSA (*National Security Agency*) pudiese utilizarlos para descifrar los mensajes cifrados sin necesidad de conocer previamente la llave de 56 bits.

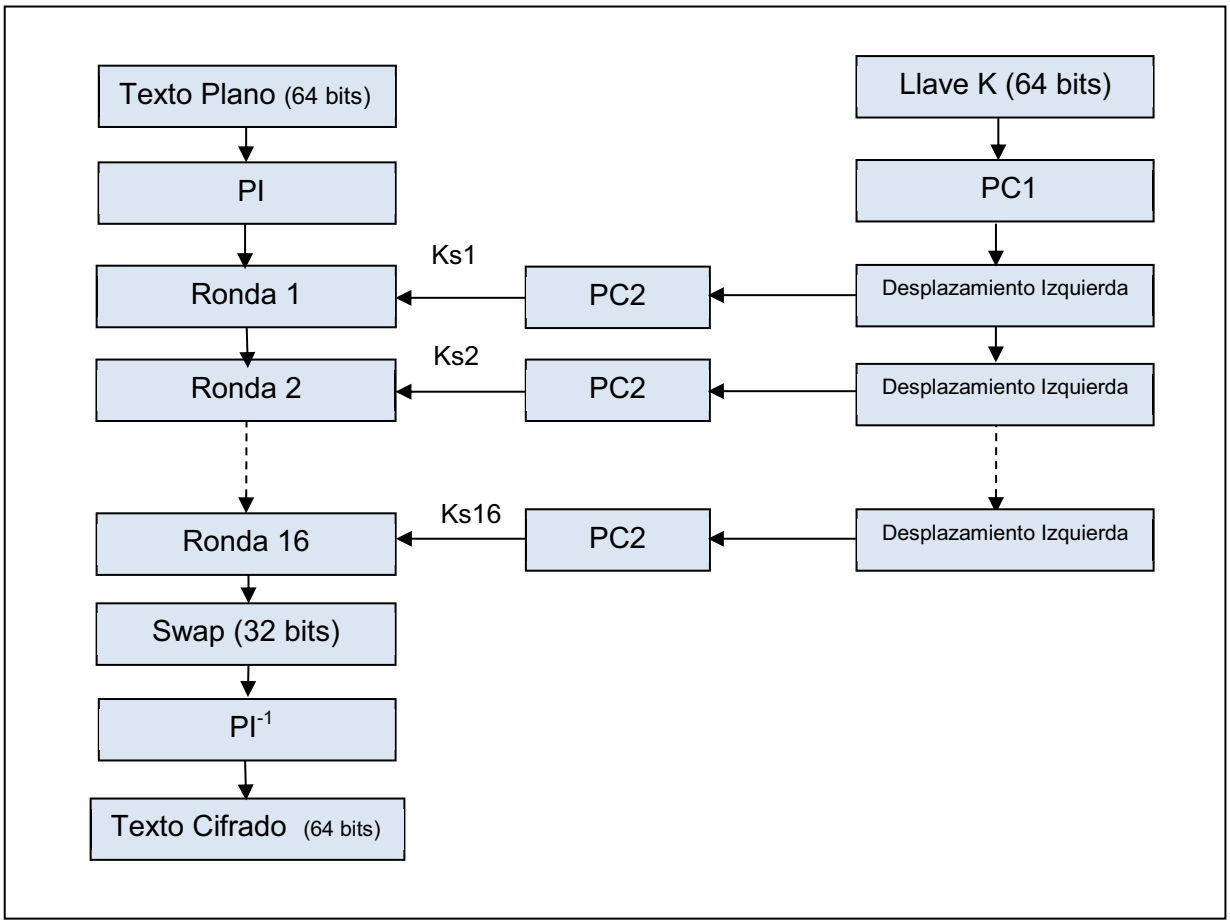
Eventos siguientes basados en el criptoanálisis diferencial revelaron que DES poseía una estructura interna fuerte, además las cajas S fueron cambiadas con las sugerencias dadas por la NSA que removieron las vulnerabilidades encontradas en el proceso de evaluación.

En 1994 DES, el NIST (*National Institute of Standards and Technology*) reafirmó a DES para uso federal por otros cinco años, además el NIST recomendó el uso de dicho estándar en aplicaciones que se encuentren manejando información secreta.

DES se basa en una estructura de Feistel y para su estudio se divide en dos etapas:

- Generación de subllaves
- Cifrado

Un esquema global se presenta en la figura 2-31.



Fuente [2.1]

Figura 2-31. Cifrado DES.

2.2.1 GENERACIÓN SUBLLAVES Ks1, Ks2,…, Ks16

La etapa de generación de las subllaves consiste en generar a partir de la llave inicial **K**, 16 subllaves. En esta etapa se realizan operaciones de permutación y desplazamiento a la llave inicial para obtener cada una de las 16 subllaves, como se ilustra en la figura 2-31.

Las operaciones generadoras de las subllaves se las enumera a continuación:

- Permutación PC1
- Desplazamiento Izquierda
- Permutación PC2

Si se considera a la llave de 64 bits como **K**, cada bit correspondiente a **K** se representa como:

$$K = K_1, K_2, K_3, K_4, \ldots\ldots\ldots\ldots\ldots\ldots K_{64}$$

Ejemplo:

K= 0110010011001010111001001101110011000010110111001100100011000 01

2.2.1.1 Permutación PC1

En la permutación PC1 se ingresan los 64 bits de llave **K**, estos bits se distribuyen en grupos de 8 bits donde cada bit menos significativo es un bit de paridad, quedando 56 bits para la generación de subllaves. Este proceso se observa en la figura 2-32.

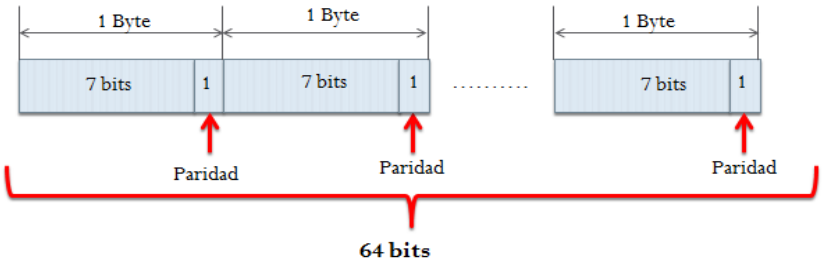


Figura 2-32. Permutación PC-1.

Ejemplo:

K=0110011001100101011100100110111001100001011011100110010001100001

Los bits de paridad representados de color rojo en el ejemplo anterior se utilizan para detectar errores en la llave **K**, antes de generar las subllaves.

Con los 64 bits, correspondientes a **K,** se realiza una permutación PC-1, donde los bits son ordenados de tal forma que a la salida se obtiene el bit 1 como el bit 57, el bit 2 es el 49……etc. Nótese que los bits 8, 16, 24, 32,40, 48, 56, y 64 no están en la tabla ya que son los bits de paridad. A continuación la tabla 2-7, explica la regla de la permutación.

PC-1													
57	49	41	33	25	17	9	1	58	50	42	34	26	18
10	2	59	51	43	35	27	19	11	3	60	52	44	36
63	55	47	39	31	23	15	7	62	54	46	38	30	22
14	6	61	53	45	37	29	21	13	5	28	20	12	4

Tabla 2-7. Permutación PC1. [2.1]

Ejemplo:

Dato de entrada:
K = 0110010011001010111001001101110011000010110111001100100010100001

Dato de salida:
KPC = 00000000111111111111111100000010110101101011001010000100

Diagrama de flujo de la permutación PC1

El diagrama de flujo correspondiente a la permutación PC1 se indica en la figura 2-33.

2.2.1.2 Desplazamiento izquierda

Como se indica en la figura 2-31, el siguiente paso es realizar el desplazamiento circular a la izquierda. Se toma los 56 bits, resultantes de la permutación PC1. Este

grupo de bits se divide en mitades iguales y luego se realiza un desplazamiento a la izquierda como se muestra en la figura 2-34.

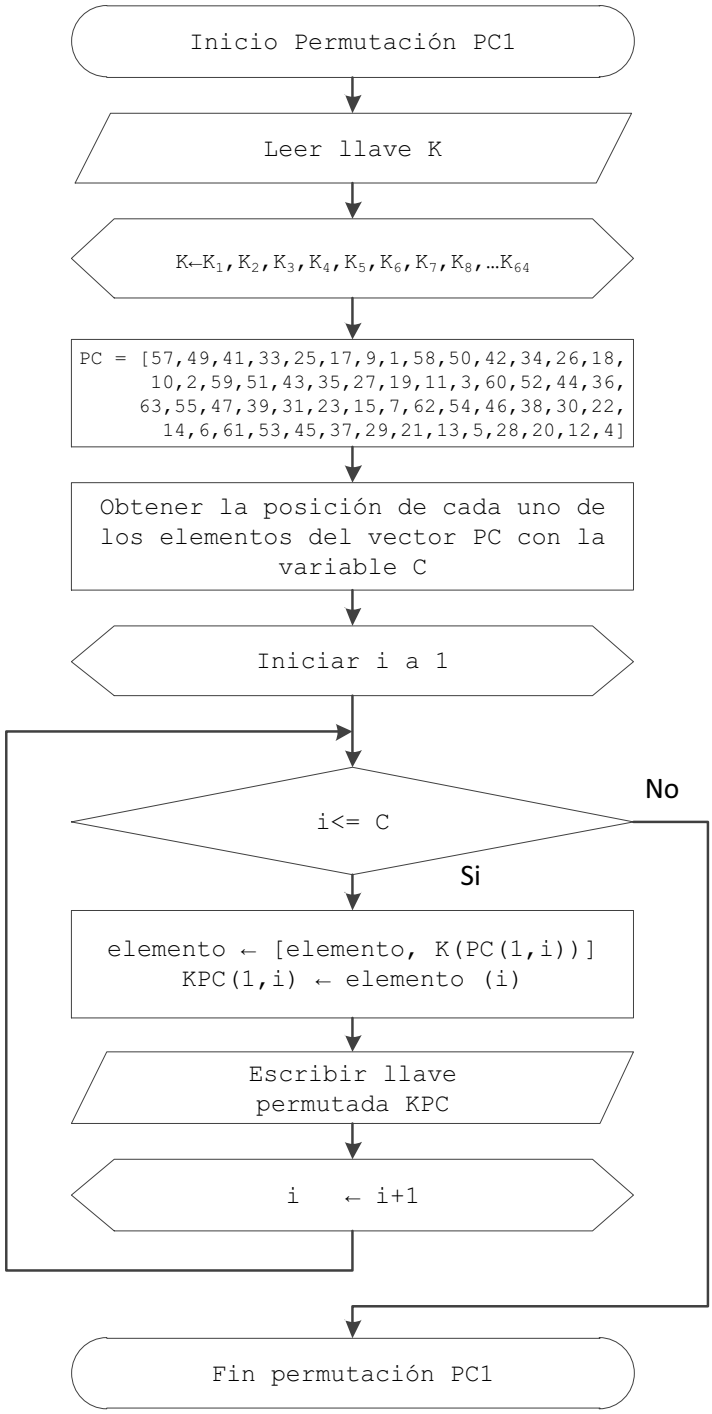


Figura 2-33. Diagrama de flujo correspondiente a la permutación PC1.

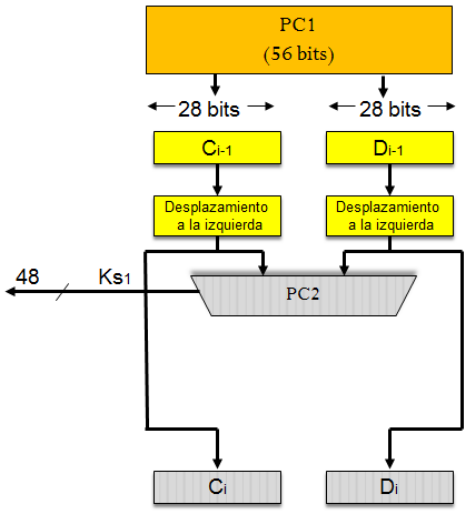
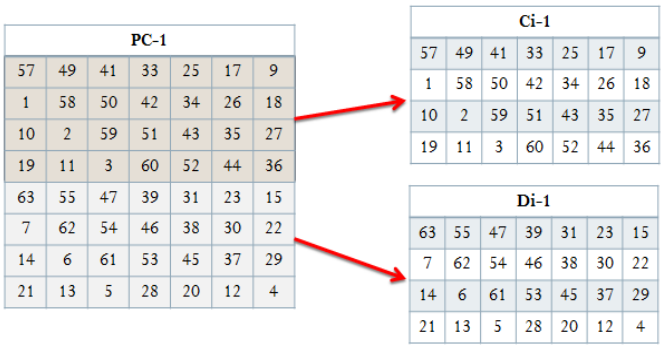


Figura 2-34. Representación gráfica del desplazamiento a la izquierda.

Antes de realizar el proceso de desplazamiento circular a la izquierda, la matriz PC1 se divide en dos matrices C_i y D_i (para la izquierda y la derecha respectivamente), cada una ellas de 28 bits. Se toman los primeros 28 bits de la matriz PC1 (1 al 28) y estos son los elementos de la matriz C_i, los elementos del 29 al 56 de la matriz PC1 son los elementos de la matriz D_i como se muestra en la figura 2-35.



PC-1						
57	49	41	33	25	17	9
1	58	50	42	34	26	18
10	2	59	51	43	35	27
19	11	3	60	52	44	36
63	55	47	39	31	23	15
7	62	54	46	38	30	22
14	6	61	53	45	37	29
21	13	5	28	20	12	4

Ci-1						
57	49	41	33	25	17	9
1	58	50	42	34	26	18
10	2	59	51	43	35	27
19	11	3	60	52	44	36

Di-1						
63	55	47	39	31	23	15
7	62	54	46	38	30	22
14	6	61	53	45	37	29
21	13	5	28	20	12	4

Figura 2-35. Ci-1 y Di-1.

Los bloques Ci-1 y Di-1, se rotan hacia la izquierda, de manera que los bits que estaban en la segunda posición pasan a la primera, aquellos que estaban en tercera posición pasan a la segunda, etc. Los bits que estaban en la primera posición se mueven hacia la última posición. Esta operación se puede observar con más claridad en la figura 2-36.

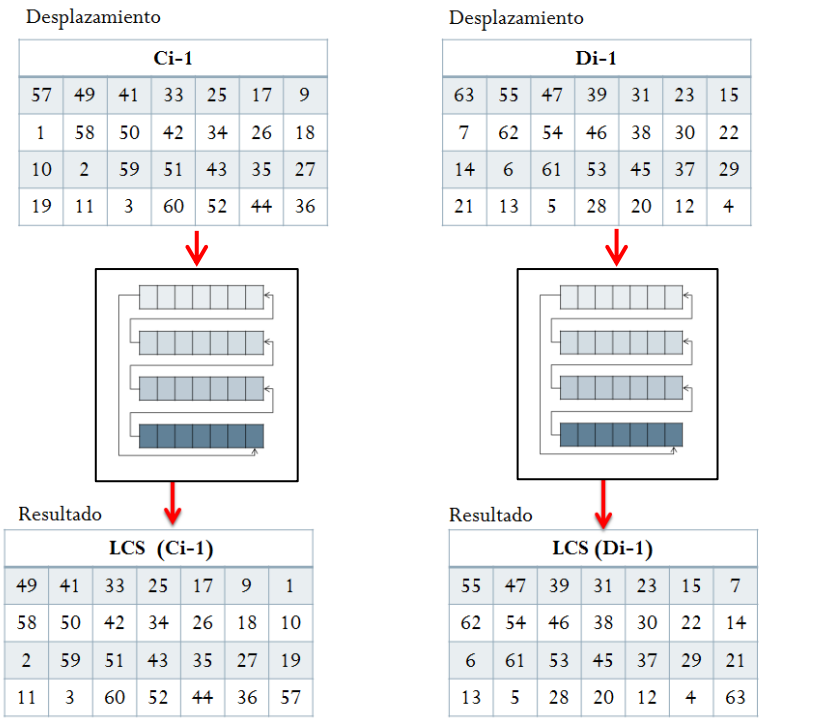


Figura 2-36. Desplazamiento Ci-1 y Di-1.

Posteriormente los dos bloques de 28 bits se vuelven a unir en un bloque de 56 bits, como se indica en la figura 2-37.

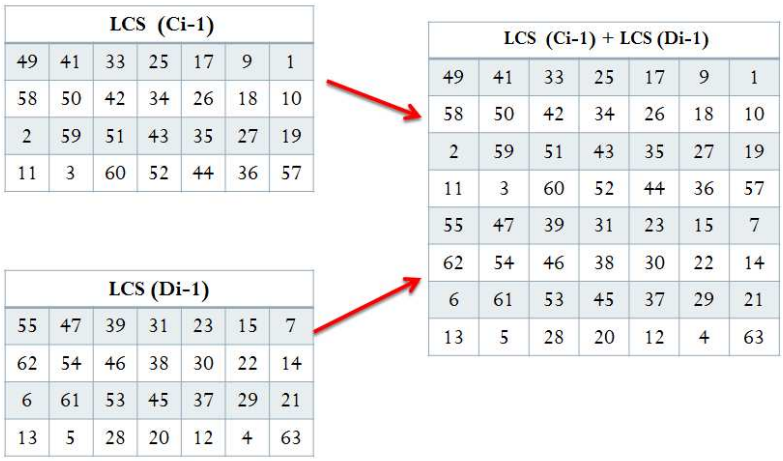


Figura 2-37. Agrupación LCS_T.

Ejemplo:

KPC = 0000000011111111111111110000 00101101011010110010100001 00

Ci-1 = 00000000111111111111111110000

Di-1 = 00101101011010110010100001000

LCS (Ci-1) = 00000001111111111111111100000

LCS (Di-1) = 01011010110101100101000010000

LCST = [LCS (Ci-1), LCS (Di-1)]

LCS_T = 00000001111111111111111100000010110101101011001010000100000

Diagrama de flujo de desplazamiento a la izquierda

El diagrama de flujo, para la realizar la operación desplazamiento izquierda se indica en la figura 2-38.

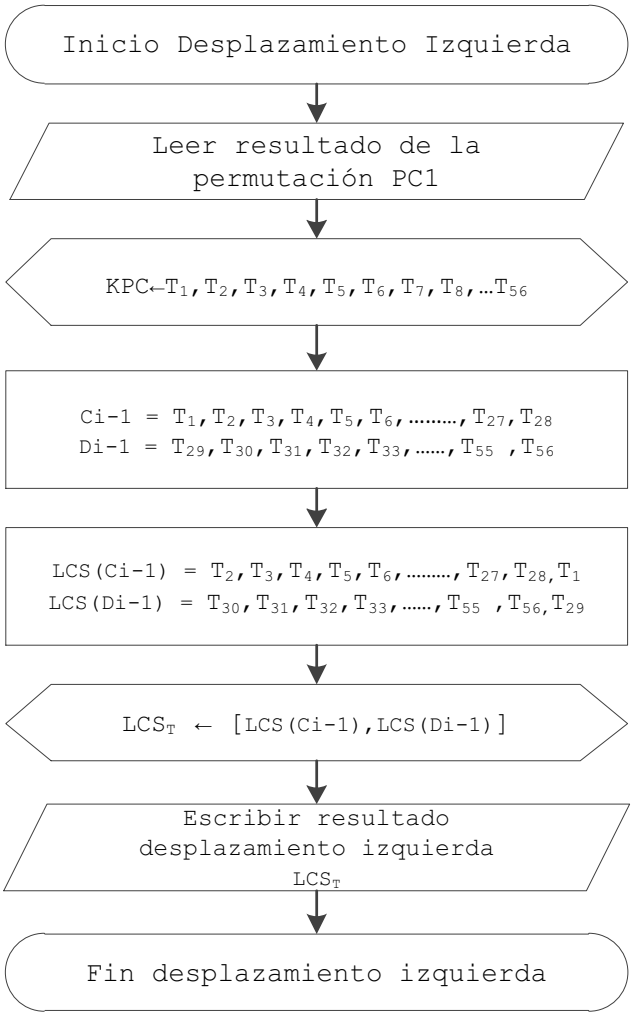


Figura 2-38. Diagrama de flujo correspondiente al desplazamiento a la izquierda.

2.2.1.3 Permutación PC2

Según la figura 2-31, la siguiente etapa consiste en realizar la permutación PC2. En el paso anterior los dos bloques de 28 bits se agrupan en un bloque de 56 bits, ahora pasan por la permutación, denominada PC2, dando como resultado un bloque de 48 bits que representa la subllave K_{s1}. A continuación la tabla 2-8, explica la regla de la permutación.

PC2															
14	17	11	24	1	5	3	28	15	6	21	10	23	19	12	4
26	8	16	7	27	20	13	2	41	52	31	37	47	55	30	40
51	45	33	48	44	49	39	56	34	53	46	42	50	36	29	32

Tabla 2-8. Permutación PC2. [2.1]

Ejemplo:

LCS_T = 00000001111111111111111000000101101011010110010100001000
KPC2 = 11100000101111100110011000010011001100000011110001
Ks1 = KPC2

Diagrama de flujo de la permutación PC2

El diagrama de flujo correspondiente a la permutación PC2 se indica en la figura 2-39.

2.2.1.4 Generación de las Subllaves Ks2, Ks3, Ks4,…, Ks16

Siguiendo la figura 2-31, la siguiente etapa consiste en generar las subllaves Ks2, Ks3, Ks4,…,Ks16. Para generar las subllaves se sigue exactamente el mismo proceso que para generar la subllave Ks1, pero en esta ocasión no se aplica la permutación PC1, por lo tanto los datos a ser desplazados son C_{i-1} y D_{i-1} haciendo referencia al valor anterior de C_i y D_i como se muestra en la figura 2-40.

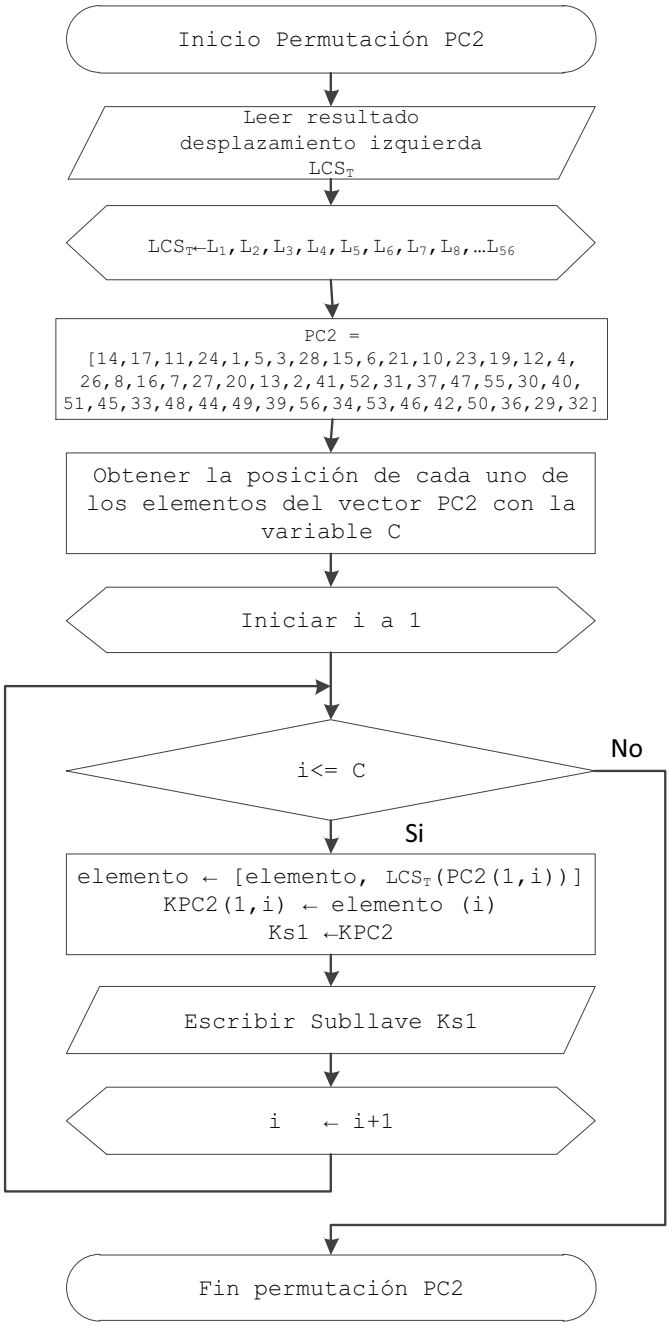


Figura 2-39. Diagrama de flujo correspondiente a la permutación PC2.

Otra diferencia es el número de desplazamientos circulares a la izquierda que deben hacerse, en algunos casos se realiza un desplazamiento y en otros casos dos. El desplazamiento a la izquierda para la generación de las subllaves, dependerá del número de ronda, el mismo que representa a la subllave. Esto se ilustra en la tabla 2-9.

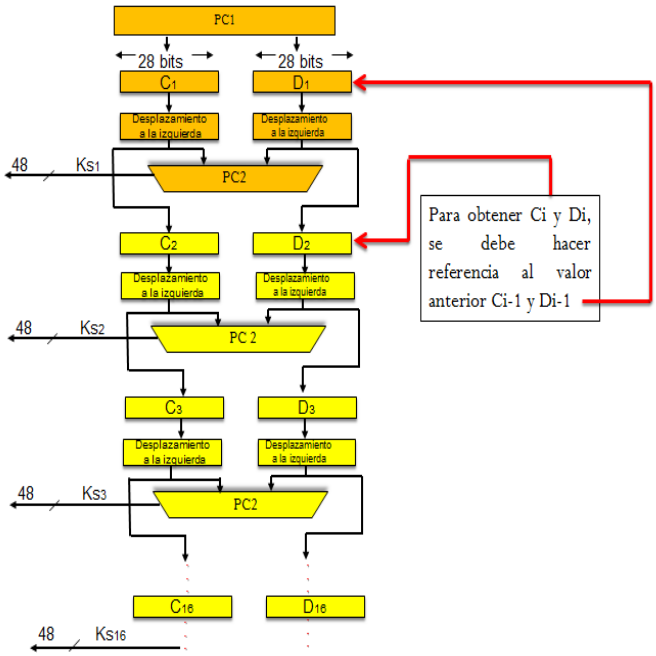


Figura 2-40. Generación subllaves Ks1,Ks2….Ks16.

La permutación PC2 es la misma y la salida de esta constituye la subllave generada, el proceso se repite 15 veces y se obtienen las otras restantes.

Número de ronda	Número de desplazamientos a la izquierda	Subllave generada
1	1	Ks1
2	1	Ks2
3	2	Ks3
4	2	Ks4
5	2	Ks5
6	2	Ks6
7	2	Ks7
8	2	Ks8
9	1	Ks9
10	2	Ks10
11	2	Ks11
12	2	Ks12
13	2	Ks13
14	2	Ks14
15	2	Ks15
16	1	Ks16

Tabla 2-9. Valor de desplazamiento para generación de subllaves.

Ejemplo:

Para las subllaves que se generan en base a la figura 2-40 y la tabla 2-9, el resultado se observa a continuación:

Ks2 = 111000001011011001110110111110000100001010010111
Ks3 = 111001001101011001110110000101110100001010001011
Ks4 = 111001101101001101110010100101100011000101000001
Ks5 = 101011101101001101110011101000101010001101100100
Ks6 = 101011110101001101011011011100001010111110000010
Ks7 = 001011110101001111011001011111000000010000011011
Ks8 = 000111110101100111011001010011110111000001001010
Ks9 = 000111110100100111011001010010001000111110100011
Ks10 = 000111110110100110011101010111100100110000011001
Ks11 = 000111110010110110001101010010110101000101011000
Ks12 = 010110110010110010101101100000011111000100101000
Ks13 = 110110011010110010101100111000000001111000100100
Ks14 = 110100001010111010101110110110000000101010111110
Ks15 = 111100001011111000100110000101010101101010011001
Ks16 = 111100001011111000100110101010000010111010000100

2.2.2 CIFRADO

Continuando con la figura 2-31, una vez generadas las 16 subllaves, se sigue con el proceso de cifrado.

El estándar DES es un algoritmo de cifrado por bloques, si la información a ser cifrada se presenta como un flujo de bits lo que se debe hacer es dividir ese flujo de bits en bloques de 64 bits de longitud.

Para cifrar el texto plano, este debe ingresar en bloques de 64 bits, en el caso de que un bloque no cumpla con los 64 bits, se debe realizar un relleno. Para este

caso se realizará con bits ceros. La figura 2-41, indica el texto plano dividido en bloques de 64 bits.

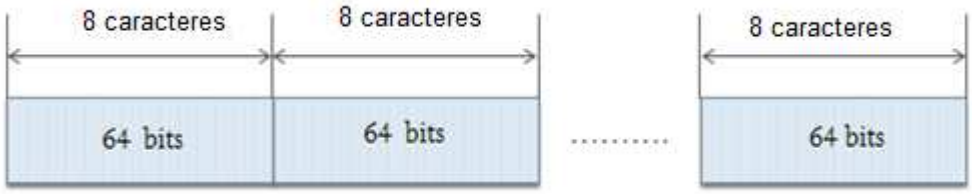


Figura 2-41. División del texto plano en bloques de 8 caracteres.

Ejemplo:

Si se considera al texto plano de 64 bits como ***TP***, cada bit correspondiente a ***TP*** se representa como:

$TP = TP_1, TP_2, TP_3, TP_4 \ldots TP_{64}$

TP =0110010011001010111001001101110011000010110111001100100001100001

Como se ve en la figura 2-31, los pasos a realizar son los siguientes:

- Permutación PI
- Rondas 1, 2, 3, 4, 5…..16
- Swap
- Permutación PI^{-1}

A continuación se detalla cada una de estas etapas:

2.2.2.1 Permutación IP

En la figura 2-31, se ilustra el proceso de cifrado donde la primera etapa corresponde a la permutación inicial ***IP.*** La permutación inicial ***IP*** recibe como parámetro de entrada los 64 bits del texto plano ***TP*** y estos son reordenados o permutados según la regla que se muestra a continuación en la tabla 2-10.

Permutación Inicial							
58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

Tabla 2-10. Permutación IP. [2.1]

Ejemplo:

$TP = TP_1, TP_2, TP_3, TP_4 \ldots TP_{64}$

TP= 0110010011001010111001001101110011000010110111001100100001100001

$IP = TP_{58}, TP_{50}, TP_{42}, TP_{34} \ldots TP_7$

IP = 1111111100000100011010111001001000000000111111110010100000101101

Diagrama de flujo de la permutación IP

El diagrama de flujo correspondiente a la permutación IP se indica en la figura 2-42.

2.2.2.2 Rondas

Continuando con el proceso de cifrado, la siguiente etapa es realizar una ronda, se le llama así porque este procedimiento debe ser repetido 16 veces. Cada ronda contiene operaciones específicas como permutaciones, operación xor y sustitución.

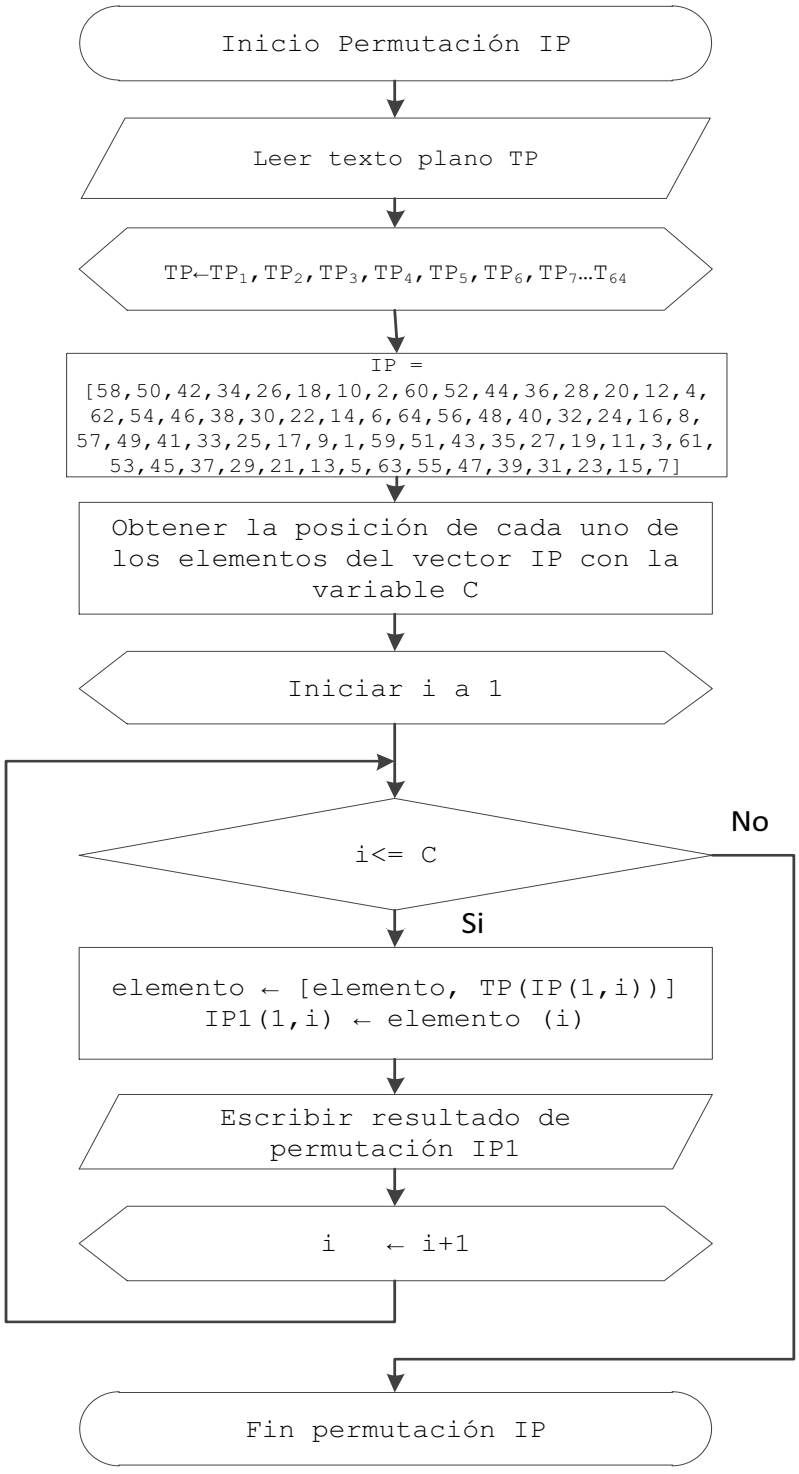


Figura 2-42. Diagrama de flujo correspondiente a la permutación IP.

Cada ronda recibe como datos de entrada 64 bits, la figura 2-43 muestra la estructura interna de una ronda.

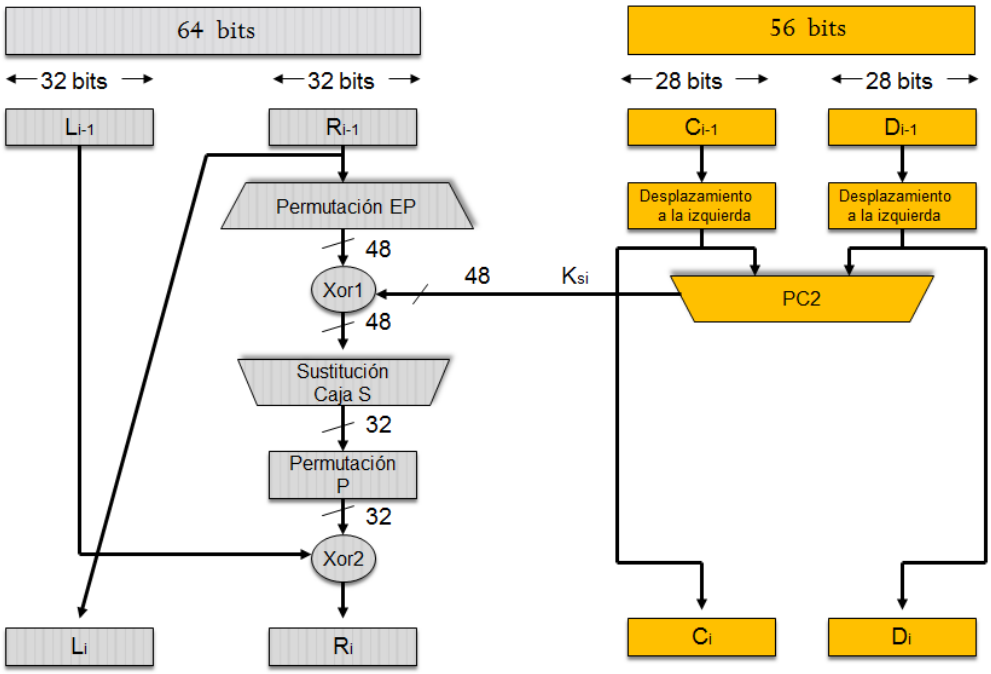


Figura 2-43. Estructura interna de una ronda.

A. Ronda 1

Cada una de las rondas como se indica en la figura 2-43, está constituída por las siguientes etapas:

A.1 División del resultado de la permutación IP.
A.2 Permutación EP.
A.3 Xor1.
A.4 Sustitución caja S.
A.5 Permutación P.
A.6 Xor2.

A.1 División del resultado de la permutación IP

Inicialmente se divide el resultado de la permutación inicial IP en dos bloques de 32 bits cada uno, el de la izquierda se denominara L por su nombre en inglés *left*, y el de la derecha R por su nombre en inglés *right*, como se muestra en la figura 2-43.

Ejemplo:

$IP = IP_1, IP_2, IP_3, IP_4 \ldots IP_{64}$

IP= 11111111000001000110101110010010

000000001111111100101000000101101

$L_1 = IP_1, IP_2, IP_3, IP_4 \ldots IP_{32}$

L_1 = 11111111000001000110101110010010

$R_1 = IP_{33}, IP_{34}, IP_{35}, IP_{36} \ldots IP_{64}$

R_1= 00000000111111110010100000101101

A.2 Permutación EP

Los 32 bits R, del resultado anterior ingresan a la permutación EP, se obtienen 48 bits a la salida. La permutación EP, sigue la regla de la tabla 2-11.

Permutación EP											
32	1	2	3	4	5	4	5	6	7	8	9
8	9	10	11	12	13	12	13	14	15	16	17
16	17	18	19	20	21	20	21	22	23	24	25
24	25	26	27	28	29	28	29	30	31	32	1

Tabla 2-11. Permutación EP. [2.1]

Como se puede observar en la tabla 2-11, algunas posiciones se replican permitiendo de esta manera expandir los 32 bits a 48 bits.

Ejemplo:

$R = R_1, R_2, R_3, R_4 \ldots R_{32}$

R = 00000000111111110010100000101101

$E/P = R_{32}, R_1, R_2, R_3 \ldots R_1$

E/P =100000000001011111111110100101010000000101011010

Diagrama de flujo de la permutación EP

El diagrama de flujo correspondiente a la permutación EP se indica en la figura 2-44.

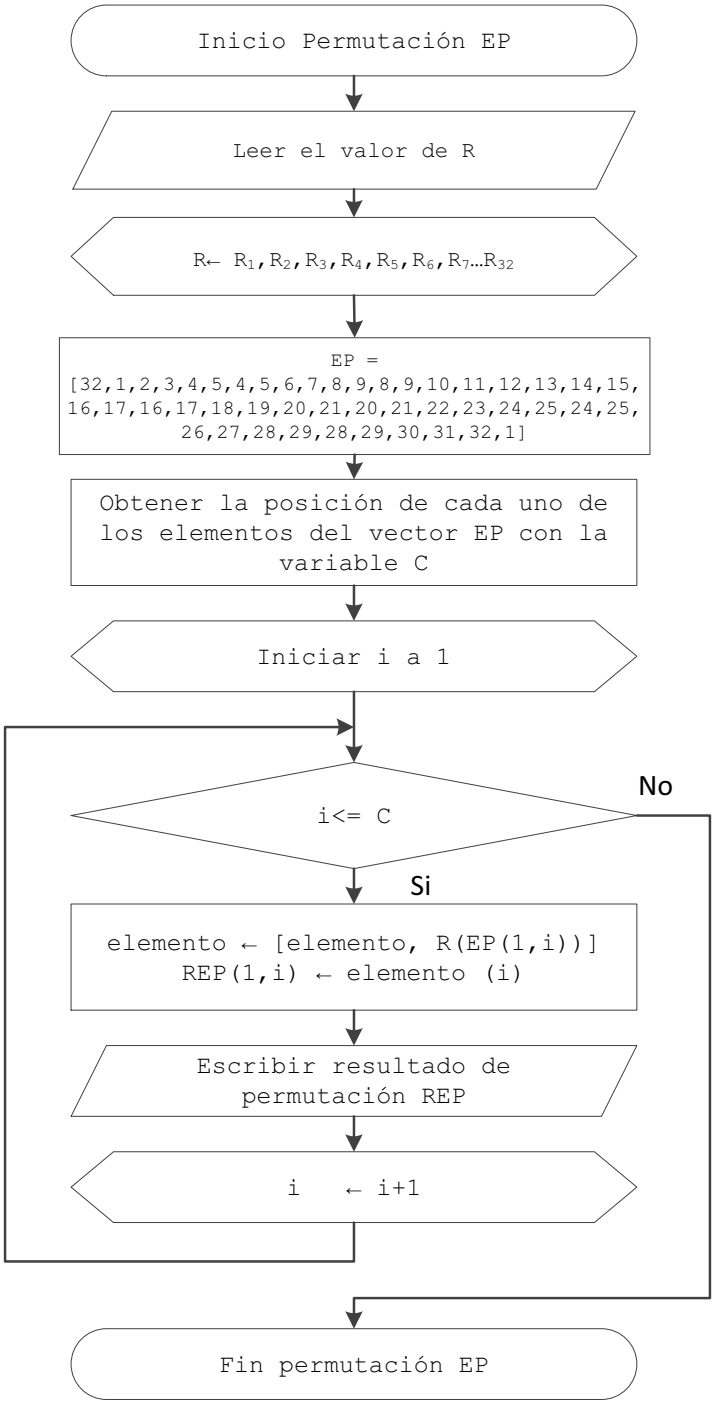


Figura 2-44. Diagrama de flujo correspondiente a la permutación EP.

A.3 Xor1

Como indica la figura 2-43, se debe realizar la operación con el resultado de la permutación EP y la subllave que corresponde a la ronda 1, para este caso corresponde ***Ks1***.

Ejemplo:

Xor1 = EP ⊕ Ks1
EP = 10000000 00010111111111101001010100000001010110 10
Ks1 = 11100000 10111110011001100001001100110000011100 01
Xor1 = 01100000 10101001100110001000011000110001001010 11

A.4 Sustitución cajas S

Como indica la figura 2-43, se realiza una sustitución de los 48 bits de la función Xor1. Estos son sustituidos por valores presentes en las cajas S, además las cajas cumplen la función de comprimir los bits a 32. El procedimiento de sustitución consiste en dividir los 48 bits, resultantes de la operación Xor1, en 8 grupos de 6 bits.

El primer grupo de 6 bits ingresa en la caja S1, el segundo grupo de 6 bits ingresa a la segunda caja S2 y se sigue con este proceso hasta terminar con los 8 grupos de bits. En la figura 2-45 se muestra este procedimiento.

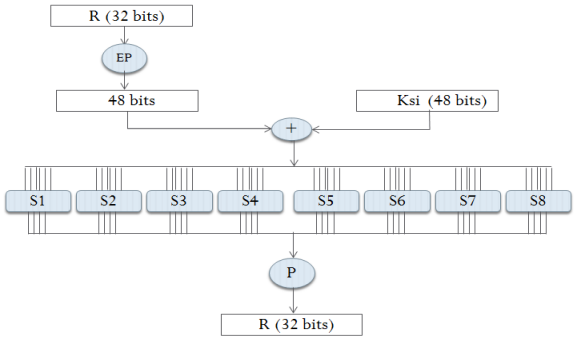


Figura 2-45. Proceso de sustitución de las cajas S.

Una caja S es una matriz de 4 filas por 16 columnas. Cada una de dichas cajas se detalla en la tabla 2-12.

Fila	Columna																S-Caja
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7	S_1
1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8	
2	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0	
3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13	
0	15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10	S_2
1	3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5	
2	0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15	
3	13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9	
0	10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8	S_3
1	13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1	
2	13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7	
3	1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12	
0	7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15	S_4
1	13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9	
2	10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4	
3	3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14	
0	2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9	S_5
1	14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6	
2	4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14	
3	11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3	
0	12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11	S_6
1	10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8	
2	9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6	
3	4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13	
0	4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1	S_7
1	13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6	
2	1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2	
3	6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12	
0	13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7	S_8
1	1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2	
2	7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8	
3	2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11	

[2.3]

Tabla 2-12. Cajas S del algoritmo DES.

Ejemplo:

Xor1 = 011000001010100110011000100001100011000100101011

Si → corresponde a las cajas S

Los valores de la operación Xor1 se dividen en 8 bloques de 6 bits, cada bloque se usará para ser sustituido por un valor en la caja S que le corresponda.

Xor1B1 = 011000 Xor1B2 = 001010

Xor1B3 = 100110 Xor1B4 = 011000

Xor1B5 = 100001 Xor1B6 = 100011

Xor1B7 = 000100 Xor1B8 = 101011

El primer bloque de 6 bits correspondiente a Xor1B1 es: 011000 dado que se trata del primer grupo de bits, este tiene que ser sustituido por los valores de la caja S1, luego se toman los bits de los extremos, los mismos

que indican la fila de la caja S1, los cuatro bits intermedios indican la columna de la caja S1, como se ilustra en la figura 2-46.

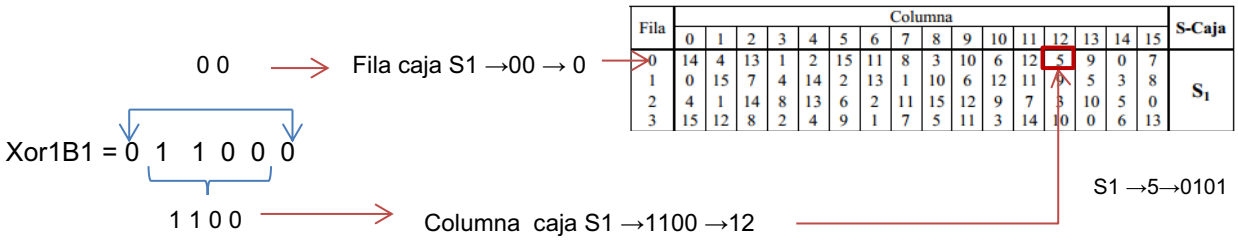


Figura 2-46. Ejemplo de selección de los valores de la Caja S1.

El resultado de los 7 bloques restantes, se indican en la tabla 2-13.

	XorBi	Caja S	Fila caja S	Columna caja S	Valor decimal caja S	Valor binario caja S
Xo1B1	011000	1	00	1100	5	0101
Xor1B2	001010	2	00	0101	11	1011
Xor1B3	100110	3	10	0011	9	1001
Xor1B4	011000	4	00	1100	11	1011
Xor1B5	100001	5	11	0000	11	1011
Xor1B6	100011	6	11	0001	3	0011
Xor1B7	000100	7	00	0010	2	0010
Xor1B8	101011	8	11	0101	10	1010

Tabla 2-13. Resultado cajas S del algoritmo DES.

El resultado final es la agrupación de cada uno de los 8 bloques de las cajas S, en un solo bloque denominado ***S,*** de 32 bits, como se muestra a continuación:

S = 01011011100110111011001100101010

Diagrama de flujo de la sustitución de cajas S

El diagrama de flujo correspondiente ***Sustitución cajas S,*** se indica en la figura 2-47.

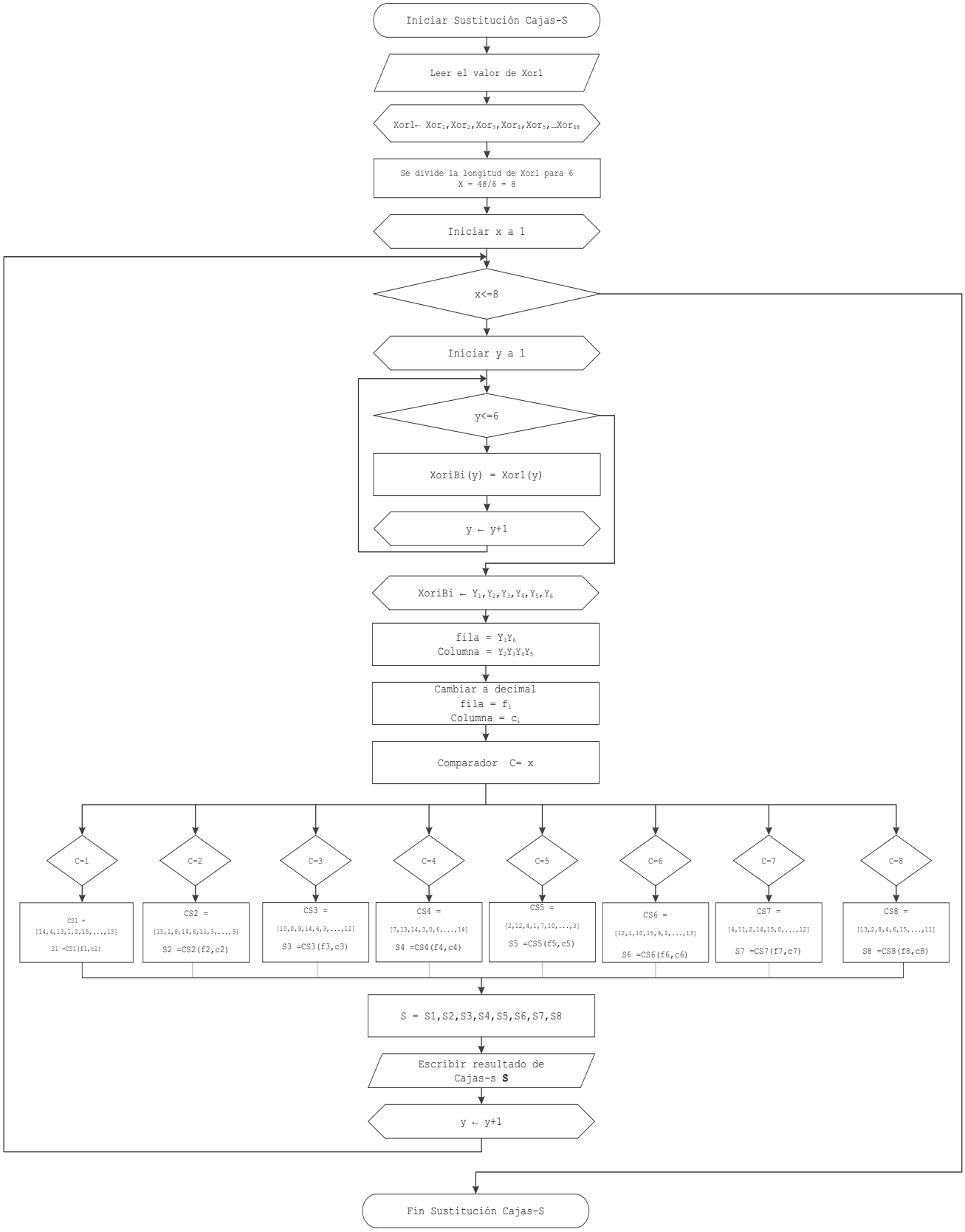


Figura 2-47. Diagrama de flujo de las cajas S.

A.5 Permutación P

El resultado de los 32 bits de la operación de las cajas-S, son permutados de acuerdo a la permutación P que se muestra en la tabla 2-14.

Permutación P							
16	7	20	21	29	12	28	17
1	15	23	26	5	18	31	10
2	8	24	14	32	27	3	9
19	13	30	6	22	11	4	25

Tabla 2-14. Permutación P. [2.1]

Ejemplo:

$S = S_1, S_2, S_3, \ldots\ldots S_{32}$

S = 01011011100110111011001100101010

$P = S_{16}, S_7, S_{20}, \ldots.. S_{25}$

P = 11101101011010101110010111000010

Diagrama de flujo de la permutación P

El diagrama de flujo correspondiente a la permutación P***,*** se indica en la figura 2-48.

A.6 Xor2

Como indica la figura 2-43, se debe realizar la operación xor con el resultado de la permutación P y el valor de L_{i-1}, el valor de ***L*** es el resultado de realizar la operación división de la permutación IP. El resultado de la operación Xor2, se denominara R_i.

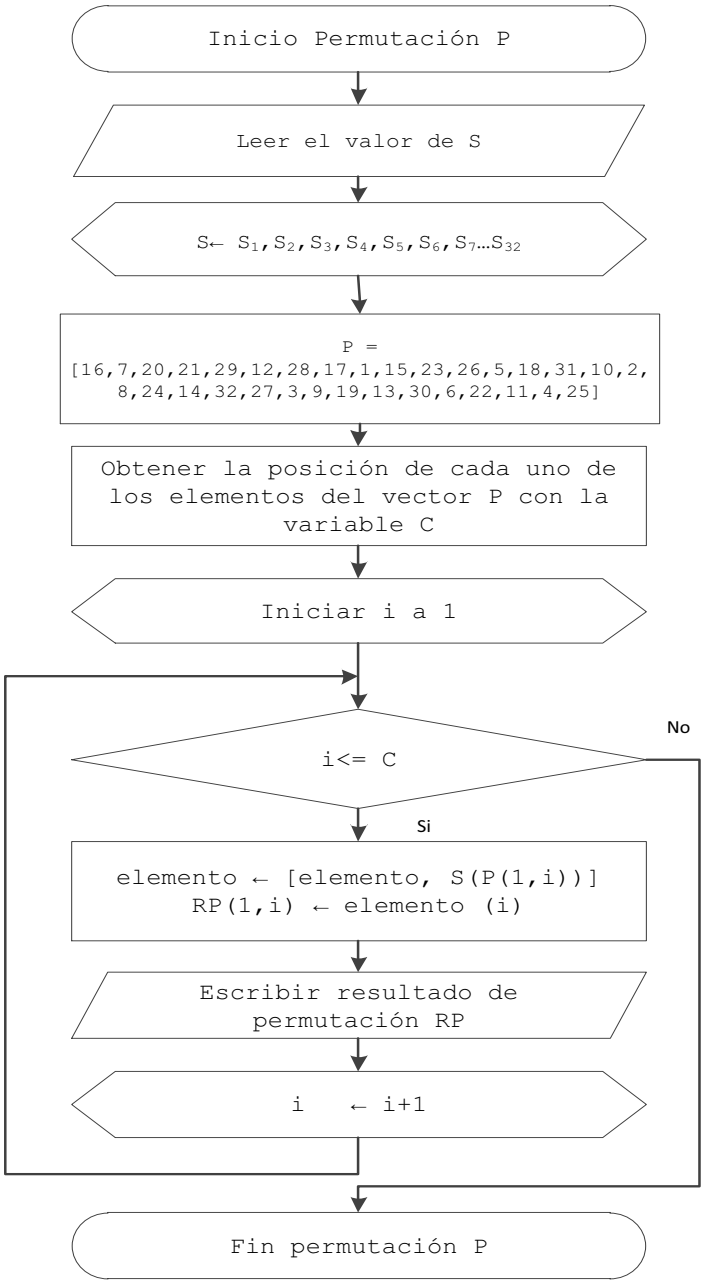


Figura 2-48. Diagrama de flujo de la permutación P.

Ejemplo:

Xor2 = L ⊕ P

L_1 = 11111111000001000110101110010010

P = 11101101011010101110010111000010

R_2 = 00010010011011101000111001010000

Como se muestra en la figura 2-43, los resultados de L_i y R_i, que se obtienen en la ronda 1, corresponden a los valores de entrada de la ronda 2. El resultado de la ronda 1 corresponde a:

$L_2 = R_1$

$L_2 =$ 00000000111111110010100000101

$R_2 =$ 00010010011011101000111001010000

B. Rondas R2, R3,……..R16

La etapa ronda se repite 16 veces como se indica en la figura 2-31, el de la ronda 1 corresponde a los datos de entrada de la ronda 2, el resultado de la ronda 2 corresponde a los datos de entrada de la ronda 3 así sucesivamente.

Ejemplo:

Los resultados de este ejemplo se realizan en base a la ronda 1 y la figura 2-43.

Ronda 1 = L2,R2

Ronda1= 00000000111111110010100000101101000100100110111010000111001010000

Ronda2= 00010010011011101000111001010000111100010100101000101011001001 01

Ronda3 = 11110001010010100010101100100101010011100001111000001001111111101

Ronda4= 01001110000111100001001111111101100001110011011101100001110000111

Ronda5= 10000111001101110110000111000111010100100101000010100010101111111

Ronda6= 01010010010100001010001010111111101000010111101101001000011111111

Ronda7= 10100001011110110100100001111111011101101100110101001001111100 1101

Ronda8= 01110110110011010010011111001101100011000111000011101000001010 11

Ronda9= 10001100011100001110100000101011110011101110011011110000111010011

Ronda10=11001110111001101111000111010011010001000101110010101001001111 11

Ronda11=01000100010111001010100100111111000111111011001000111011110111 01

Ronda12=00011111101100100011101111011101011010000010000010110010101111 00

Ronda13=01101000001000001011001010111100110001001000101101001110101101 01

Ronda14=11000100100010110100111010110101111011101000111011000001000011 11

Ronda15=11101110100011101100000100001111010101111110001000011110100010 00

Ronda16=01010111111000100001111010001000100110101101110000111010000101 01

2.2.2.3 SWAP

Como se indica en la figura 2-31, una vez que se realizan las 16 rondas, la siguiente etapa es la operación de swap. Al final de la ronda 16 se tiene los grupos L y R cada uno de 32 bits, estos intercambian posiciones el uno con el otro a esto se le conoce como la operación SWAP.

Ejemplo:

Ronda 16 = L_{r16}, R_{r16}

$L_{r16} = L_1, L_2, L_3 \ldots L_{16}$

L_{r16} = 01010111111000100001111010001000

$R_{r16} = R_1, R_2, R_3 \ldots R_{16}$

R_{r16} = 10011010110111000011101000010101

Swap = R_{r16}, L_{r16}

Swap=10011010110111000011101000010101010111111000100001111010001000

2.2.2.4 Permutación IP^{-1}

Finalmente el bloque de 64 bits resultante de la operación *SWAP* es permutado de acuerdo a la regla de la permutación inicial inversa (IP^{-1}) como muestra la figura 2-31. La regla de permutación se muestra en la tabla 2-15. El resultado de esta permutación corresponde al texto cifrado.

Permutación Inicial Inversa															
40	8	48	16	56	24	64	32	39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30	37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28	35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26	33	1	41	9	49	17	57	25

Tabla 2-15. Permutación IP^{-1}. [2.1]

Ejemplo:

Swap = $Sw_1, Sw_2, Sw_3 \ldots\ldots Sw_{64}$

Swap = 10011010110111000011101000010101010101111110001000011110100010000

$IP^{-1} = Sw_{40}, Sw_8, Sw_{48}....Sw_{25.}$

IP^{-1} = 10000001111011001001100101011110110111010010010010110000011100 10

Texto cifrado = IP^{-1} = 81EC995EDD24B072

Diagrama de flujo de la permutación IP^{-1}

El diagrama de flujo correspondiente a la permutación IP^{-1}**,** se indica en la figura 2-49.

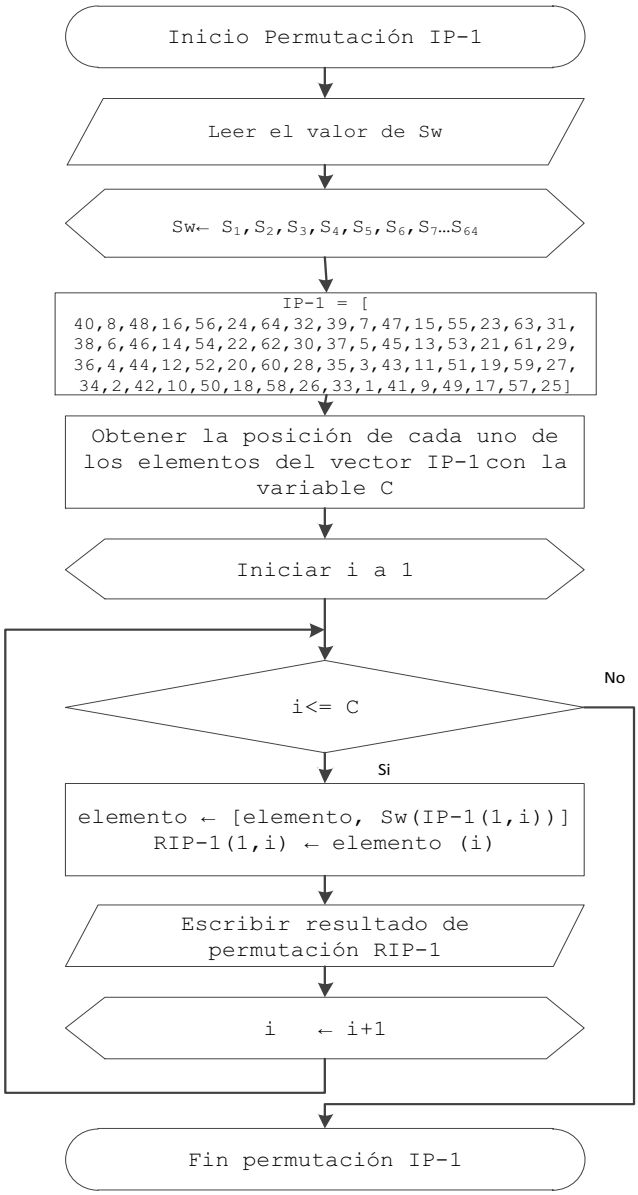


Figura 2-49. Diagrama de flujo de las permutación IP^{-1}.

2.2.3 DESCRIPCIÓN INTERFAZ GRÁFICA DE MATLAB PARA EL ALGORITMO DES

La interfaz gráfica, se desarrolló en MatLab 7.6.0 (R2008a). El usuario deberá ingresar un texto plano y la llave en formato ASCII. La interfaz gráfica se observa en la figura 2-50.

Figura 2-50. Interfaz gráfica desarrollada en MatLab para el algoritmo DES.

La interfaz diseñada es muy sencilla de usar, únicamente el usuario debe ingresar el texto que desee cifrar, colocar una clave y presionar el botón ***Enter***. Si el usuario desea cifrar otro texto solo debe presionar el botón ***Nuevo*** y este reinicializa el programa.

La figura 2-51, muestra el diagrama de flujo acerca del comportamiento de la interfaz. El código fuente correspondiente a las etapas de cifrado e interfaz gráfica se indica en el anexo 8.

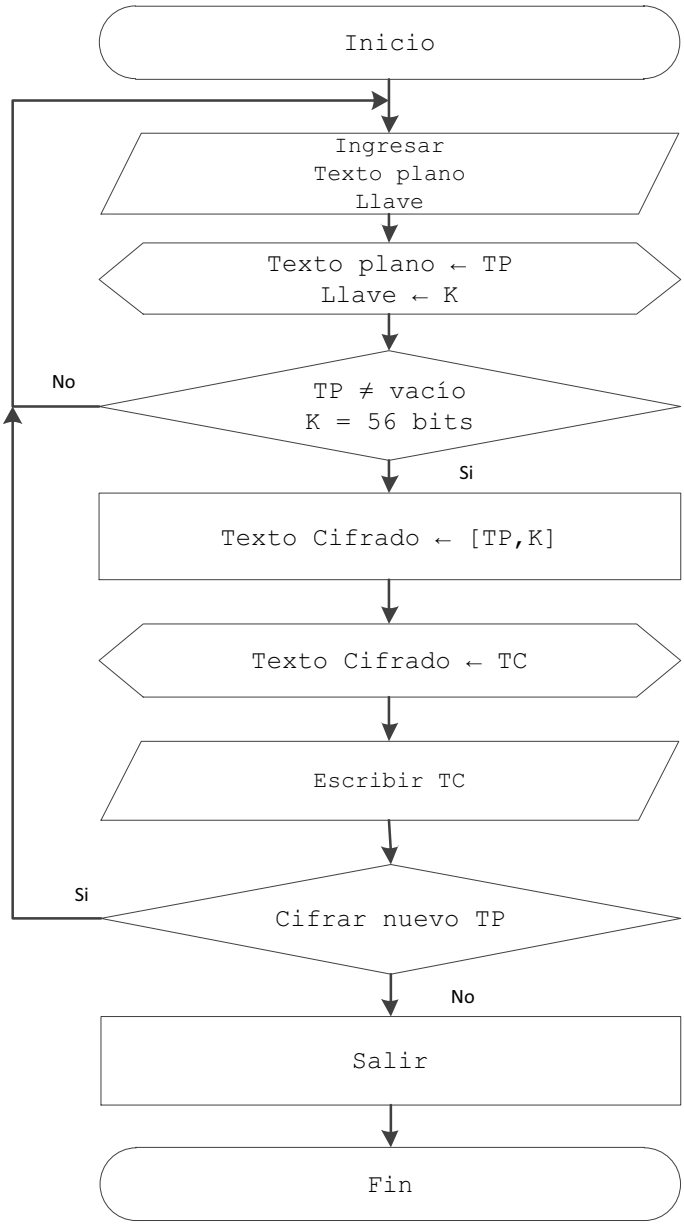


Figura 2-51. Diagrama de flujo del comportamiento de la interfaz en MatLab.

REFERENCIAS

[2.1] W. Stallings, *Cryptography and Network Security Principles and Practices.* Segunda Edición, 1995.

[2.2] P. Chu, *FPGA Prototyping by VHDL Examples*. Primera Edición, 2008.

[2.3] (1999) Descripción del algoritmo DES, Jorge Sánchez.[Online]. Disponible: http://www.tierradelazaro.com/public/libros/des.pdf

CAPÍTULO 3

ESTÁNDAR DE CIFRADO AVANZADO (AES)

3.1 INTRODUCCIÓN AES

El estándar AES, se encuentra descrito en la publicación FIPS PUB 197 (*Federal Information Processing Standards Publications*), que es emitida por el NIST (*National Institute of Standards and Technology*) de los Estados Unidos de Norteamérica.

En 1997 el NIST realizó un concurso de propuestas para el desarrollo de un nuevo estándar de cifrado avanzado (AES), el mismo que debería cumplir con las siguientes condiciones:

- Ser tan robusto, o más que, 3DES.
- Ser un cifrador simétrico de bloque, con una longitud de 128 bits por bloque.
- Permitir longitudes de llave de: 128, 192 y 256 bits.

Los criterios de evaluación se basarían en:

- Seguridad.
- Eficiencia computacional.
- Requisitos de memoria.
- Idoneidad para hardware y software.

3.1.1 PROCESO DE SELECCIÓN

En la primera etapa de evaluación se aceptaron 15 de los algoritmos propuestos. En agosto de 1999, se realizó la segunda etapa y se los redujo a cinco:

- MARS

- RC6
- Rijndael
- Serpent
- Twofish

Las siguientes razones determinaron, por qué no se eligió a los otros diez candidatos:

- CAST 256: Este algoritmo era similar al algoritmo Serpenter, pero con un alto costo de implementación.
- Crypton: Comparable con Rijndael y Twofish, pero con un margen de seguridad bajo.
- DFC: Margen de seguridad bajo y mal rendimiento en procesadores de 64 bits.
- E2: Comparable con Rijndael y Twofish en la estructura, pero con un margen de seguridad bajo y un alto costo de implementación.
- SAFER: Alto margen de seguridad similar a Serpenter, pero demasiado lento.

Luego de que el NIST anunció a los cinco finalistas, una notable contribución se originó a partir de la NSA: la presentación de los resultados de las simulaciones de rendimiento de hardware realizadas para los cinco finalistas en las cuales Rijndael y Twofish tuvieron resultados excelentes, demostrando ser muy adecuados para la implementación en hardware, mientras que RC6 presentó un alto costo en la implementación en hardware debido al uso de 32 bits de multiplicación; MARS resultó ser bastante costoso en recursos de hardware.

El 2 de octubre del 2000, el NIST dio a conocer que Rijndael, podría ser seleccionado como el nuevo estándar del NIST (AES), por su seguridad y alto rendimiento a nivel de hardware y software. El estándar final fue publicado en noviembre del 2001.

Rijndael, fue desarrollado y presentado por dos criptógrafos belgas: Dr. Joan Daemen y Dr. Vincent Rijimen.

3.1.2 BLOQUES DE ENTRADA Y SALIDA

El tamaño de un bloque en el estándar AES es de 128 bits, tanto para el proceso de cifrado como para el proceso de descifrado. El estándar también soporta bloques de longitudes de llave de 128, 192 y 256 bits. Una representación gráfica se indica en la figura 3-1.

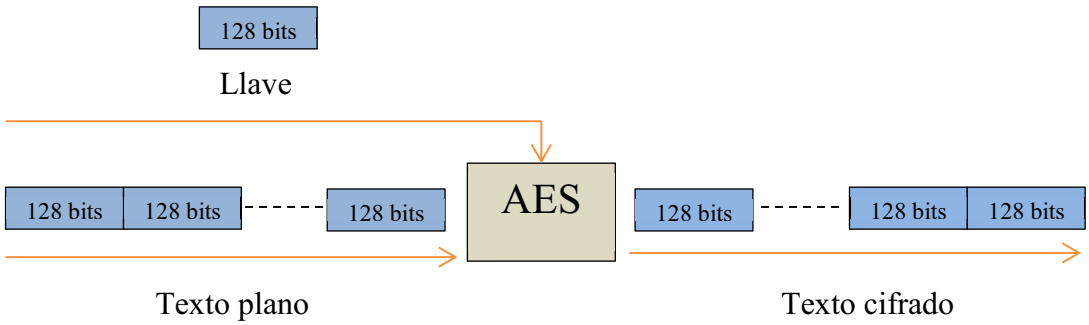


Figura 3-1. Bloques de entrada y salida AES, para la etapa de cifrado.

3.1.3 NOTACIÓN

Bytes.- La unidad básica para el procesamiento de AES es un **byte,** una secuencia de ocho bits tratado como una sola entidad. Un bloque de entrada, salida o una llave, puede ser expresado como un arreglo ***a,*** y ser representado de dos formas: a_n o a[n]. [3.4]

Ejemplos:

- Si la longitud de la llave es de 128 bits, entonces se tiene un bloque de 16 bytes. En este documento es el que se va a desarrollar.

- Si la longitud de la llave es de 192 bits, entonces se tiene un bloque de 24 bytes.
- Si la longitud de la llave es de 256 bits, entonces se tiene un bloque de 32 bytes.

Todos los bytes de AES se presentan como la concatenación de sus bits individuales, valores (0 ó 1), ordenados de la siguiente manera: ***{ b7, b6, b5, b4, b3, b2, b1, b0};*** siendo el bit más significativo el b7. Este puede ser mostrado como elemento de un campo finito usando una representación polinomial:

$$b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x^1 + b_0 = \sum_{i=0}^{7} b_i x^i \qquad [3.4]$$

Ejemplo:

{01000110}, puede ser representado como: $\boldsymbol{x^6 + x^2 + x}$, donde:

0	1	0	0	0	1	1	0
b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0

$$\sum_{i=0}^{7} b_i x^i = 0x^7 + 1x^6 + 0x^5 + 0x^4 + 0x^3 + 1x^2 + 1x^1 + 0 = x^6 + x^2 + x$$

Otra forma de indicar un byte es utilizando una representación hexadecimal, agrupando 4 bits, como se muestra a continuación:

Representación hexadecimal

$$\underbrace{0\ 1\ 0\ 0}_{b_7\ b_6\ b_5\ b_4}\ \underbrace{0\ 1\ 1\ 0}_{b_3\ b_2\ b_1\ b_0}$$

46

La tabla 3-1, indica los valores hexadecimales correspondientes a los 4 bits.

Patrón de bits	Carácter	Patrón de bits	Carácter	Patrón de bits	Carácter	Patrón de bits	Carácter
0000	0	0100	4	1000	8	1100	C
0001	1	0101	5	1001	9	1101	D
0010	2	0110	6	1010	A	1110	E
0011	3	0111	7	1011	B	1111	F

Tabla 3-1. Valores hexadecimales.

Estado.- Un estado es un arreglo de bytes ordenados de forma matricial. Esta distribución o arreglo se toma como base para las operaciones de cifrado y descifrado del estándar AES. Está compuesto de cuatro filas y el número de columnas se denota por ***Nb*** y es igual a la longitud del bloque dividido para 32, por lo tanto se puede tener los siguientes tipos de estados:

- Si la longitud del bloque es de 128 bits o 16 bytes, entonces se tiene un estado de 4 filas y 4 columnas, ya que ***Nb*** = 128 / 32 = 4.

  Se tiene 16 bytes ordenados como: $S_{f,c} = S_{0,0}, S_{1,0}, S_{2,0}, S_{3,0}, S_{0,1}.....S_{3,3}$.

$S_{0,0}$	$S_{0,1}$	$S_{0,2}$	$S_{0,3}$
$S_{1,0}$	$S_{1,1}$	$S_{1,2}$	$S_{1,3}$
$S_{2,0}$	$S_{2,1}$	$S_{2,2}$	$S_{2,3}$
$S_{3,0}$	$S_{3,1}$	$S_{3,2}$	$S_{3,3}$

- Si la longitud del bloque es de 192 bits o 24 bytes, entonces se tiene un estado de 4 filas y 6 columnas, ya que ***Nb*** = 192 / 32 = 6.

  Se tiene 24 bytes ordenados como: $S_{0,0}, S_{1,0}, S_{2,0}, S_{3,0}, S_{0,1}.....S_{3,5}$.

$S_{0,0}$	$S_{0,1}$	$S_{0,2}$	$S_{0,3}$	$S_{0,4}$	$S_{0,5}$
$S_{1,0}$	$S_{1,1}$	$S_{1,2}$	$S_{1,3}$	$S_{1,4}$	$S_{1,5}$
$S_{2,0}$	$S_{2,1}$	$S_{2,2}$	$S_{2,3}$	$S_{2,4}$	$S_{2,5}$
$S_{3,0}$	$S_{3,1}$	$S_{3,2}$	$S_{3,3}$	$S_{3,4}$	$S_{3,5}$

- Si la longitud del bloque es de 256 bits o 32 bytes, entonces se tiene un estado de 4 filas y 8 columnas, ya que ***Nb*** = 256 / 32 = 8.

  Se tiene 16 bytes ordenados como: $S_{f,c} = S_{0,0}, S_{1,0}, S_{2,0}, S_{3,0}, S_{0,1}.....S_{3,7}$.

$S_{0,0}$	$S_{0,1}$	$S_{0,2}$	$S_{0,3}$	$S_{0,4}$	$S_{0,5}$	$S_{0,6}$	$S_{0,7}$
$S_{1,0}$	$S_{1,1}$	$S_{1,2}$	$S_{1,3}$	$S_{1,4}$	$S_{1,5}$	$S_{1,6}$	$S_{1,7}$
$S_{2,0}$	$S_{2,1}$	$S_{2,2}$	$S_{2,3}$	$S_{2,4}$	$S_{2,5}$	$S_{2,6}$	$S_{2,7}$
$S_{3,0}$	$S_{3,1}$	$S_{3,2}$	$S_{3,3}$	$S_{3,4}$	$S_{3,5}$	$S_{3,6}$	$S_{3,7}$

Ejemplo:

16 bytes ordenados como: $S_{f,c}$ = 3b 6d 11 61 56 ab e2 36 ae ff 55 08 c9 11 4a 33

3b	56	ae	c9
6d	Ab	ff	11
11	e2	55	4a
61	36	08	33

Como se indicó anteriormente, la longitud de la llave puede tener 128, 192 o 256 bits. La longitud de la llave se la representa por ***Nk***, la misma que puede tomar valores de 4, 6 o 8, estos valores reflejan el número de columnas de la llave de cifrado. También se debe considerar que el número de rondas, representado por ***Nr***, depende del tamaño de la llave, es decir que ***Nr*** depende del valor de ***Nk***: Por

ejemplo: Nr = 10 cuando Nk = 4, Nr = 12 cuando Nk = 6 y Nr = 14 cuando Nk = 8. La tabla 3-2, muestra estos resultados.

	Longitud de la Llave Nk	Tamaño del Bloque Nb	Número de Rondas Nr
AES-128	4	4	10
AES-192	6	4	12
AES-256	8	4	14

Tabla 3-2. Valores de Nk y Nr.

3.1.4 AES Y CAMPOS FINITOS

El rendimiento en procesadores de 8 bits constituye una cuestión importante, ya que la mayoría de tarjetas inteligentes tienen un procesador de 8 bits y muchas aplicaciones criptográficas corren en estas tarjetas inteligentes.

AES opera con 8 bits (1 byte). Según la teoría de campos finitos, todos los bytes del algoritmo son interpretados como elementos de un campo finito, por lo tanto estos elementos pueden ser sumados y multiplicados.

Con ocho bits se puede representar enteros en un rango de 0 a 255 por lo que se concluye que el campo finito sería de orden 256. El orden de un campo finito es el número máximo de elementos del campo. Si se considera a **m** como el orden de un campo, entonces un campo con orden **m** existe si y solamente si **m** es una potencia de un número primo: m $= p^n$, para algún número entero **n** y un número primo **p**. La notación para un campo finito es: ***GF (p^n).***

En el anexo 9, se detalla la teoría de campos finitos y operaciones de aritmética modular, dos temas importantes para entender el funcionamiento del algoritmo AES. El campo finito que utiliza AES es ***GF (2^8),*** y el polinomio irreducible es $m(x) = x^8 + x^4 + x^3 + x + 1$. Para todas las operaciones que se realicen en el

proceso de cifrado y generación de subllaves se debe tomar en cuenta las propiedades de aritmética modular.

3.2 FUNCIONAMIENTO DE AES

El funcionamiento del algoritmo AES consiste de dos partes:

- *Proceso de generación de subllaves o expansión de la llave.*- Consiste en expandir la llave inicial, la misma que puede tener una longitud de 128, 192 y 256 bits. Luego de haber pasado por el proceso de expansión de la llave se generarán 44 subllaves o también llamadas palabras ***W***, cada una de estas palabras es utilizada en cada etapa del proceso de cifrado.

- *Proceso de cifrado*.- Consta de diez rondas y una ronda inicial. Cada ronda se encuentra conformada por operaciones básicas de bytes, entre las que se tiene: SubBytes, ShiftRows, MixColumns y Xor.

Una representación gráfica de cada una de las etapas de cifrado se ilustra en figura 3-2.

3.2.1 EXPANSIÓN DE LLAVE

El proceso de expansión, toma la llave de cifrado ingresada por el usuario, realiza una rutina de expansión para generar un total de ***Nb(Nr + 1)*** subllaves o palabras, las mismas que intervienen en las rondas del proceso de cifrado del algoritmo. La longitud de la llave ingresada por el usuario inicialmente es de 128 bits, luego estos bits se ordenan en un arreglo de 16 bytes organizados de forma matricial.

Cada palabra o subllave generada en el proceso de expansión de llave, se la denota de la siguiente manera:

W [i]

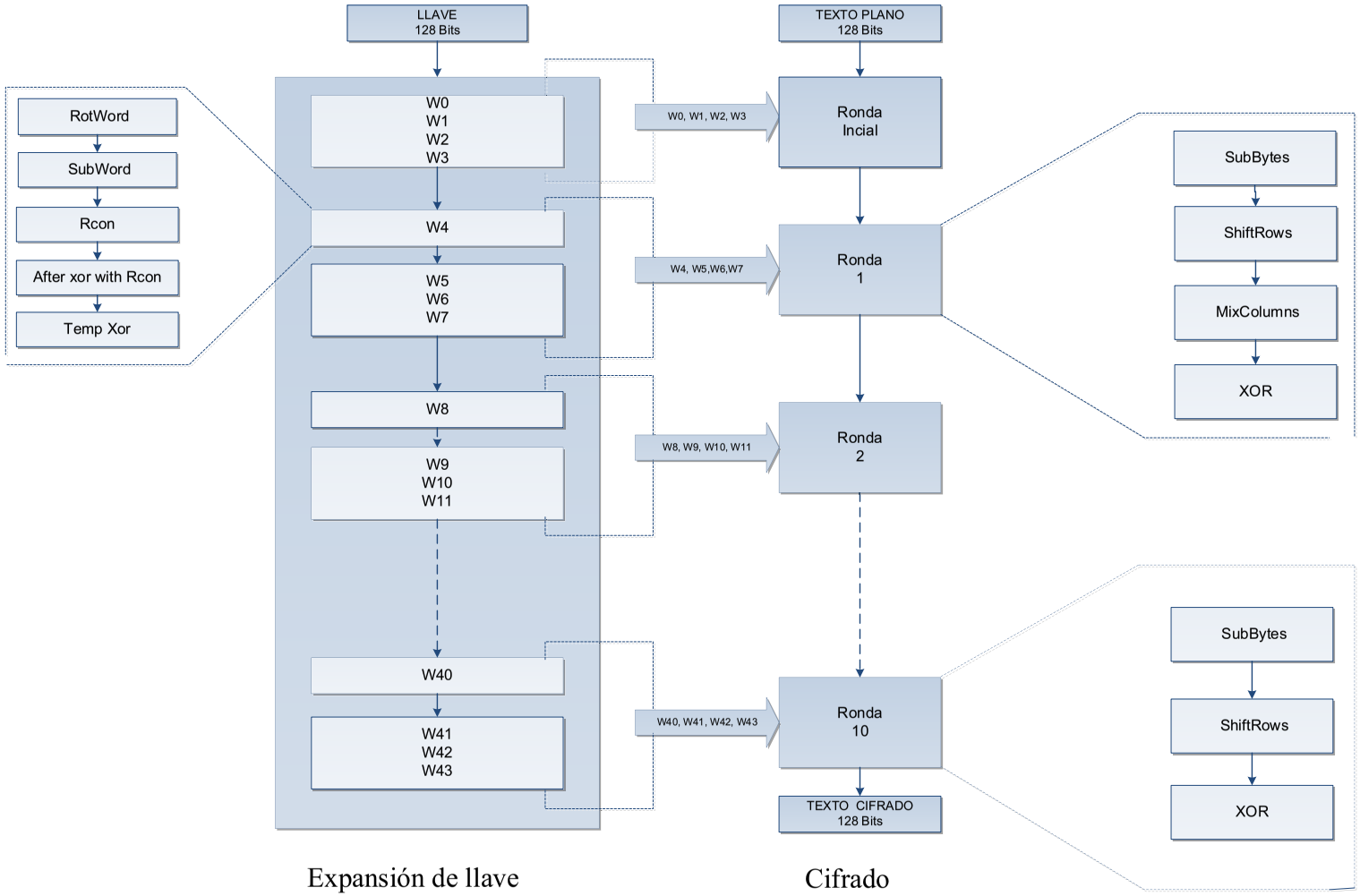


Figura 3-2. Representación gráfica cifrado AES, para el caso de una llave de 128 bits.

En donde **i** representa un número que sigue la regla **Nb(Nr + 1)**, como se indica en la tabla 3-2.

Ejemplo:

Si se tiene una longitud de llave de 128 bits, es decir, 16 bytes, entonces se tiene un número de rondas igual a 10, por lo tanto si se aplica la regla ***Nb(Nr + 1)***, se concluye que el número de palabras generadas será 44, numeradas de 0 a 43, de la siguiente manera:

W0, W1,W2,…,W43.

Para expandir la llave, se sigue dos procedimientos, uno para las llaves que son múltiplos de **Nk** y otro para las llaves que no son múltiplos de **Nk**. El valor de **Nk,** se indica en la tabla 3-2.

La generación de cada palabra ***W***, es de la siguiente manera:

- Las palabras W[0], W[1], W[2] y W[3], se generan directamente con los valores de la llave ingresada.
- Las palabras múltiplos de Nk =4, correspondientes a W[4], W[8], W[12], W[16], W[20], W[24], W[28], W[32], W[36], W[40], se generan siguiendo una serie de operaciones, que se van a explicar más adelante.
- Las palabras que nos son múltiplos de Nk =4, W[5], W[6],W[7],W[9]……W[43], se encuentran realizando una operación xor con la palabra que le precede.

3.2.1.1 Palabras W[0], W[1], W[2] y W[3]

Para encontrar estas palabras se organiza en forma de matriz, el valor de la llave original. Para realizar este ordenamiento se debe tener en cuenta la notación de la matriz de estado descrita anteriormente, es decir, si la llave original es representada por el siguiente flujo de bytes:

$$K = K_0, K_1, K_2, K_3, K_4, K_5, K_6, K_7, K_8, K_9, K_{10}, K_{11}, K_{12}, K_{13}, K_{14}, K_{15}$$

Organizando en formato matricial se tiene:

K_0	K_4	K_8	K_{12}
K_1	K_5	K_9	K_{13}
K_2	K_6	K_{10}	K_{14}
K_3	K_7	K_{11}	K_{15}

Tomando en cuenta la notación de la matriz de estado se tiene la matriz de la llave, en donde cada columna representa las primeras 4 palabras extendidas W[0], W[1], W[2] y W[3]:

W[0]	W[1]	W[2]	W[3]
$S_{0,0}$	$S_{0,1}$	$S_{0,2}$	$S_{0,3}$
$S_{1,0}$	$S_{1,1}$	$S_{1,2}$	$S_{1,3}$
$S_{2,0}$	$S_{2,1}$	$S_{2,2}$	$S_{2,3}$
$S_{3,0}$	$S_{3,1}$	$S_{3,2}$	$S_{3,3}$

Ejemplo:

Se tiene la llave ingresada por el usuario, en formato ASCII y en formato hexadecimal:

llave (ASCII) = A B C D E F G H I J K L M N O P

llave (HEX) = 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F 50

W[0]	W[1]	W[2]	W[3]
41	45	49	4D
42	46	4A	4E
43	47	4B	4F
44	48	4C	50

3.2.1.2 Palabras múltiplos de W[Nk]

Para encontrar la palabra W[Nk], así como sus múltiplos, las palabras W[nNk] (donde n va desde 1 hasta el número de rondas) pasan por las siguientes transformaciones:

A. RotWord
B. SubWord
C. Rcon [i/Nk]
D. Xor con Rcon
E. W[i-Nk]
F. W[i] = tem XOR w[i-Nk]

En este estudio se analizará la generación de las palabras múltiplos de Nk=4, con 10 rondas de cifrado (Nr = 10), generando las palabras:

W[4], W[8], W[12], W[16], W[20], W[24], W[28], W[32], W[36], W[40].

Como se indicó en la figura 3-2, una vez generadas las palabras W[0], W[1], W[2] y W[3], se debe generar la palabra W[4], que corresponde a la primera palabra múltiplo de Nk = 4. Cada una de las etapas de generación de W[4], se describen a continuación:

A. Transformación RotWord

Esta transformación toma la palabra W[Nk - 1] = [a_0, a_1, a_2, a_3] como entrada, realiza una permutación cíclica del primer elemento de la columna y lo desplaza hacia el último elemento de la columna, dando como resultado la palabra WRotWord = [a_1, a_2, a_3, a_0]. Para el caso de W[4]:

W[Nk - 1] = W [4 - 1] = W[3]

La figura 3-3, representa en forma gráfica la permutación cíclica.

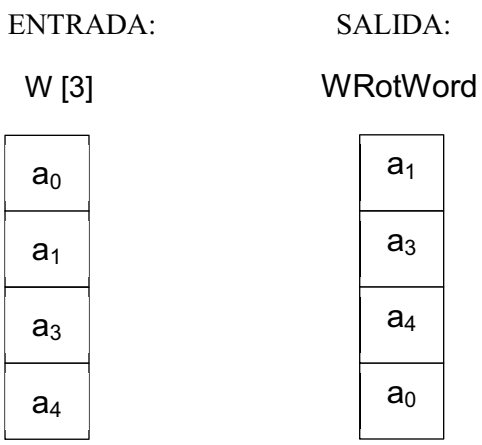


Figura 3-3. Representación gráfica de la permutación RotWord.

Ejemplo:

W[Nk - 1] = W [4 - 1] = W[3]

W[3] = [4D 4E 4F 50]

WAfterRotWord = [4E 4F 50 4D]

Diagrama de flujo de la transformación RotWord

El diagrama de flujo correspondiente a la transformación RotWord, se muestra en la figura 3-4.

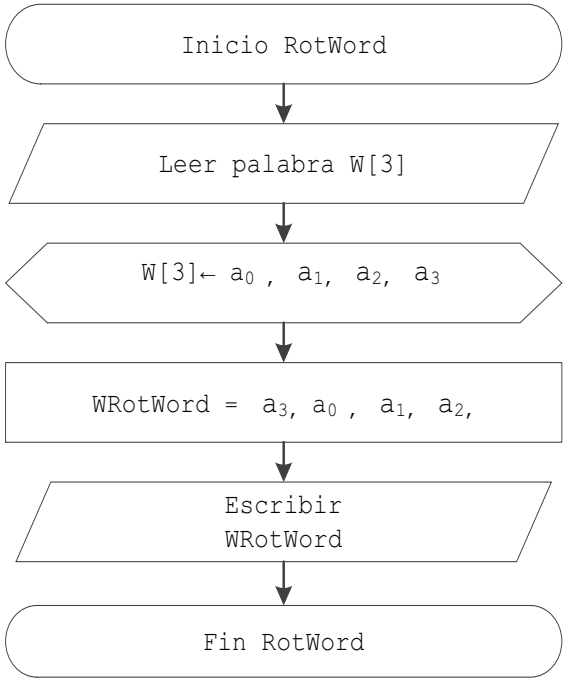


Figura 3-4. Diagrama de flujo de la transformación RotWord.

B. Transformación SubWord

La transformación SubWord tiene por objetivo romper la linealidad de la información de los datos originales, ya que para construir la misma se toman en cuenta criterios como la ***no linealidad*** y ***complejidad algebraica***.

No Linealidad.- En este criterio se toman aspectos como:

- Correlación**:** La máxima correlación entre los valores de entrada y salida debe ser tan pequeña como sea posible.
- Diferencia de Propagación de la Probabilidad: La máxima diferencia de propagación de probabilidad debe ser tan pequeña como sea posible.

*Complejidad Algebraica***.**- La expresión algebraica sobre el campo finito GF(2^8) tiene que ser compleja para que el algoritmo sea fuerte frente a un criptoanálisis diferencial.

Esta transformación toma como entrada una palabra de cuatro bytes. Estos son sustituidos uno a uno con un valor de la caja-S (tabla 3-3).

		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	01	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	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
	4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
	5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
	6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	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	0b	db
	a	e0	32	3a	0a	49	06	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	08
	c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
	d	70	3e	b5	66	48	03	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	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Tabla 3-3. Caja-S. [3.2]

El fundamento matemático de la generación de la caja-S, se describe en el anexo 10.

Ejemplo:

De la etapa anterior se obtuvo: WRotWord = [4E 4F 50 4D]. Cada byte se ubica en la tabla 3-3, su salida correspondiente. En este ejemplo se tiene los siguientes datos:

Entrada: 4 E 4 F 5 0 4 D

x y x y x y x y

Salida: 2 F 8 4 5 3 E 3

En la figura 3-5, se detalla de forma gráfica el valor correspondiente a 4E.

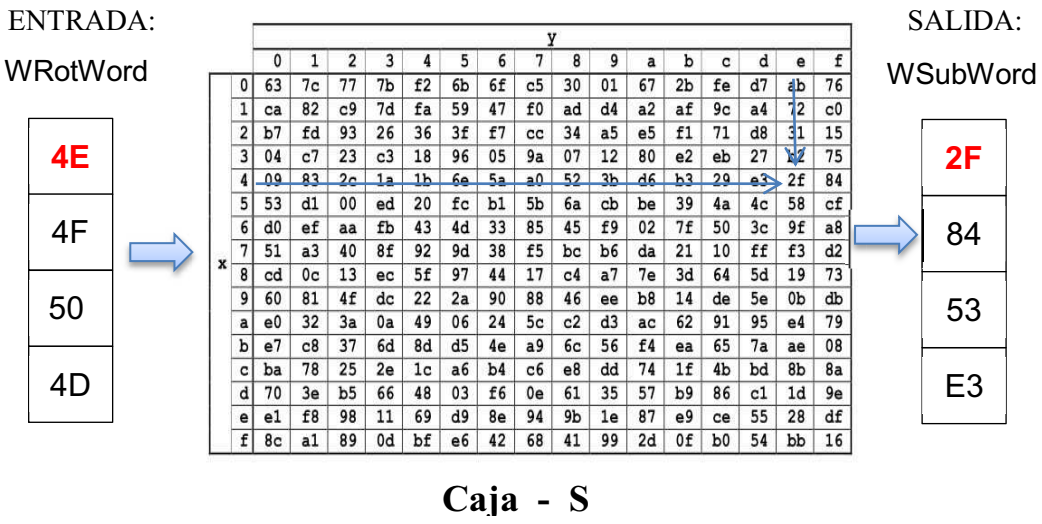


Figura 3-5. Representación gráfica de la transformación WSubWord.

Diagrama de flujo de la transformación SubWord

El diagrama de flujo correspondiente a la transformación SubWord se ilustra en la figura 3-6.

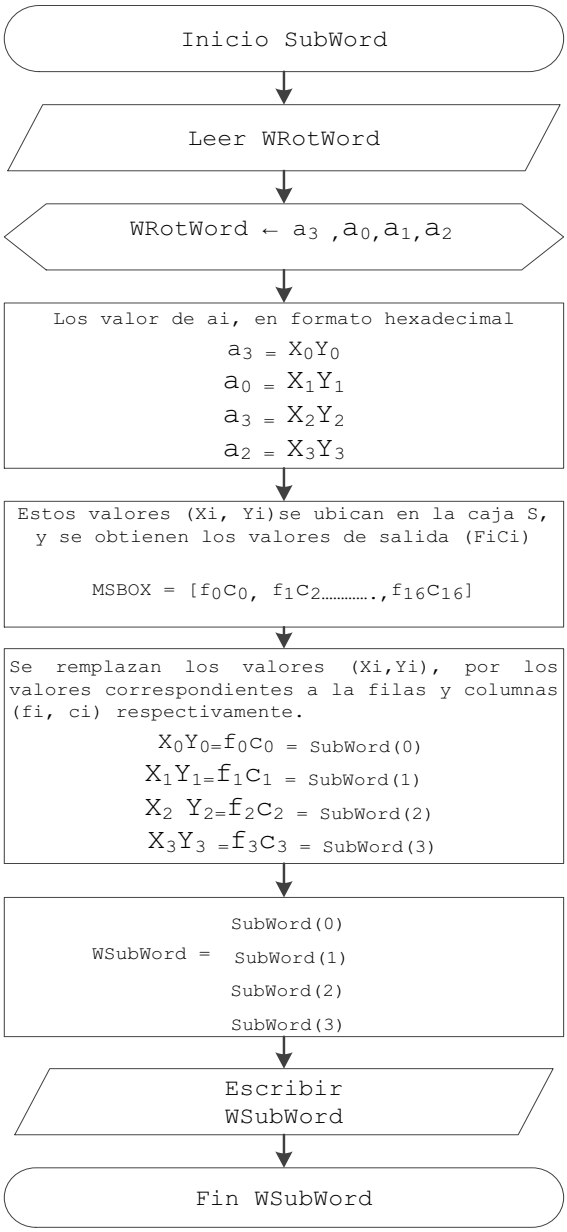


Figura 3-6. Diagrama de flujo de la transformación SubWord.

C. Transformación RCON

Esta transformación contiene los 10 valores constantes los mismos que son obtenidos de $\left[x^{i-1},\{00\},\{00\},\{00\}\right]$, cada valor es usado en las diferentes rondas del proceso de cifrado. Estos valores son los que se indican en la figura 3-7, y el fundamento matemático para conseguir estos valores se detallan en el anexo 10.

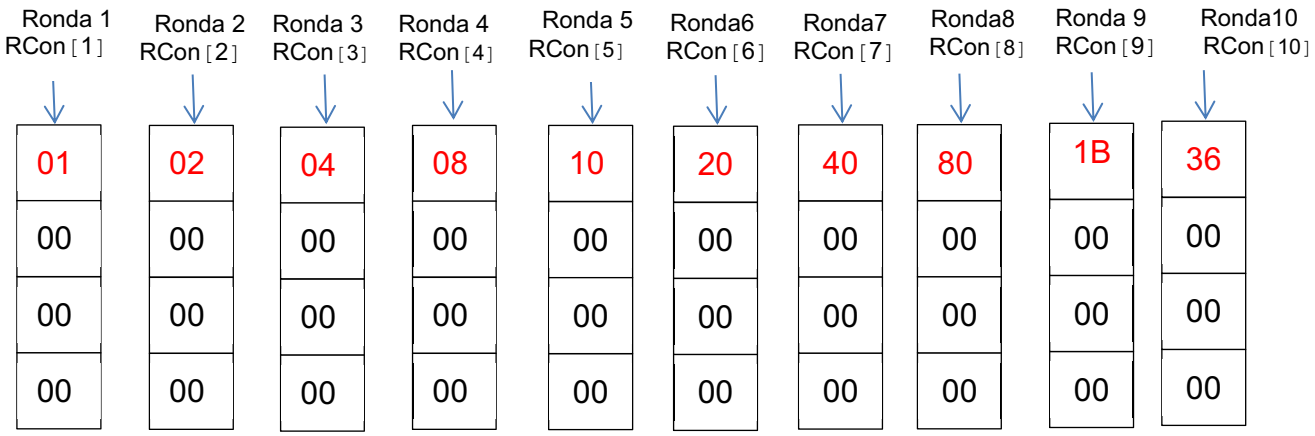


Figura 3-7. Representación gráfica de los valores de Rcon(i).

D. Transformación XOR con RCON

Consiste en realizar la operación xor, entre el resultado de SubWord y el valor de Rcon [i/Nk] correspondiente al número de ronda.

Ejemplo:

WSubWord = [2F 84 53 E3]

Rcon[i/Nk] = Rcon [4/4] = Rcon [1] = [01 00 00 00]

La representación gráfica de la operación xor, se indica en la figura 3-8.

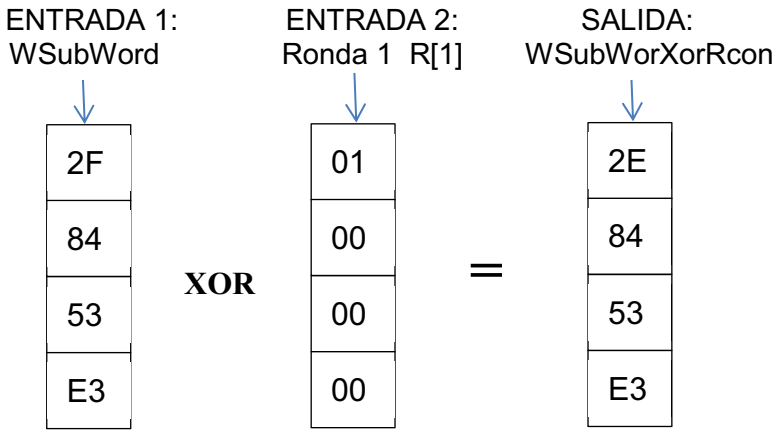


Figura 3-8. Operación xor entre WSubWord y Rcon [1].

Diagrama de flujo de transformación Xor con Rcon

El diagrama de flujo correspondiente a la transformación Xor con Rcon, se indica en la figura 3-9.

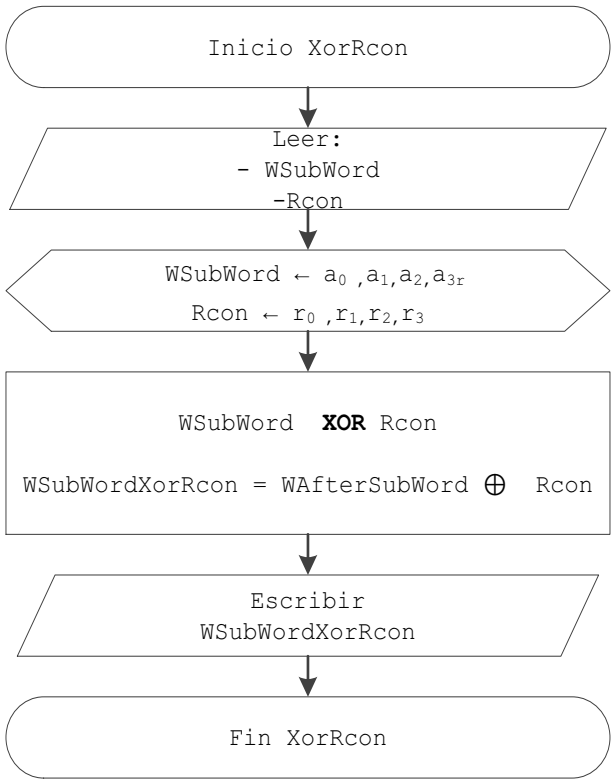


Figura 3-9. Diagrama de flujo de la transformación XorRcon.

E. W [i - Nk]

Consiste en encontrar la palabra w[i-Nk], donde, i corresponde al número de palabra múltiplo de Nk que se está calculando, y el valor de Nk = 4.

$$W [i - Nk] = W [4-4] = W[0]$$

Donde el valor de W[0], se obtuvo de los valores de la llave.

Ejemplo:

La representación gráfica de la operación W [i -Nk], se indica en la figura 3-10.

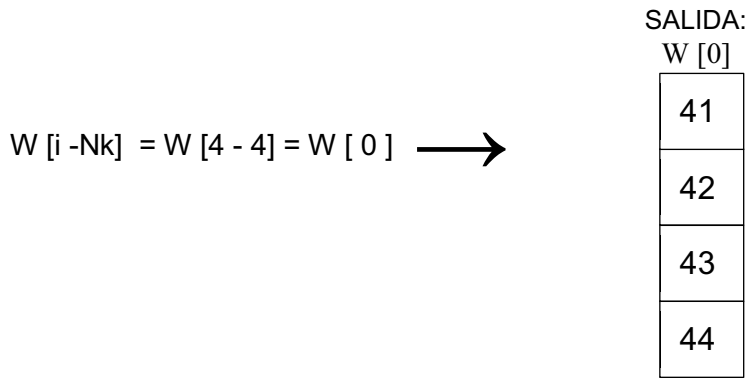


Figura 3-10. W[i- Nk].

F. W[i] =TEMP XOR W[i-Nk]

Esta operación consiste en realizar la operación xor con el resultado de la transformación *(Xor con Rcon)* y el valor de W [i – Nk], encontrado en el paso anterior. El resultado corresponde a la palabra W[4], como se indica en la figura 3-2.

Ejemplo:

WXorRcon = [2E 84 53 E3]
W[i-Nk] = W [4-4] = W [0]
W [0] = [41 42 43 44]

La representación gráfica se ilustra en la figura 3-11.

De la misma forma que se encontró W [4], se obtienen las siguientes palabras múltiplos de cuatro. En el caso de W[8], se inicia la operación de RotWord, con W[7]; para W[12] con W[11] y así sucesivamente.

Diagrama de Flujo transformación Xor con Rcon

El diagrama de flujo correspondiente a la transformación Xor con Rcon, se indica en la figura 3-12.

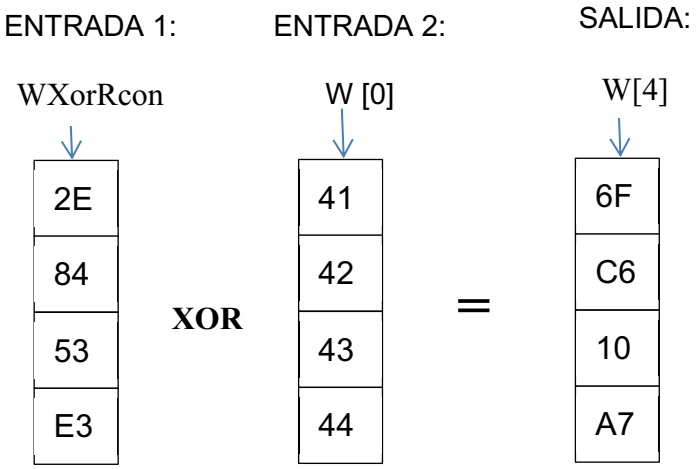


Figura 3-11. W [i] = tem XOR W[i-Nk].

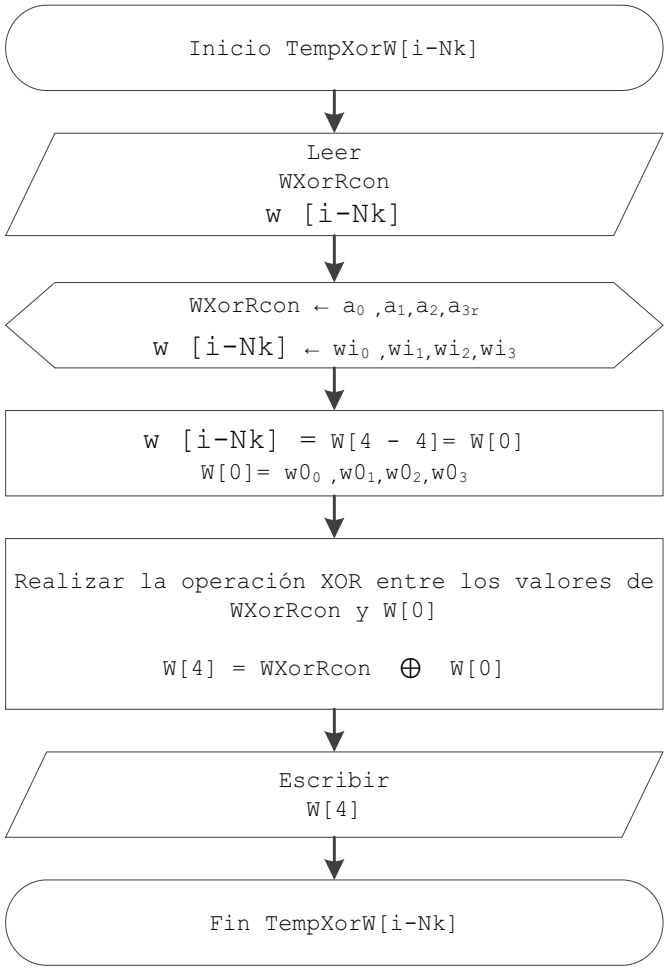


Figura 3-12. Diagrama de flujo correspondiente a TempXorW[i-Nk].

3.2.1.3 PALABRAS NO MÚLTIPLOS DE Nk=4, W[5], W[6], W[7], W[9], ….W[43]

Se obtiene realizando un xor, entre la palabra que le precede a la que se desea encontrar y la palabra W[i-Nk], donde i corresponde a la palabra que se va obtener y Nk = 4. De esta forma se tiene:

W[5] = W[4] xor W[1]
W[6] = W[5] xor W[2]
W[7] = W[6] xor W[3]
W[8] = Se encuentra con la funciones de generación de palabra múltiplo de Nk.
W[9] = W[8] xor W[5]
W[10] = W[9] xor W[6]
W[11] = W[10] xor W[7]
W[12] = Se encuentra con la funciones de generación de palabra múltiplo de Nk.
W[13] = W[12] xor W[9], ……W[43] = W[42] – W[39].

El resultado de cada una de las palabras se indica en la tabla 3-4, en representación hexadecimal.

W[i]	Valor	W[i]	Valor	W[i]	Valor	W[i]	Valor
W[0]	41424344	W[11]	55E50529	W[22]	C31CB0C3	W[33]	871501FF
W[1]	45464748	W[12]	EF40B86A	W[23]	2E601EA6	W[34]	33102C01
W[2]	494A4B4C	W[13]	F7EBF213	W[24]	BFE4303B	W[35]	228220E7
W[3]	4D4E4F50	W[14]	8C8AA4C9	W[25]	48EB8F7D	W[36]	B050872C
W[4]	6FC610A7	W[15]	D96FA1E0	W[26]	8BF73FBE	W[37]	374586D3
W[5]	2A8057EF	W[16]	4F72595F	W[27]	A5972118	W[38]	0455AAD2
W[6]	63CA1CA3	W[17]	B899AB4C	W[28]	77199D3D	W[39]	26D78A35
W[7]	2E8453F3	W[18]	34130F85	W[29]	3FF21240	W[40]	882E11DB
W[8]	322B1D96	W[19]	ED7CAE65	W[30]	B4052DFE	W[41]	BF6B9708
W[9]	18AB4A79	W[20]	4F96140A	W[31]	11920CE6	W[42]	BB3E3DDA
W[10]	7B6156DA	W[21]	F70FBF46	W[32]	B8E713BF	W[43]	9DE9B7EF

Tabla 3-4. Palabras W[i].

Diagrama de flujo de las palabras no múltiplos de Nk=4

El diagrama de flujo correspondiente a las palabras no múltiplo de cuatro, se indica en la figura 3-13.

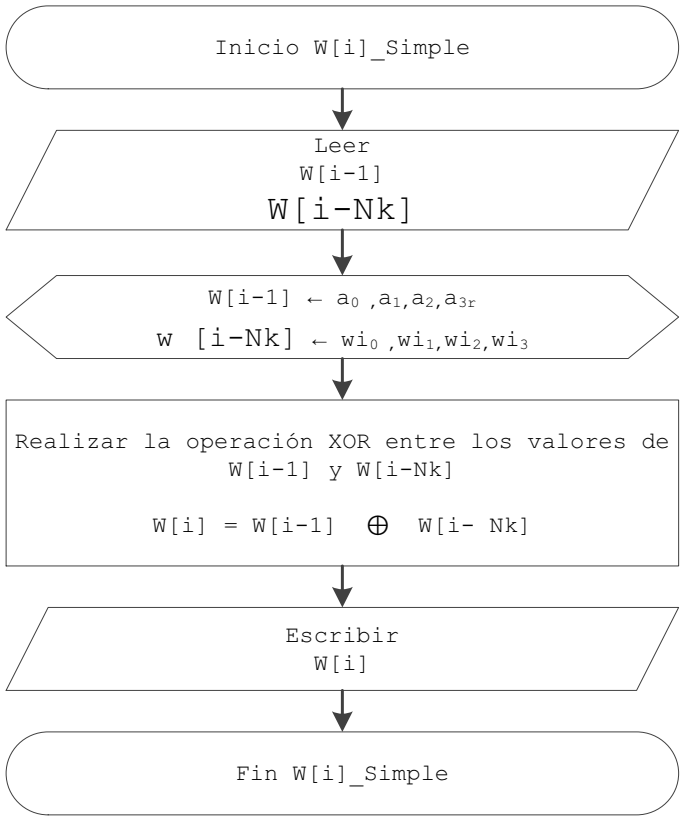


Figura 3-13. Diagrama de flujo correspondiente a palabras no múltiplos de cuatro.

3.2.2 CIFRADO

Como se indica en la figura 3-2, una vez encontradas las 44 palabras (W[0], W[1], W[2]….W[43]) en el proceso de expansión de llave, ya se puede realizar el cifrado.

Se ha tomado como ejemplo de texto plano un bloque de 128 bits denotado como:

Texto plano (ASCII) = F E R N A N D A F E R N A N D A
Texto plano (HEX) = 46 45 52 4E 41 4E 44 41 46 45 52 4E 41 4E 44 41

3.2.2.1 Ronda inicial

Consiste en realizar la operación XOR entre el texto plano y el valor de la llave, ingresados por el usuario, ambos puestos en forma matricial.

Ejemplo:

Expresado en forma hexadecimal se tiene:

Texto plano (HEX) = 46 45 52 4E 41 4E 44 41 46 45 52 4E 41 4E 44 41

Llave (HEX) = 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F 50

La representación matricial de la ronda inicial, se ilustra en la figura 3-14.

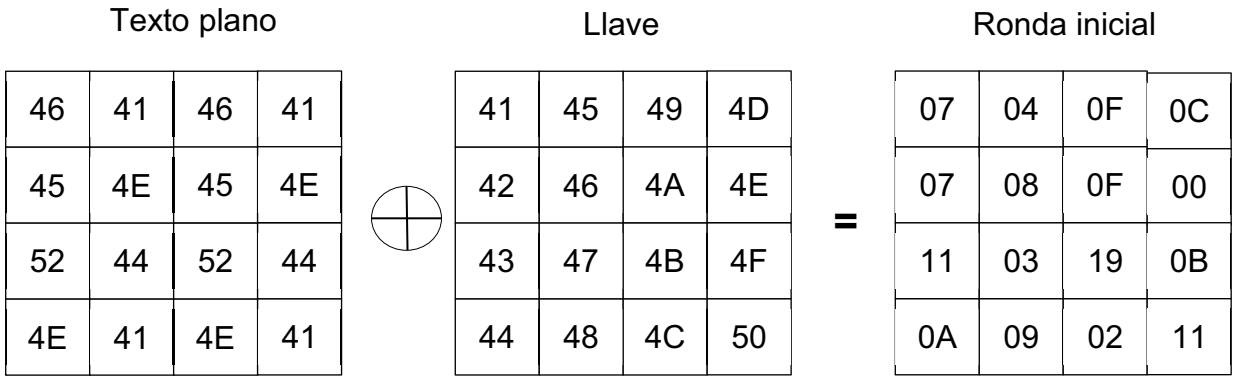


Figura 3-14. Ronda Inicial: resultado de la operación xor entre el texto plano y la llave.

Diagrama de Flujo de la ronda inicial

El diagrama de flujo correspondiente a la ronda inicial, se indica en la figura 3-15.

3.2.2.2 Ronda 1

Como se ilustra en la figura 3-2, la siguiente etapa corresponde a la Ronda 1, proceso que contiene cuatro operaciones:

A. SubByte
B. ShiftRows
C. MixColumns
D. Xor

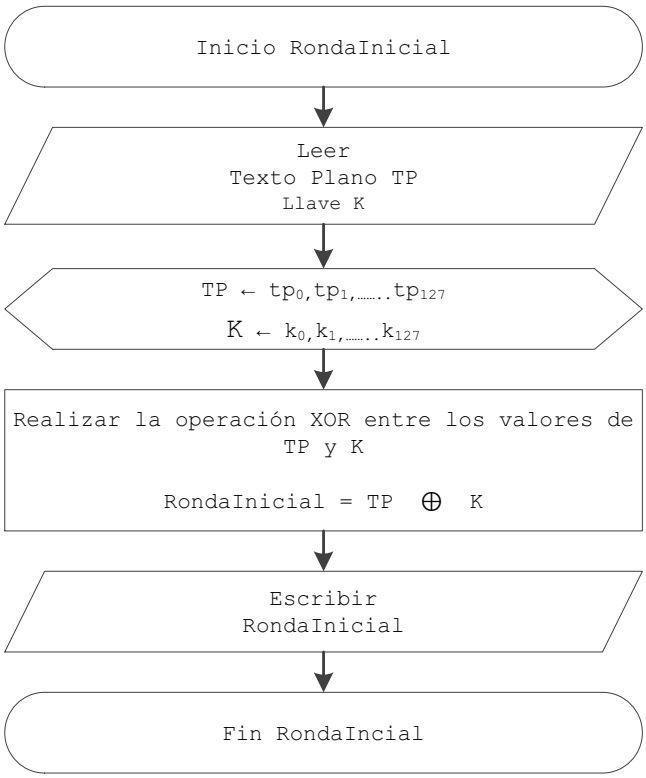


Figura 3-15. Diagrama de flujo correspondiente a la ronda inicial.

A. SubBytes

Para la sustitución de bytes se usa la caja S (tabla 3.3), con el mismo procedimiento que ya se empleó en la expansión de la llave en la figura 3-5.

Ejemplo:

La representación matricial de la sustitución de bytes, se ilustra en la figura 3-16.

El diagrama de flujo correspondiente a la operación SubByte, se indica en la figura 3-17.

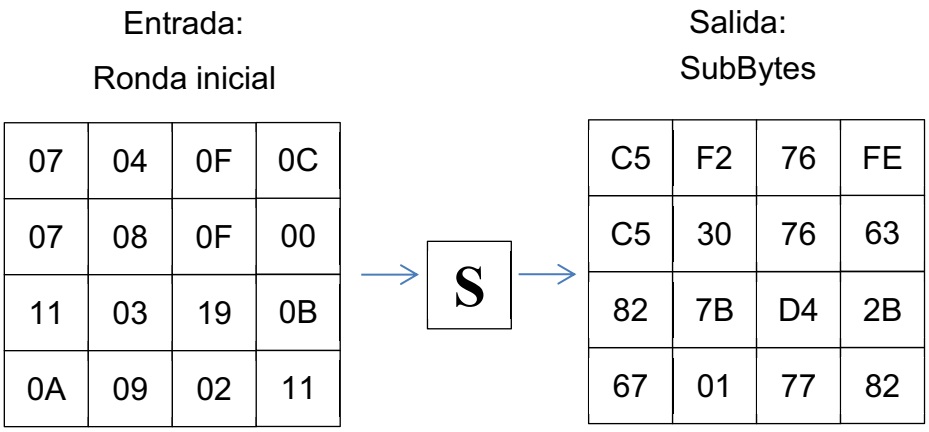


Figura 3-16. Ejemplo práctico: SubBytes, ronda 1.

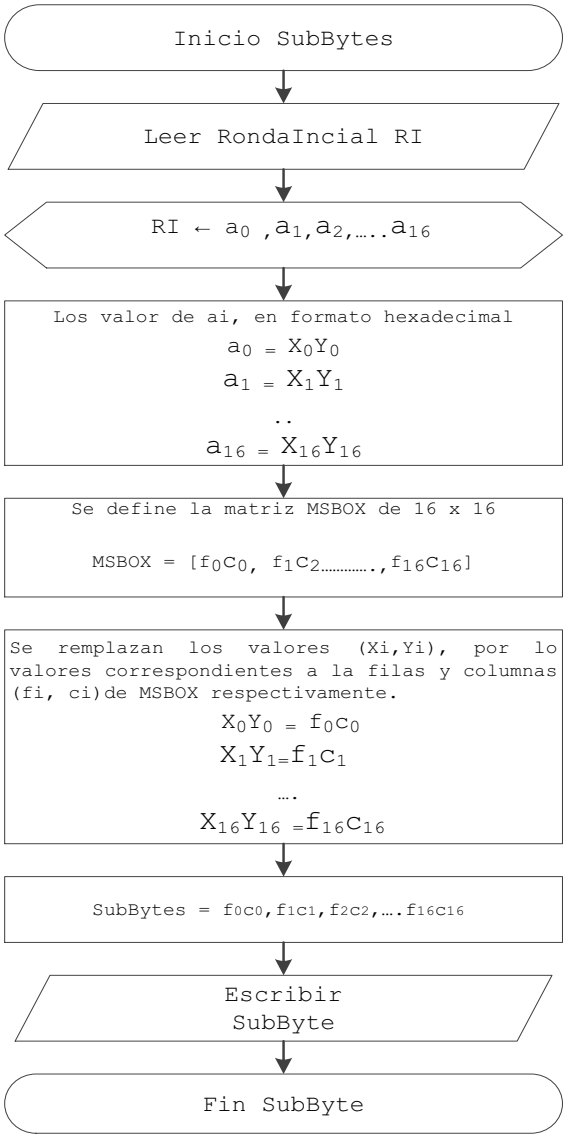


Figura 3-17. Diagrama de flujo correspondiente a SubByte.

B. ShiftRows

Consiste en aplicar el desplazamiento de filas al resultado de la matriz SubByte, tal como indica la figura 3-18.

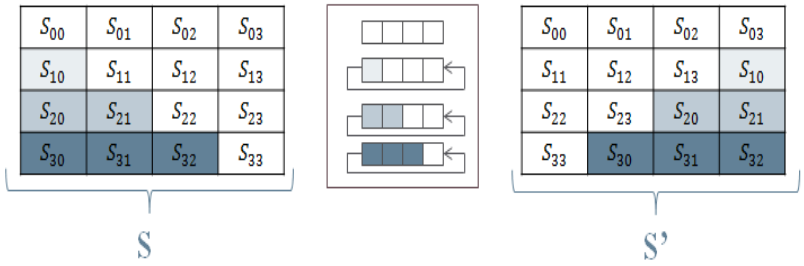


Figura 3-18. Permutación de ShiftRows.

Ejemplo:

La representación matricial de la permutación ShiftRows, se ilustra en la figura 3- 19.

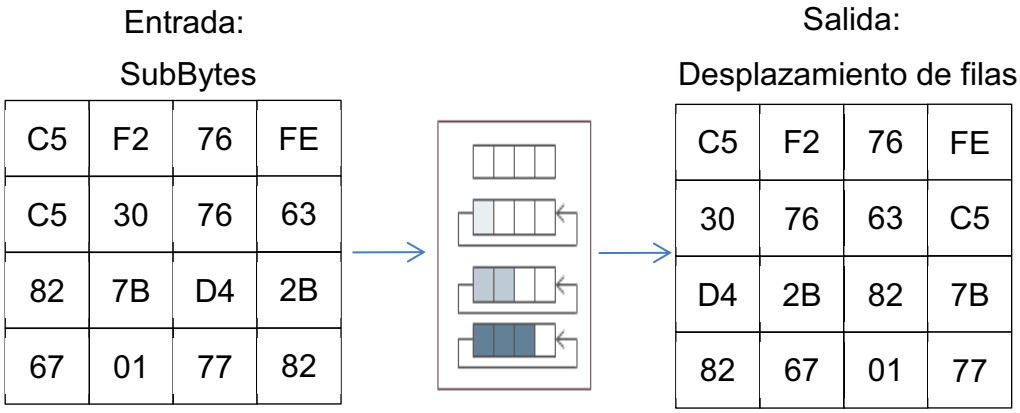


Figura 3-19. Ejemplo práctico: ShirfRows, ronda 1.

Diagrama de flujo de la operación ShiftRows

El diagrama de flujo correspondiente a la operación ShiftRows, se indica en la figura 3-20.

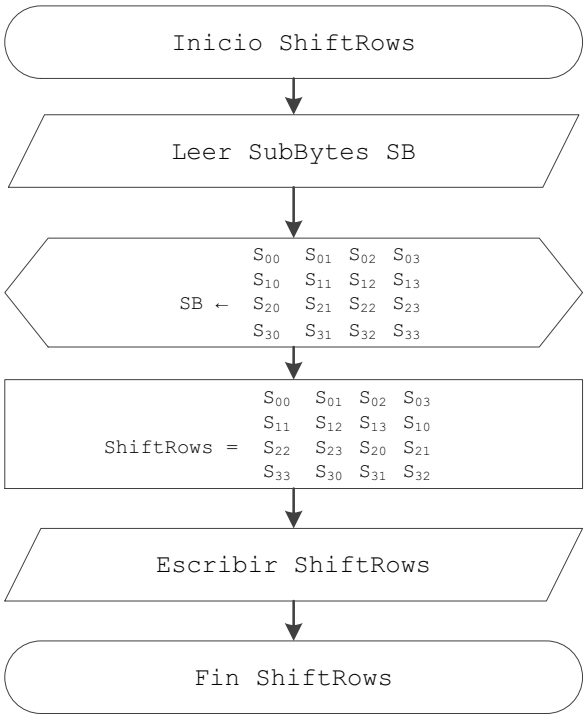


Figura 3-20. Diagrama de flujo correspondiente a ShiftRows.

C. MixColumns

Consiste en multiplicar cada columna de la matriz de entrada, por una columna constante que se obtiene en GF(2^8) [X]/(X^4+1).

Una representación gráfica se indica en la figura 3-21.

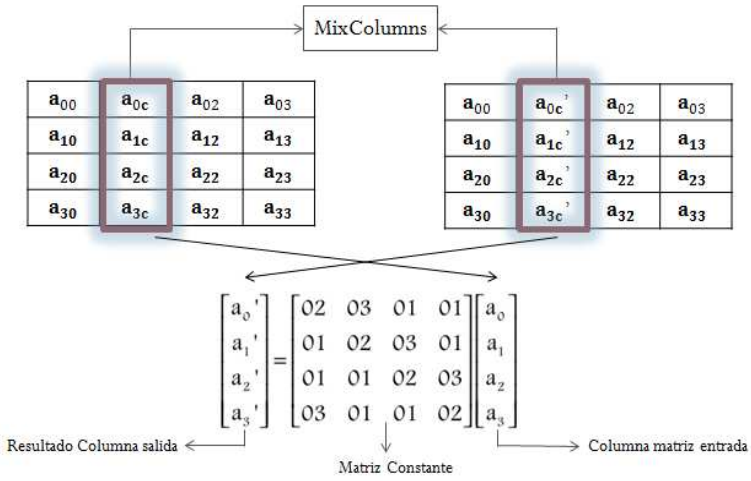


Figura 3-21. MixColumns.

Para obtener la representación matricial, las columnas se consideran polinomios sobre GF(2^8), y se multiplican modulo (x^4 + 1), con un polinomio fijo c(x), donde:

$$c(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}$$

El desarrollo matemático de esta sección se encuentra en el anexo 10.

Ejemplo:

La representación gráfica de la operación MixColumns, se ilustra en la figura 3-22.

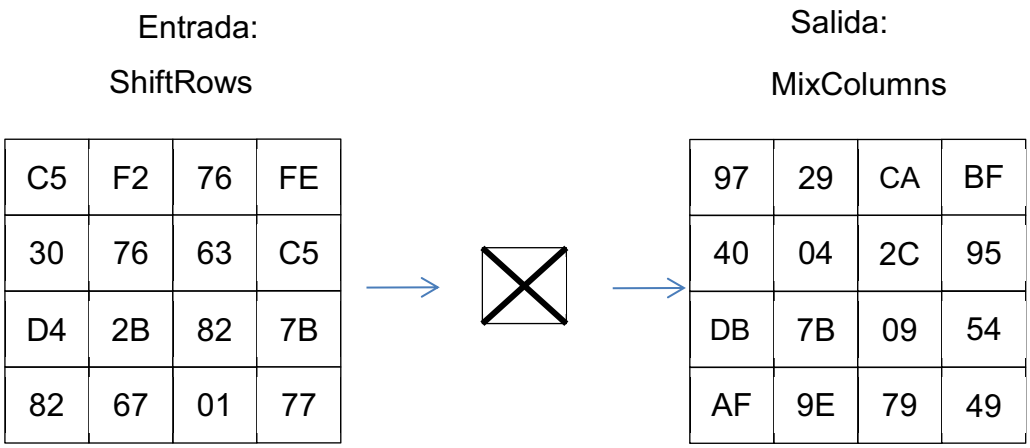


Figura 3-22. Ejemplo práctico: MixColumns, ronda 1.

Diagrama de flujo de la operación MixColumns

El diagrama de flujo correspondiente a la operación MixColumns, se indica en la figura 3-23.

D. Xor

Se realiza la operación xor, entre el resultado de la operación MixColumns (MC), y las palabras correspondientes a la ronda1 las cuales son: W[4], W[5], W[6], W[7].

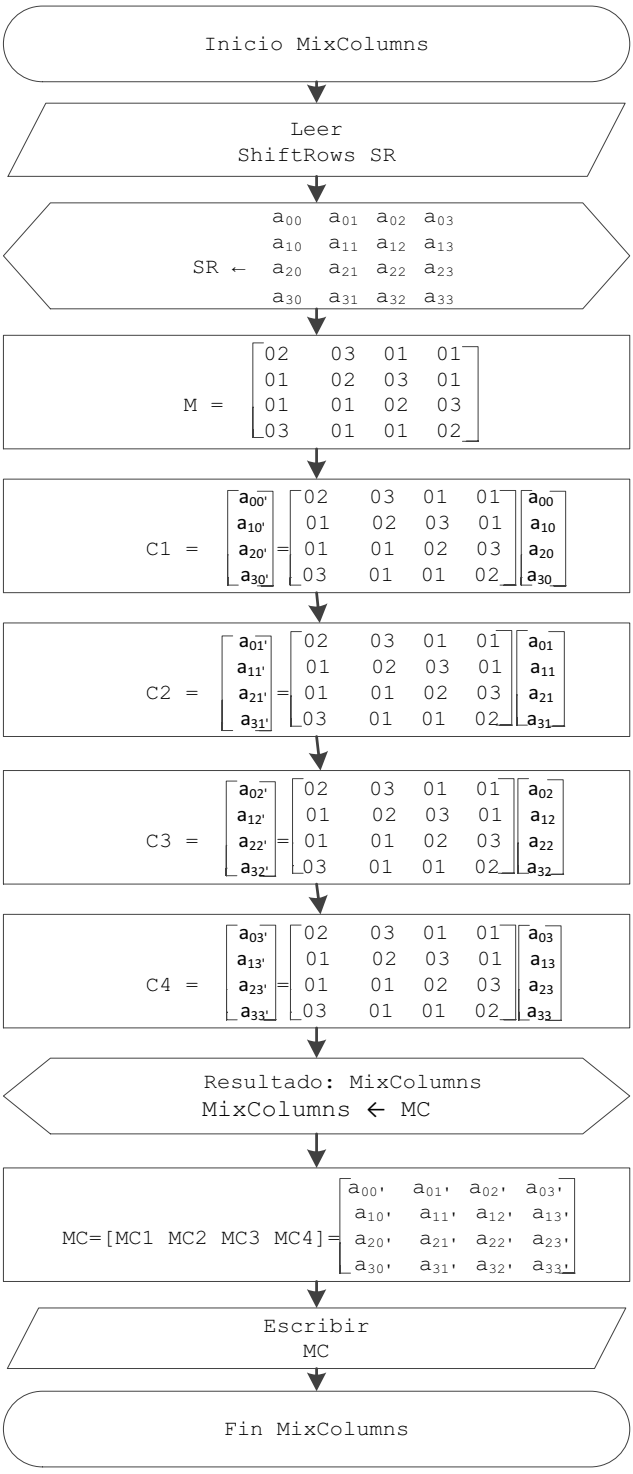


Figura 3-23. Diagrama de flujo correspondiente a MixColumns.

Ejemplo:

La representación gráfica del ejemplo práctico, se ilustra en la figura 3-24.

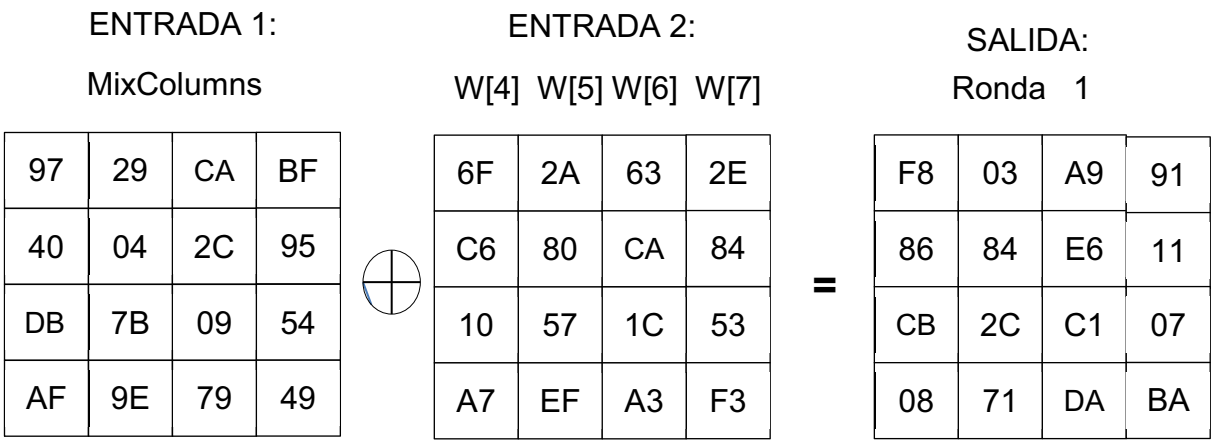


Figura 3-24. Ejemplo práctico: Xor, ronda 1.

Diagrama de flujo de la operación Xor

El diagrama de flujo correspondiente a la operación xor, se indica en la figura 3-25.

3.2.2.3 Rondas 2, 3, 4, 5, 6, 7, 8 y 9

Como se indica en la figura 3-2, para la ronda 2, se ejecuta con el dato de entrada correspondiente al resultado de la ronda 1 y las palabras W[8], W[9], W[10], y W[11]. El dato de entrada para la ronda tres será el resultado de la ronda dos y las palabras W[12], W[13], W[14], y W[15]. Lo mismo hasta la ronda nueve.

Realizando estos cálculos, se tiene los siguientes resultados:

Ronda 1: F803A9918684E611CB2C15070871DABA
Ronda 2: FC92E7A6CBB32F28B67EC7DBA0DF742A
Ronda 3: 000F33D19EBE3AE9C6FA21CADBB5335F
Ronda 4: 5202EA6285D1FEE03F65C59F2C4BC09C
Ronda 5: 75652834C5126E92044DF26C355DB8F6
Ronda 6: 15AE8CEB32E51C8AAB5C6FF64352C7AD
Ronda 7: 880B9C319FE93B25F2C0F303969D4E38

Ronda 8: 030702540FFC6E12DA952622F520728A

Ronda 9: 04694A962CEDE5253BD078481A5CA089

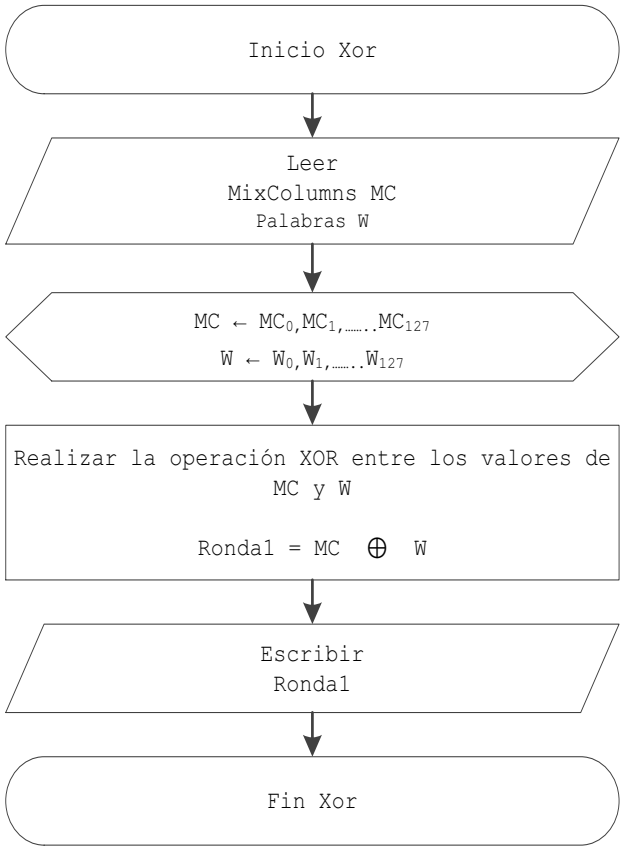


Figura 3-25. Diagrama de flujo correspondiente a operación Xor.

3.2.2.4 Ronda 10

Como se indica en la figura 3-2, esta ronda realiza todos los pasos de las rondas anteriores, excepto la operación MixColumns. Es decir, la salida de esta ronda es calculada mediante los pasos siguientes:

A. SubBytes
B. ShiftRows
C. Xor

El desarrollo de estas etapas es la siguiente:

A. SubBytes

En la figura 3-26, se ilustra el resultado de la operación SubBytes correspondiente a la ronda 10.

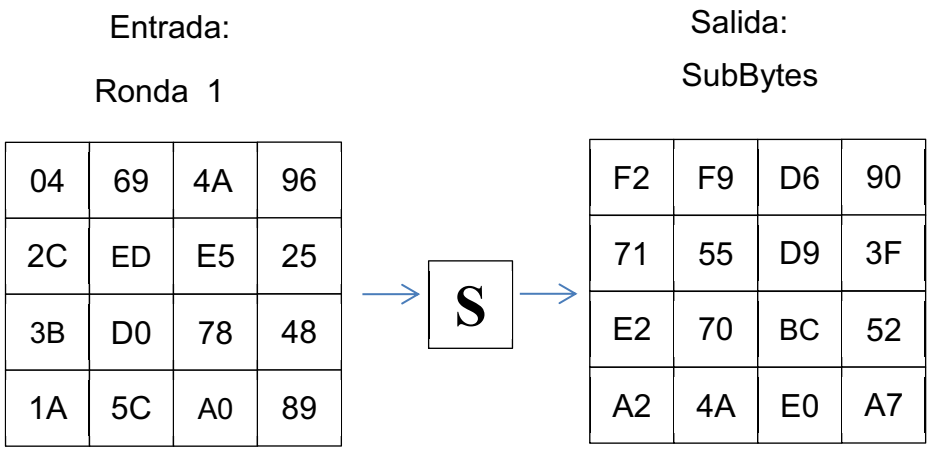


Figura 3-26. Ejemplo práctico: SubBytes, ronda 10.

B. ShiftRows

En la figura 3-27, se ilustra el resultado de la operación ShiftRows correspondiente a la ronda 10.

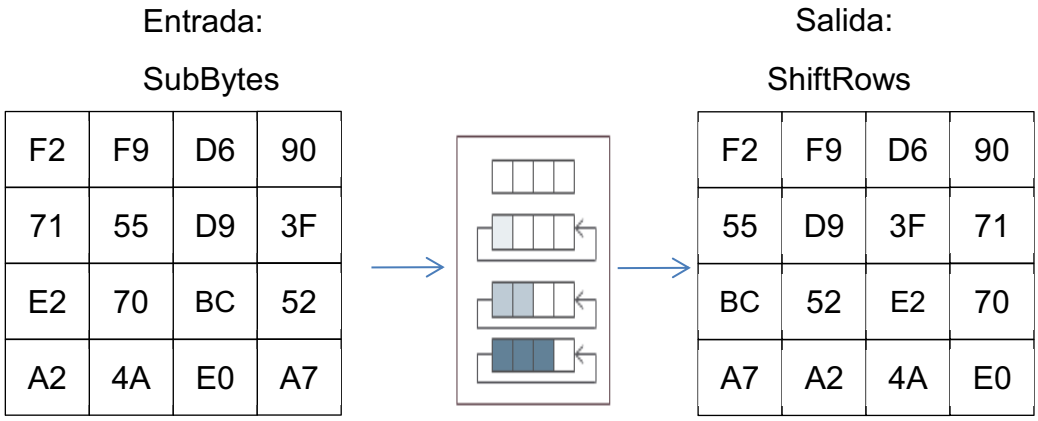


Figura 3-27. Ejemplo práctico: ShirftRows, ronda 10.

C. Xor

En la figura 3-28, se ilustra el resultado de la operación Xor correspondiente a la ronda 10.

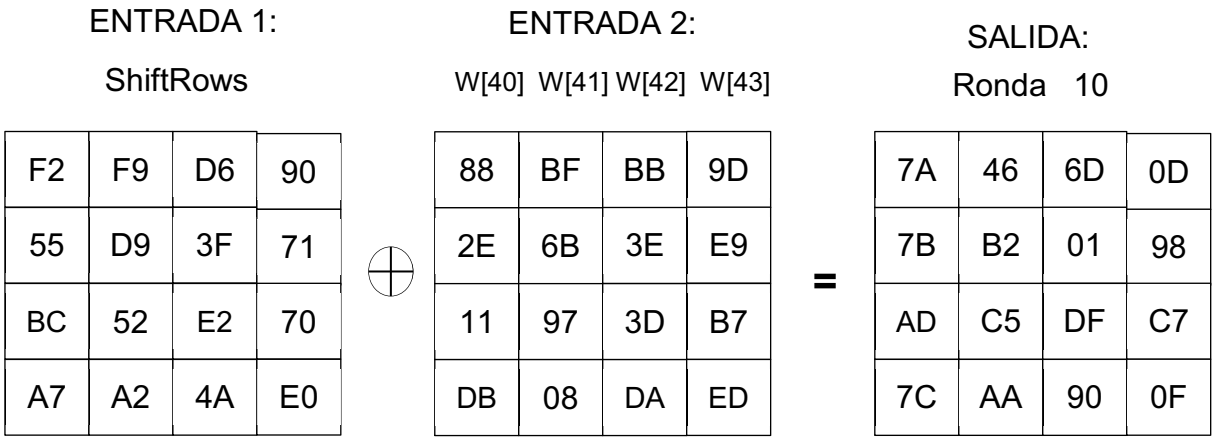


Figura 3-28. Suma llave de etapa, ronda 10.

El resultado obtenido en la ronda 10 es el texto cifrado; el cual es leído de izquierda a derecha:

Texto cifrado = 7A7BAD7C46B2C5AA6D01DF900D98C70F

3.2.3 DESCIFRADO

El descifrado sigue la estructura de la figura 3-29. El proceso de expansión de llave es el mismo que se empleó en el proceso de cifrado.

3.2.3.1 Ronda 1

Como se ilustra en la figura 3-29, la etapa que corresponde a la ronda 1, contiene tres operaciones:

A. Xor
B. ShiftRows Inversa
C. SubBytes Inversa

Cada una de estas etapas se detalla a continuación:

A. Xor

La operación xor se realiza entre el texto cifrado, y las palabras: W[40], W[41], W[42], W[43].

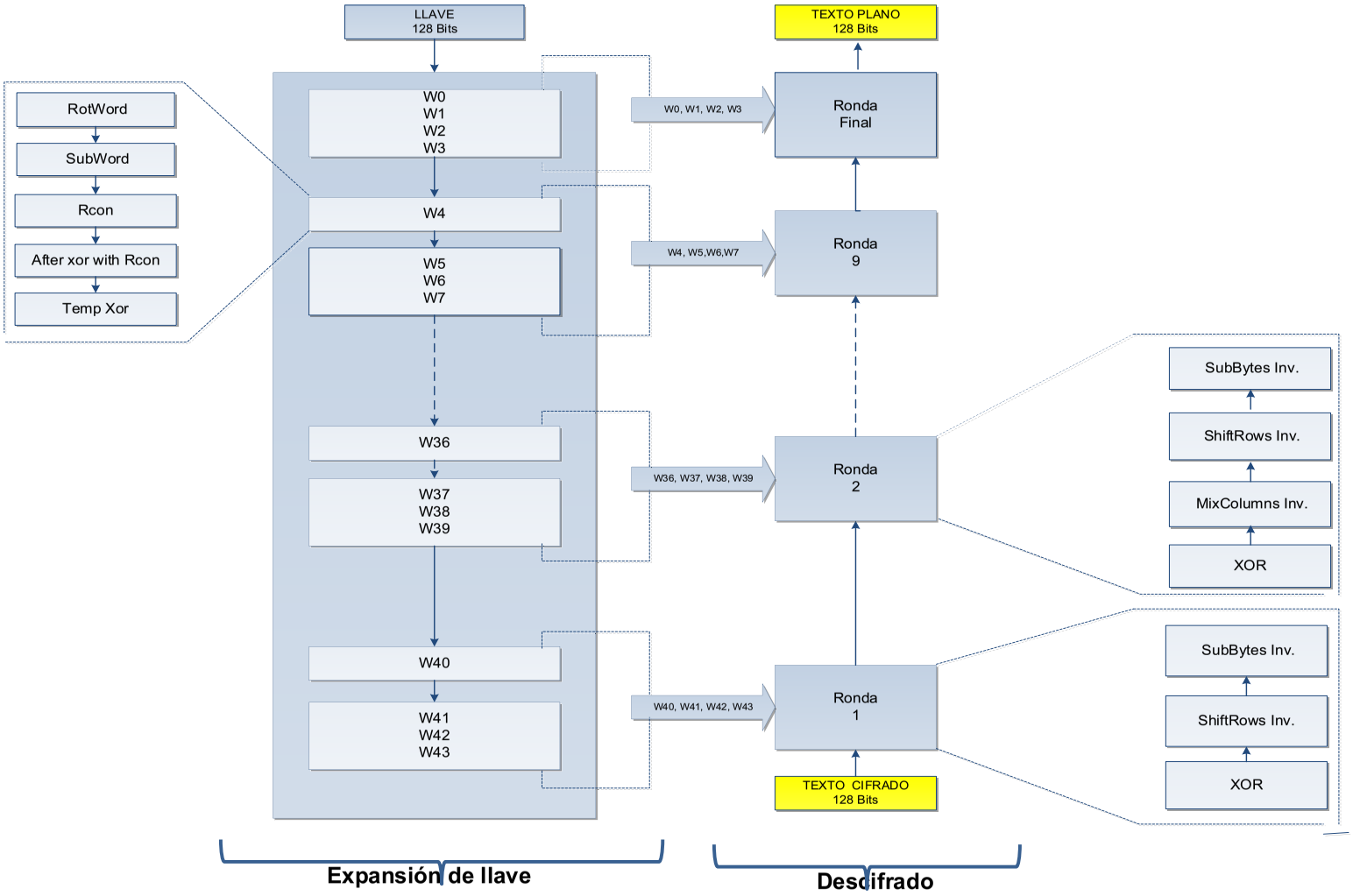


Figura 3-29. Representación gráfica del descifrado AES.

B. ShiftRows inversa

La operación ShiftRows, de forma inversa se realiza como una permutación, tal como indica la figura 3-30.

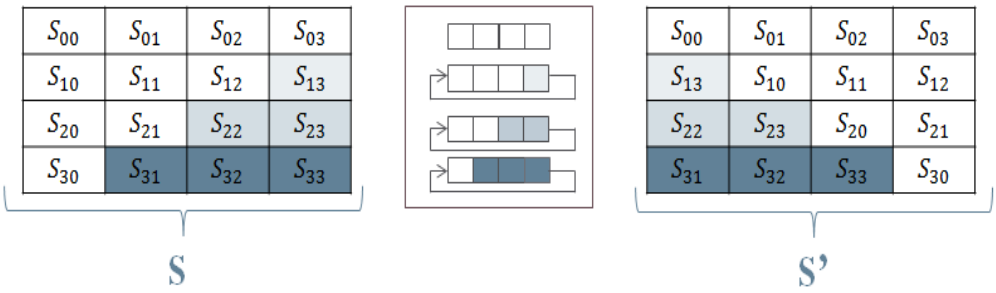


Figura 3-30. ShiftRows inversa.

C. SubBytes inversa

Esta operación se realiza de la misma forma que en el cifrado, pero la sustitución de los valores se los aplica con la caja-S inversa. Los valores de la caja-S inversa se muestran en la tabla 3-5. El fundamento matemático para la obtención de estos valores se desarrolla en el anexo 10.

		y														
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	0	52	09	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	0b	42	fa	c3	4e
	3	08	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	00	8c	bc	d3	0a	f7	e4	58	05	b8	b3	45	06
	7	d0	2c	1e	8f	ca	3f	0f	02	c1	af	bd	03	01	13	8a	6b
	8	3a	91	11	41	4f	67	dc	ea	97	f2	cf	ce	f0	b4	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	0e	aa	18	be	1b
	b	fc	56	3e	4b	c6	d2	79	20	9a	db	c0	fe	78	cd	5a	f4
	c	1f	dd	a8	33	88	07	c7	31	b1	12	10	59	27	80	ec	5f
	d	60	51	7f	a9	19	b5	4a	0d	2d	e5	7a	9f	93	c9	9c	ef
	e	a0	e0	3b	4d	ae	2a	f5	b0	c8	eb	bb	3c	83	53	99	61
	f	17	2b	04	7e	ba	77	d6	26	e1	69	14	63	55	21	0c	7d

Tabla 3-5. Caja -S, inversa. [3.4]

3.2.3.2 Rondas 2, 3, 4, 5, 6, 7, 8 y 9

La siguiente ronda (ronda 2), como se indica en la figura 3-29, se ejecuta con el dato de entrada correspondiente a la ronda uno y las palabras W[36], W[37],

W[38] y W[39]. El dato de entrada para la ronda tres será el resultado de la ronda dos y las palabras W[35], W[36], W[37] y W[38], y así sucesivamente. Las etapas que corresponden a cada una de estas rondas son:

A. Xor
B. MixColumns Inversa
C. ShiftRows Inversa
D. SubByte Inversa

A continuación se detalla el funcionamiento de la etapa MixColumns inversa, ya que las otras etapas se realizan de la misma forma que se describió en el punto anterior.

B. MixColumns inversa

Consiste en multiplicar cada columna de la matriz de entrada, por una columna constante que se obtiene en $GF(2^8)$ $[X]/(X^4+1)$. Una representación gráfica se indica en la figura 3-31, y el desarrollo matemático para obtener la matriz constante se muestra en el anexo 10.

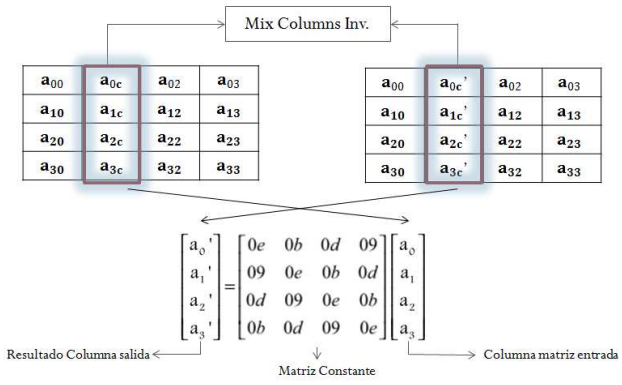


Figura 3-31. MixColums inversa.

3.2.3.3 Ronda Final

Como se ilustra en la figura 3-29, al finalizar se realiza la ronda final, en la cual se realiza la operación xor, con el resultado de la ronda 9 y las palabras W[0], W[1], W[2] y W[3]. El resultado de la ronda final corresponde al texto plano.

3.2.4 DESCRIPCIÓN INTERFAZ GRÁFICA DE MATLAB PARA AES

La interfaz gráfica, se desarrolló en MatLab 7.6.0 (R2008a). El usuario deberá ingresar un texto plano y la llave en formato ASCII. La interfaz gráfica se observa en la figura 3-32.

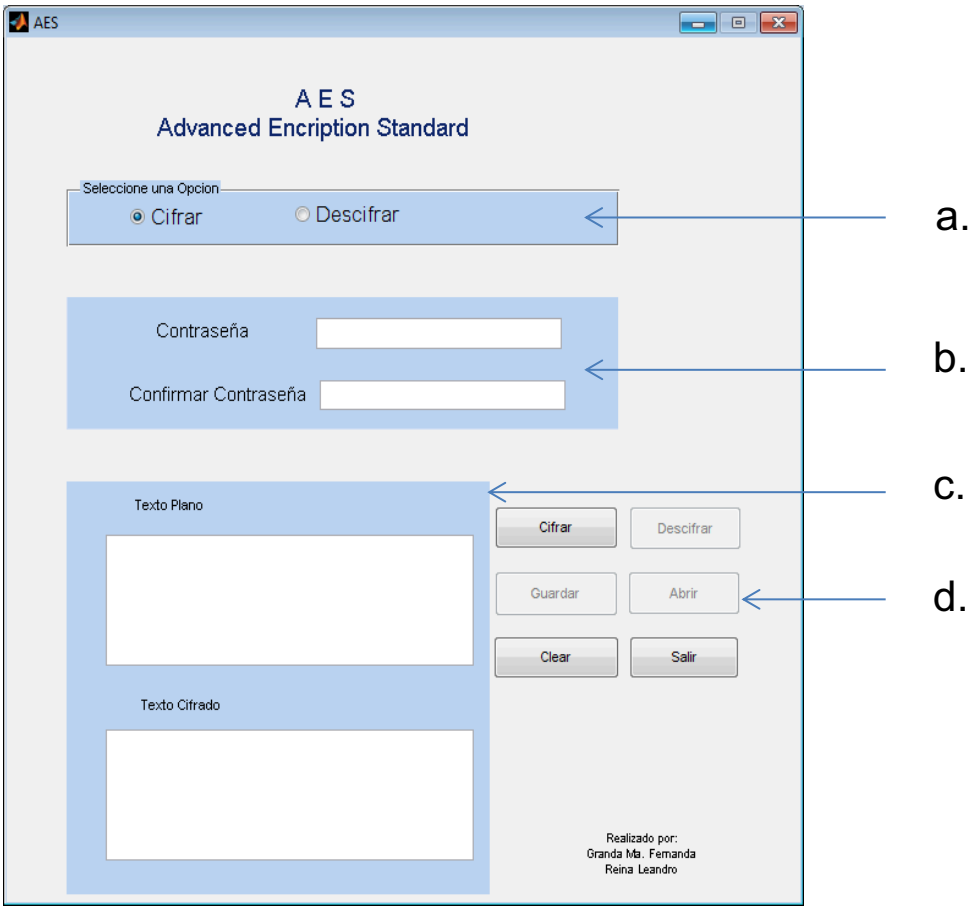


Figura 3-32. Interfaz gráfica del cifrado AES.

a. El usuario debe seleccionar la opción que desea realizar: cifrado o descifrado.

- Cifrar: habilita los botones cifrar, clear y salir. Una vez que se termine el proceso de cifrado, el resultado es guardado en un archivo txt, para poder descifrar.

- Descifrar: habilita los botones descifrar, abrir, clear y salir. Para realizar esta opción se debe abrir un archivo txt, previamente cifrado con el botón abrir.

b. La contraseña corresponde a la llave y debe ser confirmada, para poder realizar el paso siguiente. Se debe ingresar 16 caracteres en formato ASCII.

c. En este panel se indicará los resultados del proceso de cifrado y descifrado.

d. Es el conjunto de cuatro botones que realizaran las operaciones requeridas por el usuario :

- Botón Cifrar
- Botón Descifrar
- Botón Guardar
- Botón Abrir
- Botón Clear
- Botón Salir

El diagrama de flujo que responde al requerimiento de la interfaz gráfica se describe en la figura 3-33.

El código fuente correspondiente a la implementación del cifrado y descifrado del estándar AES realizado en MatLab se muestra en el anexo 11.

3.3 DISEÑO DE UN SISTEMA DE CIFRADO AES USANDO FPGA

La arquitectura planteada para el cifrador AES es presentada en la figura 3-34. Aquí se puede observar claramente los distintos módulos o componentes del sistema, y el flujo de bits entre ellos por medio de los buses de datos.

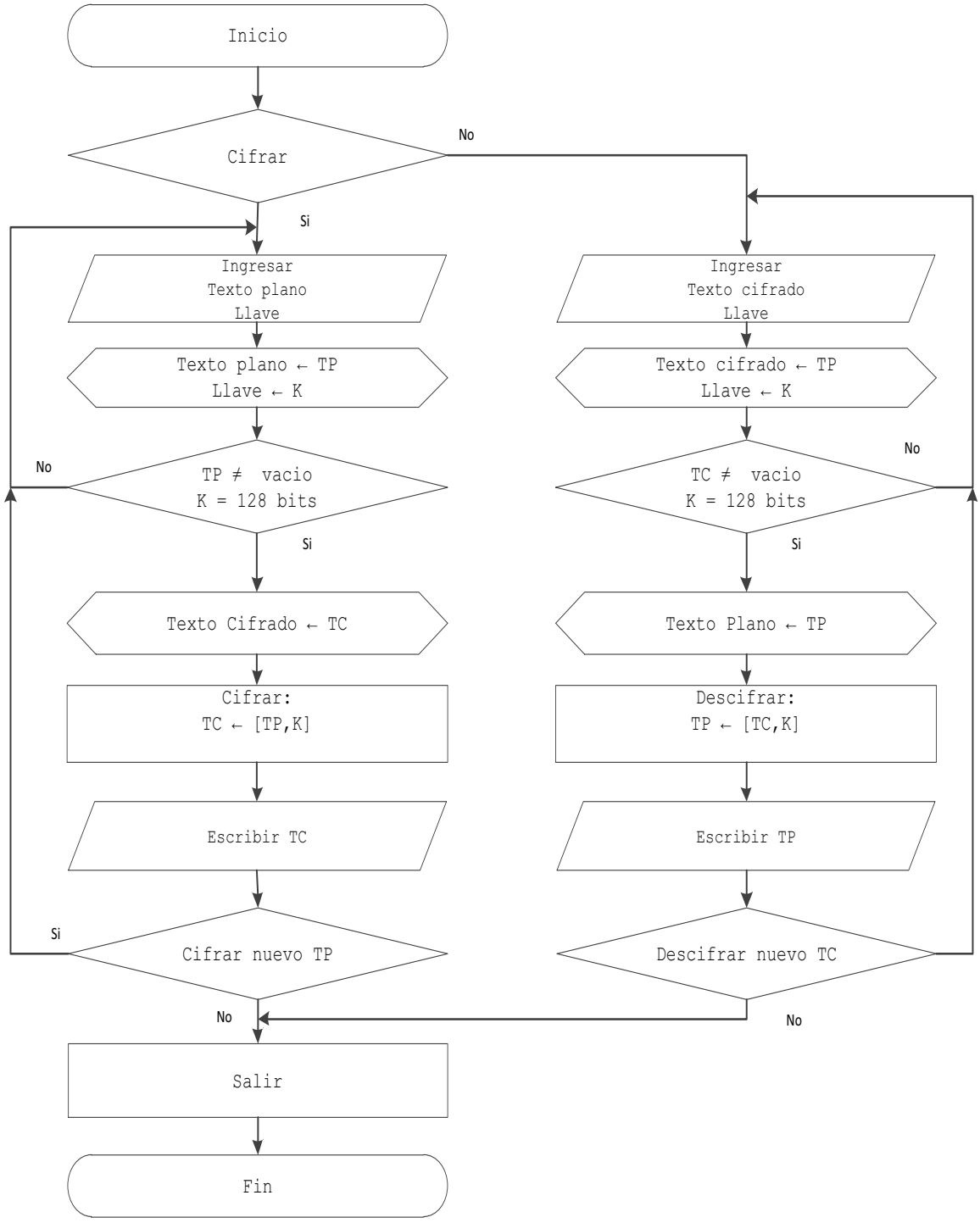


Figura 3-33. Diagrama de flujo correspondiente a la interfaz gráfica.

El bloque de color verde representa a la aplicación realizada en MatLab, las flechas de color negro representan a los buses que transportan datos entre los

módulos y los bloques de color celeste representan componentes que se encuentran en el interior del chip FPGA Virtex 5.

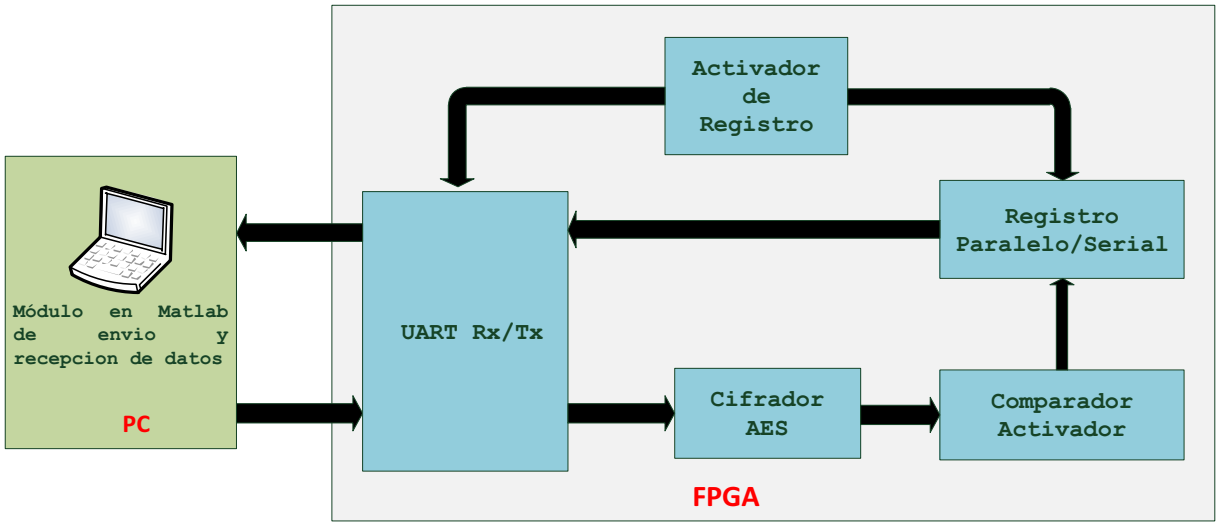


Figura 3-34. Arquitectura del cifrador AES.

Como se observó en la etapa de descripción del algoritmo AES, las etapas que lo constituyen, se encuentran representadas en la figura 3-2, el algoritmo presenta una estructura secuencial, por lo que se debe diseñar un circuito secuencial en VHDL, que permita describir el comportamiento del algoritmo.

3.3.1 CONSIDERACIONES DE DISEÑO

Para la realización de este diseño son necesarios doce bloques, diez bloques de estos representan a las diez iteraciones o rondas que realiza el algoritmo, un bloque adicional representa la ronda inicial y el restante indica la expansión de la llave.

Cada bloque de una ronda posee 384 puertos, 256 de entrada y 128 de salida. Los primeros 128 de entrada representan los datos de entrada de la ronda anterior, los segundos 128 representan la entrada de las 4 subllaves y los 128 pines de salida corresponden a la salida de cada ronda.

El bloque de la ronda inicial de igual manera también posee 384 puertos, 256 de entrada y 128 de salida. Los primeros 128 de entrada representan los datos del

texto plano, los segundos 128 de entrada representan los datos de entrada de las 4 subllaves y los 128 puertos de salida corresponden a la salida de la ronda inicial y la entrada a la ronda uno.

Por otra parte el bloque expansión de la llave posee 1536 puertos, de los cuales 128 son de entrada y los restantes 1408 son de salida. Los 128 de entrada corresponden la llave que ingresa el usuario y los puertos de salida corresponden a las 44 subllaves generadas a partir de la llave inicial. En este bloque se tendría que las 44 subllaves son generadas en paralelo.

Como ventaja de este diseño se tiene la velocidad en cuanto al procesamiento de los datos ya que estos en su mayoría se calculan, en forma paralela. La desventaja seria el uso de recursos en cuanto a CLBs y espacios de almacenamiento, pero debido a la gran cantidad de recursos que se dispone en el dispositivo Virtex 5, el diseño se acepta como válido.

3.3.2 DESCRIPCIÓN EN VHDL DEL ESTÁNDAR AES

El módulo del sistema de cifrado contiene cinco entidades y cada una de estas entidades representa un bloque en la arquitectura descrita en la figura 3-34. Las cinco entidades se unen mediante ***components*** en una entidad global denominada aes_total, y esta es la que se sintetiza e implementa en el kit de entrenamiento. La estructura interna de la entidad aes_total se describe en el diagrama RTL de la figura 3-35.

3.3.2.1 Bloque UART

Para el bloque UART que se utiliza para la transmisión y recepción de datos, los puertos asignados a esta entidad se observan en la figura 3-36.

A diferencia de la UART desarrollada en el capítulo 2, para la implementación del cifrado S-DES, en este diseño se incorporan las entidades: uart_tx y fifoTransmision. Dichas entidades serán descritas de forma detallada.

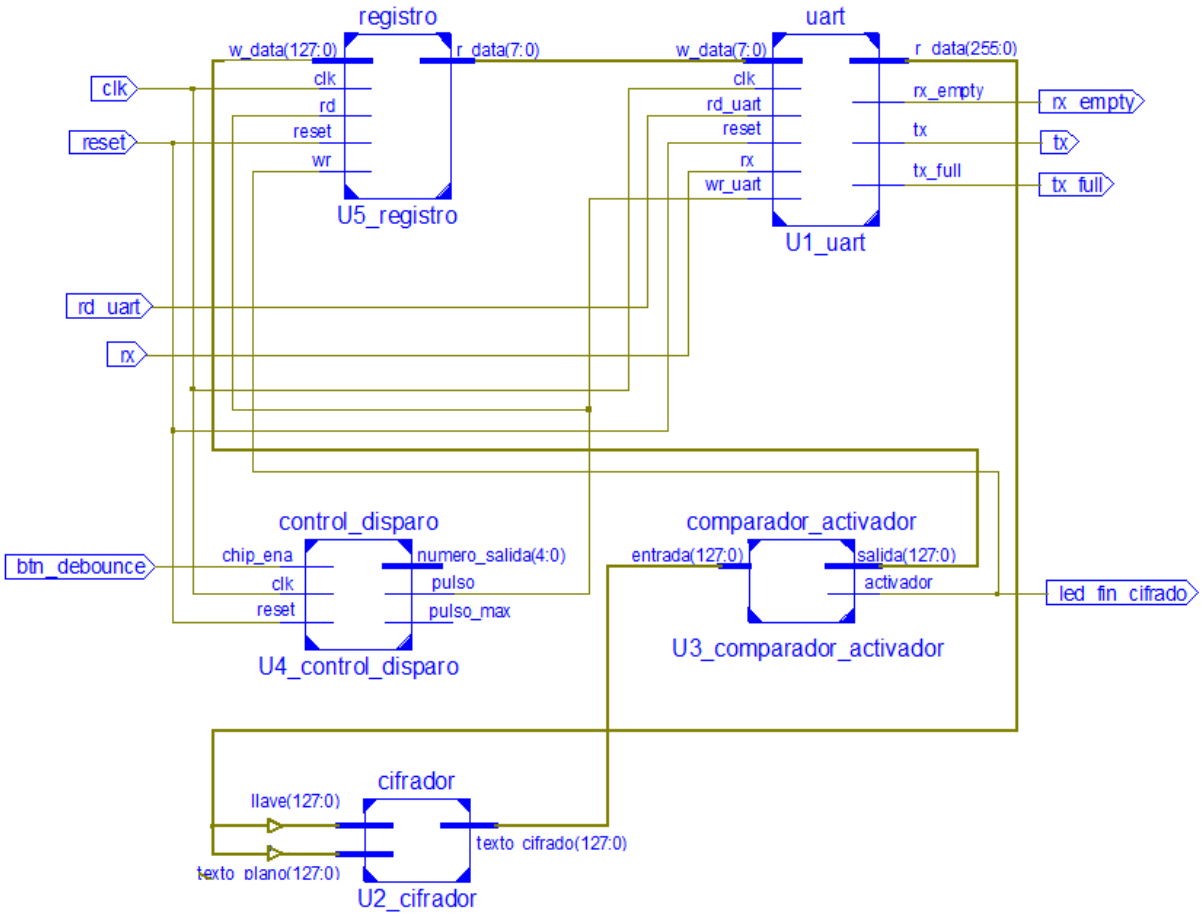


Figura 3-35. Diagrama RTL interno del módulo FPGA.

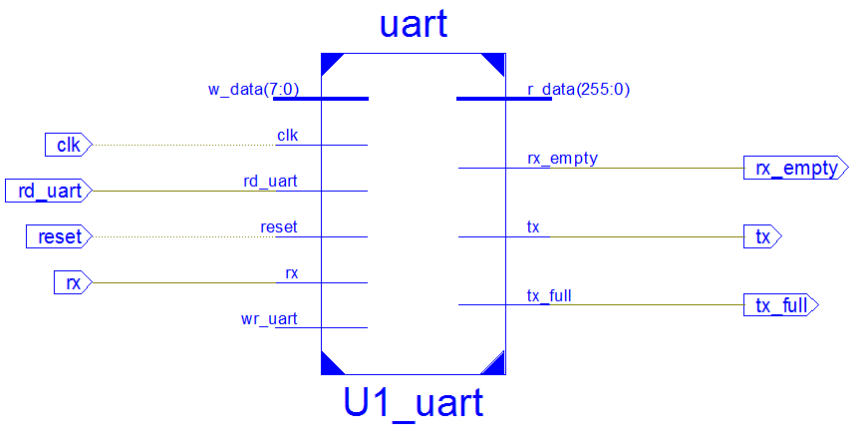


Figura 3-36. Diagrama RTL total UART.

Internamente el bloque UART, presenta la estructura que se describe en la figura 3-37.

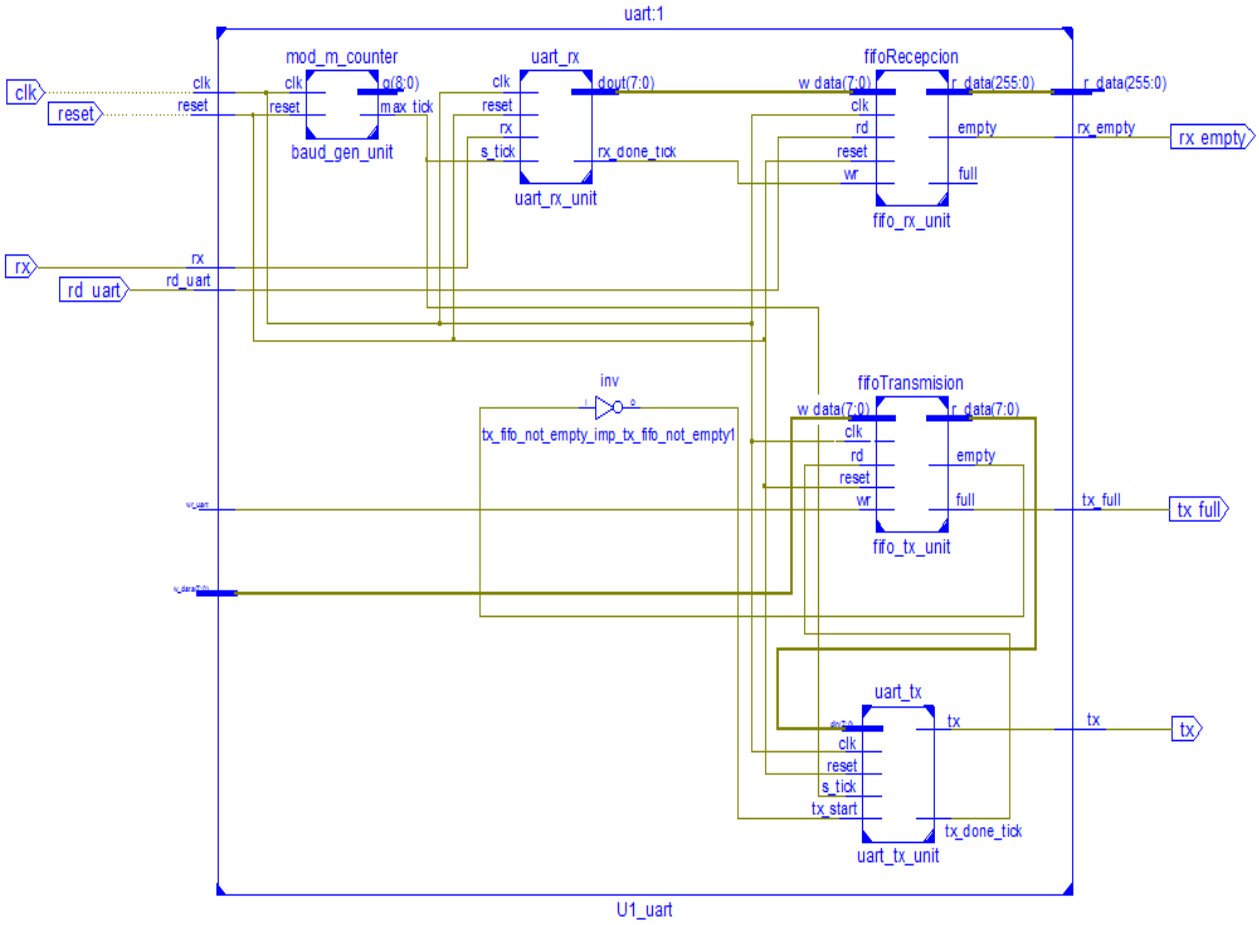


Figura 3-37. Diagrama RTL interno UART.

Las entidades internas son las siguientes:

- Entidad → uart_rx, recibe los datos, enviados por la interfaz gráfica en MatLab y los envia a la fifo de recepción:

```
clk : in  STD_LOGIC;
reset : in  STD_LOGIC;
rx : in  STD_LOGIC;
s_tick : in  STD_LOGIC;
rx_done_tick : out  STD_LOGIC;
dout : out  STD_LOGIC_VECTOR (7 downto 0);
```

La función de este bloque es la de recibir una trama serial y extraer los bits de datos, para ello usa un esquema de sobremuestreo comúnmente

utilizado para localizar la posición media de cada bit transmitido cuando se tome la muestra real. La frecuencia de muestreo más común es de 16 veces la frecuencia de la señal. Por lo tanto cada bit se muestrea 16 veces, pero solo una muestra se guarda. Dicho procedimiento se explicó a detalle en el capítulo 2. [3.6]

- Entidad → fifoRecepcion, almacena los datos correspondientes al texto plano y la llave. [3.6] Los datos son enviados a la entidad cifrador. Los puertos de entrada y salida son:

```
clk : in  STD_LOGIC;
reset : in  STD_LOGIC;
rd : in  STD_LOGIC;
wr : in  STD_LOGIC;
w_data : in  STD_LOGIC_VECTOR (B-1 downto 0);
empty : out  STD_LOGIC;
full : out  STD_LOGIC;
r_data : out  STD_LOGIC_VECTOR (255 downto 0);
```

Donde los valores de B corresponden a un tipo de dato *generic* denotado como:

```
generic( B: natural := 8);
```

La fifo de recepción está diseñada para almacenar 256 bits, de los cuales los primeros 128 corresponden al texto plano, y los segundos a la llave. Estos datos son enviados a la entidad cifrador.

- Entidad → fifoTransmision, recibe los datos cifrados los almaceno y envia a la entidad uart_tx, para que realice el proceso de transmisión del texto cifrado. Los puertos de entrada y salida son:

```
clk : in  STD_LOGIC;
reset : in  STD_LOGIC;
```

```
rd : in  STD_LOGIC;
wr : in  STD_LOGIC;
w_data : in  STD_LOGIC_VECTOR (B-1 downto 0);
empty : out  STD_LOGIC;
full : out  STD_LOGIC;
r_data : out  STD_LOGIC_VECTOR (B-1 downto 0);
```

Donde los valores de B corresponden a un tipo de dato *generic* denotado como:

```
generic( B: natural := 8);
```

A diferencia de la fifo de recepción, aquí se almacenan 128 bits correspondientes al texto cifrado. Los datos en la fifo de transmisión, se reciben serialmente mediante la entidad registro. Una vez que se llena la fifo se envía la bandera de full que es visualizada mediante un led en la tarjeta y se envían los datos manualmente a la uart_tx. La lectura y envió de los datos se realiza de uno en uno en bloques de 8 bits, como indica la figura 3-38.

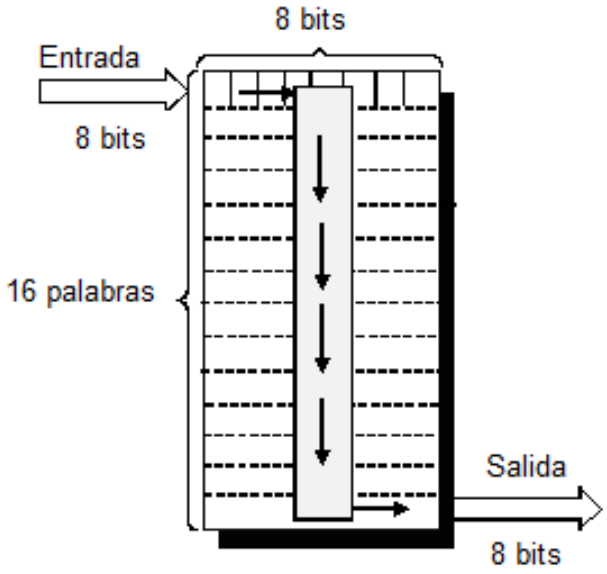


Figura 3-38. Fifo de transmisión.

- Entidad → uart_tx, envía los datos cifrados a la interfaz gráfica para que sean visualizados por los usuarios. [3.6]

  Los puertos de entrada y salida son:

  ```
  clk : in  STD_LOGIC;
  reset : in  STD_LOGIC;
  tx_start : in  STD_LOGIC;
  s_tick : in  STD_LOGIC;
  din : in  STD_LOGIC_VECTOR (7 downto 0);
  tx_done_tick : out  STD_LOGIC;
  tx : out  STD_LOGIC;
  ```

  El diseño de este bloque contempla una máquina de estados finitos. En este diseño se tiene cuatro estados:

  - *Idle*
  - *Start*
  - *Data*
  - *Stop*

  La transición entre uno u otro estado es controlado por señales internas que se activan o desactivan dependiendo de la operación que se vaya a realizar.

  - **Estado *Idle*:** En este estado, la señal tx_next es la encargada de transmitir los datos hacia el puerto serial de la PC, la misma se activa en uno lógico ya que esto indica al puerto serie de la PC que todavía no se realizará ningún envío de datos. [3.6]

    Cuando la señal tx_start toma el valor el valor de “1”, entonces se pasa al estado *start*, se inicializa el contador “s” en cero y se almacena los datos recibidos de la entidad registro en la señal

b_next. En la figura 3-39, se indica un fragmento del código que realiza este procedimiento.

```
when idle =>
   tx_next <= '1';
   if tx_start = '1' then
      state_next <= start;
      s_next <= (others => '0');
      b_next <= din;
   end if;
```

Figura 3-39. Fragmento de código del estado *idle*.

- **Estado *Start*:** En este estado, la señal tx_next toma el valor de cero para indicar que va iniciar un envío de datos, cuando la señal s_tick toma el valor de uno y la señal s_reg toma el valor de 15, entonces se pasa al estado data , se reinicializa el contador “s” y el contador “n” en cero. En la figura 3-40, se indica un fragmento del código que realiza este procedimiento. [3.6]

```
when start =>
   tx_next <= '0';
   if (s_tick = '1') then
      if s_reg= 15 then
         state_next <= data;
         s_next <= (others => '0');
         n_next <= (others => '0');
       else
         s_next <= s_reg + 1;
      end if;
   end if;
```

Figura 3-40. Fragmento de código del estado *start*.

- **Estado *Data*:** En este estado, la señal tx_next toma el valor del primer elemento almacenado en la señal b_reg, luego se verifica que el valor de la señal s_tick sea “1” y que el contador “s”, alcance el valor de “15”, para garantizar el esquema de muestreo, cuando esto ocurre se almacena el valor de los datos de entrada en la señal

b_next. En la figura 3-41, se indica un fragmento del código que realiza este procedimiento. [3.6]

```
when data =>
   tx_next <= b_reg (0);
   if (s_tick = '1') then
      if s_reg = 15 then
         s_next <= (others => '0');
         b_next <= '0' & b_reg(7 downto 1);
         if n_reg =(DBIT-1) then
            state_next <= stop;
            else
               n_next <= n_reg+1;
         end if;
      else
         s_next <= s_reg + 1;
      end if;
   end if;
```

Figura 3-41. Fragmento de código del estado *data*.

- **Estado *Stop*:** En este estado, la señal tx_next toma el valor de uno, de esta manera se indica que la transmisión ha culminado y se pasa al estado *Idle* completando el ciclo. En la figura 3-42, se ilustra un fragmento del código que realiza este procedimiento. [3.6]

```
when stop =>
   tx_next <= '1';
      if (s_tick = '1') then
         if s_reg = (SB_TICK-1) then
            state_next <= idle;
            tx_done_tick <= '1';
          else
            s_next <= s_reg + 1;
         end if;
      end if;
```

Figura 3-42. Fragmento de código del estado *stop*.

- Entidad → mod_m_counter, genera una señal de muestreo. Los puertos de entrada y salida son:

clk : in STD_LOGIC;
reset : in STD_LOGIC;

```
max_tick : out  STD_LOGIC;
q : out  STD_LOGIC_VECTOR(N-1 downto 0);
```

Donde los valores de N corresponden a un tipo de dato *generic* denotado como:

```
generic( N: integer := 9);
```

Genera una señal de muestreo cuya frecuencia es exactamente 16 veces la frecuencia de la velocidad de señal de la UART de la PC. [3.6]

3.3.2.2 Bloque Cifrador

Este bloque contiene todo el proceso de cifrado que contempla el estándar AES, siguiendo el esquema de la figura 3-35, la entidad cifrador recibe de la entidad fifo de recepción el texto plano y la llave, una vez que se efectúan las rondas internas del algoritmo, se devuelve el texto cifrado. Los puertos de entrada y salida correspondientes a la entidad cifrador se detallan en la figura 3-43:

```
llave : in  STD_LOGIC_VECTOR (127 downto 0);
texto_plano : in  STD_LOGIC_VECTOR (127 downto 0);
texto_cifrado : out  STD_LOGIC_VECTOR (127 downto 0);
```

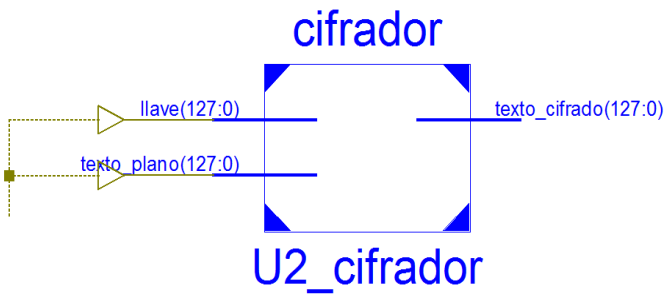


Figura 3-43. Diagrama RTL correspondiente a la entidad cifrador.

Internamente la entidad cifrador está compuesta por el diagrama de la figura 3-44 correspondiente a cada una de sus entidades internas.

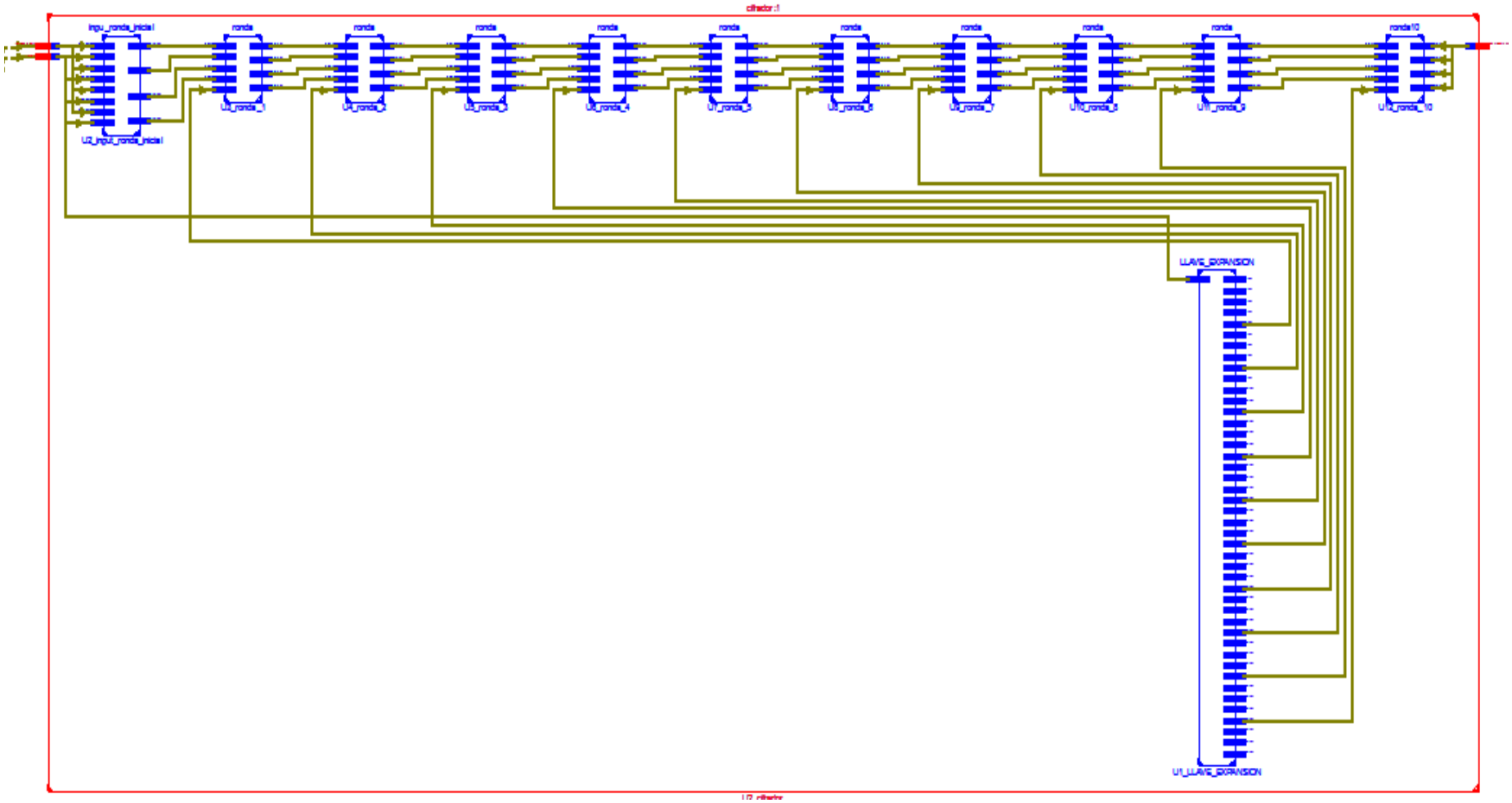


Figura 3-44. Diagrama RTL interno de la entidad cifrador.

Las entidades internas que componen a la entidad cifrador se unen mediante components y son las siguientes:

- Entidad → inpu_ronda_inicial, realiza la operación correspondiente al cifrado de la ronda inicial. Los puertos de entrada y salida son:

```
cin1_1 : in  STD_LOGIC_VECTOR (31 downto 0);
cin2_1 : in  STD_LOGIC_VECTOR (31 downto 0);
cin3_1 : in  STD_LOGIC_VECTOR (31 downto 0);
cin4_1 : in  STD_LOGIC_VECTOR (31 downto 0);
cin1_2 : in  STD_LOGIC_VECTOR (31 downto 0);
cin2_2 : in  STD_LOGIC_VECTOR (31 downto 0);
cin3_2 : in  STD_LOGIC_VECTOR (31 downto 0);
cin4_2 : in  STD_LOGIC_VECTOR (31 downto 0);
cout1 : out  STD_LOGIC_VECTOR (31 downto 0);
cout2 : out  STD_LOGIC_VECTOR (31 downto 0);
cout3 : out  STD_LOGIC_VECTOR (31 downto 0);
cout4 : out  STD_LOGIC_VECTOR (31 downto 0);
```

La figura 3-45 indica los resultados de la simulación realizada para la ronda inicial. Los siguientes valores corresponden al texto plano:

```
cin1_1 : 4645524E
cin2_1 : 414E4441
cin3_1 : 4645524E
cin4_1 : 414E4441
```

Los valores corresponden a la llave:

```
cin1_2 : 41424344
cin2_2 : 414E4441
cin3_2 : 4645524E
cin4_2 : 414E4441
```

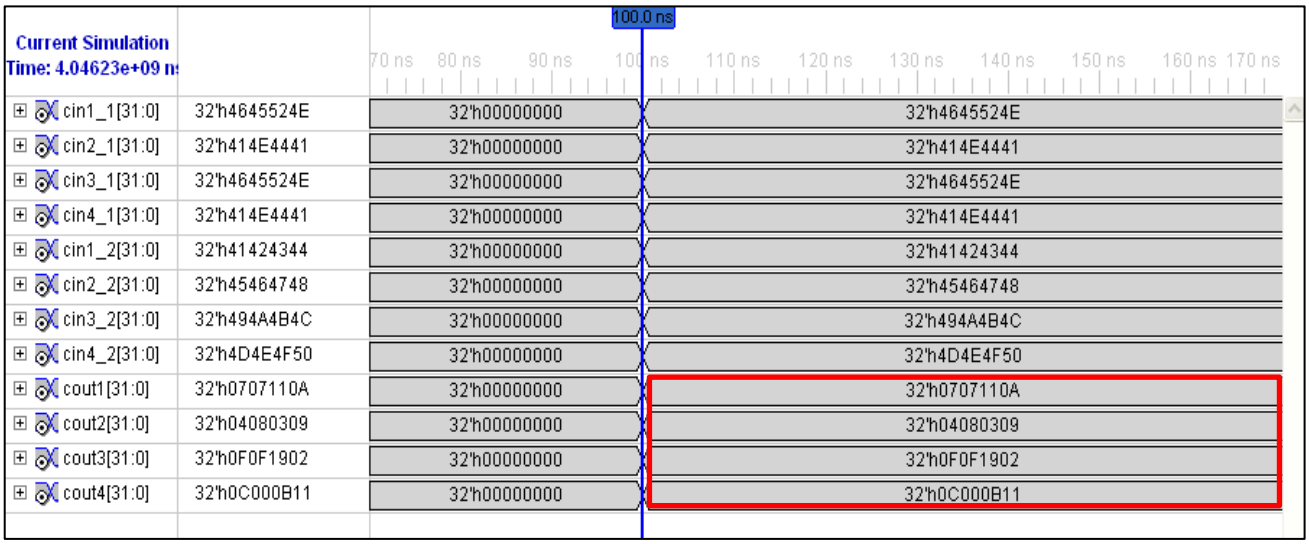


Figura 3-45. Simulación entidad inpu_ronda_inicial.

Como se muestra en la figura 3-45, el resultado es el esperado ya que los valores de cout1, cout2, cout3, cout4, corresponden a los valores obtenidos en la sección 3.2.2.1.

- Entidad → ronda, esta entidad es generada una sola vez, pero es clonada nueve veces correspondientes a las nueve rondas del proceso de cifrado. Los puertos de entrada y salida son:

```
cin1 : in  STD_LOGIC_VECTOR (31 downto 0);
cin2 : in  STD_LOGIC_VECTOR (31 downto 0);
cin3 : in  STD_LOGIC_VECTOR (31 downto 0);
cin4 : in  STD_LOGIC_VECTOR (31 downto 0);
subllave: in std_logic_vector (127 downto 0);
cout1 : out  STD_LOGIC_VECTOR (31 downto 0);
cout2 : out  STD_LOGIC_VECTOR (31 downto 0);
cout3 : out  STD_LOGIC_VECTOR (31 downto 0);
cout4 : out  STD_LOGIC_VECTOR (31 downto 0);
```

De este proceso es importante describir la lógica de descripción de hardware de cada una de las entidades internas:

A. subByte
B. shiftRows
C. mix_column
D. ope_xor

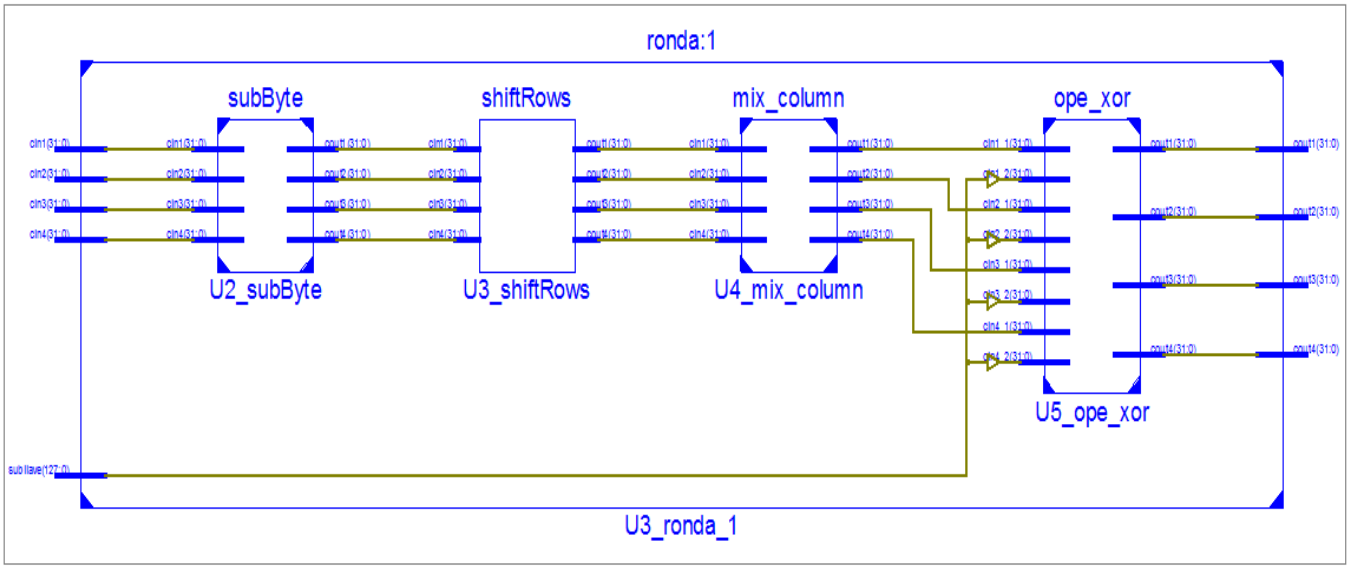


Figura 3-46. Diagrama RTL interno de la entidad ronda.

A. Entidad subByte

Este bloque constituye una memoria ROM, ya que en su interior se almacena una matriz con 256 valores escritos en formato hexadecimal, que corresponden a la caja-S. La función de esta entidad es la de sustituir los valores de entrada, por los valores de la caja-S, para ello los valores de entrada sirven como direcciones de memoria para leer la tabla.

Para describir la caja-S, se declaró un tipo de dato definido por el usuario llamado TYPE_MATRIZ como un arreglo de 256 elementos y cada elemento era de tipo STD_LOGIC_VECTOR (7 DOWNTO 0), luego se declaró como valor constante SBOX cuyo tipo de dato era TYPE_MATRIZ; en la constante SBOX se almacenaron los valores de la caja-S. La manera en la que se lee un valor de la tabla SBOX se muestra en la figura 3-47.

```
process(byte_Uno)
      begin
        aux(31 downto 24) <= SBOX(to_integer(unsigned(byte_Uno)));

end process;
```

Figura 3-47. Fragmento del código de la entidad subByte.

En donde byte_Uno representa un elemento de la columna de entrada, luego este es pasado a formato sin signo y luego a entero. Cada valor de byte_Uno está comprendido entre 0 y 255, y dado que el arreglo SBOX posee posiciones de memoria de 0 a 255 por lo que de esta forma se pueden leer cada uno de los elementos de SBOX.

La figura 3-48 indica los resultados de la simulación realizada para la operación SubByte, correspondiente a la ronda uno. Los siguientes valores corresponden a los datos de entrada:

cin1: 0707110A
cin2: 04080309
cin3: 0F0F1902
cin4: 0C000B11

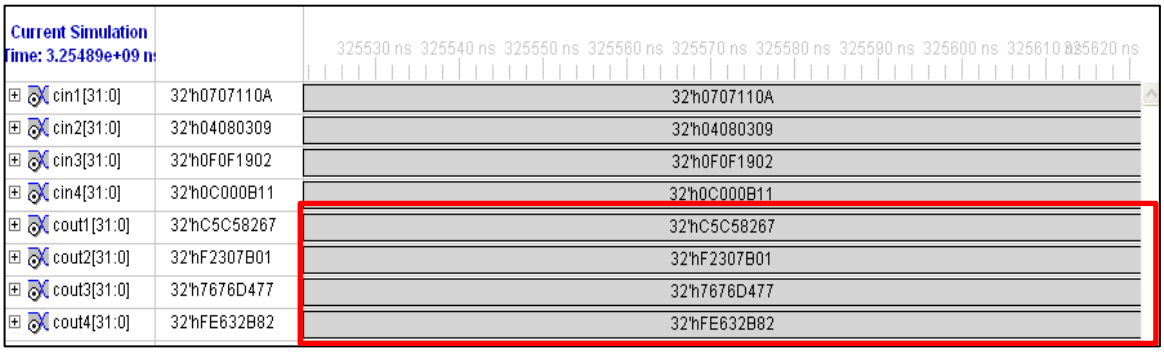


Figura 3-48. Simulación entidad subByte, ronda1.

Como se puede observar, los resultados obtenidos son iguales a los desarrollados en la sección 3.2.2.2.

B. Entidad shiftRows

Este bloque realiza un desplazamiento de las filas de la matriz de estado. Para representar este desplazamiento en este diseño se realiza el reordenamiento de los elementos de las columnas de la matriz de estado ya que en este bloque se tiene como elementos de entrada columnas. La figura 3-49, contiene fragmento de código de esta permutación.

```
architecture arch of shiftRows is

begin
cout1<=cin1(31 downto 24)&cin2(23 downto 16)&cin3(15 downto 8)&cin4(7 downto 0);
cout2<=cin2(31 downto 24)&cin3(23 downto 16)&cin4(15 downto 8)&cin1(7 downto 0);
cout3<=cin3(31 downto 24)&cin4(23 downto 16)&cin1(15 downto 8)&cin2(7 downto 0);
cout4<=cin4(31 downto 24)&cin1(23 downto 16)&cin2(15 downto 8)&cin3(7 downto 0);

end arch;
```

Figura 3-49. Fragmento del código de la entidad shiftRows.

La figura 3-50 muestra los resultados de la simulación realizada de la entidad shiftRows, correspondiente a la ronda uno. Los siguientes valores corresponden a los datos de entrada:

cin1: C5C58267
cin2: F2307B01
cin3: 7676D477
cin4: FE632B82

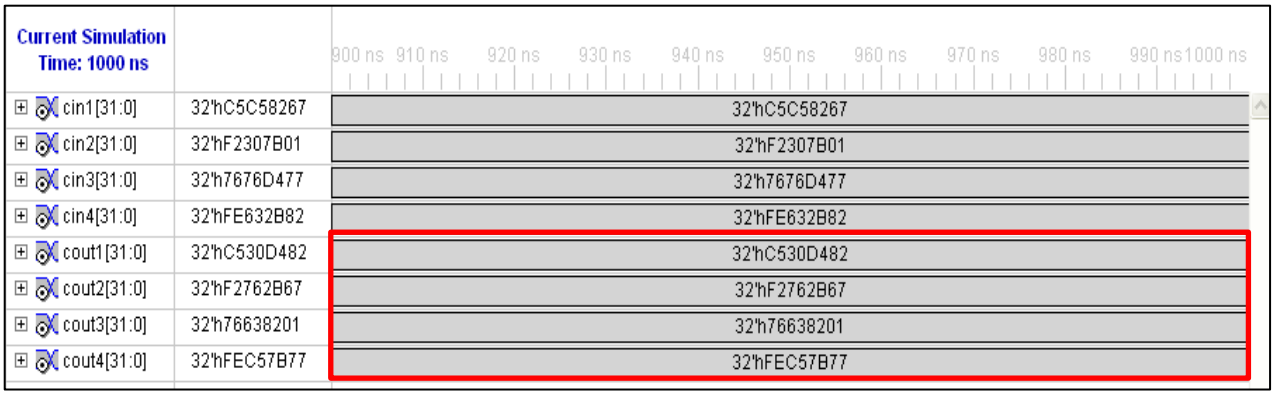


Figura 3-50. Simulación entidad shiftRows, ronda1.

Como se puede observar, los resultados obtenidos son iguales a los desarrollados en la sección 3.2.2.2.

C. Entidad mix_column

Como se explicó en la operación MixColumns, se debe multiplicar cada columna de la matriz de entrada por una columna constante que se obtiene en GF(2^8) [X]/(X^4+1), esta operación en MatLab no resulta muy compleja ya que usa los recursos computacionales de esta herramienta en la multiplicación de matrices y su reducción en GF(2^8). Este proceso en VHDL ya no resulta tan fácil, pues realizar la multiplicación de polinomios y la división para reducir en el campo GF(2^8) utilizaría muchos recurso del dispositivo FPGA, es por esta razón que se utiliza las propiedades de aritmética modular para encontrar la forma de multiplicar matrices. Teniendo en cuenta la figura 3-21, se tiene los siguientes resultados:

$$C_0' = (C_0 * \{02\}) \oplus (C_1 * \{03\}) \oplus (C_2) \oplus (C_3)$$
$$C_1' = (C_0) \oplus (C_1 * \{02\}) \oplus (C_2 * \{03\}) \oplus (C_3)$$
$$C_2' = (C_0) \oplus (C_1) \oplus (C_2 * \{02\}) \oplus (C_3 * \{03\})$$
$$C_3' = (C_0 * \{03\}) \oplus (C_1) \oplus (C_2) \oplus (C_3 * \{02\})$$

Donde C_0', C_1', C_2', C_3', son el resultado de la MixColumn, y los valores de C_0 , C_1 , C_2 , C_3 , son los valores correspondientes a la columna de datos de entrada. Se puede observar en la matriz de resultados que únicamente se debe determinar la multiplicación de cada uno de los valores de la columna de entrada por los valores constantes {02} y {03}; y reagrupar este resultado como se indica en la ecuación anterior.

Ejemplo:

Se realizará el ejemplo práctico de dos valores distintos de C_0, es decir cuando toma el valor de uno y cero y se procederá a multiplicar por los valores constantes {02} y {03}.

Caso 1 → $x^7 = 1$

C_0 = 11110000 = F0 = $x^7 + x^6 + x^5 + x^4 + 0 + 0 + 0 + 0$

$\{02\} = 00000010 = 02 = x$

$\{03\} = 00000011 = 03 = x + 1$

Multiplicación de C_0 por {02}

$$\begin{array}{r} x^7 + x^6 + x^5 + x^4 \\ \times \quad\quad\quad\quad x \\ \hline x^8 + x^7 + x^6 + x^5 \end{array} \longrightarrow \begin{array}{r} 11110000 \\ \times \quad\quad\quad 10 \\ \hline 111100000 \end{array}$$

Reducir a módulo 11B

Reducir módulo 11B

$$\begin{array}{l} x^8 + x^7 + x^6 + x^5 \quad \big| \; x^8+x^4+x^3+x+1 \\ \underline{x^8 + x^4 + x^3 + x + 1} \qquad\quad 1 \\ x^7 + x^6 + x^5 + x^4 + x^3 + x + 1 \end{array} \longrightarrow \begin{array}{r} 111100000 \\ \oplus 100011011 \\ \hline 011111011 \end{array}$$

Multiplicación de C_0 por {03}

$$\begin{array}{r} x^7 + x^6 + x^5 + x^4 \\ \times \quad\quad\quad x + 1 \\ \hline x^8 + x^4 \end{array} \longrightarrow \begin{array}{r} 11110000 \\ \times \quad\quad\quad 11 \\ \hline 100010000 \end{array}$$

Reducir a módulo 11B

Reducir módulo 11B

$$\begin{array}{l} x^8 + x^4 \quad\quad\quad\quad \big| \; x^8+x^4+x^3+x+1 \\ \underline{x^8 + x^4 + x^3 + x + 1} \qquad 1 \\ x^7 + x^6 + x^5 + x^4 + x^3 + x + 1 \end{array} \longrightarrow \begin{array}{r} 100010000 \\ \oplus 100011011 \\ \hline 000001011 \end{array}$$

De las operaciones anteriormente realizadas se obtiene los siguientes resultados:

$(C_0 * \{02\}) = x^7 + x^6 + x^5 + x^4 + x^3 + x + 1 = 011111011$

$(C_0 * \{03\}) = (C_0 * \{02\}) \oplus C_0 = x^8 + x^4 = 000001011$

Caso 2 $\rightarrow x^7 = 0$

C_0 = 01110000 = 70 = $x^6 + x^5 + x^4 + 0 + 0 + 0 + 0$

$\{02\} = 00000010 = 02 = x$

$\{03\} = 00000011 = 03 = x + 1$

$$\begin{array}{r} x^6 + x^5 + x^4 \\ \hline x \\ \hline x^7 + x^6 + x^5 \end{array} \longrightarrow \begin{array}{r} 01110000 \\ 10 \\ \hline 011100000 \end{array}$$

No es necesario reducir a módulo 11B

$(C_0 * \{02\}) = x^7 + x^6 + x^5 = 011100000$
$(C_0 * \{03\}) = (C_0 * \{02\}) \oplus C_0 = 10010000$

De lo explicado en el caso 1 y 2, se puede realizar las operaciónes de mix_columun en VHDL como se indica en el la figura 3-51.

```
process (c1,polinomio_red)
begin
-- Multiplicación C0 *{02};

      if  c1(7) = '1' then
            c1_red <= c1(6 downto 0)&'0' xor polinomio_red;
       elsif (c1(7) = '0')  then
            c1_red <= c1(6 downto 0)&'0';
      end if;
end process;
-- Multiplicación C0 *{03};
c1_p3<=c1_red xor c1;
```

Figura 3-51. Fragmento de código de la entidad mix_columns.

En estas operaciones se realizan los valores C_1, C_2, C_3, y luego se reagrupan para formar la matriz de salida como se indica a continuación.

$$C_0' = (C_0 * \{02\}) \oplus (C_1 * \{03\}) \oplus (C_2) \oplus (C_3)$$

$$C_1' = (C_0) \oplus (C_1 * \{02\}) \oplus (C_2 * \{03\}) \oplus (C_3)$$
$$C_2' = (C_0) \oplus (C_1) \oplus (C_2 * \{02\}) \oplus (C_3 * \{03\})$$
$$C_3' = (C_0 * \{03\}) \oplus (C_1) \oplus (C_2) \oplus (C_3 * \{02\})$$

La figura 3-52 muestra los resultados de la simulación realizada de la entidad mix_columns, correspondiente a la ronda uno. Los siguientes valores corresponden a los datos de entrada:

cin1: C530D482
cin2 : F2762B67
cin3 : 76638201
cin4 : FEC57B77

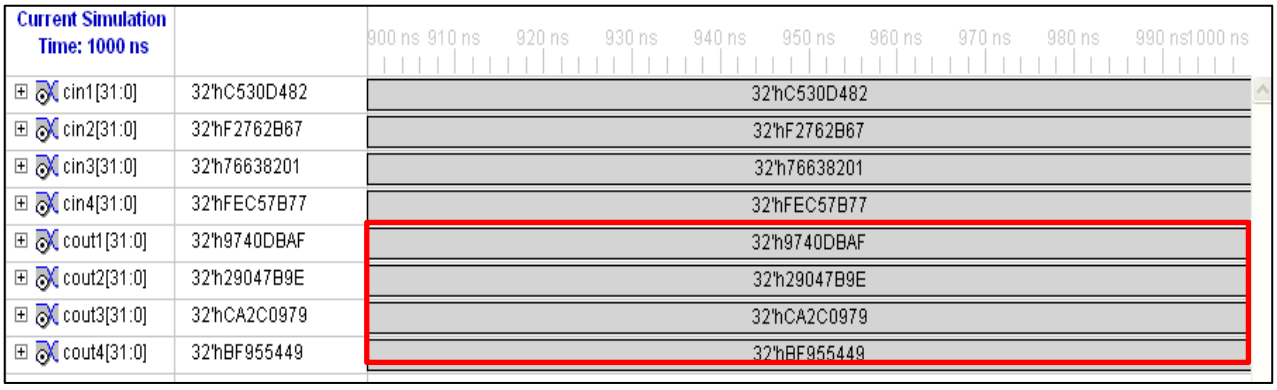


Figura 3-52. Simulación entidad mix_columns, ronda1.

Como se puede observar, los resultados obtenidos son iguales a los desarrollados en la sección 3.2.2.2.

D. Bloque Operación Xor con la llave

En este bloque únicamente se realiza una operación xor entre el resultado que se obtiene de la entidad mix_column y la llave. El código que realiza esta operación se muestra en la figura 3-53.

```
architecture Behavioral of ope_xor is
begin
cout1 <= cin1_1 xor cin1_2;
cout2 <= cin2_1 xor cin2_2;
cout3 <= cin3_1 xor cin3_2;
cout4 <= cin4_1 xor cin4_2;
end Behavioral;
```

Figura 3-53. Fragmento de código de la entidad opr_xor.

La figura 3-54 muestra los resultados de la simulación realizada de la entidad opr_xor, correspondiente a la ronda uno. Los siguientes valores corresponden a los datos de entrada:

cin1_1: 9740DBAF

cin2_1 : 29047B9E

cin3_1: CA2C0979

cin4_1: BF955449

Datos de la llave correspondiente a las palabras: W [4], W [5], W [6] y W [7] respectivamente:

cin1_2: 6FC610A7

cin2_2 : 2A8057EF

cin3_2: 63CA1CA3

cin4_2: 2E8453F3

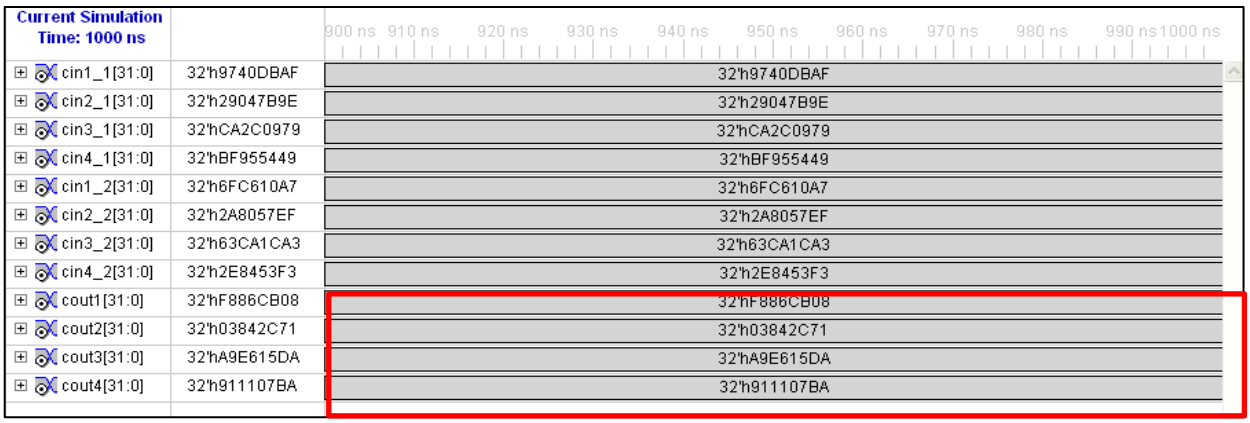


Figura 3-54. Simulación entidad opr_xor, ronda1.

Como se puede observar, los resultados obtenidos son iguales a los desarrollados en la sección 3.2.2.2. Estos valores corresponden al resultado final de la ronda 1.

- Entidad → ronda10, corresponde a la última ronda en el proceso de cifrado. Los puertos de entrada y salida son:

```
cin1 : in  STD_LOGIC_VECTOR (31 downto 0);
cin2 : in  STD_LOGIC_VECTOR (31 downto 0);
cin3 : in  STD_LOGIC_VECTOR (31 downto 0);
cin4 : in  STD_LOGIC_VECTOR (31 downto 0);
subllave: in std_logic_vector (127 downto 0);
cout1 : out  STD_LOGIC_VECTOR (31 downto 0);
cout2 : out  STD_LOGIC_VECTOR (31 downto 0);
cout3 : out  STD_LOGIC_VECTOR (31 downto 0);
cout4 : out  STD_LOGIC_VECTOR (31 downto 0));
```

Como ya se observó en el proceso de simulación del algoritmo en MatLab, esta entidad se encuentra formada por las entidades subByte, shiftRows y la operación xor con la llave. En la figura 3-55, se muestra el diagrama de entidades las mismas que constituyen la entidad ronda10.

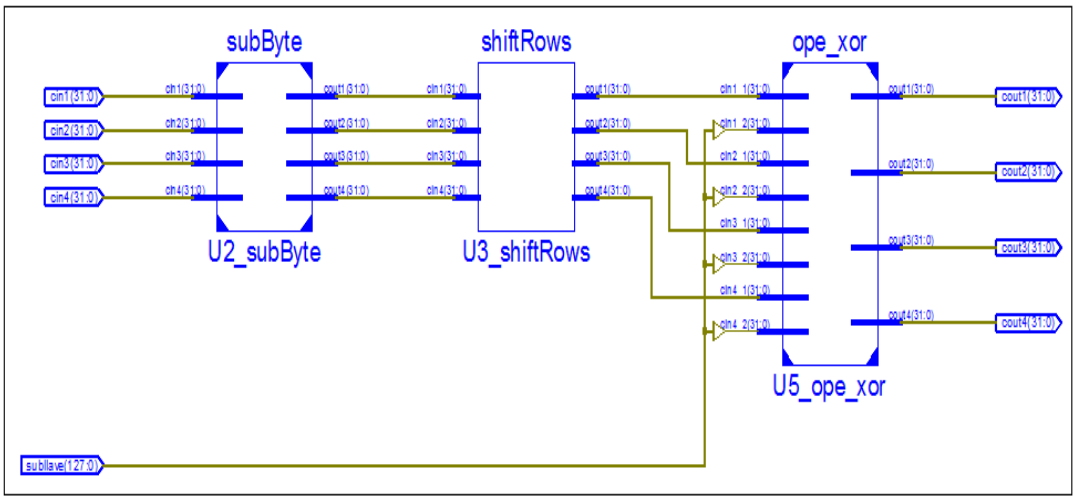


Figura 3-55. Diagrama RTL interno de la entidad ronda 10.

- Entidad → LLAVE_EXPANSION, esta entidad recibe los datos correspondientes a la llave ingresada por el usuario y genera las 44 palabras, que se implementan en el cifrado. Los puertos de entrada corresponden a la llave y los puertos de salida corresponden a las 44 palabras cada una de 32 bits:

```
llave : in  STD_LOGIC_VECTOR (127 downto 0);
w0,w1,w2,w3,w4,w5 : out  STD_LOGIC_VECTOR (31 downto 0);
w6,w7,w8,w9,w10,w11,w12: out  STD_LOGIC_VECTOR (31 downto 0);
w13,w14,w15,w16,w17,w18: out STD_LOGIC_VECTOR (31 downto 0);
w19,w20,w21,w22,w23,w24: out STD_LOGIC_VECTOR (31 downto 0);
w25,w26,w27,w28,w29,w30: out STD_LOGIC_VECTOR (31 downto 0);
w31,w32,w33,w34,w35,w36: out STD_LOGIC_VECTOR (31 downto 0);
w37,w38,w39,w40,w41,w42: out STD_LOGIC_VECTOR (31 downto 0);
w43: out STD_LOGIC_VECTOR (31 downto 0);
```

Las palabras múltiplos de cuatro se ejecutan con una entidad interna que contiene las operaciones AfterRotWord, AfterSubWord, SubBytes, XorRCON, tempXor; mientras que las palabras no múltiplos de cuatro se generan con una operación xor.

La entidad LLAVE_MULTIPLO_4, contiene 4 bloques internos y la interconexión entre ellos. Cada uno de estos bloques corresponde a una operación de transformación de la información de la llave original:

A. AfterRotWord
B. AfterSubWord
C. XorRCON
D. tempXor

Esta entidad es generada una vez y clonada diez veces correspondientes a las palabras mencionadas. El diagrama RTL correspondiente a esta entidad es el de la figura 3-56.

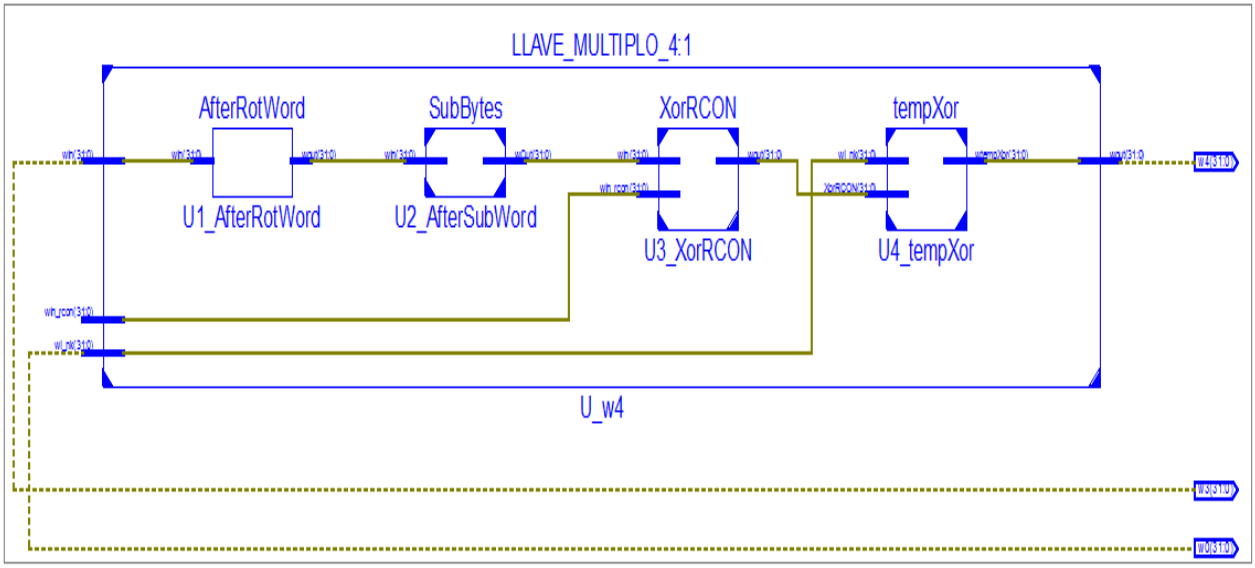


Figura 3-56. Diagrama RTL interno de la entidad LLAVE_MULTIPLO_4.

A. Entidad → AfterRotWord, este bloque fue descrito como un circuito de desplazamiento, recibe como dato de entrada una palabra correspondiente a W3 de 32 bits. Estos bits en el interior se almacenan en un vector std_logic_vector (31 downto 0), y se los desplaza de acuerdo a la posición en la cual se ubiquen en el interior del vector.

Cabe recalcar que el bit más significativo se ubica en la posición 31 y el menos significativo en la posición cero; según la regla de desplazamiento se debe desplazar el vector en 8 posiciones hacia la izquierda. Los puertos de entrada y salida son:

```
win : in  STD_LOGIC_VECTOR (31 downto 0);
wout : out  STD_LOGIC_VECTOR (31 downto 0);
```

B. Entidad → AfterSubWord, en su interior se encuentra una tabla SBOX con 256 valores escritos en formato hexadecimal. La función de esta entidad es la de sustituir los valores de entrada provenientes del bloque AfterRotWord por los valores de la tabla SBOX, para ello los valores de entrada sirven como direcciones de memoria para leer la tabla.

```
architecture Behavioral of AfterRotWord is

      begin
      wout <= wIN(23 DOWNTO 0) & wIN(31 DOWNTO 24);

end Behavioral;
```

Figura 3-57. Fragmento de código de la entidad AfterRotWord.

Para describir esta tabla SBOX, se declaró un tipo de dato definido por el usuario llamado TYPE_MATRIZ como un arreglo de 256 elementos y cada elemento era de tipo STD_LOGIC_VECTOR (7 DOWNTO 0), luego se declaró como valor constante SBOX cuyo tipo de dato era TYPE_MATRIZ, en la constante SBOX se almacenaron los valores de la caja S. La manera en la que se lee un valor de la tabla SBOX se muestra en la figura 3-58.

```
process(byte_Uno)
     begin
     aux(31 downto 24) <= SBOX(to_integer(unsigned(byte_Uno)));

end process;
```

Figura 3-58. Fragmento de código de la entidad AfterSubWord.

En donde byte_Uno representa un byte, luego este es pasado a formato sin signo luego es pasado a entero, este valor está comprendido entre 0 y 255, de esta forma se pueden leer cada uno de los elementos de SBOX.

C. Entidad → XorRCON, la funcionalidad de este bloque es simple, consiste en realizar una operación XOR entre la palabra W que sale del bloque AfterSubWord y el vector RCON.

D. Entidad → tempXor, de manera similar que el anterior bloque, consiste en realizar una operación XOR entre la salida del bloque XorRCON y la palabra Wi_Nk.

La figura 3-59 ilustra la simulación de la entidad *cifrador,* con los siguientes datos de entrada:

- Texto plano →FERNANDAFERNANDA
- Llave → ABCDEFGHIJKLMNOP

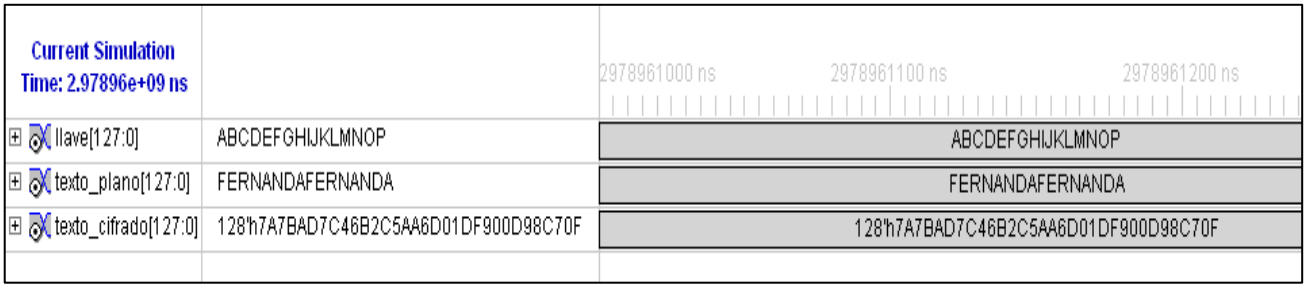


Figura 3-59. Simulación entidad cifrador.

Como se puede observar el resultado corresponde al ejemplo práctico desarrollado en sección 3.2.2.

3.3.2.3 Bloque comparador activador

El propósito de este bloque es el de comparar el valor de salida de la entidad cifrador; si dicho resultado corresponde a cifrar un texto plano donde todos sus elementos son cero, el pin de salida *activador* se pone en cero, caso contrario el pin *activador* se pone en uno y el valor de los pines de *entrada* se colocan en los pines de salida. El diagrama RTL correspondiente a esta entidad es el de la figura 3-60.

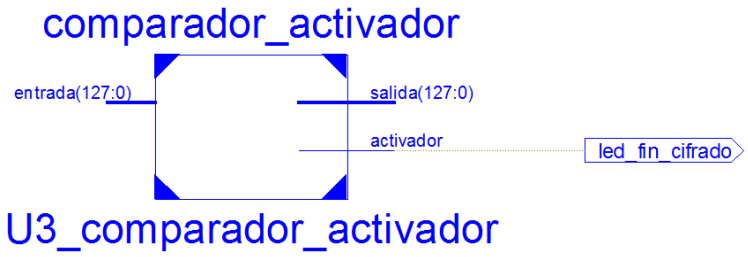


Figura 3-60. Diagrama RTL interno de la entidad comparador_activador.

La entidad interna es la siguiente:

- Entidad → comparador_activador, recibe los datos de la entidad cifrador y los envía a la entidad registro, adicionalmente se enciende un led si la operación se realizó correctamnete. Los puertos de entrada y salida son los siguientes:

```
entrada : in  STD_LOGIC_VECTOR (127 downto 0);
salida : out  STD_LOGIC_VECTOR (127 downto 0);
activador : out  STD_LOGIC;
```

3.3.2.4 Bloque paralelo serial

El propósito de este bloque es el de recibir 128 bits de entrada, almacenarlos en un registro interno para luego devolverlos de 8 en 8 bits. Este bloque está formado principalmente por dos partes: un registro y el control de lectura de ese registro. El registro es un arreglo de 16 palabras, en donde cada palabra es un byte es decir 8 bits, por lo tanto este registro permite almacenar hasta 128 bits, este registro se llena con los 128 bits de entrada cuando el pin de entrada *wr* se encuentra en 1, el proceso de llenado se realiza de la siguiente manera, los 8 bits más significativos se ubican en el registro (0), los siguientes 8 bits en el registro (1) y así sucesivamente.

El proceso de control de lectura utiliza un contador y este proporciona las direcciones de memoria para leer el registro. Para realizar el contador se utiliza la técnica de transferencia de registros. El diagrama RTL correspondiente a esta entidad es el de la figura 3-61. La entidad interna es la siguiente:

- Entidad → registro, recibe un bloque de 128 bits y los envía de 8 en 8 bits. Adicionalmente si el proceso no tiene inconvenientes se enciende un led. Los puertos de entrada y salida son los siguientes:

```
clk : in  STD_LOGIC;
reset : in  STD_LOGIC;
```

```
wr : in  STD_LOGIC;
rd : in  STD_LOGIC;
w_data : in  STD_LOGIC_VECTOR (127 downto 0);
r_data : out  STD_LOGIC_VECTOR (7 downto 0);
```

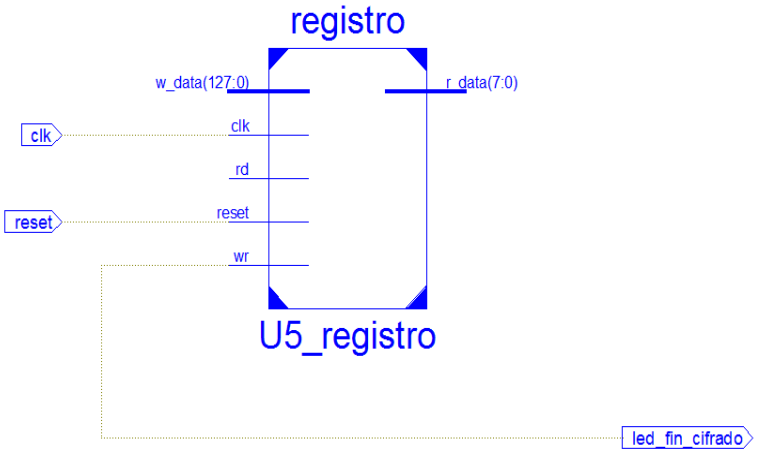


Figura 3-61. Diagrama RTL interno de la entidad registro.

En el siguiente fragmento de código se puede observar el proceso de lectura de la memoria que se encuentra en la entidad registro, el proceso de lectura se realiza mediante un contador que se activa cuando la señal de control *rd* tome el valor de uno, este valor es tomado como una dirección de memoria o puntero. En la figura 3-62, se muestra un fragmento de código que realiza este procedimiento.

```
process(r_ptr_reg, rd, r_ptr_succ)
      begin
            r_ptr_next <= r_ptr_reg;
            if(rd = '1') then
                  r_ptr_next <= r_ptr_succ;
            end if;
end process;
r_ptr_succ <= std_logic_vector(unsigned(r_ptr_reg) + 1);
r_data <= registro(to_integer(unsigned(r_ptr_reg)));
```

Figura 3-62. Fragmento de código de la entidad registro.

El código fuente correspondiente a cada una de las etapas de cifrado en VHDL, se detalla en el anexo 12.

3.3.3 MÓDULO EN MATLAB DE ENVIÓ Y RECEPCIÓN DE DATOS

La interfaz comunicación serial permite comunicar la PC con la tarjeta de entrenamiento por medio de los puertos seriales de ambas. Esta interfaz se indica en la figura 3-63.

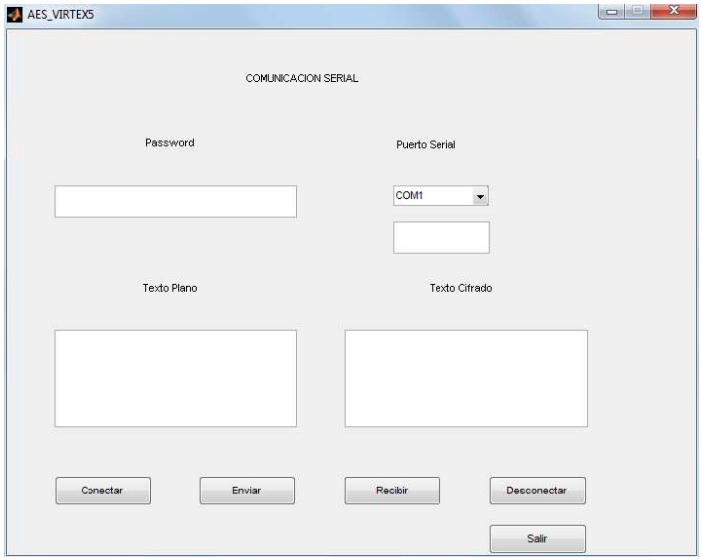


Figura 3-63. Interfaz gráfica de envío y recepción de datos.

La interfaz de comunicación serial se desarrolló en MatLab 7.6.0(R2008a), el usuario deberá ingresar los datos de la siguiente manera y en este orden:

- Ingresar un password o contraseña.
- Ingresar un texto plano que se desee cifrar.
- Escoger un puerto de comunicación serial.
- Presionar el botón Conectar.
- Presionar el botón Enviar.
- Luego se debe verificar el led en la tarjeta de entrenamiento que corresponde al fin de cifrado, es decir este led indica que la tarjeta terminó el proceso de cifrado.

- Se debe presionar en la interfaz de comunicación serial el botón de Recibir, este proceso advierte a la PC que la tarjeta transmitirá datos.
- Los datos correspondientes al texto cifrado se muestran en el *edit box:* Texto Cifrado.
- Finalmente se debe pulsar el botón Desconectar para liberar el puerto serial.

El código fuente correspondiente al módulo en MatLab de envío y recepción de datos, se detalla en el anexo 12.

REFERENCIAS

[3.1] (2012) Grupo (Matemática). [Online]. Disponible: http://es.wikipedia.org/wiki/Grupo_(matem%C3%A1tica)

[3.2] W. Stallings, *Cryptography and Network Security Principles and Practices*. Cuarta Edición, 2005.

[3.3] J. Daemen y V. Rijrnen, *The Design of Rijndael AES – The Advanced Encryption Standard*, Primera Edición, 2002.

[3.4] Advanced Encryption Standard (AES), Federal Information Processing Standards Publication 197, 2001.

[3.5] W. Stallings, *Fundamentos de Seguridad en Redes, Aplicaciones y Estándares*. Segunda Edición, Madrid, 2004.

[3.6] P. Chu, *FPGA Prototyping by VHDL Examples*. Primera Edición, 2008.

CAPÍTULO 4

PRUEBAS Y RESULTADOS

Una vez que se ha expuesto el cifrado S-DES, los estándares DES y AES en MatLab como en VHDL; en este capítulo se procede a realizar la verificación de los resultados obtenidos a nivel de software y hardware.

4.1 RESULTADOS S-DES

El proceso de pruebas realizado con el cifrador S-DES se divide en:

- S-DES en MatLab.
- S-DES en VHDL.

4.1.1 S-DES EN MATLAB

Cada una de las etapas de cifrado y la interfaz gráfica, se desarrollaron en MatLab 7.6.0 (R2008a). El usuario deberá ingresar un texto plano y la llave en formato binario. El proceso de cifrado se ejecutará con el botón ***Enter***.

Para realizar el ejemplo práctico se tomarán los siguientes datos:

- Texto plano → A→ 01000001
- Llave → 0100000101

El resultado se indica en la figura 4-1, en formato hexadecimal:

- Texto cifrado (hexadecimal) → D1

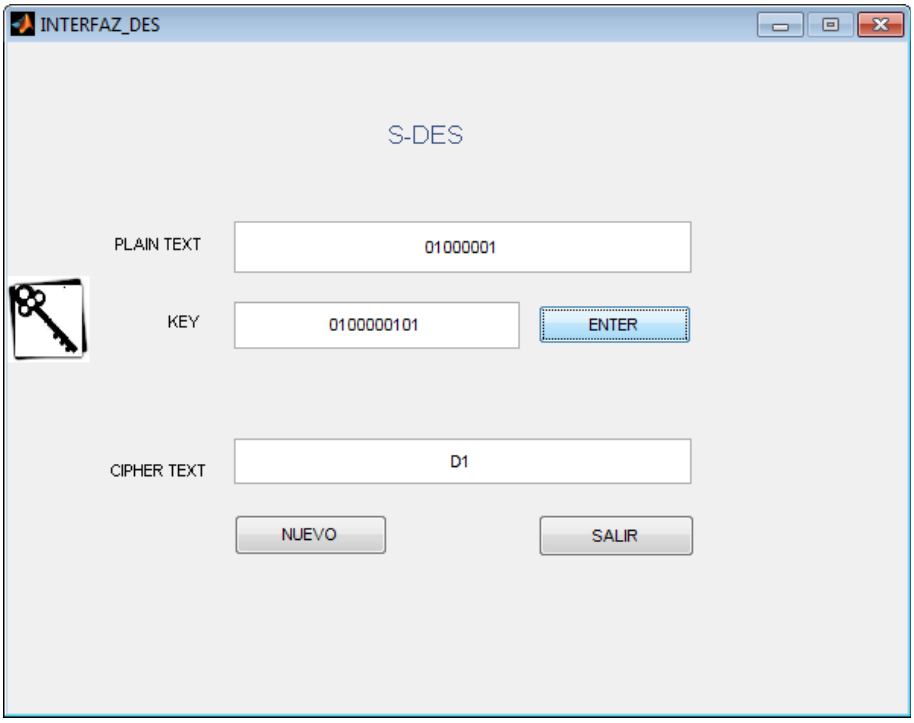


Figura 4-1. Ejemplo cifrado S-DES.

Para realizar la comprobación de resultados se emplea el programa *Simplified DES Calculator (SDES), v1.0.* Este programa se encuentra en la página web:

http://buzzard.ups.edu/sdes/sdes.html

Donde se debe ingresar el texto plano y la llave en formato binario. El programa permite observar cada etapa de cifrado paso a paso.

Para el ejemplo práctico se tiene los siguientes datos:

- Texto plano → A→ 01000001
- Llave → 0100000101

El resultado se observa en la figura 4-2.

- Texto cifrado → D1 →11010001

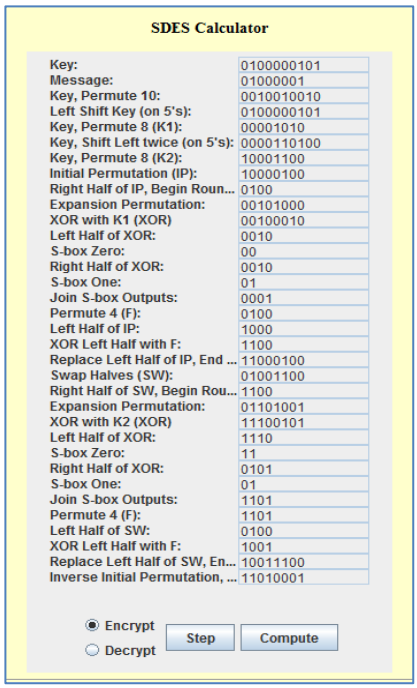


Figura 4-2. Ejemplo realizado en *Simplified DES Calculator.*

El resultado es el esperado, el programa realiza el proceso de cifrado de la forma correcta. Como se explicó al inicio del capítulo 2, S-DES, es una versión reducida del estándar DES. Tiene propiedades similares a DES pero este abarca un texto plano y una llave más reducido. El objetivo de realizar este programa es comprender de una forma educativa las distintas etapas del algoritmo de cifrado DES.

4.1.2 S-DES EN VHDL

4.1.2.1 Síntesis

Una vez que se realiza el proceso de síntesis, la capacidad del dipositivo FPGA, que se ha utilizado se indica a continuación:

Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	164	69120	0%
Number of Slice LUTs	387	69120	0%
Number of fully used LUT-FF pairs	164	387	42%
Number of bonded IOBs	12	640	1%
Number of BUFG/BUFGCTRLs	1	32	3%

Se puede observar que los recursos utilizados son pocos, ya que este es un ejemplo básico introductorio a VHDL y la complejidad del algoritmo S-DES en relación a las operaciones matemáticas necesarias para su implementación son mínimas comparada con diseños posteriores.

Se usa el 1% de IOBs correspondiente a 12 de 640 puertos disponibles de entrada y salida.

Para el diseño de los bloques en hardware se usan 387 de 69190 slice. Como elementos de almacenamiento se usa un 42% correspondiente a 164 de 387 LUT-FF.

4.1.2.2 Implementación

Una vez realizada la síntesis y al no haber errores en esta etapas se procede a generar el archivo UCF, de asignación de pines para la los puertos de entrada y salida. Los puertos del dispositivo FPGA correspondientes a xc5vlx110t se encuentra en el achivo UG247 correspondiente al anexo 13. El archivo .ucf que se genera se observa en la figura 4-3.

```
1
2   # PlanAhead Generated physical constraints
3
4   NET "btn" LOC = AJ6;
5   NET "clk" LOC = AH15;
6   NET "led[0]" LOC = H18;
7   NET "led[1]" LOC = L18;
8   NET "led[2]" LOC = G15;
9   NET "led[3]" LOC = AD26;
10  NET "led[4]" LOC = G16;
11  NET "led[5]" LOC = AD25;
12  NET "led[6]" LOC = AD24;
13  NET "led[7]" LOC = AE24;
14  NET "reset" LOC = AC24;
15  NET "rx" LOC = AG15;
```

Figura 4-3. Archivo .UCF de asignación de pines en el dispositivo FPGA.

Se continúa con las etapas de *translate, map y place & route*, en estas etapas es muy importante que no existan *warnings* pues el diseño en el dispositivo FPGA no

funciona correctamente. Los puertos de entrada y salida correspondientes a este sistema de cifrado son:

1. Reloj 100MHz →clk : in STD_LOGIC;
2. Botón reset → reset : in STD_LOGIC;
3. Botón enviar datos →btn : std_logic_vector(2 downto 0);
4. Puerto recepción datos RS-232→ rx : in STD_LOGIC;
5. Los 8 led de la tarjera visualización resultado → led : out STD_LOGIC_VECTOR (7 downto 0);

La ubicacón física de estos puertos se indica en la figura 4-4.

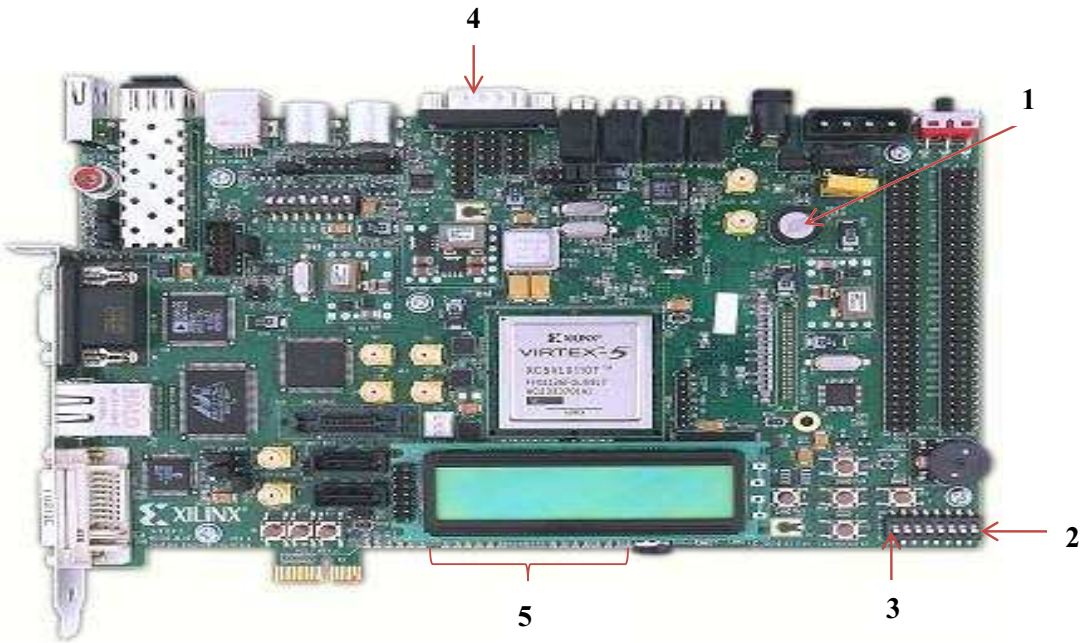


Figura 4-4. Ubicación de puertos del sistema de cifrado S-DES.

El siguiente paso es generar el archivo .bit, en la opción ***Generate Programming File.***

El archivo generado es ***testuart.bit,*** dicho archivo se carga en el dispositivo FPGA usando el programa ***ISE Impact,*** como indica la figura 4-5.

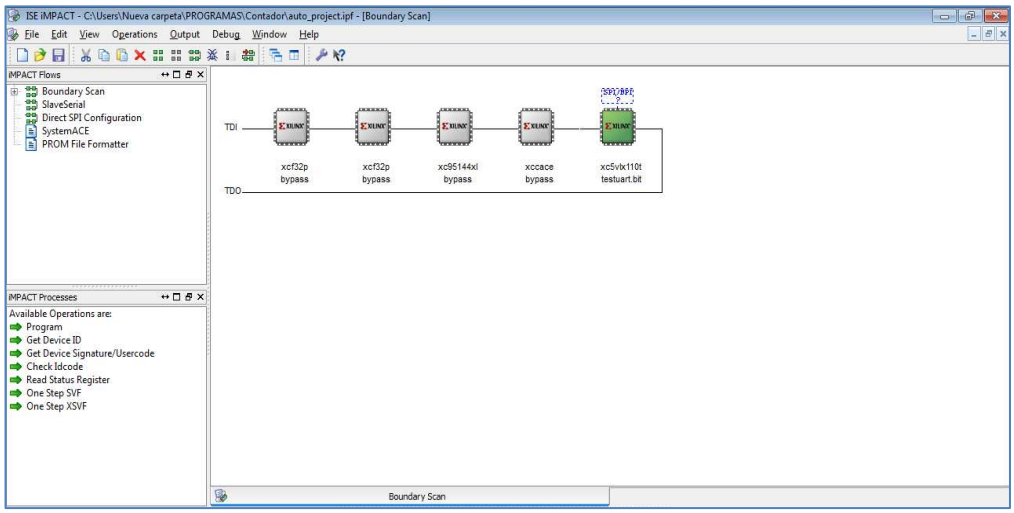


Figura 4-5. Archivo testuart.bit, cargado en el dispositivo FPGA.

Este proceso de describe a detalle en el anexo 4. Una vez que se carga el archivo testuart.bit, en el dispositivo FPGA, se muestra un mensaje de ***Program Succeeded,*** como indica la figura 4-6.

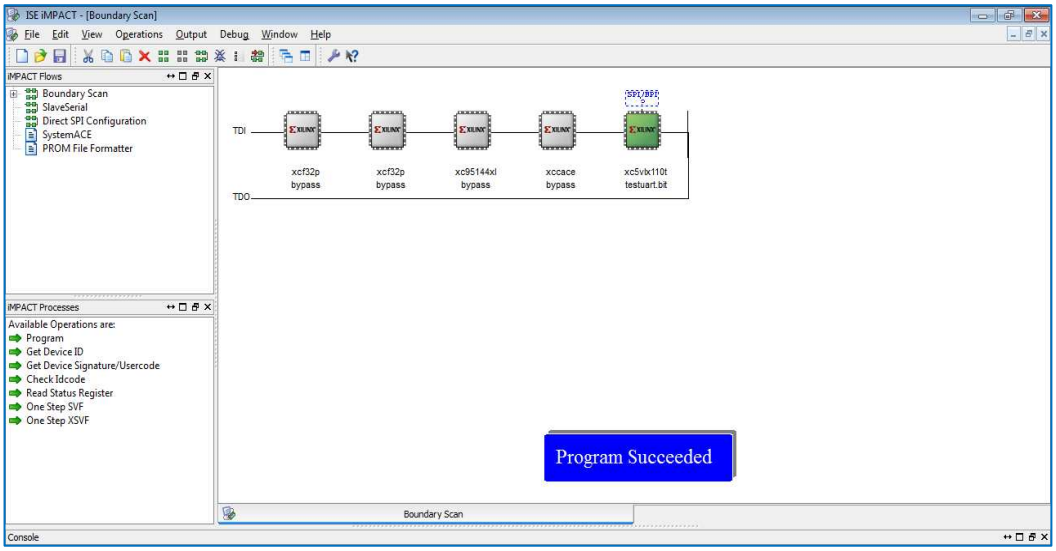


Figura 4-6. Archivo testuart.bit, cargado en el dispositivo FPGA en forma satisfactoria.

4.1.2.3 Pruebas

Para realizar las pruebas se debe contar con los siguientes elementos:

1. Una Laptop, que tenga instalado hyperterminal para la trasmisión serial.
2. Kit de desarrollo Virtex 5:

    - Una tarjeta de entrenamiento XUPV5-LX110T.
    - Una tarjeta flash de 1GB.
    - Cables XUP USB para programar la tarjeta de entrenamiento.
    - Una fuente de alimentación 6A.

3. Un cable USB – Serial RS-232.
4. Un cable RS-232 con terminales hembra – hembra.

En la figura 4-7, se indican cada uno de los elementos mencionados.

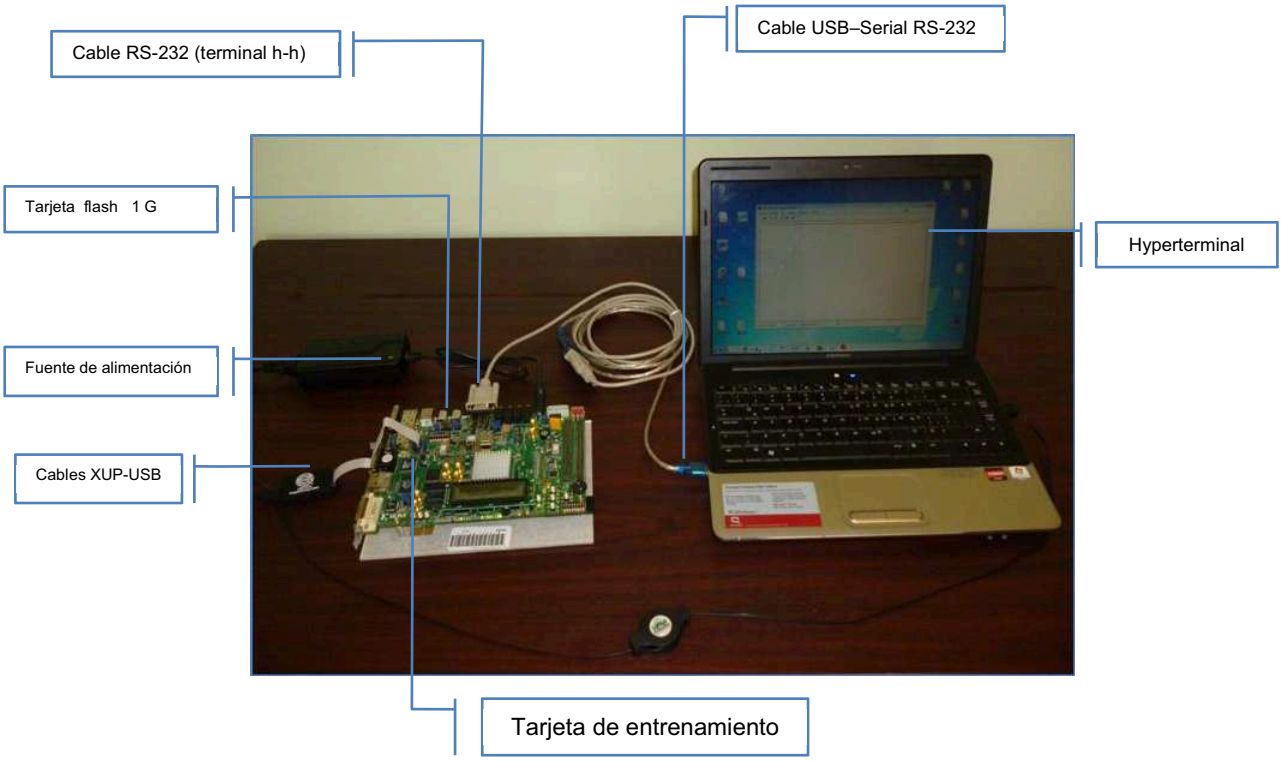
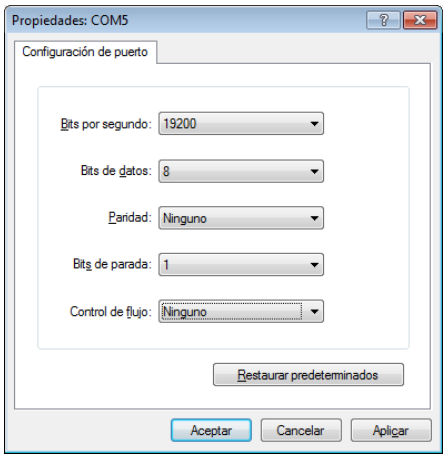


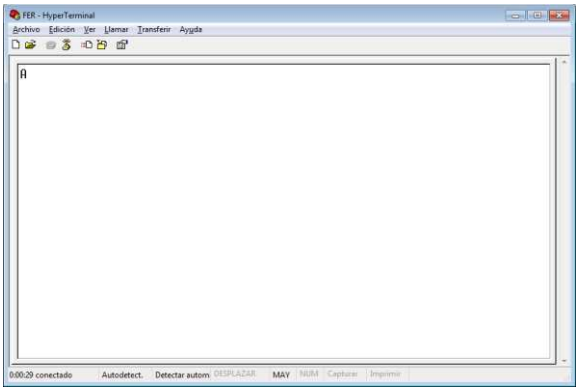
Figura 4-7. Elementos para ejecutar el cifrador S-DES con un dispositivo FPGA.

Las propiedades de comunicación en el hyperterminal son las siguientes:



Dato a ser cifrado:

- Texto plano →A →01000001



Llave generada en el dispositivo FPGA:

- Llave →0100000101

Resultado:

- Texto Cifrado →1 1 0 1 0 0 0 1

Como se explicó anteriormente el resultado se va mostrar en la tarjeta de entrenamiento usando los leds, como indica la figura 4-8. El resultado se puede leer de derecha a izquierda.

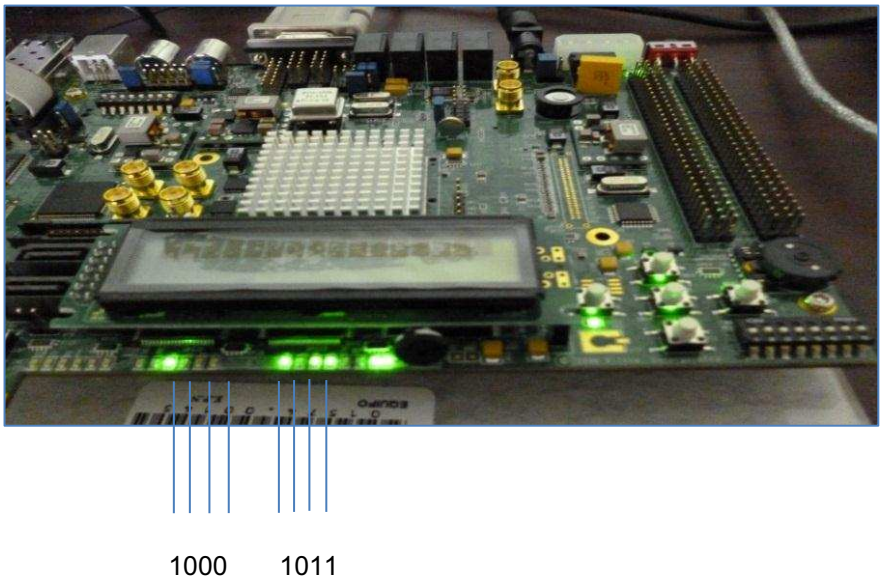


Figura 4-8. Resultado del cifrador obtenido en la tarjeta de entrenamiento.

4.2 RESULTADOS DES

4.2.1 DES EN MATLAB

Cada una de las etapas de cifrado y la interfaz gráfica, se desarrollaron en MatLab 7.6.0 (R2008a). El usuario deberá ingresar un texto plano y la llave en formato ASCII. El proceso de cifrado se ejecutará con el botón ***Enter***.

Para realizar el ejemplo práctico se tomarán los siguientes datos:

- Texto plano → fernanda
- Llave → fernanda

El resultado se observa en la figura 4-9.

- Texto cifrado (hexadecimal) → 81EC995EDD24B072

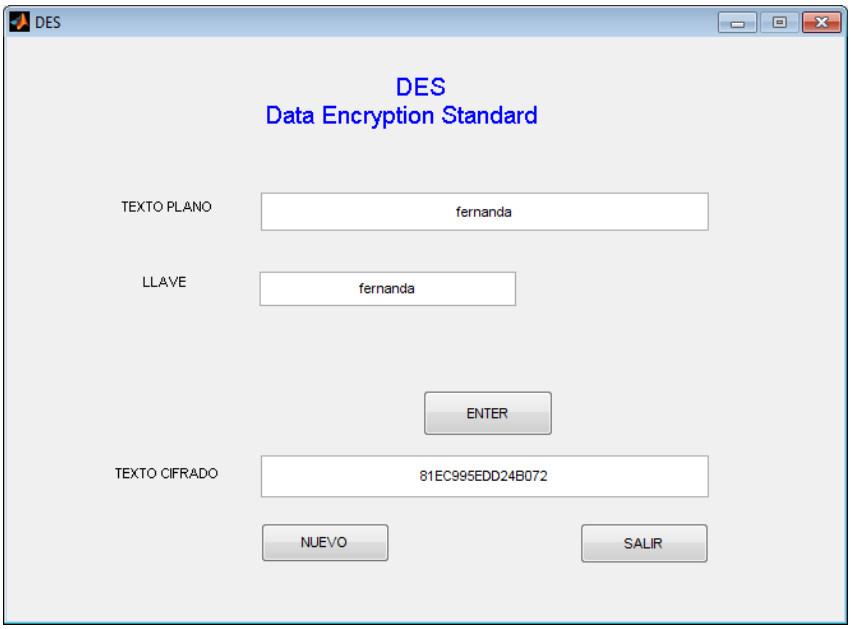


Figura 4-9. Ejemplo cifrado en algoritmo DES.

El proceso de verificación de resultados se realizará con el programa safeDes, que se obtiene de la página web:

http://www.criptored.upm.es/software/sw_m001j.htm.

La interfaz se observa en la figura 4-10.

Figura 4-10. Interfaz safeDES.

Los pasos para realizar el cifrado se describen a continuación:

- Ingreso texto plano → fernanda

En la figura 4-11, se ilustra el ingreso del texto plano en formato ASCII.

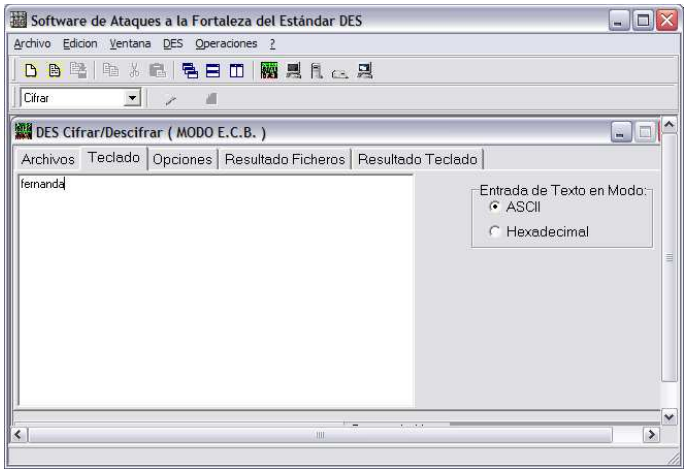


Figura 4-11. Ingreso del texto plano en la interfaz safeDES.

- Ingreso de la llave (ASCII) → fernanda

En la figura 4-12, se ilustra el ingreso de la llave en formato ASCII.

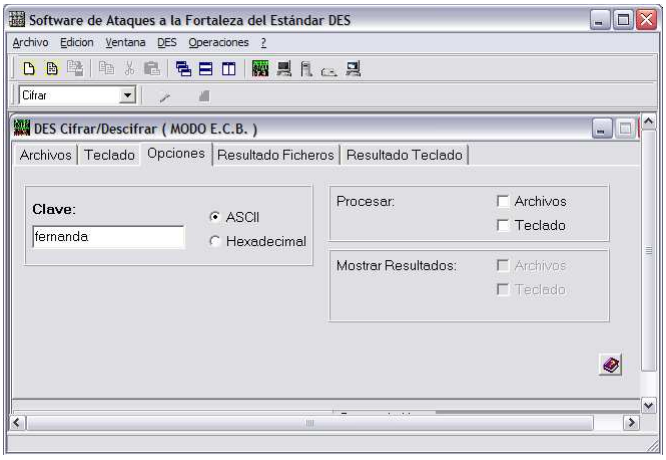


Figura 4-12. Ingreso de la llave en la interfaz safeDES.

- Texto cifrado (HEX) → 81EC995EDD24B072

En la figura 4-13, se indica el texto cifrado en formato hexadecimal.

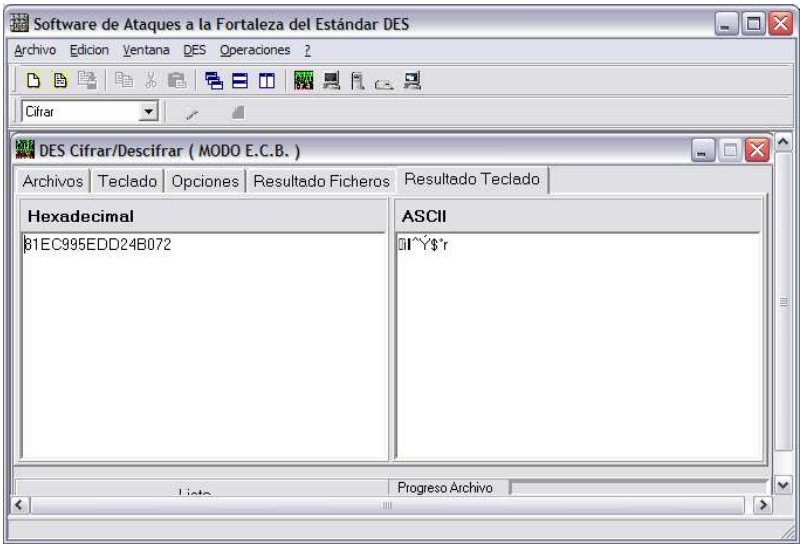


Figura 4-13. Texto cifrado en el programa safeDES.

4.3 RESULTADOS AES

El proceso de pruebas realizado en el estándar AES se divide en:

- AES en MatLab.
- AES en VHDL.

4.3.1 AES EN MATLAB

Cada una de las etapas de cifrado y la interfaz gráfica, se desarrollaron en MatLab 7.6.0 (R2008a). En este diseño el usuario puede usar el programa para:

A. Cifrar. – El usuario deberá ingresar un texto plano y la llave en formato ASCII. El proceso de cifrado se ejecutará con el botón ***Cifrar*** y es necesario realizar la confirmación de la llave denominada contraseña.
Para realizar el ejemplo práctico se tomarán los siguientes datos:

   - Texto plano → FERNANDAFERNANDA
   - Llave → ABCDEFGHIJKLMNOP

El resultado se observa en la figura 4-14.

- Texto cifrado (HEX) → 7A7BAD7C46B2C5AA6D01DF900D98C70F

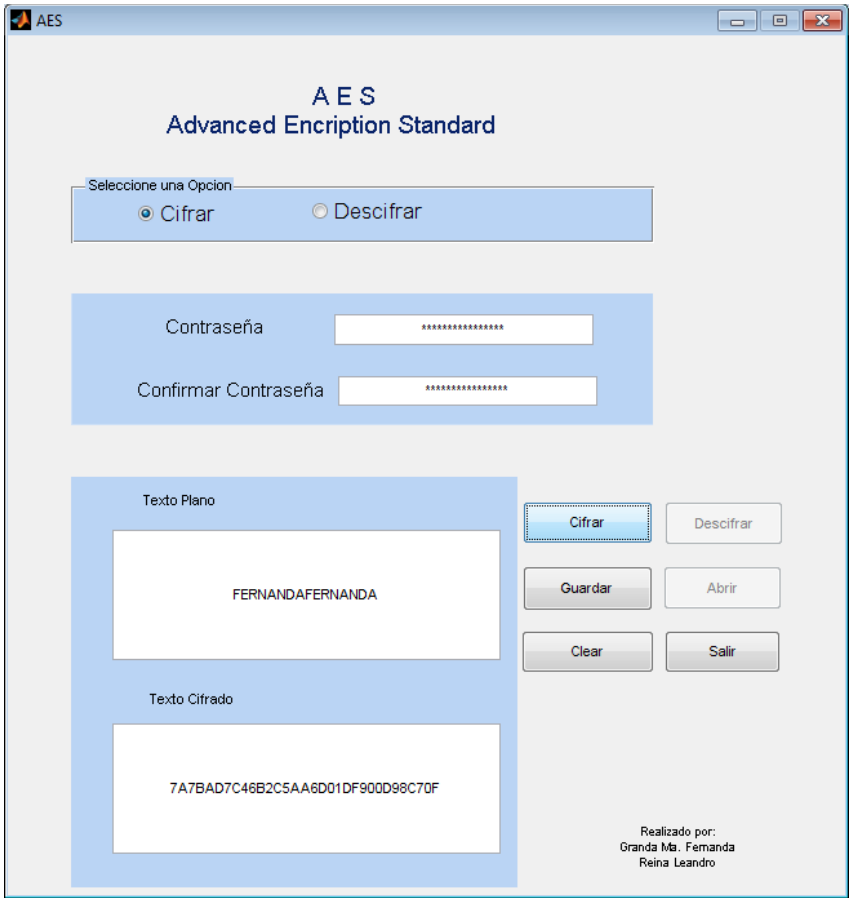


Figura 4-14. Ejemplo práctico: cifrado con AES en MatLab.

El usuario tiene la opción de guardar en un archivo el texto cifrado, usando el botón **Guardar,** en caso de requerir dicho archivo para el proceso de descifrado.

B. Descifrar. – El usuario deberá ingresar un texto cifrado en formato hexadecimal y la llave en ASCII. El proceso de descifrado se ejecutará con el botón **Descifrar** y es necesario realizar la confirmación de la llave denominada contraseña. Se puede hace uso del botón **Abrir,** si el usuario ha guardado algún texto cifrado. Para realizar el ejemplo práctico se tomarán los siguientes datos:

- Texto cifrado (HEX) → 7A7BAD7C46B2C5AA6D01DF900D98C70F
- Llave → ABCDEFGHIJKLMNOP

El resultado se observa en la figura 4-15.

- Texto plano → FERNANDAFERNANDA

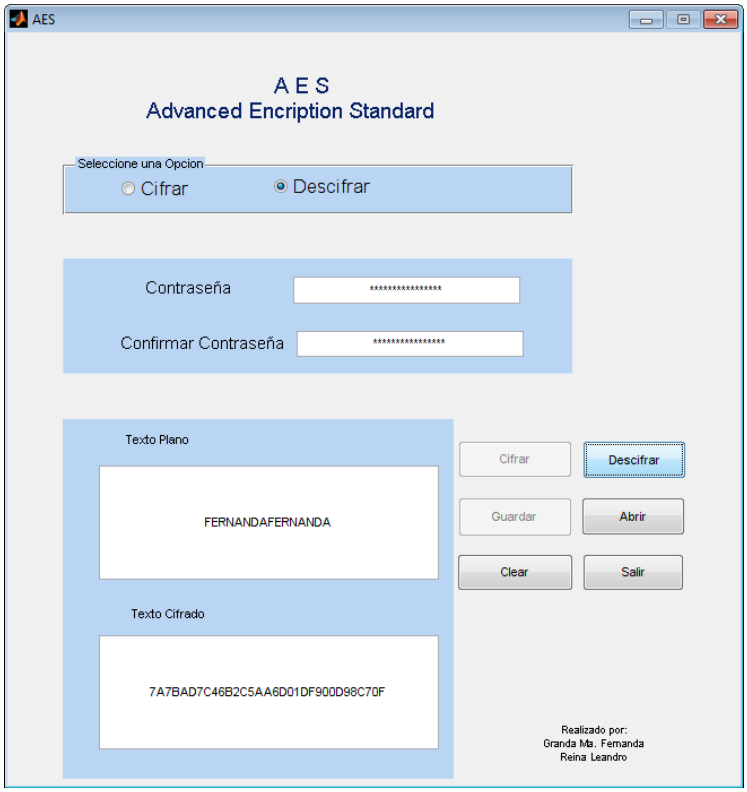


Figura 4-15. Ejemplo práctico: descifrado con AES en MatLab.

4.3.2 AES EN VHDL

4.3.2.1 Síntesis

Una vez que se realiza el proceso de síntesis la capacidad del dipositivo FPGA, que se ha utilizado se indica a continuación:

Device Utilization Summary				[-]
Slice Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Registers	613	69,120	1%	
Number used as Flip Flops	613			
Number of Slice LUTs	9,732	69,120	14%	
Number used as logic	9,730	69,120	14%	
Number of occupied Slices	3,325	17,280	19%	
Number of LUT Flip Flop pairs used	10,088			
Number with an unused Flip Flop	9,475	10,088	93%	
Number with an unused LUT	356	10,088	3%	
Number of fully used LUT-FF pairs	257	10,088	2%	
Number of slice register sites lost to control set restrictions	7	69,120	1%	
Number of bonded IOBs	9	640	1%	
Number of LOCed IOBs	9	9	100%	
Number of BUFG/BUFGCTRLs	1	32	3%	
Average Fanout of Non-Clock Nets	11.30			

Se puede observar que en este diseño, no se sobrepasan los recursos disponibles en el dispositivo FPGA.

Se usa el 1% de IOBs correspondiente a 9 de 640 puertos disponibles de entrada y salida, los cuales son usados al 100%.

Para el diseño de los bloques en hardware se usan 3325 de 17280 slice, correspondientes al 19%. Como elementos de almacenamiento se usa de un 2% correspondiente a 357 de 10088 LUT-FF.

4.3.2.2 Implementación

Una vez realizada la síntesis, se procede a generar el archivo UCF, de asignación de pines para la los puertos de entrada y salida. Los puertos del dispositivo FPGA

correspondientes a xc5vlx110t se encuentra en el achivo UG247 correspondiente al anexo 13. El archivo .ucf que se genera se observa en la figura 4-16.

```
1
2  # PlanAhead Generated physical constraints
3
4  NET "btn_debounce" LOC = AE26;
5  NET "clk" LOC = AH15;
6  NET "led_fin_cifrado" LOC = E8;
7  NET "rd_uart" LOC = AC24;
8  NET "reset" LOC = AC25;
9  NET "rx" LOC = AG15;
10 NET "rx_empty" LOC = AF13;
11 NET "tx" LOC = AG20;
12 NET "tx_full" LOC = AG12;
```

Figura 4-16. Archivo .UCF de asignación de pines en el dispositivo FPGA.

Los puertos de entrada y salida correspondientes a este sistema de cifrado son:

1. Reloj 100MHz →clk : in STD_LOGIC;
2. Botón reset → reset : in STD_LOGIC;
3. Botón enviar datos →btn_debouce : STD_LOGIC;
4. Puerto recepción datos RS-232→ rx : in STD_LOGIC;
5. Puerto transmisión datos RS-232→tx : out STD_LOGIC;
6. Led fin de cifrado → led : out STD_LOGIC
7. Recepción → rx_empty: : out STD_LOGIC;
8. Transmisión full → tx_full: out STD_LOGIC;
9. Lectura UART → rd_uart: in STD_LOGIC;

La ubicación física de estos puertos se indica en la figura 4-17.

Se continúa con las etapas de *translate*, *map* y *place & route*. El siguiente paso es generar el archivo .bit, en la opción ***Generate Programming File.*** El archivo generado es ***aes_total.bit,*** dicho archivo se carga en el dispositivo FPGA usando el programa ***ISE Impact***. Este proceso de describe a detalle en el anexo 13.

Una vez que se carga el archivo testuart.bit, en el dispositivo FPGA, se muestra un mensaje de ***Program Succeeded.***

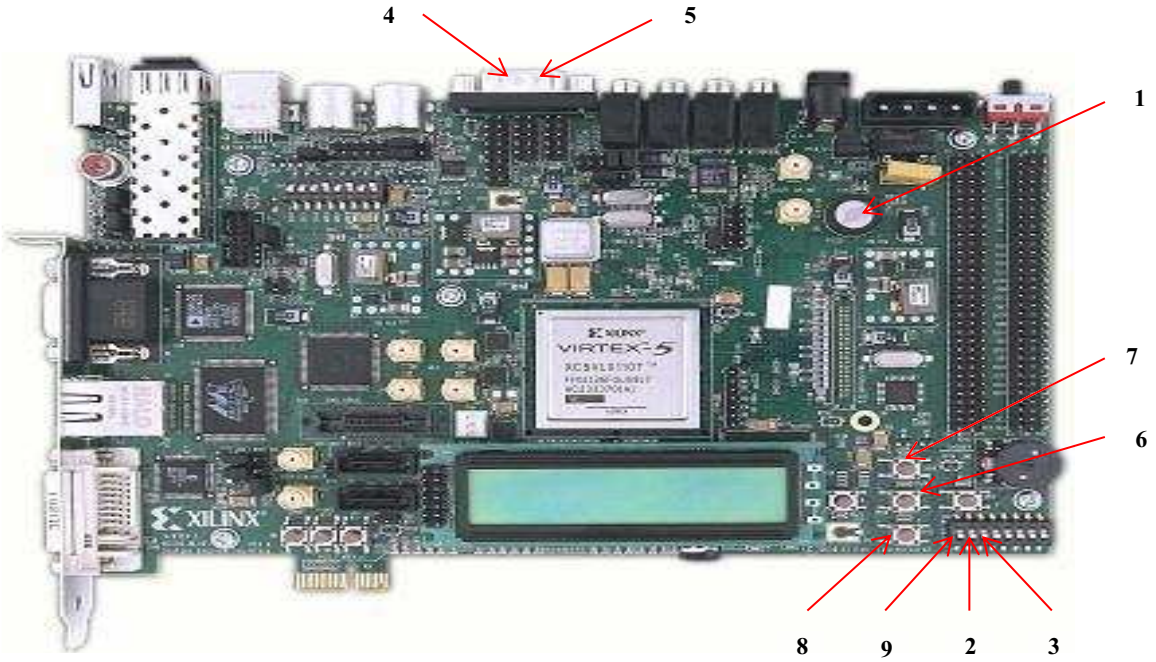


Figura 4-17. Ubicación de puertos del sistema de cifrado AES.

4.3.2.3 Pruebas

Para la realizar las pruebas se debe contar con los siguientes elementos:

1. Una Laptop, que tenga instalado hyperterminal para la trasmisión serial.
2. Kit de desarrollo Virtex 5:
   - Una tarjeta de entrenamiento XUPV5-LX110T.
   - Una tarjeta flash de 1GB.
   - Cables XUP USB para programar la tarjeta de entrenamiento.
   - Una fuente de alimentación 6A.
3. Un cable USB – Serial RS-232
4. Un cable RS-232 con terminales hembra - hembra

En la figura 4-7, se indican cada uno de los elementos mencionados.

Dato a ser cifrado:

- Texto plano →FERNANDAFERNANDA

Llave ingresada por el usuario:

- Llave → ABCDEFGHIJKLMNOP

En ingreso de los datos antes descritos, se observa en la figura 4-18, para mayor facilidad del usuario el texto plano y la llave se ingresan en formato ASCII; y para la comprobación del resultado el texto cifrado se muestra en formato hexadecimal. El usuario debe seleccionar el puerto de comunicación serial:

- Puerto comunicación serial → COM7

Resultado:

- Texto Cifrado → 7A7BAD7C46B2C5AA6D01DF900D98C70F

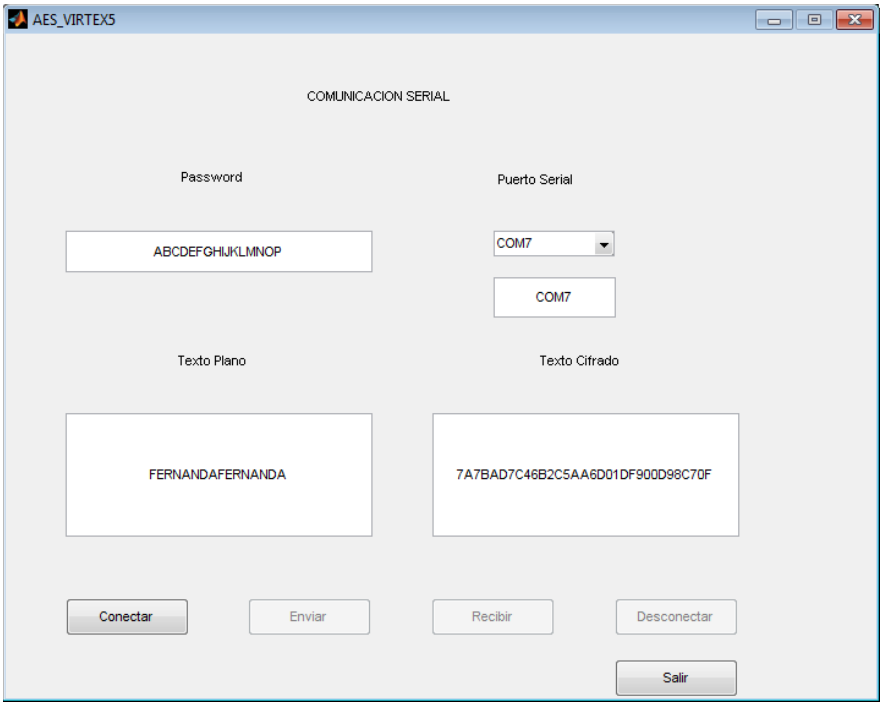


Figura 4-18. Ejemplo práctico: sistema de cifrado AES usando un dispositivo FPGA.

El proceso de verificación de resultados se realizará con la librería cripto de OPENSSL, para lo cual se siguen los siguientes pasos:

1. Se debe crear un archivo .txt, con el texto plano que se desee cifrar, como se observa, en la figura 4-19.

   El nombre del archivo es textoplano.txt y el texto plano a cifrar es:

   FERNANDAFERNANDA

Figura 4-19. Ingreso texto plano para el proceso de cifrado AES en openssl.

2. Para cifrar el archivo textoplano.txt se utiliza el comando:

openssl enc -aes-128-cbc -in textoplano.txt -out textocifrado.enc -nosalt -iv 0000000000000000 -K 41424344454647484 94A4B4C4D4E4F50

3. Para observar el texto cifrado en formato hexadecimal se utiliza el siguiente comando:

xxd -l 120 -c 12 textocifrado.enc

Como se observa en la figura 4-20, el resultado es el esperado y se indica en formato hexadecimal.

Figura 4-20. Resultado cifrado AES en openssl.

CAPÍTULO 5

CONCLUSIONES Y RECOMENDACIONES

5.1 CONCLUSIONES

Se verifica que el estándar de cifrado AES, posee una estructura óptima y eficiente para ser implementado en hardware, ya que las operaciones que se utilizan para transformar la información no consumen muchos recursos de hardware.

El diseño planteado, es en su mayoría un diseño combinacional es decir que el valor de las salidas dependen del valor de las entradas, la ventaja de este tipo de diseño es que se obtiene mayor rapidez para realizar las operaciones de cifrado, la desventaja de este tipo de diseño es que se utilizan más recursos del FPGA. Otra alternativa de diseño sería el diseño con retroalimentación ya que este nos permite en su mayoría reutilizar el hardware, el problema de este diseño es que los resultados son más lentos ya que el valor de las salidas dependen del valor de las entradas y de un estado anterior, dado que el FPGA utilizado corresponde a la familia Virtex 5, el misma que posee más recursos de hardware se escogió el diseño combinacional.

El bloque que realiza la operación de la mezcla de columnas (mix columns), en un diseño inicial se describió como una operación de multiplicación y reducción de polinomios, dicha solución consumía demasiados recursos de hardware a tal punto que el esquema total de diseño del estándar AES no podía ser realizado, por tal motivo este bloque fue rediseñado.

En un diseño inicial para la implementación de S-DES, se intentó usar el código realizado en MatLab, asimilarlo y convertirlo en VHDL, para lo cual se usa subprogramas (funciones, subrutinas y procesos). Este método no resultó efectivo, ya que se creó código con la sintaxis correcta, pero no sintetizable, ya

que lo que se intenta hacer en VHDL, no es programar en un nuevo lenguaje, sino modelar el algoritmo S-DES en hardware. Una vez comprendido el objetivo de VHDL, se usa una descripción estructural del algoritmo, creando un circuito basado en componentes más pequeños, especificando cada una de sus partes y conexiones. La unión de cada una de estas partes denominadas entidades se realiza mediante components.

Se garantiza que el diseño realizado del estándar AES y el texto cifrado resultante realizado en el FPGA es el correcto, ya que se comprobó el funcionamiento bajo el sistema operativo Ubuntu, mediante la herramienta OPENSSL; cuyos resultados obtenidos son exactamente los mismos que los del sistema implementado físicamente. Adicionalmente se implementó un escenario de pruebas que comprueben el correcto funcionamiento del diseño, las cuales consistieron en ingresar un texto plano en la interfaz gráfica de MatLab, luego estos datos son enviados al dispositivo FPGA, el que realiza el proceso de cifrado y se obtiene el texto codificado como se esperaba.

El diseño jerárquico y por bloques, fue el más adecuado para la realización de este trabajo, ya que se puede implementar el esquema general de forma individual entidad por entidad, de esta manera se prueba el funcionamiento individual de cada una.

El diseño del sistema que se implementó, se basó en el número de recursos de hardware disponibles en el FPGA y en la velocidad de procesamiento del mismo, para lograr estos objetivos se analizó la arquitectura interna del FPGA y de este análisis se concluye que para lograr la velocidad de respuesta del cifrador se realizó un diseño que en su mayoría sea un circuito paralelo, es decir que realice las operaciones del estándar de cifrado de forma simultánea.

5.2 RECOMENDACIONES

Se recomienda implementar un sistema de cifrado AES más equilibrado en el sentido de que se use la menor cantidad de recursos del FPGA, sin que esto disminuya la velocidad de procesamiento del sistema implementado.

Para optimizar la cantidad de recursos utilizados en los FPGAs, se debe utilizar una metodología de descripción de hardware que se conoce como la transferencia de registros (RT por sus siglas en inglés), esta metodología contempla la reutilización de registros y localidades de memoria del chip, adicionalmente este tipo de descripción permite implementar modelos o sistemas de programación secuenciales.

Se debe tener muy en cuenta la sincronización cuando se realice un determinado diseño de un sistema, ya que si se realiza una errónea forma de descripción del circuito, cuando este se sintetiza se infiere la creación de *latches* no previstos, los mismos que dan lugar a la desincronización de las entidades internas del circuito principal, produciéndose resultados de salida erróneos.

Se recomienda tener mucho cuidado cuando se describe un determinado circuito utilizando el lenguaje VHDL, cuando se realiza los procesos *de traslate, map y place & route*, si en este proceso se generan *warnings* se debe prestar mucha atención a los mismos, ya que ellos nos pueden informar sobre la creación de *latches* o la desconexión de un determinado puerto de una entidad.

ANEXO 1

CÓDIGO MATLAB DEL ALGORITMO DE CIFRADO SDES

(Se encuentra en digital)

ANEXO 2

ARREGLO DE COMPUERTAS PROGRAMABLES (FPGA)

2.1 DISPOSITIVOS FPGAs

Un FPGA (*Field Programmable Gate Array*), es un dispositivo lógico programable, que contiene bloques de lógica configurable y las interconexiones entre estos bloques lógicos. Un bloque lógico puede ser configurado o programado para realizar funciones simples, y las interconexiones son configuradas para permitir la unión entre los bloques lógicos.

El primer FPGA basado en memoria estática fue propuesto por Wahlstrom en 1967. Los FPGAs contemporáneos de matriz de celdas homólogas, permiten funciones de entrada de toda lógica y elementos de almacenamiento que podrían aplicarse en cada celda lógica. Además, las conexiones entre bloques programables podrían fácilmente cambiarse (a través de la configuración de la memoria) para permitir la aplicación de una variedad de topologías de circuito, aunque la memoria estática ofrece un enfoque más flexible de programación del dispositivo, requiere un aumento significativo en el área del semiconductor por conmutador programable en comparación con las implementaciones de ROM (*Read Only Memory*).

Es probable que este problema retrasó la introducción comercial de dispositivos programables de memoria hasta mediados de los 1980, cuando el costo por transistor se redujo.

Xilinx en 1984, introdujo el primer FPGA de época moderna. Contenía la clásica matriz de bloques de lógica configurable. De ese primer FPGA que contenía 58 entradas y 64 salidas, los FPGAs han crecido enormemente en complejidad. Los modernos FPGA ahora pueden contener aproximadamente 330000 bloques y alrededor de 1100 entradas y salidas. Además de una gran número de bloques

más especializados que han ampliado considerablemente la capacidades de los FPGAs. Estos aumentos masivos en capacidades han sido acompañados de importantes cambios en la arquitectura.

2.1.1 ESTRUCTURA INTERNA DE UN FPGA

Un FPGA es un dispositivo lógico programable formado por conjunto de bloques lógicos programables o configurables, los mismos que se encuentran conectados o comunicados por interconexiones programables. Desde un punto de vista físico un FPGA es un chip en blanco en el cual no se encuentra configurado o programado ningún tipo de operación, pero posee los elementos necesarios para realizar cualquier tipo de operación sea esta secuencial o combinacional.

Un FPGA se programa o configura "*in the field*", esto significa que no vienen configurados por el fabricante, la programación es realizada por el desarrollador, por esta razón un FPGA no es un dispositivo dedicado que realiza una función específica, su propósito es de carácter general.

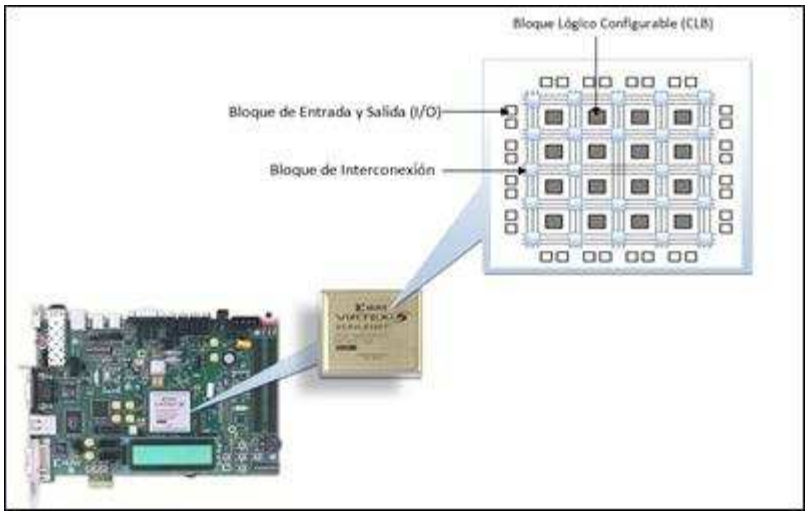


Figura A2-1. Estructura básica de un FPGA.

Un FPGA es un circuito integrado que básicamente se encuentra formado por bloques lógicos configurables CLB (*Configurable Logic Block*), bloques de entrada/salida y una matriz de interconexiones programables. En la figura A2-1, se muestra la estructura básica de un FPGA.

2.1.1.1 Bloques lógicos configurables

El bloque lógico configurable CLB es la unidad básica de un dispositivo FPGA, en este bloque se puede configurar cualquier tipo de operación. El CLB se encuentra formado por *Look-Up Tables* (LUTs*)*, dispositivos de almacenamiento como *flip flops* y *multiplexers* cuya función principal es la de enrutar los datos entre los componentes. Los *Look-Up Tables* (LUTs), vienen a constituir las tablas de verdad de los circuitos que se implementarán en el interior del CLB. En la figura A2-2, se muestra la estructura básica de un CLB.

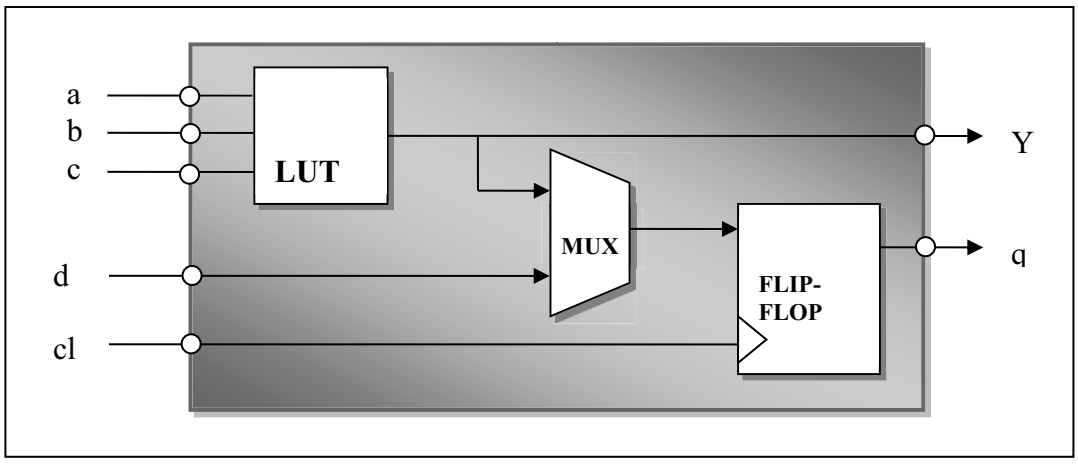


Figura A2-2. Estructura básica de un CLB.

Desde el punto de vista de sistemas digitales, un LUT constituye una memoria donde se almacena la tabla de verdad y a su vez sería un circuito combinacional que cumple con esa ley o función booleana. El multiplexor tiene la función de un conmutador de los datos de entrada y el *flip flop* constituye la parte secuencial, en donde todo el CLB se sincroniza con el reloj y a su vez con un sistema superior.

2.1.1.2 Bloques de entrada/salida

Los bloques de entrada/salida son los encargados de conectar al dispositivo FPGA con el mundo exterior, debiendo ser capaces de manejar corrientes eléctricas diferentes a las que se manejan en el interior del dispositivo FPGA, permitiendo de esta manera la entrada y salida de señales, razón por la cual los bloques de entrada/salida son bidireccionales. Cada uno de los bloques de

entrada/salida controlan a un pin del circuito integrado FPGA, haciendo posible la comunicación con el resto de dispositivos electrónicos presentes en la disposición circuital del kit de entrenamiento, como por ejemplo el FPGA puede estar conectado a un puerto usb, a un puerto fast Ethernet, entre otros.

2.1.1.3 Bloques de interconexión

La principal función de los bloques de interconexión es la de conectar las entradas y salidas de los CLB entre sí y con los bloques de entrada/salida del dispositivo FPGA. Al momento de configurar estos bloques de interconexión se debe tener en cuenta el tipo de tecnología, ya que esta puede variar según el fabricante.

2.2 TARJETA DE ENTRENAMIENTO VIRTEX 5 DE XILINX

La familia de FPGAs Virtex- 5 permite escoger cuatro nuevas plataformas:

- **LX**: familia que optimiza un alto rendimiento lógico.
- **LXT**: familia que optimiza un alto rendimiento lógico con baja potencia y conectividad serial.
- **SXT**: familia que optimizada para aplicaciones de procesamiento digital de señales, con aplicaciones de memoria intensiva con baja potencia y conectividad serial.
- **FXT**: familia que optimizada para procesamiento embebido y memoria intensiva con alta velocidad en conectividad serial.

Todas las plataformas de la familia Virtex-5 se caracterizan por presentar distinto número de ciertos recursos, por ejemplo pueden diferir en:

- El número de bloques de entrada/salida.
- El número de CLBs.
- El número de bloques de interconexión.
- El número de relojes y su distribución.
- Algunas familias poseen ciertos dispositivos para aplicaciones específicas.

		LX20T	LX30T	LX50T	LX85T	LX110T	LX155T	LX220T	LX330T
	Part Number	XC5VLX20T	XC5VLX30T	XC5VLX50T	XC5VLX85T	XC5VLX110T	XC5VLX155T	XC5VLX220T	XC5VLX330T
	EasyPath Cost Reduction Solutions [1]	-	-	-	XCE5VLX85T	XCE5VLX110T	XCE5VLX155T	XCE5VLX220T	XCE5VLX330T
Logic Resources	Slices [2]	3,120	4,800	7,200	12,960	17,280	24,320	34,560	51,840
	Logic Cells [3]	19,968	30,720	46,080	82,944	110,592	155,648	221,184	331,776
	CLB Flip-Flops	12,480	19,200	28,800	51,840	69,120	97,280	138,240	207,360
Memory Resources	Maximum Distributed RAM (Kbits)	210	320	480	840	1,120	1,640	2,280	3,420
	Block RAM/FIFO w/ECC (36 Kbits each)	26	36	60	108	148	212	212	324
	Total Block RAM (Kbits)	936	1,296	2,160	3,888	5,328	7,632	7,632	11,664
Clock Resources	Digital Clock Manager (DCM)	2	4	12	12	12	12	12	12
	Phase Locked Loop (PLL)	1	2	6	6	6	6	6	6
I/O Resources [4]	Maximum Single-Ended Pins	172	360	480	480	680	680	680	960
	Maximum Differential I/O Pairs	86	180	240	240	340	340	340	480
	I/O Standards	15, LVCMOS12, LVTTL, PCI33, PCI66, PCI-X, GTL, GTL+, HSTL I (1.2V, 1.5V, 1.8V), HSTL III (1.5V, 1.8V), HSTL IV (1							
Embedded Hard IP Resources [5]	DSP48E Slices	24	32	48	48	64	128	128	192
	PowerPC 440 Processor Blocks	-	-	-	-	-	-	-	-
	PCI Express Endpoint Blocks	1	1	1	1	1	1	1	1
	10/100/1000 Ethernet MAC Blocks	2	4	4	4	4	4	4	4
	RocketIO GTP Low-Power Transceivers	4	8	12	12	16	16	16	24
	RocketIO GTX High-Speed Transceivers	-	-	-	-	-	-	-	-
Speed Grades	Commercial	-1,-2	-1,-2,-3	-1,-2,-3	-1,-2,-3	-1,-2,-3	-1,-2,-3	-1,-2	-1,-2
	Industrial	-1,-2	-1,-2	-1,-2	-1,-2	-1,-2	-1,-2	-1,-2	-1
Configuration	Configuration Memory (Mbits)	6.3	9.4	14.1	23.4	31.2	43.1	55.2	82.7
Package [6]	Area	Available User I/Os [7]							
FFA Packages (FF): flip-chip fine-pitch BGA (1.0 mm ball									
FF324	19 x 19 mm								
FF676	27 x 27 mm								
FF1153	35 x 35 mm								
FF1760	42.5 x 42.5 mm								
FF323	19 x 19 mm	172 (4)	172 (4)						
FF665	27 x 27 mm		360 (8)	360 (8)					
FF1136	35 x 35 mm			480 (12)	480 (12)	640 (16)	640 (16)		
FF1738	42.5 x 42.5 mm					680 (16)	680 (16)	680 (16)	960 (24)

Figura A2-3. Familia Virtex-5 plataforma LXT.

Los elementos en común que poseen las plataformas de la familia Virtex-5 son:

- Bus PCI Express.
- Puerto Ethernet de tres velocidades 10/100/1000 Mbps.
- Transceivers para comunicación en serie de alta velocidad en el orden de los 5Gbps.
- Poseen tecnología CMOS de 65nm.

En la figura A2-3, se muestra las características de la familia Virtex-5 plataforma LXT que es la plataforma a la cual pertenece la tarjeta de entrenamiento XUPV5-LX110T.

En el presente proyecto se utilizará el kit de desarrollo Virtex 5, que corresponde al programa universitario de Xilinx XUP (*Xilinx University Program*), y cada uno de los dispositivos que contiene se especifican en la tabla A2-1. La tarjeta de entrenamiento XUPV5-LX110T, se indica en la figura A2-4.

Una tarjeta de entrenamiento XUPV5-LX110T.	
El software ISE Project Navigator Versión 13.1.	
Una tarjeta flash de 1GB.	
Un cable SATA.	
El cable XUP USB-JTAG para programar la tarjeta de entrenamiento.	
Un adaptador de DVI a VGA.	
Una fuente de alimentación 6A.	

Tabla A2-1. Kit de desarrollo Virtex 5.

- El chip FPGA Xilinx Virtex-5 XC5VLX110T.
- Dos plataformas flash PROMs Xilinx XCF32P de 32 MB cada una, para almacenar la configuración del dispositivo.
- Un controlador de configuración de flash Xilinx System ACE.
- Un módulo de memoria SODIMM de 256 MB.

- Una tarjeta de 32 bits ZBT sincrónica con SRAM e Intel P30 StrataFlash.
- Un puerto Ethernet de tres velocidades 10/100/1000.
- Un puerto USB.
- Un sistema generador de reloj programable.
- Un códec stereo AC97 línea in, línea out, auricular, micrófono y conectores para audio digital SPDIF.
- Un puerto RS-232.
- Un LCD de 2 filas de 16 caracteres.

Todos los componentes de la tarjeta de entrenamiento XUPV5-LX110T, se encuentran unidos al chip FPGA XC5VLX110T y estos se encuentran ubicados en lugares estratégicos.

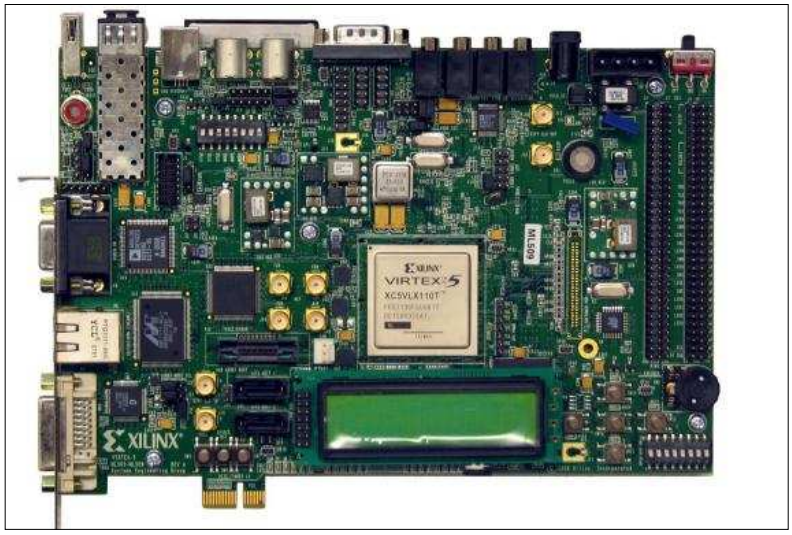


Figura A2-4. Tarjeta de entrenamiento XUPV5-LX110T.

2.2.1 ARQUITECTURA DE LOS FPGAS DE LA PLATAFORMA VIRTEX-5

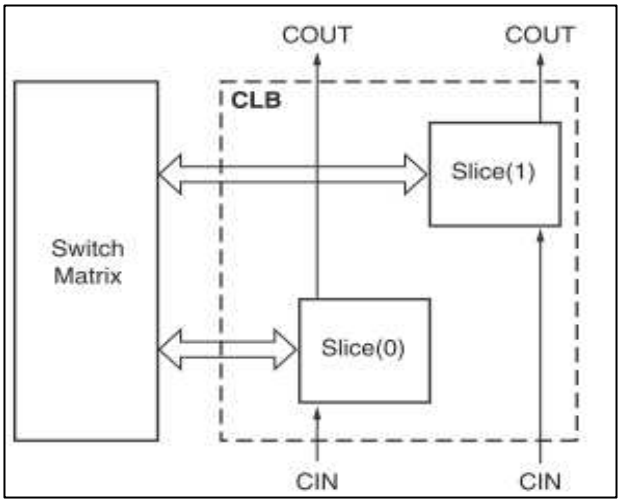
Los FPGAs que pertenecen a la plataforma VIRTEX-5 están compuestos de bloques programables que son:

- Bloques Lógicos Configurables (CLBs).
- Bloques de entrada/salida.

2.2.1.1 Bloques Lógicos Configurables

Los bloques lógicos configurables son los recursos lógicos principales que nos permiten implementar circuitos secuenciales y combinacionales. Cada CLB es conectado a una *swicth matrix* para acceder a aplicaciones generales de ruteo o interconexión con otros CLB. Cada CLB contiene un par de *slices*. Estos dos *slices* no tienen conexiones directas entre ellos; cada *slice* se encuentra organizado en forma de columna y posee un *carry chain* independiente. [2.1]

En cada CLB se etiquetan a los *slices*, el que se ubica en la parte superior derecha se le da el nombre de *slice*(1), y al *slice* que se ubica en la parte inferior izquierda recibe el nombre de slice(0). En la figura A2-5, se muestra la disposición de los sectores del CLB.

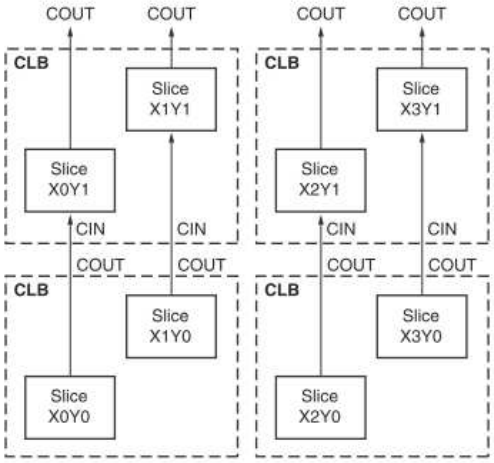


Fuente [A2.1]

Figura A2-5. Disposición de sectores en un CLB.

Las herramientas de Xilinx identifican a los *slices* de la siguiente manera:

- Una X seguida por un identificador nos dice a qué columna del CLB pertenece el *slice*.
- Una Y seguida por un identificador nos dice a qué fila del CLB pertenece el *slice*. En la figura A2-6, se muestra un ejemplo de la ubicación de los *slice* en el CLB.



Fuente [A2.1]

Figura A2-6. Ubicación de los *slice* en el CLB.

El número de recursos lógicos en un CLB se especifican en la tabla A2-2.

Slice	LUTs	Flip-flops	Arithmetic and carry chains	Distributed RAM	Shift register
2	8	8	2	256 bits	128 bits

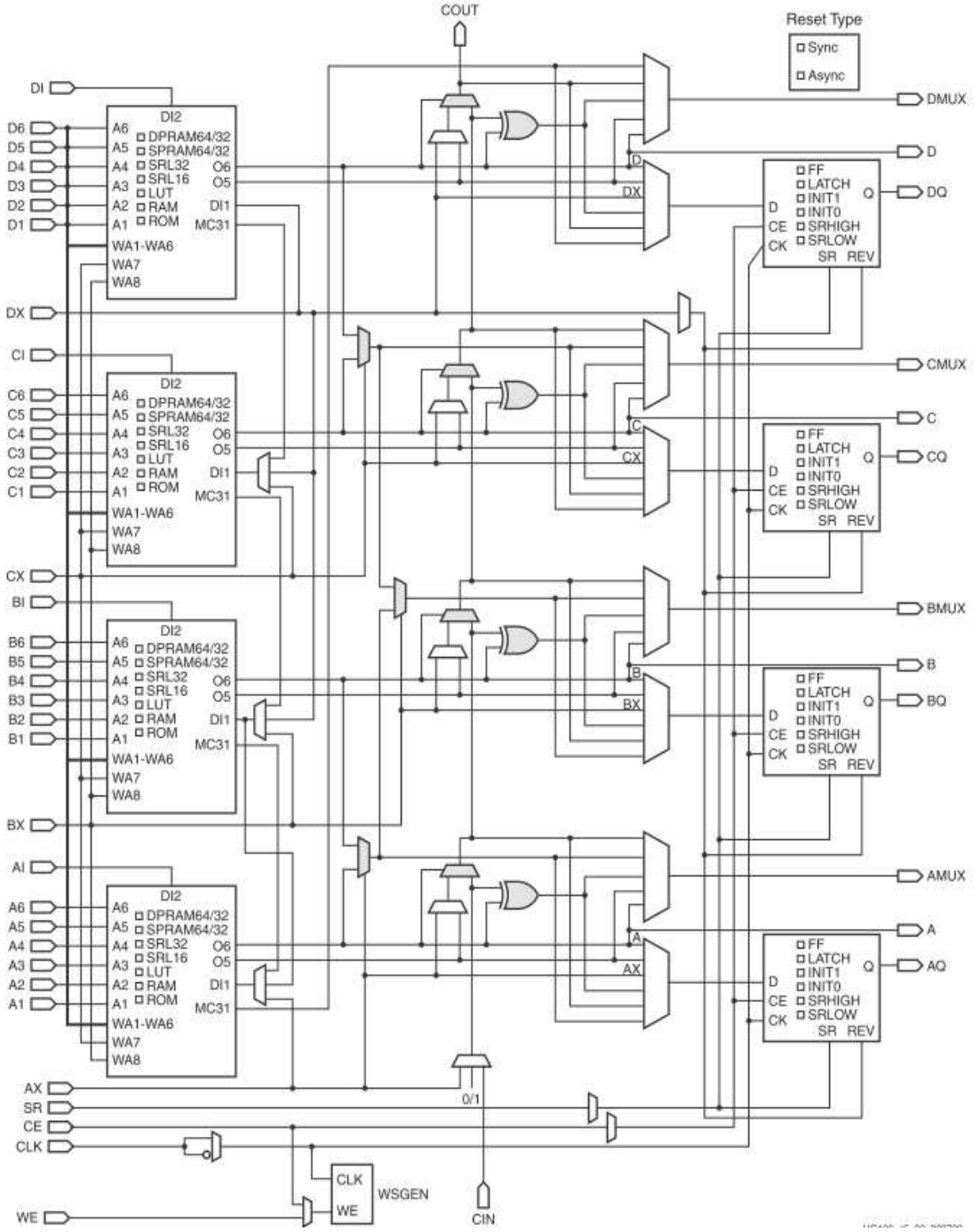
Tabla A2-2. Tabla de recursos lógicos en un CLB.

2.2.1.1.1. Slice

Un *slice* contiene cuatro generadores de funciones lógicas o también conocidas como *Look-Up Tables* (LUTs), cuatro elementos de almacenamiento, *multiplexers* y lógica de acarreo (*carry logic*). Estos elementos son utilizados por todos los *slices* para proveer operaciones lógicas, operaciones de aritmética y funciones de ROM.

En algunos casos los *slices* pueden soportar dos funciones adicionales: almacenamiento de datos utilizando una RAM distribuída y desplazamiento de datos con registros de 32 bits, estos *slices* se los conoce como: *SLICEL* y *SLICEM*.

En la figura A2-7, se muestra la estructura de *SLICEM*.



Fuente [A2.1]

Figura A2-7. Estructura de un *SLICEM.*

Los elementos que forman un *slice* son:

A. *Look_Up Tables.*

B. Elementos de almacenamiento.

C. RAM distribuída.
D. ROM.
E. Registros de desplazamiento.
F. Multiplexores.
G. Lógica de acarreo rápida.

A. Look-Up Tables

Los generadores de función *LUTs* son basados en una memoria RAM, tienen seis entradas independientes (por ejemplo si se tiene las entradas A, estas se numeran de A1 hasta A6) y dos salidas independientes (O5 y O6) para cada uno de los generadores de función en un *slice* denominados A, B, C y D. Los generadores de función pueden ser definir cualquier función booleana.

Un slice además de contener LUTs, contiene multiplexores:

- F5Mux: multiplexa las salidas de las LUTs dentro del SLICE.
- F6Mux: multpilexa las salidas de los F5Mux de un SLICE.
- F7Mux: multiplexa las salidas de los F6Mux de un CLB.
- F8Mux: multiplexa las salidas de los F7Mux de dos CLBs.

B. Elementos de almacenamiento

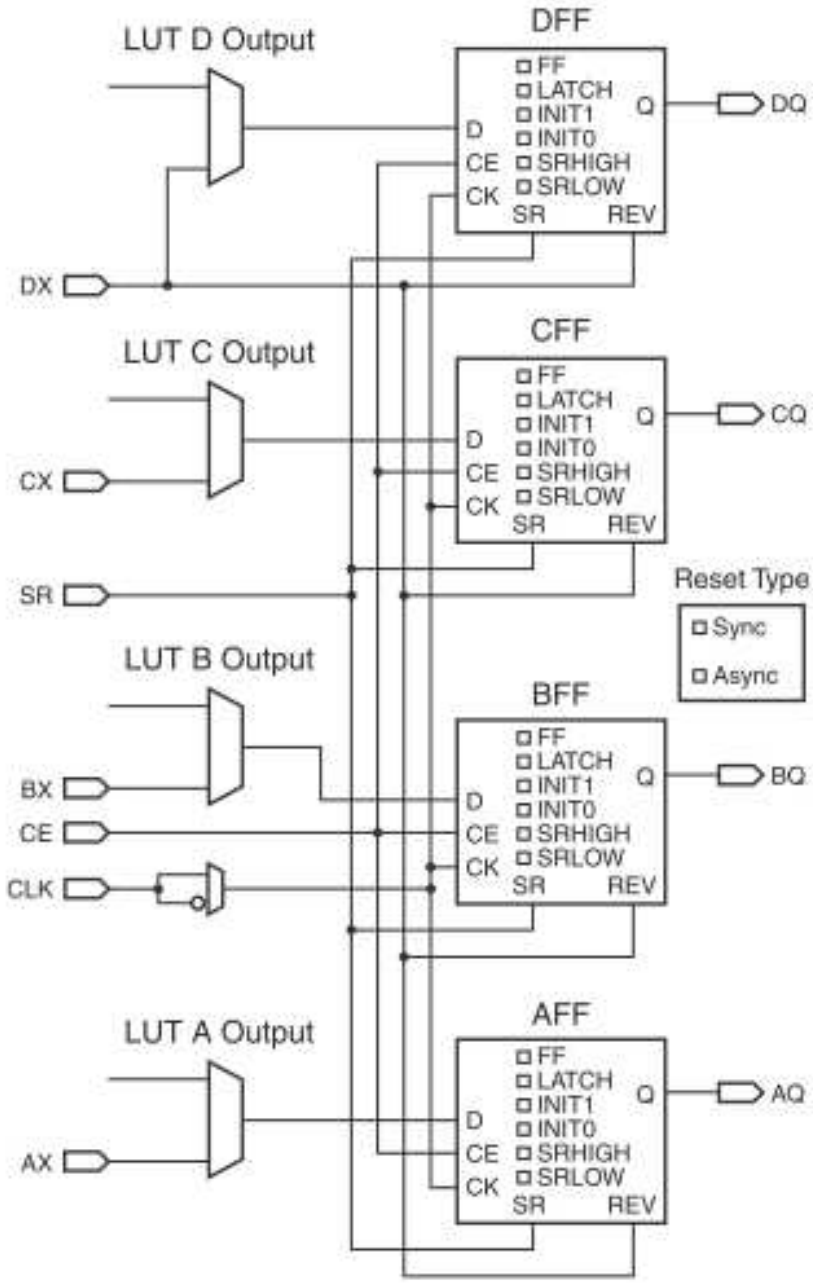
Como elementos de almacenamiento en un *slice* se tiene un arreglo de *flip-flops* tipo D o *latches* sensibles a nivel, las entradas para los *flip-flops* tipo D son manejadas por multiplexores.

En la figura A2-8, se muestra la configuración de los *flip-flops* en un *slice*.

C. RAM distribuída

La RAM distribuída solo puede estar habilitada en el *SLICEM*, en esencia múltiples LUTs pueden ser combinados de varias maneras para almacenar grandes cantidades de datos. Los módulos de RAM distribuída son sincrónicos

cuando se realiza la operación de escritura. Para que la operación de lectura sea sincrónica la RAM distribuída, debe ser implementada con elementos de almacenamiento o *flip-flops* que se encuentren ubicados en el mismo *slice*.



Fuente [A2.1]

Figura A2-8. Configuración de los flip-flops en un *slice*.

En la tabla A2-3, se muestra el número de *LUTs* (4 por slice) ocupados por cada configuración de RAM distribuída.

RAM	Número de LUTs
32 x 1S	1
32 x 1D	2
32 x 2Q	4
32 x 6 SDP	4
64 x 1S	1
64 x 1D	2
64 x 1Q	4
64 x 3SDP	4
128 x 1S	2
128 x 1D	4
256 x 1S	4

Tabla A2-3. Número de *LUTs* por *RAM* distribuída.

S: puerto simple Q: puerto cuádruple
SDP: puerto dual simple D: puerto dual

D. ROM

Cada generador de función en *SLICEM* o en *SLICEL* puede implementar una memoria ROM de 64 x 1 bit. De este esquema resulta tres posibles configuraciones de ROM: ROM 64 x 1, ROM 128 x 1 y ROM 256 x 1

ROM	Número de LUTs
64 x 1	1
128 x 1	2
256 x 1	4

Tabla A2-4. Número de *LUTs* por *ROM* distribuída.

La configuración de ROM y el número de LUTs se presentan en la tabla A2-4.

E. Registros de desplazamiento

Los registros de desplazamiento únicamente se encuentran habilitados en el *SLICEM*. Un generador de función de un *SLICEM* también puede ser configurado como un registro de desplazamiento de 32 bits, sin usar los *flip-flops* presentes en ese *slice*. Ciertas aplicaciones requieren cierto retardo o latencia por lo que hacen uso de estos registros de desplazamiento para realizar diseños eficientes. Los registros de desplazamiento son muy usados para implementar una memoria FIFO.

F. Multiplexores

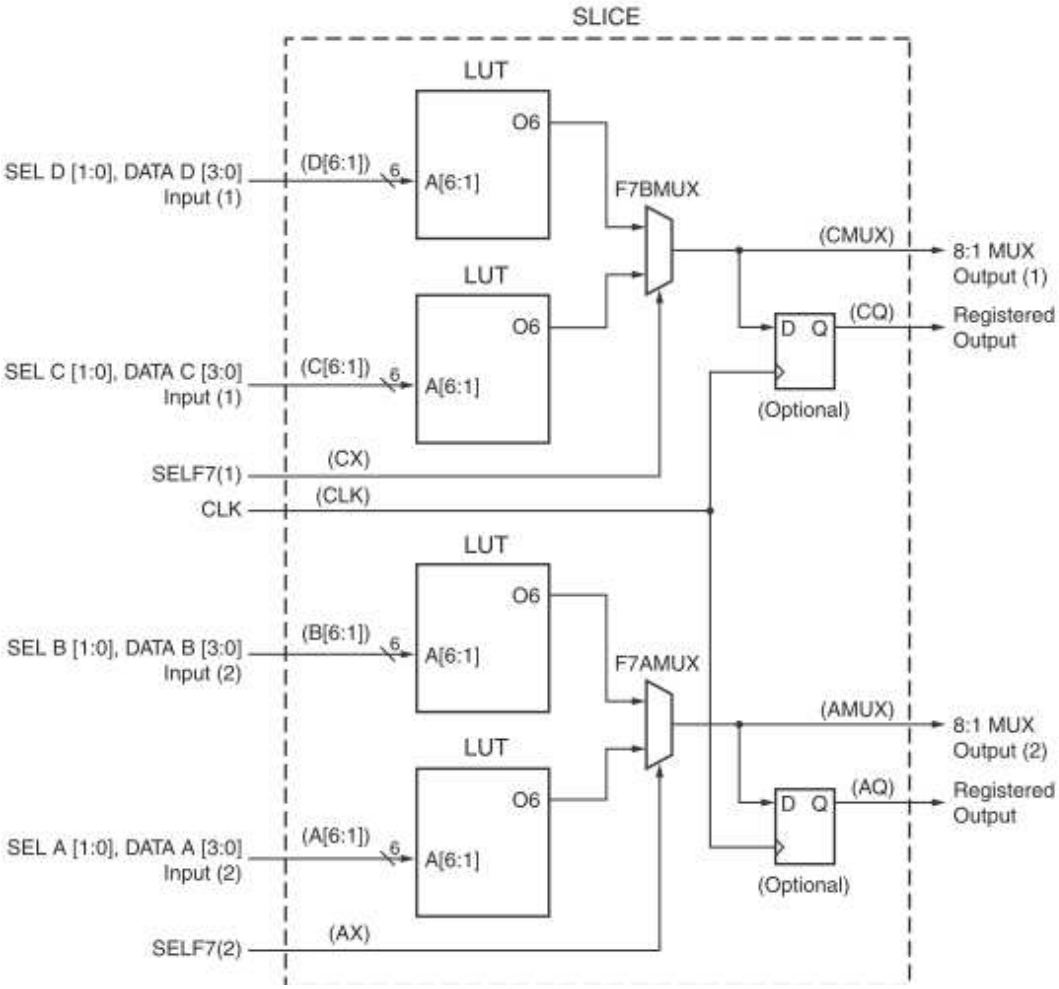
Los generadores de funciones y los multiplexores presentes en el *FPGA* Virtex-5, pueden implementar los siguientes multiplexores:

- Multiplexores 4:1 utilizando únicamente un LUT.
- Multiplexores 8:1 utilizando dos LUTs.
- Multiplexores 16:1 utilizando cuatro LUTs.

Para formar los multiplexores anteriormente mencionados se deben combinar los LUTs por medio de los multiplexores F7AMUX, F7BMUX y F8MUX que se encuentran en el FPGA Virtex-5. En la figura A2-9, se muestra la implementación de un multiplexor 8:1 construido mediante la combinación de LUTs. Como cada *slice* tiene dos multiplexores F7AMUX y F7BMUX, estos sirven para unir a los LUTs y formar una función que tiene hasta 13 entradas, lo que es válido porque solo se necesitan ocho entradas, como un *slice* posee cuatro LUTs se realiza el mismo proceso de unión de las otras dos LUTs, de esta manera al final se obtienen dos multiplexores de 8:1.

G. Lógica de acarreo rápida

La lógica de acarreo rápida permite realizar operaciones aritméticas como la suma y resta de una manera más rápida en el interior de un *slice*. Cada *CLB* del FPGA Virtex-5 tiene dos *carry chains* separados uno del otro.



Fuente [A2.1]

Figura A2-9. Configuración de los *flip-flops* en un *slice*.

2.2.1.2 Bloques RAM

En adición con la memoria RAM distribuída, los dispositivos Virtex-5 presentan un gran número de bloques RAM de 36Kb. Cada bloque RAM contiene dos bloques RAM de 18Kb cada uno, los mismos que son controlados independientes uno del otro.

Los bloques RAM están ubicados en columnas y el número total de bloques *RAM* depende del tamaño del dispositivo *Virtex-5*, además los bloques de 36Kb se encuentran ubicados en cascada para que cuando estos sean habilitados la penalización en tiempo sea mínima.

2.2.1.2.1 Mejoras en los bloque RAM en el FPGA virtex-5

Incremento de la memoria y en la capacidad de almacenamiento por bloque. Cada bloque RAM puede almacenar hasta 36Kb. Soporta dos bloques independientes de 18Kb o un solo bloque de 36Kb.

El FPGA *Virtex-5* posee un atributo para configurar a un bloque RAM como una memoria FIFO sincrónica, cada bloque RAM puede ser configurado como una memoria FIFO de 18Kb o 36Kb.

Los bloques RAM de 36 Kb permiten ser configurados en cascada para obtener mayor espacio de almacenamiento o anchos de palabra mayores, con tiempos mínimos de retardo.

2.2.1.2.2 Memoria RAM sincrónica de doble y simple puerto

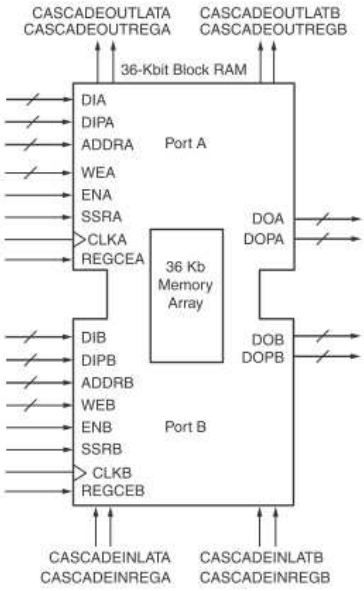
Las memorias RAM de doble puerto de 36Kb, consisten de un área de almacenamiento de 36Kb de dos puertos de acceso, A y B, completamente independientes. De manera similar un bloque *RAM* de 18Kb de doble puerto consiste de un área de almacenamiento de 18Kb y dos puertos de acceso, A y B, completamente independientes. La estructura es altamente simétrica y ambos puertos son intercambiables. En la figura A2-10, se muestra una memoria RAM de puerto dual.

Los datos pueden ser escritos o leídos desde ambos puertos. El acceso de escritura es síncrono y cada puerto tiene su propio bus de datos de entrada y salida, bus de direcciones, señal de reloj, habilitación de bloqueo y habilitación de escritura.

2.2.1.2.3 Operaciones de la memoria RAM

- Lectura. - La dirección de lectura es registrada en el puerto de lectura, y el dato almacenado es cargado en la salida.

- Escritura.- La dirección de escritura es registrada en el puerto de escritura y el dato de entrada es almacenado en la memoria.



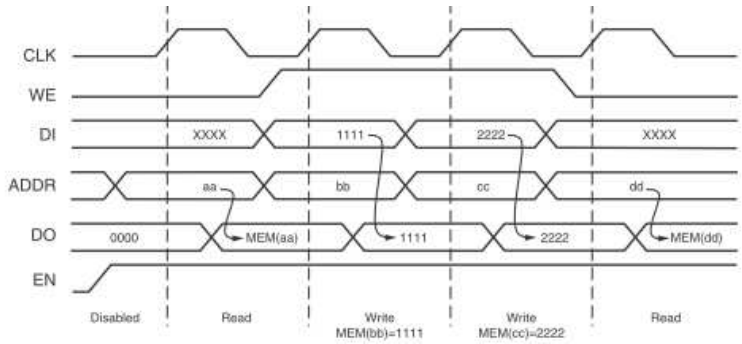
Fuente [A2.1]

Figura A2-10. Memoria *RAM* de puerto dual.

2.2.1.2.4 Modos de escritura en la memoria RAM

- Write_first

  Es el modo por *default*, el dato de entrada es escrito simultáneamente en la memoria y almacenado en el dato de salida. En la figura A2-11, se muestra las formas de onda correspondientes a este modo de escritura.

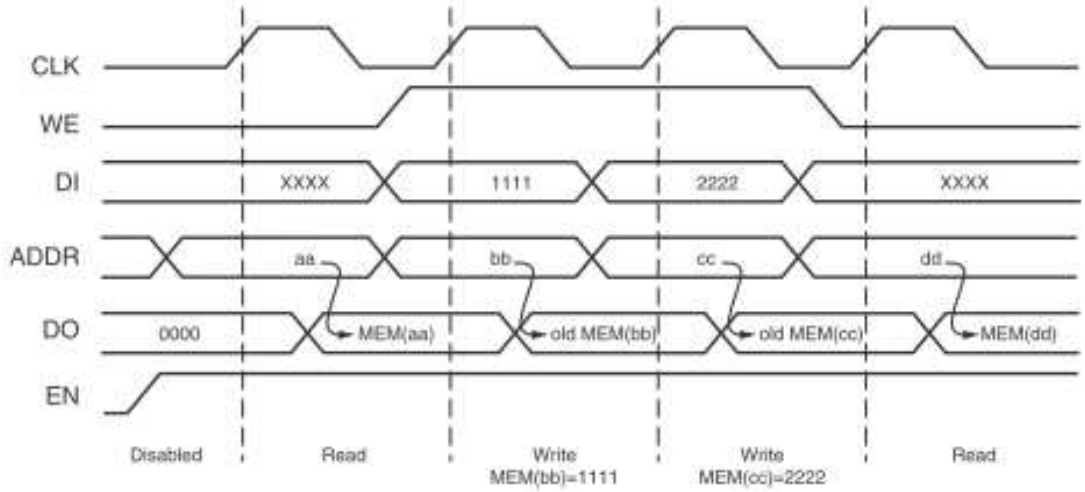


Fuente [A2.1]

Figura A2- 11. Formas de onda para el modo *WRITE_FIRST*.

- Read_First

  En este modo los datos deben ser previamente almacenados para que luego estos puedan ser leídos. En la figura A2-12, se muestra las formas de onda del modo *READ_FIRST*.



Fuente [A2.1]

Figura A2-12. Formas de onda para el modo READ_FIRST.

2.2.1.2.5 Memorias FIFO

Muchos diseños de FPGA, usan los bloques RAM para implementar memorias FIFO. En la arquitectura Virtex-5, lógica dedicada en los bloques RAM permiten a los usuarios implementar de manera sencilla una memoria FIFO sincrónica o memorias FIFO asincrónicas *multirate*. Esto elimina la necesidad de utilizar un CLB adicional para realizar un contador, comparador o un generador de banderas, por lo tanto solo se usa un bloque RAM por memoria FIFO.

Una FIFO puede ser configurada como una memoria de 18Kb o 36Kb. Cuando una memoria FIFO funciona en modo de 18Kb soporta las siguientes configuraciones: 4k x 4, 2k x 9, 1k x 18 y 512 x 36.

Las configuraciones soportadas para un bloque FIFO de 36Kb son 8k x 4, 4k x 9, 2k x 18, 1k x 36 y 512 x 72.

El bloque RAM puede ser configurado como una memoria FIFO con un reloj común para realizar las operaciones de escritura y lectura, o también con un reloj independiente. El puerto A del bloque RAM es usado como un puerto de lectura de la memoria FIFO y el puerto B del bloque RAM es usado como el puerto de escritura de la memoria FIFO. El dato es leído de la memoria FIFO con cada flanco del reloj de lectura y un dato es escrito en la memoria FIFO con cada flanco del reloj de escritura.

- **Fifo Multirate**

  Este tipo de memoria ofrece una interfaz de usuario muy sencilla. Los relojes de lectura y escritura pueden ser de idénticas o diferentes frecuencias y estás pueden ser seleccionadas hasta su límite máximo de frecuencia.

  Este diseño evita cualquier ambigüedad o problemas de estabilidad aun cuando las frecuencias sean completamente diferentes.

2.2.1.3 Bloques de entrada/salida

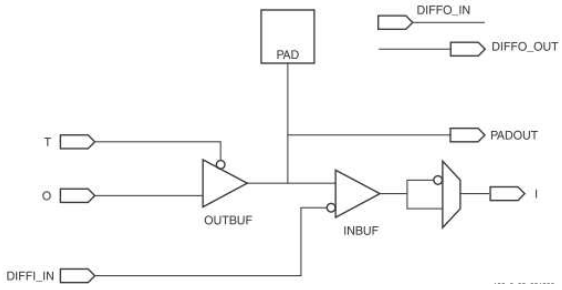
El dispositivo *FPGA* Virtex-5, tiene asociado a cada pin de entrada/salida drivers configurables conocidos como *Select* IO, los mismos que soportan una gran variedad de estándares lógicos, a estos se los denomina bloques de entrada/salida (IOB). Estos bloques permiten configurar, la corriente de salida, la impedancia (DCI) y la velocidad de variación de las señales.

Cada IOB es bidireccional, contiene los drivers para que estos bloques funcionen en 3-state.

Los IOBs se asocian en parejas, cada IOB posee dos bloques de lógica de entrada y salida denominados ILOGIC y OLOGIC.

Los IOBs también pueden ser configurados como bloques que nos permiten gestionar comunicaciones en las cuales es necesario la conversión serie/paralelo

y viceversa, a estos bloques se los denomina *ISERDES* y k. En la figura A2-13, se muestra la estructura básica de un IOB.



Fuente [A2.1]

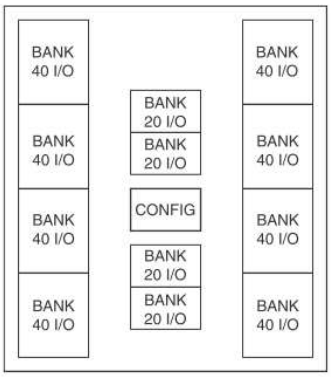
Figura A2-13. Estructura básica de un IOB.

2.2.1.3.1 Organización en bancos de los pines de entrada/salida

Dependiendo del dispositivo Virtex-5 el número de IOBs de entrada/salida varía entre 220 y 1200.

Un banco consiste de 40 IOBs cada uno, con excepción de algunos bancos centrales que son de 20 IOBs. Una tensión de alimentación VCCO es la que alimenta a cada uno de los bancos, esta tensión es común a cada uno de los pines, así como algunos buffers de entrada.

Cada bloque IOB es muy versátil y puede ser configurado con diferentes valores de tensión (1.2 a 3.3 V). Con excepción de los pines de alimentación y de los que son dedicados a señales de reloj. La figura A2-14, indica la distribución de los bancos.



Fuente [A2.1]

Figura A2-14. Distribución de los bancos.

2.2.1.3.2 Control digital de impedancia

El DCI ajusta la impedancia de salida o la terminación de entrada para que coincida, con precisión, con la impedancia característica de la línea de transmisión. El ajuste continuo de la impedancia de los IOBs se lo realiza para compensar variaciones de temperatura y fluctuaciones en el nivel de voltaje.

El DCI controla el driver de impedancia, para que coincida con dos resistencias de referencia o en caso opcional coincida con la mitad del valor de estas dos resistencias de referencia.

2.2.1.3.3 Buffers de entrada y salida

A veces las señales de entrada y salida se deben acondicionar por lo que es necesario el uso de buffers o espacios de almacenamiento. Estos buffers pueden ser creados gracias a unas primitivas que se encuentran en las librerías proporcionadas por Xilinx.

Xilinx presenta las siguientes primitivas para el FPGA Virtex-5:

- IBUF: buffer de entrada.
- IBUFG: buffer destinado a señales de reloj de entrada.
- OBUF: buffer de salida.
- OBUFT: buffer de salida con señal control de triestado.
- IOBUF: buffer de entrada y uno de salida con control de triestado que posibilita el tratamiento de señales diferenciales.

2.2.1.4 Recursos de reloj

En el dispositivo FPGA Virtex-5 los recursos de reloj se encuentran divididos en regiones. El número de regiones varía según el tamaño del dispositivo, en los dispositivos más pequeños el número de regiones es de 8 mientras que en dispositivos de mayor tamaño se puede encontrar 24 regiones.

2.2.1.4 .1 Reloj global

Cada FPGA Virtex-5 tiene 32 líneas de reloj global, que pueden dar el servicio de reloj a todos los recursos secuenciales en el FPGA (CLB, bloques *RAM* y los IOBs). Las líneas de reloj global están controladas por un buffer de reloj global, este funciona como un circuito que habilita o deshabilita las líneas de reloj.

Los relojes globales tienen asociada una red dedicada de interconexión diseñada específicamente para hacer llegar todas las entradas de reloj a los diferentes recursos de un FPGA. Estas redes han sido diseñadas para conseguir un bajo *skew* y una distorsión baja del ciclo de trabajo, un bajo consumo, y un *jitter* de tolerancia mínima.

Un buffer de reloj global es a menudo controlado por un administrador de reloj (CMT) para eliminar los retardos si se distribuye la señal de reloj, o para ajustar los retardos a otro reloj.

2.2.1.4 .2 Regiones de reloj

Los dispositivos Virtex-5 disponen una distribución de reloj dada por el uso de las regiones de reloj. Cada región de reloj puede tener hasta diez dominios de reloj global. Estos diez dominios de reloj global pueden ser controlados por cualquier combinación de los 32 buffers de reloj global. La dimensión de una región de reloj es una mezcla de 20 CLBs, 40 IOBs. La ventaja de un dispositivo Virtex-5 es que pueden soportar un amplio número de dominios de reloj que los dispositivos FPGAs anteriores. En la figura A2-15, se muestra el número de regiones de reloj en un dispositivo Virtex-5.

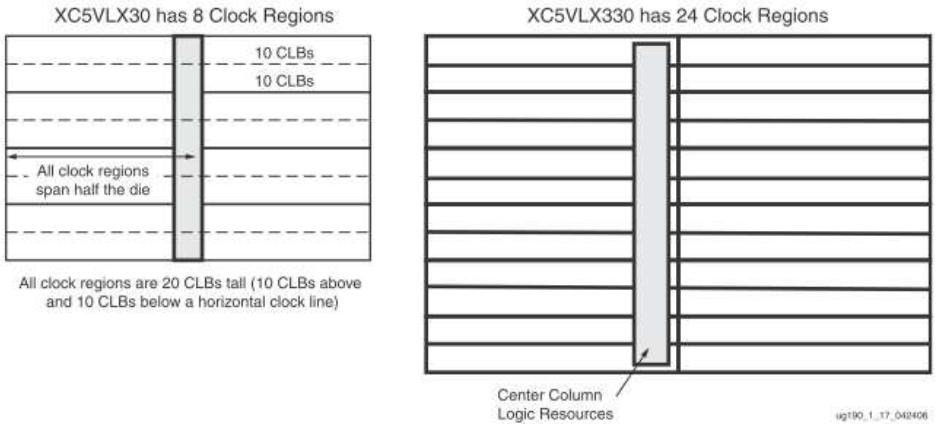
2.2.1.5 Gestión de relojes

El gestionador de relojes CMT, en los dispositivos Virtex-5 provee alta flexibilidad y un alto rendimiento de los relojes. Cada CMT contiene dos DCM y un PLL, esto se muestra en la figura A2-16.

2.2.1.5.1 DCM

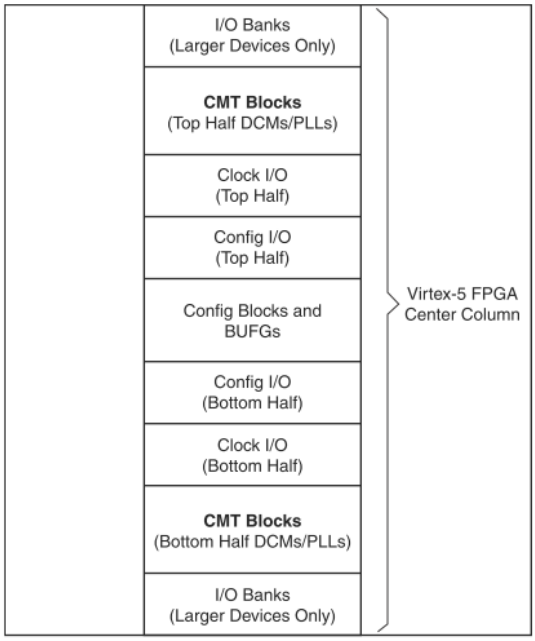
El administrador digital de relojes (DCM) en los dispositivos Virtex-5 presenta:

A. Reloj de Alineación.
B. Síntesis de Frecuencia
C. Desplazamiento de Fase



Fuente [A2.1]

Figura A2-15. Regiones de reloj.



Fuente [A2.1]

Figura A2-16. Ubicación de un CMT.

A. Reloj de Alineación

El DCM contiene un lazo cerrado de retardo (DLL) para eliminar los retardos de la distribución de reloj.

El DLL contiene elementos de retardo, unos pequeños *buffers* y lógica de control.

La lógica de control contiene un detector de fase y un selector de línea de retardo. El detector de fase compara la señal entrante de reloj (CLKIN) con una señal de entrada de retroalimentación (CLKFB) y dirige el retraso de la línea del selector, luego adiciona retardo a la salida del DCM hasta que la señal CLKIN coincida con la señal CLKFB.

B. Síntesis de Frecuencia

El usuario puede especificar cualquier número entero, un multiplicador M y un divisor D dentro del rango especificado en el *data sheet*. Una calculadora interna determina la selección apropiada para hacer que el flanco de salida coincida con el reloj de entrada cuando sea matemáticamente posible.

C. Desplazamiento de Fase

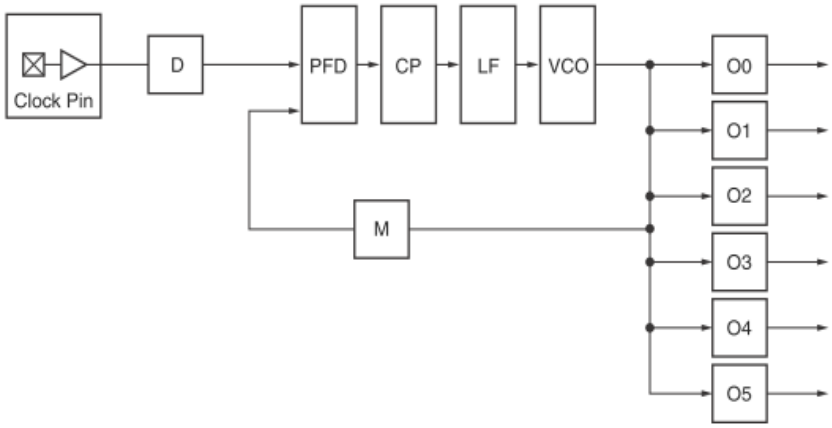
EL DCM permite desplazamiento de frecuencia fino y grueso. El desplazamiento grueso de frecuencia usa las fases de 90° 180 ° y 270 ° de CLK0 para construir CLK90 CLK180 y CLK270.

El desplazamiento fino permite a todas las salidas del reloj DCM ser desplazadas en fase con respecto al CLKIN mientras mantiene la relación entre la fase de salida gruesa.

2.2.1.5.2 PLL (phase locked loop)

El propósito principal de un PLL es servir como un sintetizador de frecuencia para un amplio rango de frecuencias y para servir como un filtro de *jitter* para relojes

internos o externos. En la figura A2-17, se muestra un diagrama de bloques de un PLL.



Fuente [A2.1]

Figura A2-17. Diagrama de bloques de un PLL.

El corazón de un PLL es un oscilador controlado por voltaje o VCO (*Voltage Controllerd Oscillator*) con un rango de frecuencias de 400MHz a 1100MHz. A partir de tres divisores de frecuencia programables (D, M y O) se adapta el VCO a la aplicación requerida.

El VCO tiene ocho salidas igualmente espaciadas (0 º, 45 º, 90 º, 135 º, 225 º, 270 º, y 315 º) cada una de las cuales puede ser seleccionada para atacar uno de los seis divisores de salida. Cada divisor puede ser configurado en un rango entero de 1 a 127.

El PLL tiene dos opciones de filtrado de *jitter* de la señal de entrada: modo de ancho de banda ancha o estrecha. El modo de ancho de banda estrecha es el que más atenúa el *jitter*, pero no debe ser usado con relojes que cambian de frecuencia rápidamente. En cambio, el modo de banda ancha produce menor atenuación del *jitter* y está pensado para relojes de entrada que pueden variar su frecuencia rápidamente.

REFERENCIAS

[A2-1] Virtex-5 User Guide UG190(v 5.3), Xilinx, 2010.

ANEXO 3

INTRODUCCIÓN A VHDL

3. VHDL

HDL por sus siglas en inglés *hardware description language*, es un lenguaje utilizado para describir el comportamiento de un circuito electrónico o un sistema. Los lenguajes HDL más utilizados son Verilog y VHDL, este proyecto de titulación se ha desarrollado en VHDL.

VHDL por el acrónimo *VHSIC Hardware Description Language*, VHSIC a su vez por *Very High Speed Integrated Circuits*, fue desarrollo en 1980 con el financiamiento del Departamento de Defensa de los Estados Unidos de Norteamérica, y años más tarde fue estandarizado por la IEEE (*Institute of Electrical and Electronics Engineers*), a través de la IEEE 1076.

VHDL está diseñado para sintetizar y simular circuitos, si bien todos los circuitos pueden ser representados por una simulación, no todos pueden ser sintetizados, esto quiere decir que no todo el código que se puede implementar con la sintaxis correcta puede ser sintetizado a nivel de hardware, razón por la cual se debe tener mucho cuidado al momento de implementar un diseño en un dispositivo FPGA.

3.1. ESTRUCTURA DE DISEÑO VHDL

Un diseño en VHDL consta de tres secciones:

- Library.
- Entity.
- Architecture

Library: Contiene las librerías que se usan en un diseño. Algunos ejemplos de librerías son: *ieee, std, work* entre otras. La sintaxis para escribir una librería es la siguiente:

```
LIBRARY  nombre_librería;
USE  nombre librería.nombre paquete.parte paquete;
```

La declaración de cada una de estas es la siguiente:

```
LIBRARY  IEEE;
USE IEEE.STD_LOGIC_1164.all;
USE IEEE.NUMERIC_STD.all;

LIBRARY  std;
USE  std.standard.all;

LIBRARY  work;
USE  work.all;
```

Dentro de la librería IEEE existen varios paquetes que se describen en la tabla A3-1.

std_logic_1164	Especifica el tipo de dato STD_LOGIC
std_logic_arith	Especifica SIGNED y UNSIGNED relacionados a operaciones aritméticas.
std_logic_signed	Especifica el tipo de dato STD_LOGIC_VECTOR de tipo SIGNED
std_logic_unsigned	Especifica el tipo de dato STD_LOGIC_VECTOR de tipo UNSIGNED

Tabla A3-1. Librería IEEE.

El paquete std_logic_1164, debe ser colocado cuando se necesite hacer uso del tipo de dato std_logic, este tipo de dato se utiliza cuando se representa una señal o variable, cuando varían de uno y cero.

El paquete std_logic_arith, especifica el tipo de dato *signed* y unsigned, resulta útil cuando se requiere realizar operaciones de tipo binario entre vectores de bits.

El paquete std_logic_signed, especifica el tipo de dato std_logic_vector, resulta útil cuando se declara un vector de bits con signo.

El paquete std_logic_unsigned, especifica el tipo de dato std_logic_vector, resulta útil cuando se declara un vector de bits sin signo.

Entity: En la entidad se declaran los puertos in/out/inout de un diseño. La sintaxis para escribir una entidad es la siguiente:

```
ENTITY  nombre_entidad  IS

PORT  (

nombre_puerto :  modo_señal  tipo_señal;
nombre_puerto :  modo_señal  tipo_señal;

...);

END  nombre_entidad;
```

Ejemplo:

Se desea implementar la compuerta lógica Xor, la entidad sería la siguiente:

```
entity Xor_compuerta is

    Port (
            operando1 : in  STD_LOGIC;

            operando2 : in  STD_LOGIC;

           salida : out  STD_LOGIC);

end Xor_compuerta;
```

Una representación gráfica de la entidad Xor_compuerta es la que indica la figura A3-1.

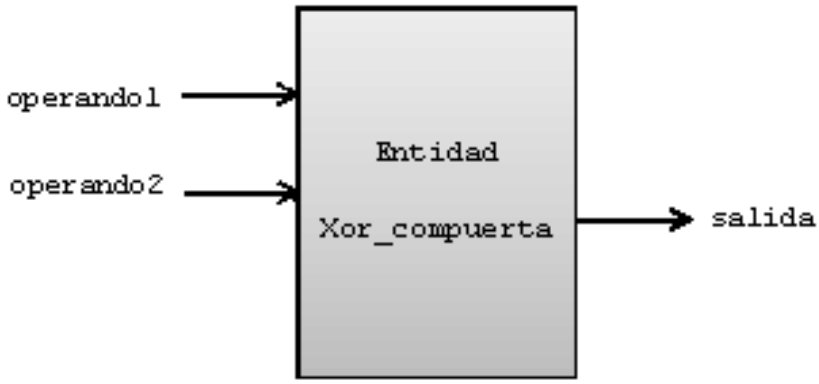


Figura A3-1. Representación gráfica de la entidad Xor_compuerta.

Architecture: Dentro de la arquitectura se describe el funcionamiento interno de un circuito. La sintaxis es la siguiente:

```
ARCHITECTURE  nombre_arquitectura  OF nombre_endidad  IS
[inicialización_señales]
BEGIN
(código)
END  nombre_arquitectura
```

La declaración de la arquitectura para la entidad Xor_compuerta es la siguiente:

```
architecture arch of Xor_compuerta is
begin
salida <= operando1 xor operando2;
end arch;
```

En VHDL no se hace distinción entre letras mayúsculas y minúsculas. El diseño completo usado para la explicación de librería, entidad, y arquitectura, basado en la realización de la compuerta xor es el siguiente:

```
-----------------------------------
-- Entidad Xor_compuerta
-----------------------------------

---------------------------------------
-- LIBRERIA
---------------------------------------

LIBRARY  IEEE;
USE IEEE.STD_LOGIC_1164.all;
USE IEEE.NUMERIC_STD.all;
USE IEEE.STD_LOGIC_UNSIGNED.all;

---------------------------------------
-- ENTIDAD
---------------------------------------
entity Xor_compuerta is
    Port ( operando1 : in  STD_LOGIC;
           operando2 : in  STD_LOGIC;
           salida : out  STD_LOGIC);

end Xor_compuerta;

---------------------------------------
-- ARQUITECTURA
---------------------------------------

architecture arch of Xor_compuerta is

begin

salida <= operando1 xor operando2;

end arch;
```

3.2 TIPOS DE DATOS

En VHDL existen tipos de datos predefinidos, los mismos que se describen a continuación:

- BIT → {0,1}
- STD_LOGIC → 8 valores lógicos definidos en la librería IEEE 1164. Diseñado para modelar señales eléctricas.

  'X' Forcing Unknown (synthesizable unknown)
  '0' Forcing Low (synthesizable logic '1')
  '1' Forcing High (synthesizable logic '0')
  'Z' High impedance (synthesizable tri-state bu¤er)
  'W' Weak unknown
  'L' Weak low

‘H’ Weak high
‘–’ Don’t care

[A3-4]

- STD_ULOGIC → 9 valores lógicos definidos en la librería IEEE 1164. (‘U’, ‘X’ , ‘0’ ‘1’, ‘Z’, ‘W’, ‘L’, ‘H’, ‘-’)
- BOLEAN → { True, False}
- INTEGER → 32 bits { desde - 2,147,483,647 hasta +2,147,483,647}
- NATURAL → { desde 0 hasta 2,147,483,647}
- REAL → { desde -1.0E38 hasta + 1.0E38} →No sintetizable
- SIGNED y UNSIGNED → tipos de datos definidos en std_logic_arith de la librería IEEE. Tienen la apariencia de STD_LOGIC_VECTOR, pero aceptan operaciones aritméticas.

3.3 OPERADORES

El lenguaje VHDL es muy versátil, ya que posee una serie de operadores, entre ellos se puede citar operadores de tipo aritmético para realizar operaciones como: suma, resta, multiplicación, exponenciación y división. Para utilizar estos operadores se debe tener en cuenta que sólo aceptan operadores de tipo entero.Existen también operadores de concatenación, operadores para realizar operaciones de comparación entre dos operandos, para verificar si un operando es igual a otro, o si no lo es, para comprobar si un operando es mayor o menor, etc.

Otro tipo de operadores son los lógicos, como la operación not, xor, and, or, pero estos sólo se aplican a tipos de datos boolean, std_logic y std_logic_vector.

Se debe tener precaución al momento de sintetizar un circuito ya que no todos los operadores son sintetizables como por ejemplo el operador de la división es un operador no sintetizable, ya que se gastaría muchos recursos de hardware para sintetizar dicha operación, por lo tanto al momento de realizar la síntesis se va a producir un error. La descripción de los operadores y los tipos de datos aplicables se resumen en la tabla A3-2.

Operator	Description	Data type of operand a	Data type of operand b	Data type of result
a **b abs a not a	exponentiation absolute value negation	integer integer boolean, bit, bit_vector	intenger	Integer integer boolean, bit, bit_vector
a * b a / b a mod b a rem b	multiplication division modulo remainder	integer	Integer	Integer
+ a - b	Identity negation	integer		Integer
a + b a – b a & b	Addition subtraction concatenation	Integer 1-D array, element	Integer 1-D array, element	Integer 1-D array
a sll b a sri b a sla b a sri b a rol b a ror b	shift-left logical shift-right logical shift-left aritmetic shift-right aritmetic rotate left rotate right	bit_vector	Integer	bit_vector
a = b a /=b a < b a <= b a > b a >= b	equal to not equal to less than less than or equal to greater than grater than or equal to	any scalar or 1-D array	same as a same as a	Boolean boolean
a and b a or b a xor b a nand b a nor b xnor b	And or xor nand nor xnor	boolean, bit bit_vector	same as a	same as a

Tabla A3-2. Operadores VHDL. [A3-3]

3.4 FORMATO DE UN PROGRAMA VHDL

Se utilizará el ejemplo de un programa que sintetiza una compuerta xor, para explicar el formato de un programa en VHDL:

```
1----------------------------------------------------------------------
-
2-- Company:
3-- Engineer:
4--
5-- Create Date:    17:45:50 05/05/2012
6-- Design Name:
7-- Module Name:    xor_entity - arch
8-- Project Name:
9-- Target Devices:
10-- Tool versions:
11-- Description:
12--
13-- Dependencies:
14--
15-- Revision:
16-- Revision 0.01 - File Created
17-- Additional Comments:
18--
19----------------------------------------------------------------------
-
20library IEEE;
21use IEEE.STD_LOGIC_1164.ALL;
22
23-- Uncomment the following library declaration if using
24-- arithmetic functions with Signed or Unsigned values
25--use IEEE.NUMERIC_STD.ALL;
26
27-- Uncomment the following library declaration if instantiating
28-- any Xilinx primitives in this code.
29--library UNISIM;
30--use UNISIM.VComponents.all;
31
32entity xor_entity is
33    Port ( a : in  STD_LOGIC;
34           b : in  STD_LOGIC;
35           salida : out  STD_LOGIC);
36end xor_entity;
37
38architecture arch of xor_entity is
39
40begin
41    salida <= a xor b;
42
43end arch;
```

La región comprendida entre las líneas 1 a la 19, se conoce como la cabecera, en esta zona se tiene información general del diseño.

La región comprendida entre las líneas 20 a la 43, es la región del programa.
En las líneas 20 y 21 se colocan las librerías necesarias para poder utilizar los tipos de datos.

En las líneas 32 a la 36 se tiene la región donde se declara la entidad y sus atributos, en el caso de la compuerta xor se tiene los puertos de entrada y el puerto de salida, como se indica a continuación:

```
32entity xor_entity is
33    Port ( a : in  STD_LOGIC;
34           b : in  STD_LOGIC;
35           salida : out  STD_LOGIC);
36end xor_entity;
```

De la línea 38 a la línea 43, se tiene la región que describe el comportamiento del circuito, para nuestro ejemplo se tiene la operación xor que recibe como parámetros de entrada al puerto de entrada a y al puerto b, el resultado se presenta en el puerto de salida, como se indica a continuación:

```
38architecture arch of xor_entity is
39
40begin
41    salida <= a xor b;
42
43end arch;
```

3.5 OBJETOS

Existen cuatro tipos de objetos en VHDL: *signal*, variable, *constant* y *file*.

Signal: Este es el objeto más utilizado en el lenguaje VHDL y se declara entre la arquitectura y la palabra reservada *begin*.

Las *signals* son de tipo global, cuando son declaradas estas pueden ser utilizadas por cualquier proceso. La sintaxis de una señal es la siguiente:

```
SIGNAL  nombre_señal : tipo_de_dato
```

También se pueden declarar de tipo *array* como por ejemplo una señal del tipo std_logic_vector:

```
signal byte : std_logic_vector;
```

Una señal puede tener un valor inicial, su sintaxis es:

```
signal a : std_logic := ´0´;
```

Variables: Una variable en VHDL lleva el mismo concepto como si se tratara de un lenguaje de programación tradicional. Una memoria es una localidad de memoria simbólica, en donde un valor puede ser almacenado y modificado. Cabe recalcar que no existe una correspondencia entre una variable y una parte de hardware. La sintaxis de una variable es:

```
VARIABLE nombre_variable : tipo_de_dato;
```

A diferencia de una señal, la variable sólo puede ser de tipo local, es decir sólo puede ser utilizado en el interior de un proceso.

Una variable también puede tomar un valor inicial:

```
variable x := ´1´;
```

Constantes: Una contaste toma un valor que no puede ser cambiado. Su sintaxis es:

```
constant nombre_constante : tipo_de_dato := valor_constante;
```

Un ejemplo de uso de una constante sería:

```
constant byte : tipo_de_dato := 8;
```

3.6 DESCRIPCIÓN DE UN CIRCUITO

El lenguaje VHDL es un lenguaje concurrente, es decir que las instrucciones de código que se escriben se ejecutan todas al mismo tiempo. Únicamente las instrucciones que se encuentran en el interior de un ***process*** se ejecutan en forma secuencial, pero en sí el lenguaje VHDL es una lenguaje concurrente en su totalidad.

Por esta razón el lenguaje VHDL presenta dos tipos diferentes de instrucciones, unas concurrentes y las otras secuenciales (*process*), se las detalla a continuación:

Instrucciones Concurrentes: Este tipo de instrucciones nos permiten describir un circuito combinacional, este tipo de circuito no necesita de un reloj ya que la salida depende únicamente del valor de las entradas, por lo tanto ocurre un cambio en la salida cuando el valor de las entradas cambia.

A continuación se describen los diferentes tipos de instrucciones concurrentes.

- **Instrucción when/else**

  Este tipo de instrucciones son similares a las instrucciones **if** y **case** en un lenguaje tradicional de programación, su sintaxis es la siguiente:

```
nombre_señal <= valor_expresion_1 when boolean_expresion_1 else
                valor_expresion_2 when boolean_expresion_1 else
                valor_expresion_n;
```

  Las expresiones booleanas son sucesivamente evaluadas hasta encontrar la expresión verdadera, cuando esta es encontrada, la señal toma el valor de **valor_expresion**.

Ejemplo:

Se representará un circuito que se activa en 1 cuando se ingrese un determinado valor, en este caso la salida será verdadera si se ingresa el ASCII que corresponde a la letra A mayúscula (01000001).

```
---------------------------------------
-- Entidad comparador
---------------------------------------
entity comparador is

  Port (  entrada : in  STD_LOGIC_VECTOR(7 downto 0);
          salida : out  STD_LOGIC);

end comparador;


---------------------------------------
-- Arquitectura arch
---------------------------------------
architecture arch of comparador is


begin
      salida <= ´1´ when (entrada = ¨01000001¨) else
          ´0´;

end arch;
```

- **Instrucción with/select/when**

Esta sentencia es muy similar a una sentencia **case** en un lenguaje de programación tradicional, su sintaxis es:

```
with seleccionador select
nombre_señal <= valor_expresion_1 when escoje_valor_1
                valor_expresion_2 when escoje_valor_2

                valor_expresion_n when others;
```

Ejemplo:

Se representará un circuito multiplexor.

```
-------------------------------------
-- Entidad multiplexor
-----------------------------------


entity multiplexor is

      Port ( entrada : in  STD_LOGIC_VECTOR(1 downto 0);
             seleccionador : in STD_LOGIC_VECTOR(1 downto 0);
             salida : out  STD_LOGIC_VECTOR(1 downto 0));

      end multiplexor;

-------------------------------------
-- Arquitectura arch
-----------------------------------


architecture arch of multiplexor is


begin
      with seleccionador select
            salida <= ¨00¨ when ¨00¨
                      ¨01¨ when ¨01¨
                       ¨10¨ when ¨10¨
                       ¨11¨ when others;
end arch;
```

Instrucciones Secuenciales: Este tipo de instrucciones son similares a las usadas en los lenguajes de programación tradicionales, estas sentencias se ejecutan en forma secuencial únicamente en el interior de una instrucción *process*.

- **Instrucción process**

  Esta instrucción nos permite encapsular instrucciones de tipo secuencial, per la instrucción *process* en si es un tipo de instrucción concurrente, es decir se ejecuta en paralelo con las demás instrucciones siempre y cuando una señal que se encuentra en el interior de su lista sensitiva cambie.

  Existe una gran variedad de instrucciones secuenciales, pero muchas de ellas no son sintetizables en *hardware*. Para no tener problemas de síntesis se debe tener en cuenta dos aspectos importantes en el uso de la instrucción *process*:

- Se debe usar un *process* para describir un circuito con sentencias **if** y **case**.
- Se debe usar un *process* para construir elementos de memoria.

La sintaxis que tiene la instrucción *process* es la siguiente:

```
process (lista sensitiva)

begin
       sentencia secuencial

end process;
```

La lista sensitiva es el conjunto de señales a las cuales la sentencia *process* responde, es decir si una señal que se encuentra en la lista sensitiva cambia, entonces esto activará a la instrucción *process*.

- **Instrucción if y else**

El significado de estas sentencias es el miso que para un lenguaje de programación tradicional. Su sintaxis es:

```
if (expresión_booleana_1) then
      sentencias_secuenciales;

elsif (expresión_booleana_2)then
      sentencias_secuenciales;

else (expresión_booleana_3) then
      sentencias_secuenciales;

end if;
```

Ejemplo:

Como ejemplo un comparador de señales, se puede evaluar condiciones para activar una u otra señal dependiendo de las señales en la lista sensitiva

```
-------------------------------------
-- Proccess
-----------------------------------

process (a, b, c)

      begin
            if (a = ´1´) then
               salida <= ¨00¨;

            elsif (b = ´1´) then
                  salida <= ¨01¨;

            else

       salida <= ¨11¨;

end process;
```

- **Sentencia case**

  El significado de estas sentencias es el miso que para un lenguaje de programación tradicional. Su sintaxis es:

```
case selector is

      when caso1 =>

            sentencias_secuenciales;

      when caso2 =>

            sentencias_secuenciales;

      when others =>

            sentencias_secuenciales;

end case;
```

3.7 COMPONENTS

Los *components* son código que es utilizado para la construcción de un diseño jerárquico.

Ejemplo:

Se representa la construcción de una compuerta NOR, mediante la unión de dos compuertas una OR y una NOT.

Paso uno: se construye una entidad Or_entity, que representa a la compuerta OR.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Or_entity is
Port ( a : in STD_LOGIC;
 b : in STD_LOGIC;
 salida_or : out STD_LOGIC);
end Or_entity;

architecture Behavioral of Or_entity is

begin

 salida_or <= a or b;

end Behavioral;
```

Paso dos: se construye una entidad Not_entity, que representa a la compuerta NOT.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Not_entity is

Port ( c : in STD_LOGIC;
salida_not : out STD_LOGIC);
end Not_entity;

architecture Behavioral of Not_entity is

begin
salida_not <= not c;

end Behavioral;
```

Paso tres: se realiza la unión de las dos entidades OR y NOT, para formar la entidad de nivel superior NOR.

```
 library IEEE;
 use IEEE.STD_LOGIC_1164.ALL;
```

```
entity Nor_entity is
Port ( operando1 : in STD_LOGIC;
operando2 : in STD_LOGIC;
salida_nor : out STD_LOGIC);
end Nor_entity;

architecture Behavioral of Nor_entity is
signal entrada_aux : std_logic;
begin

-- Declaración de la entidad Not_entity.
Not_unit : entity work.Not_entity(Behavioral)
port map(
c => entrada_aux,
salida_not => salida_nor
);

-- Declaración de la entidad Or_entity.
Or_unit : entity work.Or_entity(Behavioral)
port map(
a => operando1,
b => operando2,
salida_or => entrada_aux
);

end Behavioral;
```

En la figura A3-2, se muestra la representación gráfica de la unión de las entidades OR y NOT, mediante *component.*

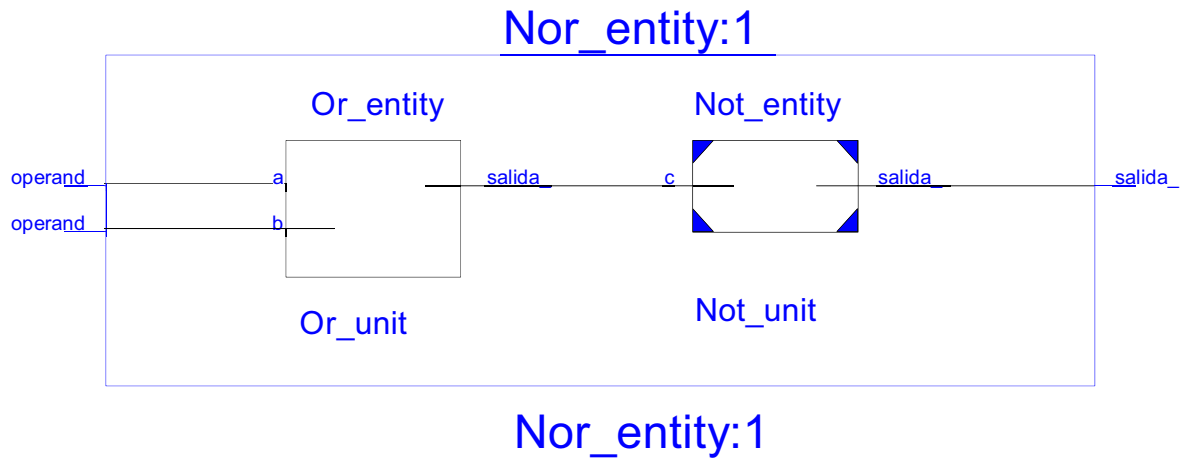


Figura A3-2. Diagrama RTL de la compuerta NOR.

REFERENCIAS

[A3.1] W. Fook , *Coding and Logic Synthesis with Synopsys*. Primera Edición, 2000.

[A3.2] P. Chu, *FPGA Prototyping by VHDL Examples*. Primera Edición, 2008.

[A3.3] P. Chu, *RTL HARDWARE DESIGN USING VHDL*. Primera Edición, 2006.

[A3.4] P. Vonei, *Circuit Design with VHDL*. Primera Edición, Massachusetts Institute of Technology, 2004.

ANEXO 4

MI PRIMER PROYECTO EN ISE PROJECT NAVIGATOR 13.1

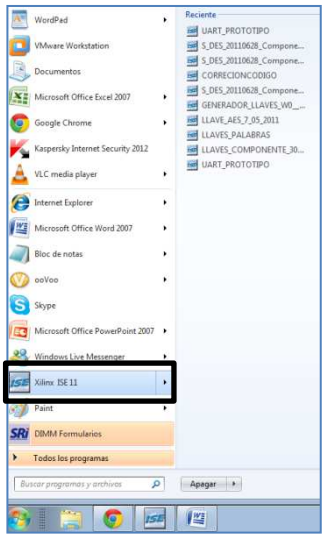
Este anexo describirá un ejemplo básico para la utilización del ISE, la versión usada es 13.1, la versión es fundamental ya que en esta se pueden realizar diseños que son ejecutados en una tarjeta de entrenamiento Virtex-5. El *software* puede ser descargado de la página oficial de xilinx:

http://www.xilinx.com/support/download/index.htm

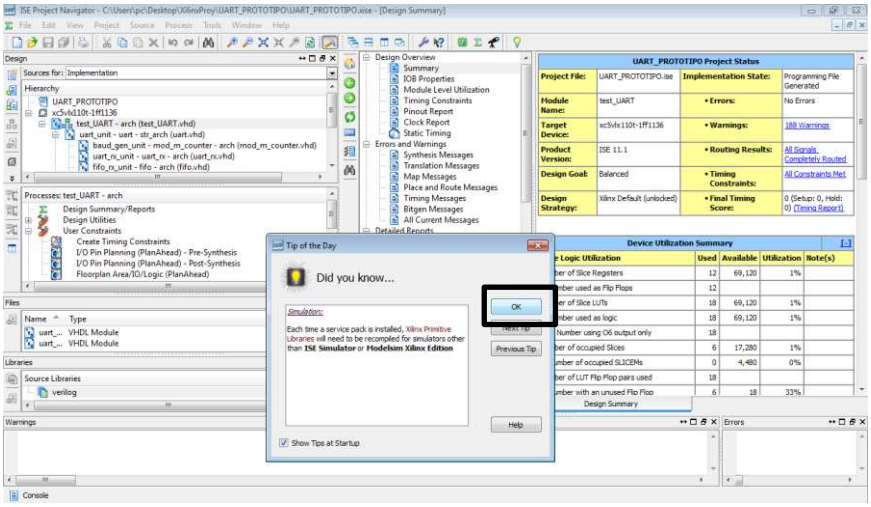
Creando una cuenta en este sitio web, se puede obtener una licencia de prueba que dura 30 días. Se procederá como ejemplo práctico la implementación de una compuerta **xor** de dos bit de entrada.

Una vez instalado el software los pasos a seguir son:

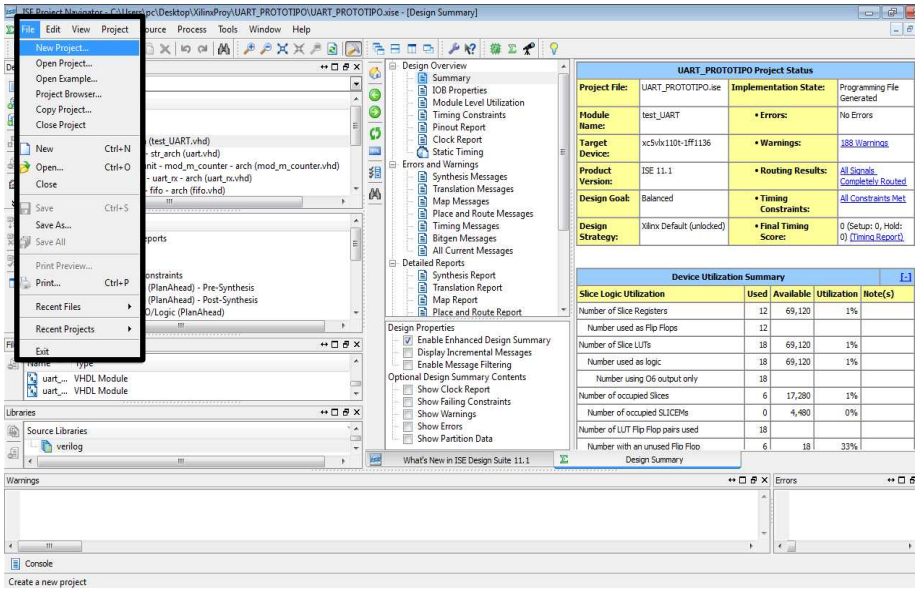
1. Abrir la Aplicación Xilinx ISE.



2. En la opción Type of the day, clic → *OK*.

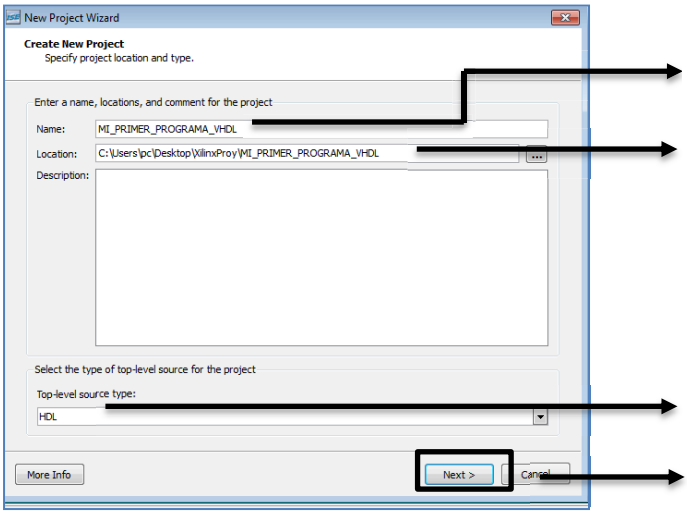


3. Se crea el primer proyecto, dando un clic en: *File → New Project.*



4. Se despliega la pantalla: *New Project Wizard.*

a. Ingresar el nombre del proyecto en este caso: MI_PRIMER_PROGRAMA_VHDL.
b. Indicar la ubicación del proyecto.
c. Top-Level: HDL.
d. Clic → *Next.*



a. Name: MI_PRIMER_PROGRAMA_VHDL

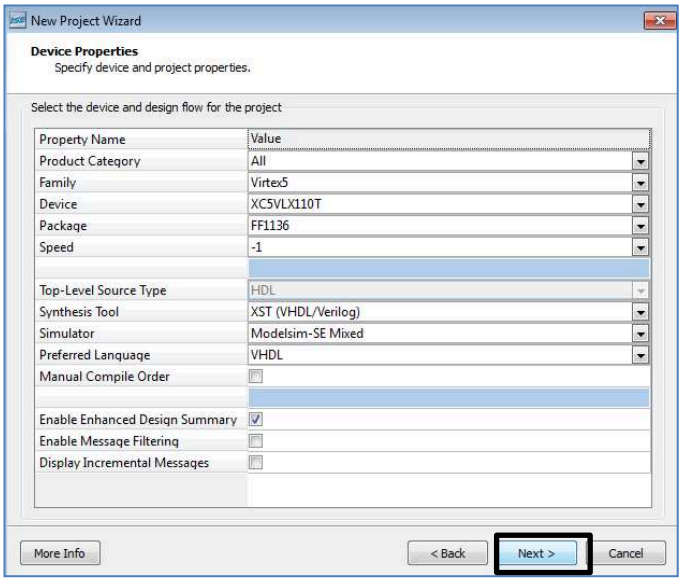
b. Location: En la carpeta XilinxProy (que se crea cuando se instala el programa)

c. Top-level source type: HDL

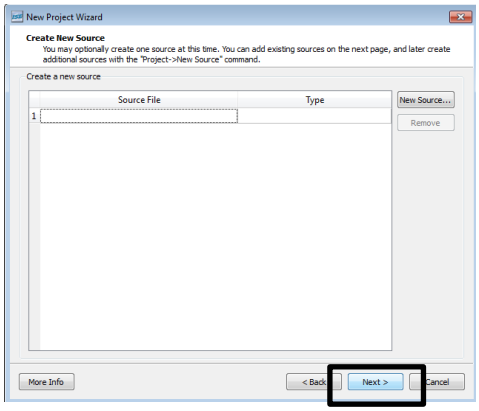
d. NEXT

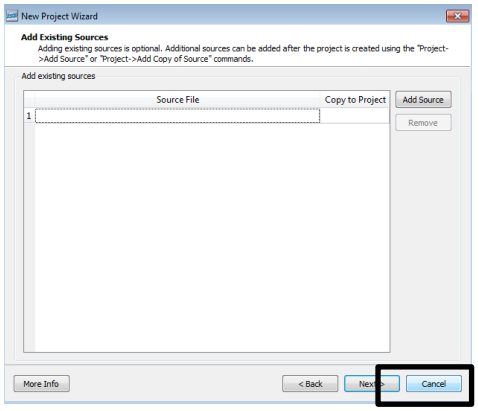
5. En la siguiente ventana, se debe especificar las características de la tarjeta, para nuestro caso una VIRTEX-5, con un fpga: XC5VLX110T.

   Estas características son propias de cada FPGA. El lenguaje de programación es VHDL.

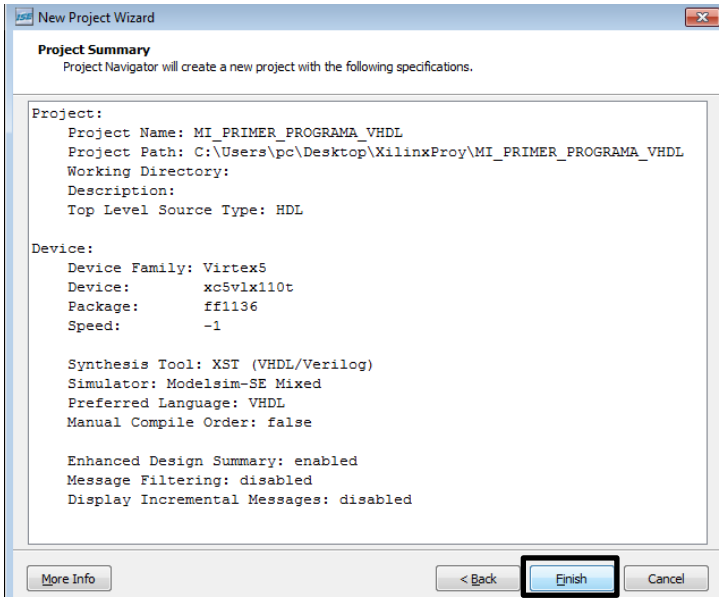


6. Como no se va adherir este proyecto a otro existe, en las siguientes dos ventanas se presiona *NEXT*.

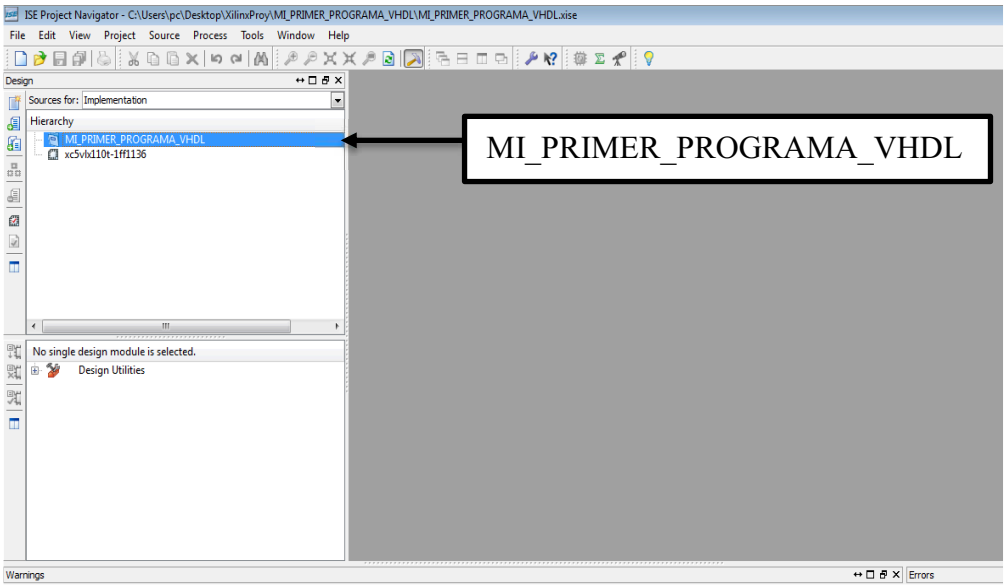




7. En la siguiente ventana: *Project Sumary*, se indica un resumen de todas las características seleccionadas, se comprueba la información y se da clic en *FINISH*.



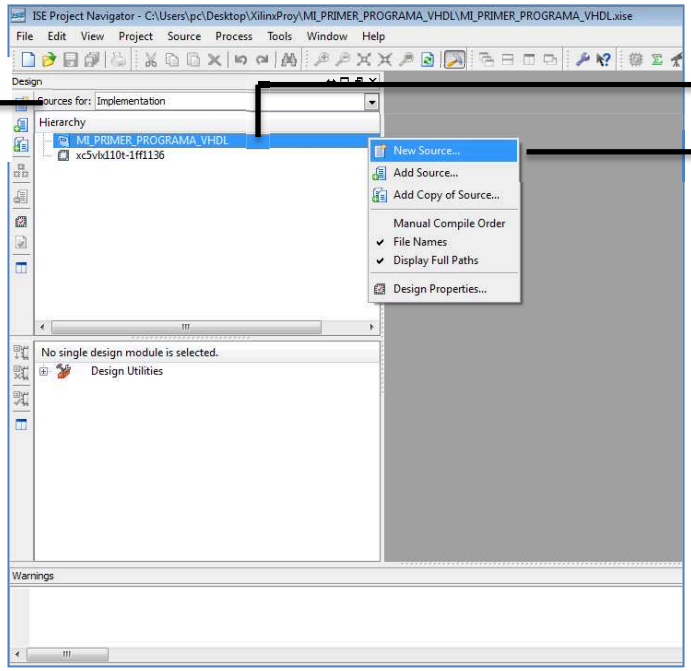
8. Una vez creado el proyecto, se puede observar en la aplicación el proyecto creado.



9. A continuación se va a crear una pequeña aplicación:

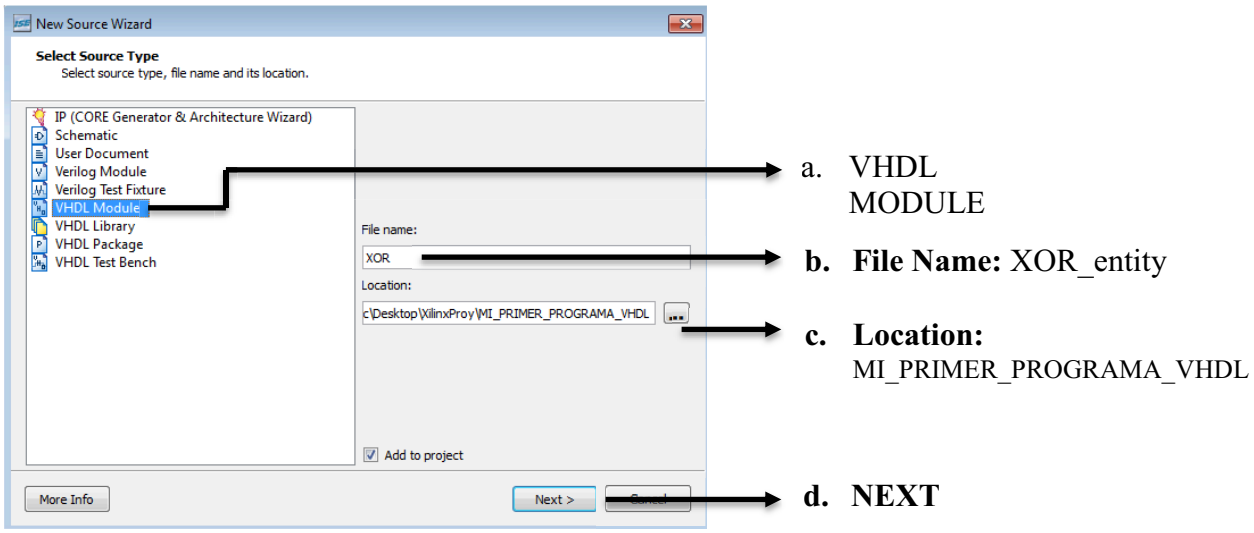
   a. Clic derecho.
   b. Seleccionar: *New Source.*
   c. La opción: *Sources for*: debe tener → "*Implementation*".

c. Source for: Implementation

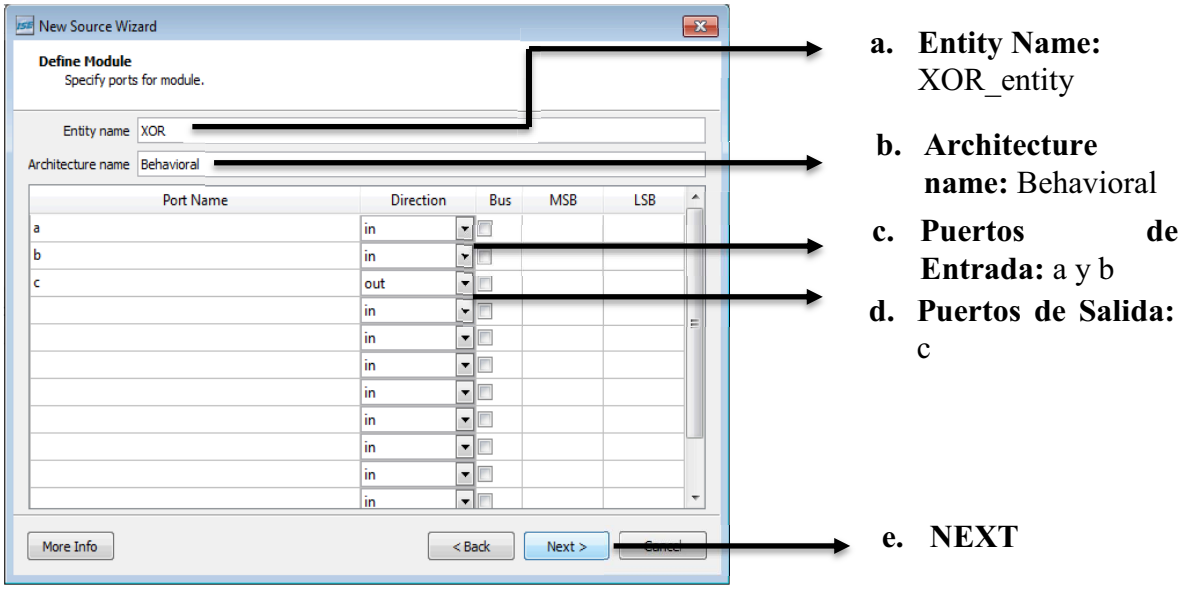


10. Se despliega una ventana donde se seleccionan los siguientes parámetros:

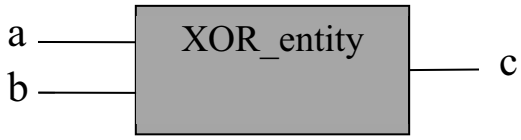
    a. VHDL *Module.*
    b. El nombre de la aplicación: XOR_entity.
    c. Comprobar la ubicación de la aplicación.
    d. *Next.*



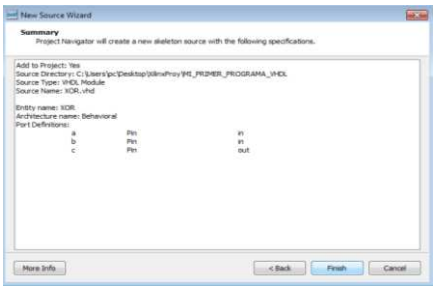
11. Se definen las características la aplicación, que desde este momento la se llama como una entidad Xor_entity, la cual contiene una arquitectura, puertos de entrada y puertos de salida.



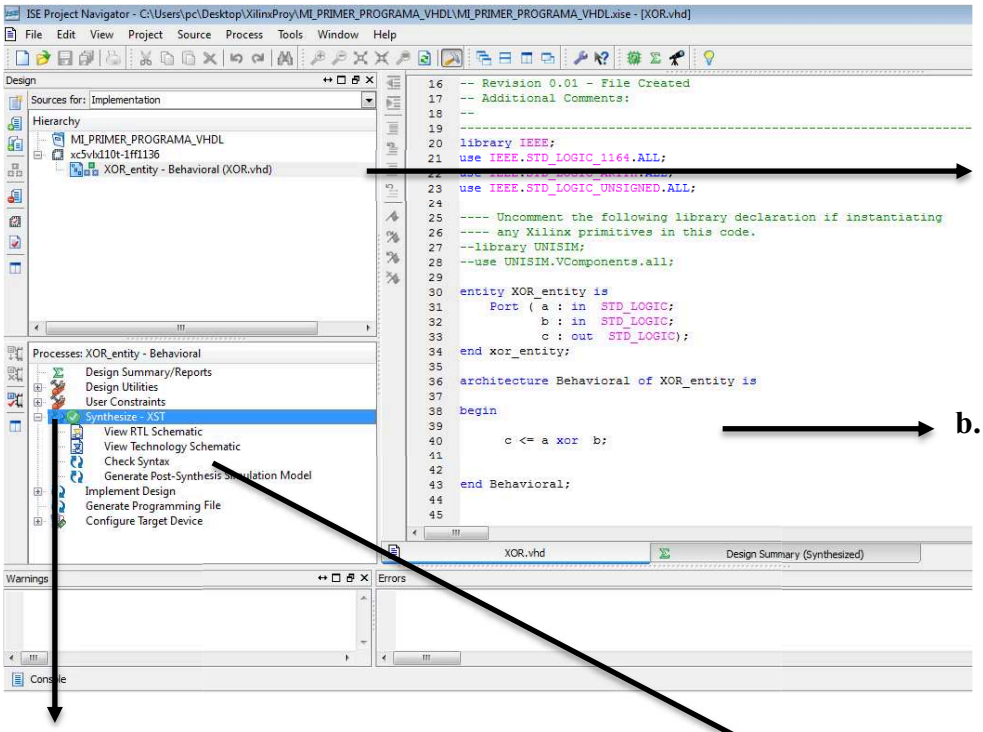
NOTA: Una representación gráfica de esta entidad, es la siguiente:



12. En la opción *Summary*, se verifica los datos ingresados y se da clic en *FINISH*.



13. A continuación se programa la entidad XOR_entiy:



a. Selecciona la entidad:
Xor_entity

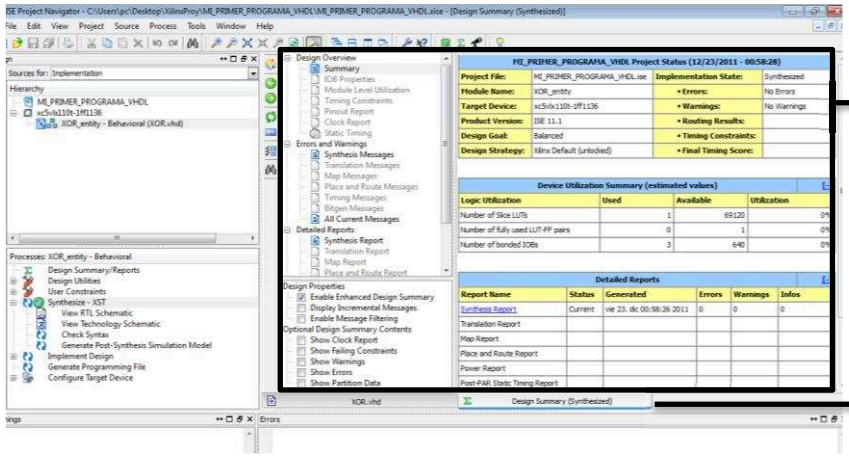
b. Se programa la entidad, con el código:
c <= a xor b;

c. Se Verifica que NO existan warnings y errores.

d. Sintetizar el programa

El software ISE ® incluye la tecnología de síntesis Xilinx (XST), sintetiza código VHDL, Verilog, o diseños mixtos. Básicamente lo que se consigue es transformar lo creado en HDL, a compuertas lógicas y FFs.

14. Se puede observar los recursos utilizados, dando clic en la parte inferior de la pantalla en la opción: *Design Summary (Sinthesized).*



b. Resumen de los recursos utilizados de la tarjeta.

a. Clic: Design Summary (Sinthesized)

CARGAR EL PROGRAMA EN LA TARJETA

1. Hay que definir los dispositivos de entrada y salida de la tarjeta, asignando los pines de la misma. Para este caso se usa:

   a. Dos pines del DIP *switch*, para las entradas: *a* y *b.*
   b. Un led, para la salida: *c.*

7. User and Error LEDs (Active-High)

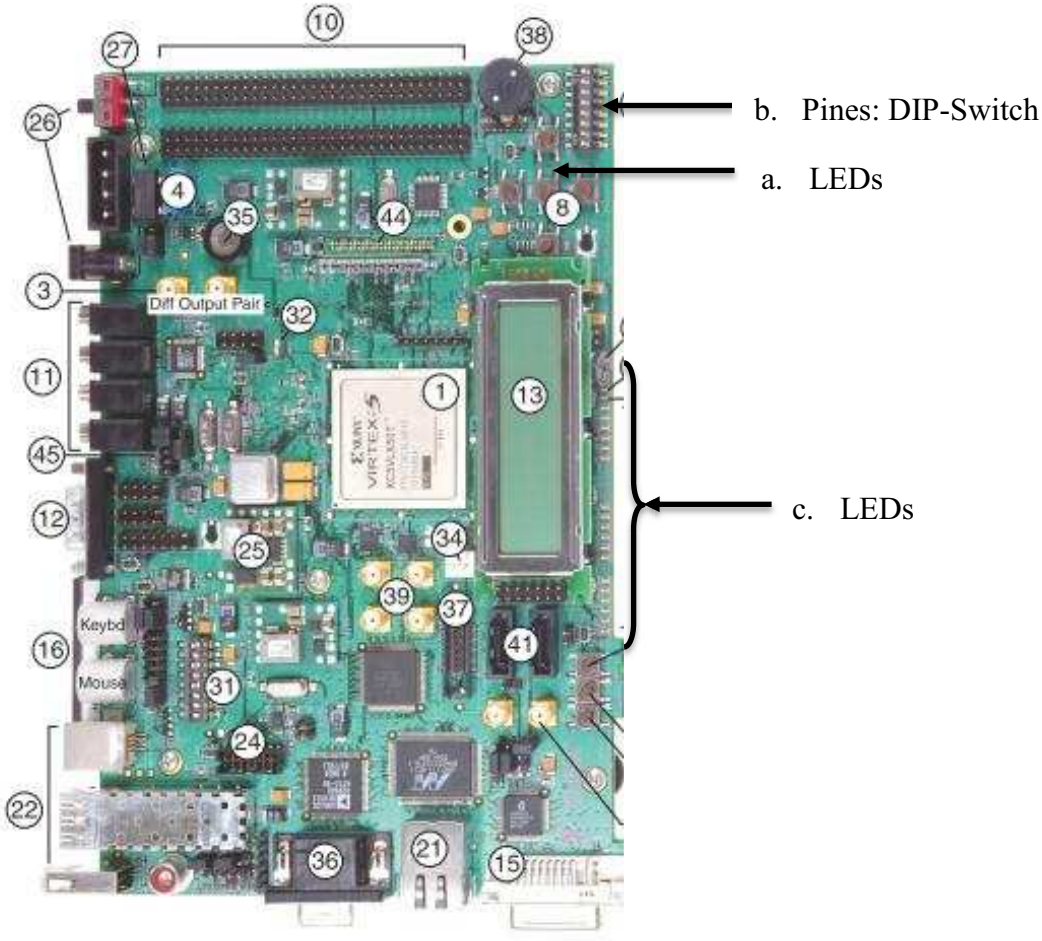
There are a total of 15 active-High LEDs directly controllable by the FPGA:

- Eight green LEDs are general purpose LEDs arranged in a row
- Five green LEDs are positioned next to the North-East-South-West-Center-oriented pushbuttons (only the *center* one is cited in Figure 1-2, page 15)
- Two red LEDs are intended to be used for signaling error conditions, such as bus errors, but can be used for any other purpose

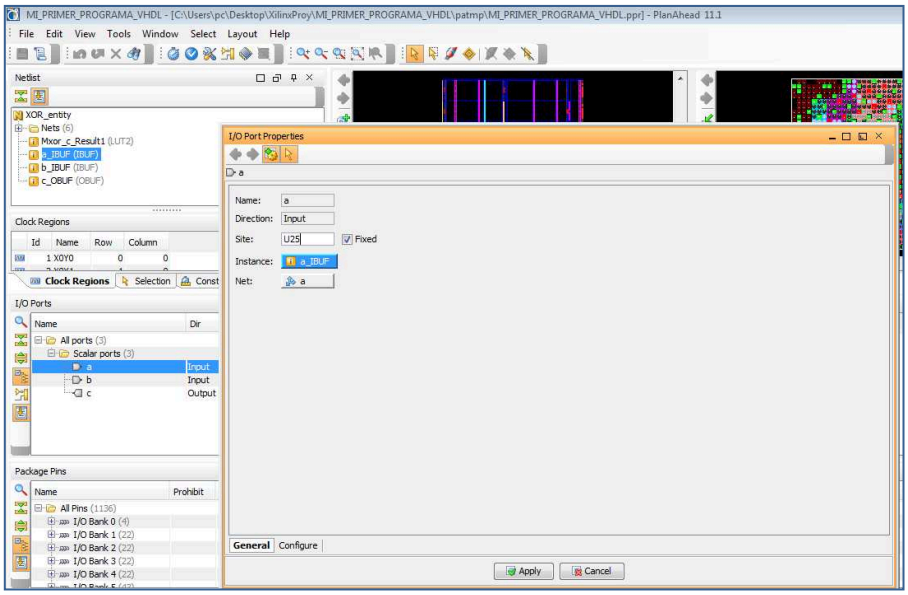
Some LEDs are buffered through the CPLD to allow the LED signals to be used as higher-performance I/O by way of the XGI expansion connector. Table 1-6 summarizes the LED definitions and connections.

Table 1-6: **User and Error LED Connections**

Reference Designator	Label/Definition	Color	FPGA Pin	Buffered
DS20	LED North	Green	AF13	Yes
DS21	LED East	Green	AG23	Yes
DS22	LED South	Green	AG12	Yes
DS23	LED West	Green	AF23	Yes
DS24	LED Center	Green	E8	Yes
DS17	GPIO LED 0	Green	H18	Yes
DS16	GPIO LED 1	Green	L18	Yes
DS15	GPIO LED 2	Green	G15	Yes
DS14	GPIO LED 3	Green	AD26	No
DS13	GPIO LED 4	Green	G16	Yes
DS12	GPIO LED 5	Green	AD25	No
DS11	GPIO LED 6	Green	AD24	No
DS10	GPIO LED 7	Green	AE24	No
DS6	Error 1	Red	F6	No
DS6	Error 2	Red	T10	No



2. Los valores de los pines correspondientes se encuentran en el documento UG247, que se puede descargar de la página web de Xilinx, para este caso se indica los valores de los pines requeridos de la aplicación.



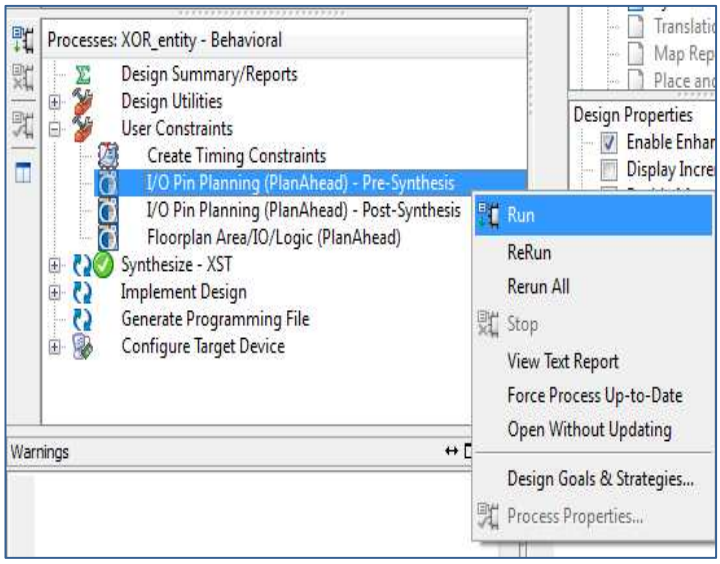
6. GPIO DIP Switches (Active-High)

Eight general-purpose (active-High) DIP switches are connected to the user I/O pins of the FPGA. Table 1-5 summarizes these connections.

Table 1-5: **DIP Switch Connections (SW8)**

SW4	FPGA Pin
GPIO_DIP_SW1	U25
GPIO_DIP_SW2	AG27
GPIO_DIP_SW3	AF25
GPIO_DIP_SW4	AF26
GPIO_DIP_SW5	AE27
GPIO_DIP_SW6	AE26
GPIO_DIP_SW7	AC25
GPIO_DIP_SW8	AC24

3. En el programa se selecciona la opción: *User Constrain*, y se ejecuta el programa, para asignar los pines de la tarjeta a la aplicación realizada. Se genera un archivo .ucf.

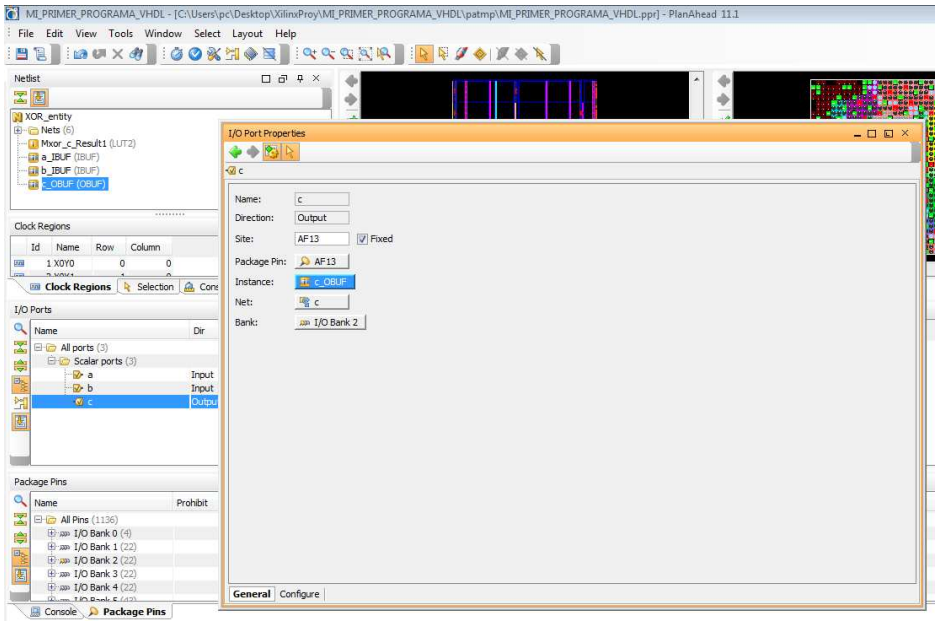


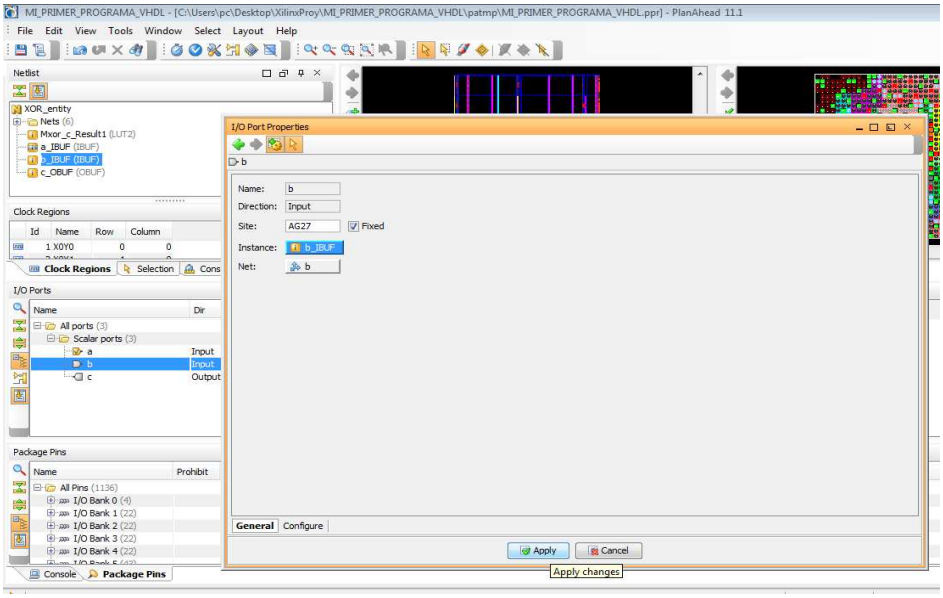
4. En la opción I/O Ports, se asignan los puertos dando doble clic en las entradas a b, y en la salida c.

a →U25

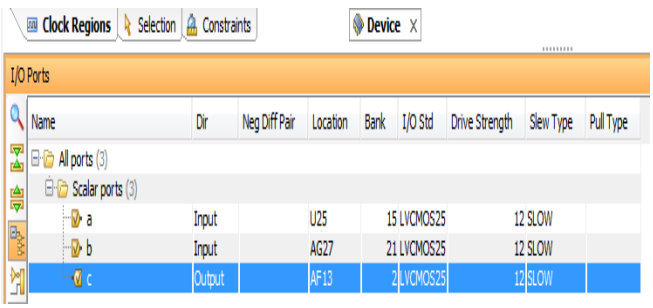
b →AG27

c→AF13

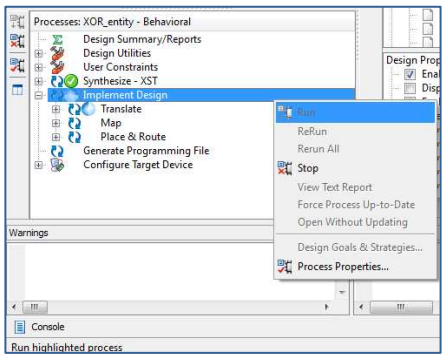




5. Verificar los puertos asignados en la ventana I/O port. Se cierra el programa y se vuelve al programa ISE.



6. En ISE, se ejecuta: *Implement Design*, y se verifica que no existan *warnings* o errores.

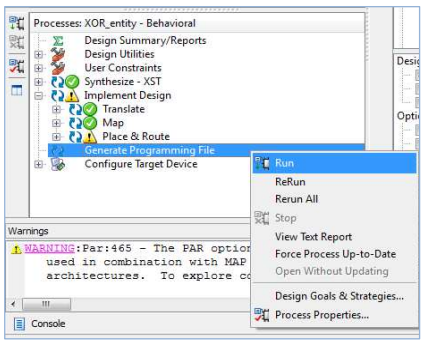


Implement Design

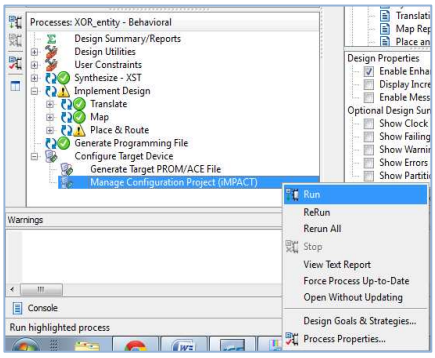
El proceso de implementación consta de 3 etapas:

- *Translate*.- Recopilar todos los *netlists*, en uno solo. Los módulos faltantes (que no son detectados por la síntesis) se marcarán en esta etapa.
- *Map*.- El *netlist* generado en la etapa anterior es comparado con los recursos del dispositivo FPGA. Es decir todas las compuertas lógicas generadas en el *netlist* son mapeados en las celdas y IOBs del FPGA. Si existen recursos insuficientes o especificados de forma incorrecta se muestra un error y se detiene la aplicación.
- *Place & Route*.- Coloca las celdas, en ubicaciones físicas y determina las rutas para conectar las señales.

7. Generar el archivo (.bit). Este archivo se carga en la tarjeta.

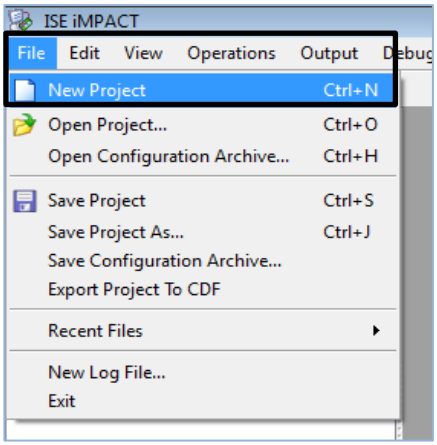


8. Ejecutar la opción *Impact*, para abrir el programa *iMPACT*, el cual permite enviar el archivo (.bit), a la tarjeta.

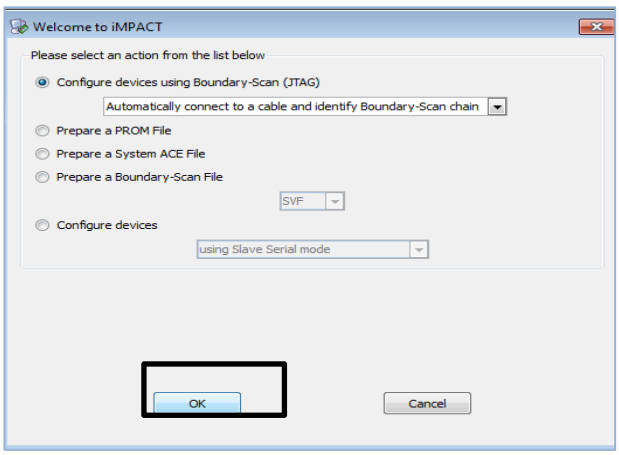


9. Una vez que se abra *iMPACT*, deben estar realizadas las conexión correspondientes entra la tarjeta de entrenamiento y la PC.

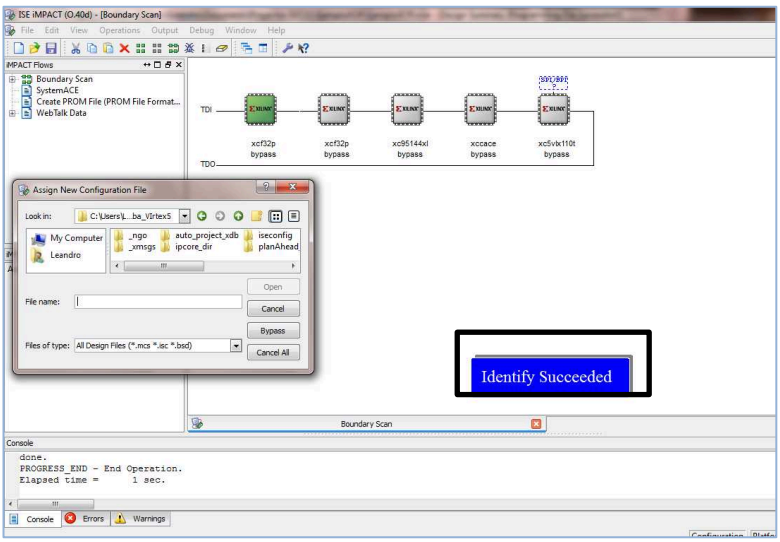
10. Se muestra la interface de *Impact*, y se selecciona un *NewProject.*



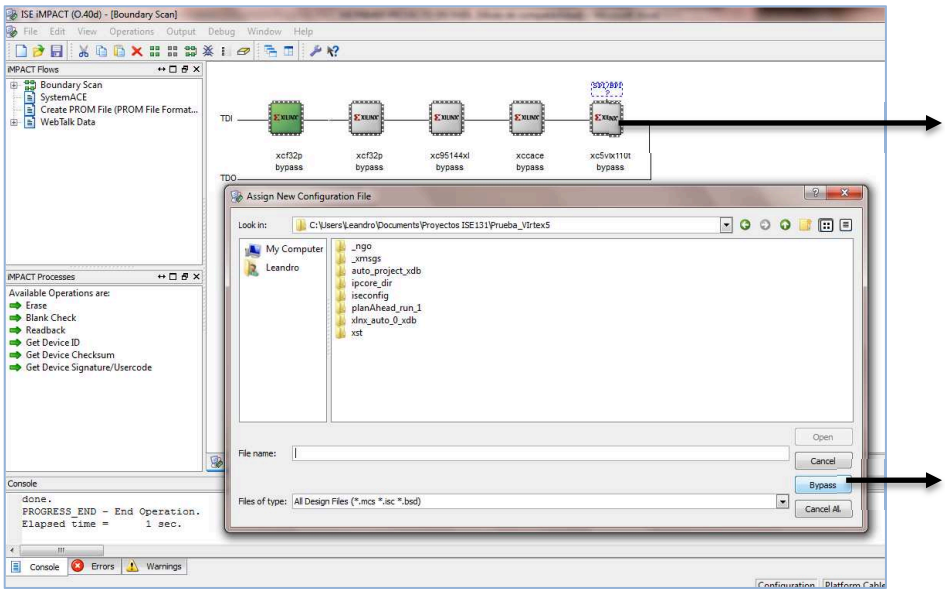
11. Seleccionar *OK*.



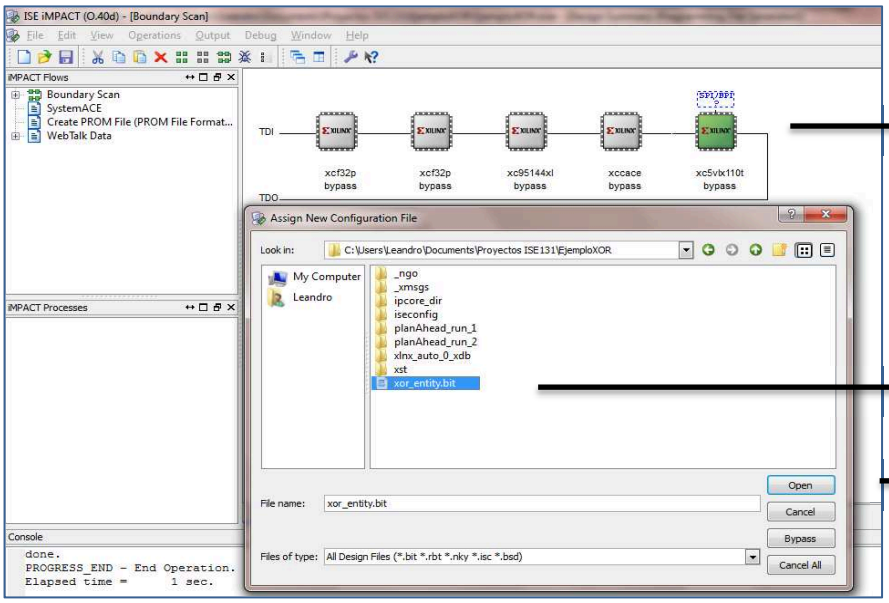
12. Se verifica si la tarjeta de entrenamiento está conectada a la PC.



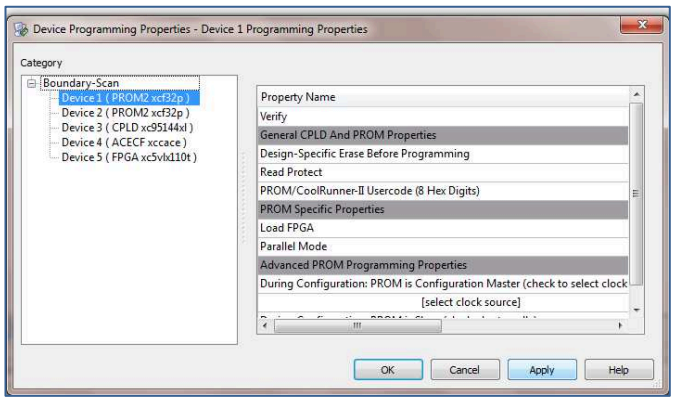
13. Para cargar el programa se observa 5 chips correspondientes a la tarjeta, en la que se va grabar es el ultima, en los anteriores, se da un clic en *Bypass*, hasta llegar al que nos corresponde.



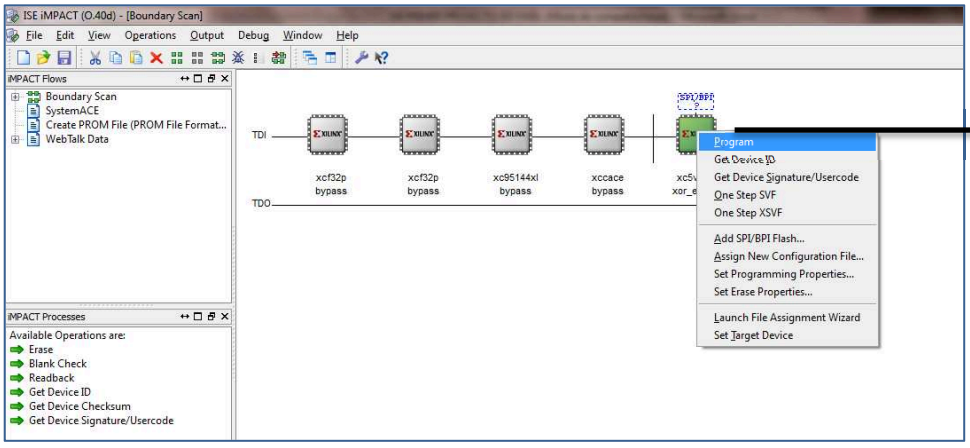
14. Cuando el chip 5, cambie de color a verde indica que se está trabajando en este. En la ventana *Asing New Configuration File*, seleccionar el archivo .bit, creado en el paso 8. Cargar el mismo y dar clic en *OPEN*.



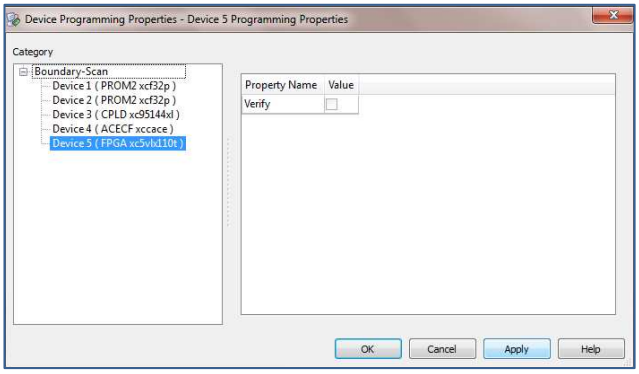
15. A continuación en la siguiente ventana, dar clic en *APPLY*, y luego en *OK*.



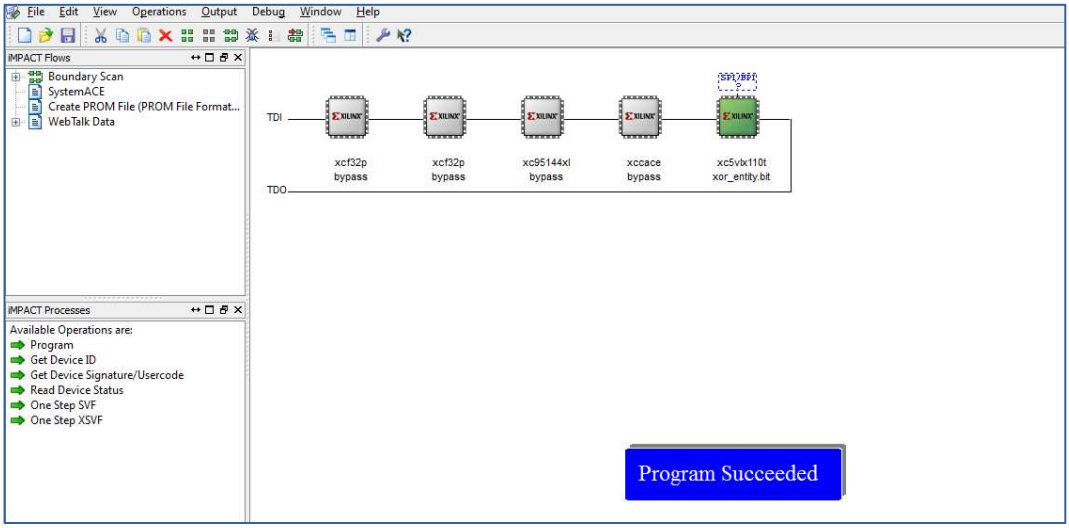
16. En el chip se da clic derecho, seleccionar *program*, y se enviar el archivo (.bit) a la tarjeta.



17. En la siguiente ventana seleccionar *Apply y OK*.



18. Verificar envió de los datos y si todo es correcto dará un mensaje: succeeded.



PARA FINALIZAR COMPROBAR EL FUNCIONAMIENTO EN LA TARJETA

ANEXO 5

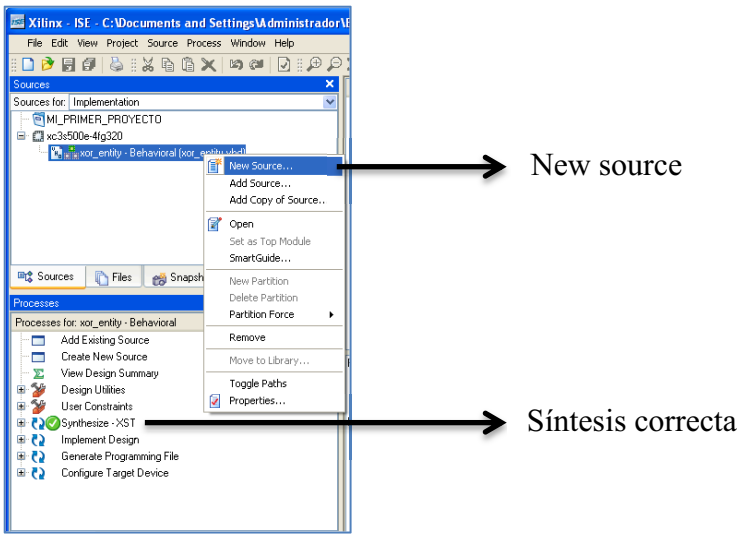
SIMULACIÓN EN ISE PROJECT NAVIGATOR 10.1

La fase de simulación se realizó en *Ise Project Navigator* 10.1, pues la versión 13.1 no tiene la herramienta *Test Bench Waveform*, que permite implementar la simulación.

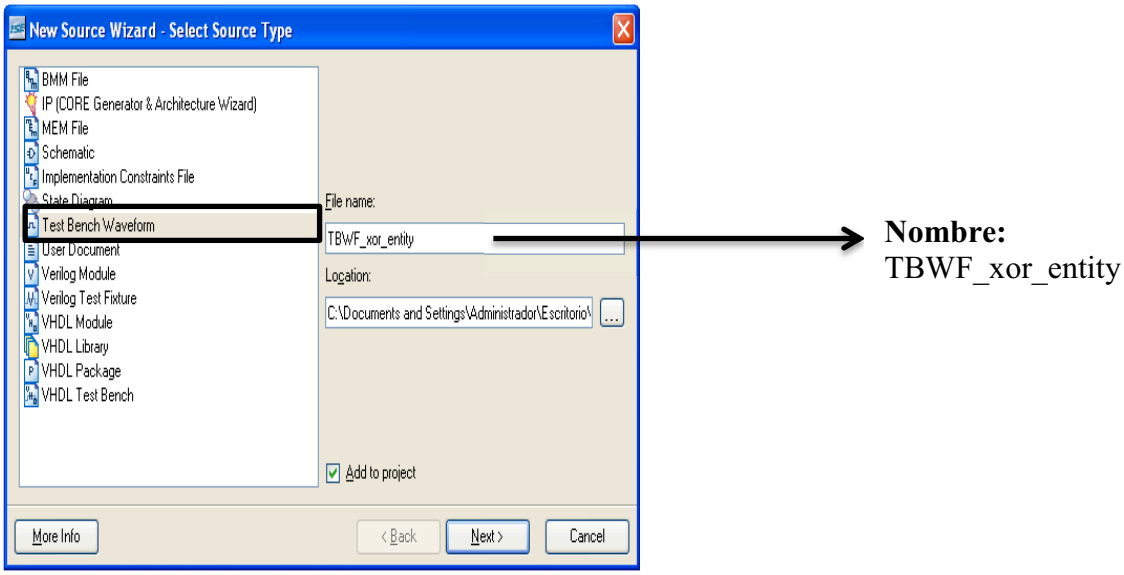
En la versión 13.1, sí se puede realizar simulaciones pero el proceso es más complejo ya que se tiene que programar los parámetros de la misma. Es por esta razón que se ha optado por usar una versión anterior solo en esta etapa del diseño. Los pasos a seguir son menos complejos y justifica usar la herramienta *Test Bench Waveform*. El ejemplo que se va a simular es el realizado en el anexo 4.

Una vez creado el proyecto y sintetizado, se siguen los siguientes pasos para la simulación:

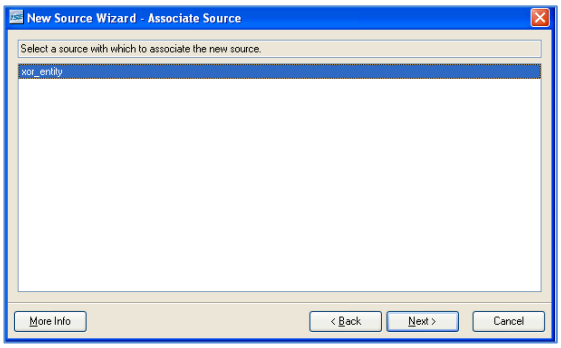
1. En la entidad xor_entity, dar clic derecho →*New source*.



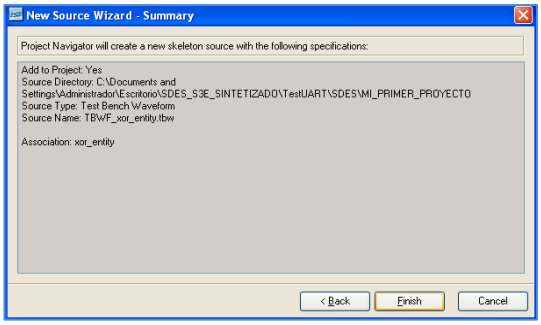
2. Seleccionar *Test Bench Waveform* y escribir el nombre del archivo que va contener la simulación.



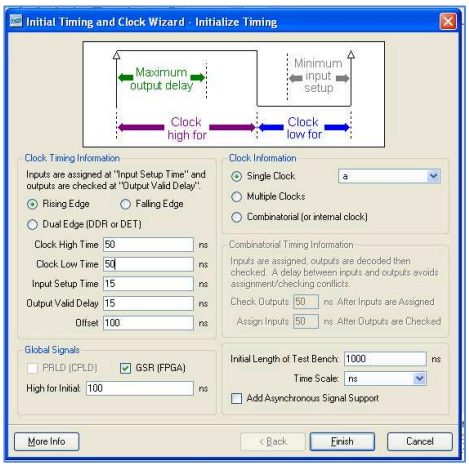
3. Clic →*Next*.



4. Clic → *Finish*.



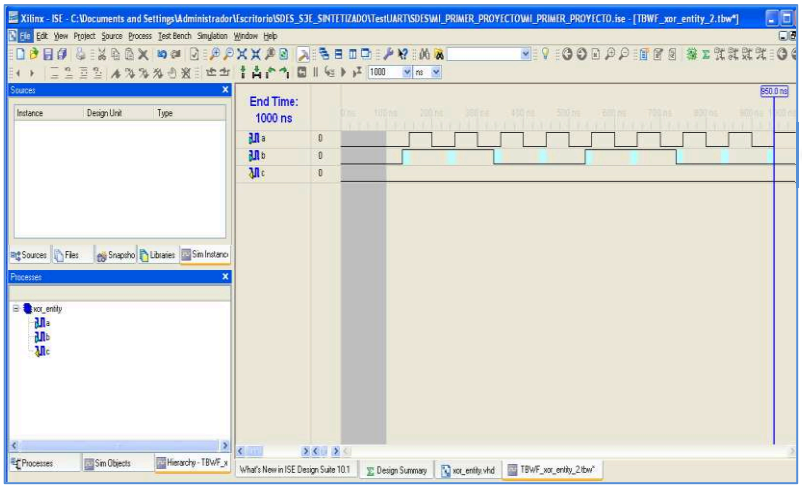
5. En esta ventana se detallan los parámetros del reloj, si todo es correcto dar clic en *Finish*.



Los parámetros variaran dependiendo de los requerimientos de cada diseño.

6. El siguiente paso es seleccionar los datos de las señales de entrada en forma gráfica, para el ejemplo implementado se tiene las entradas a y b.

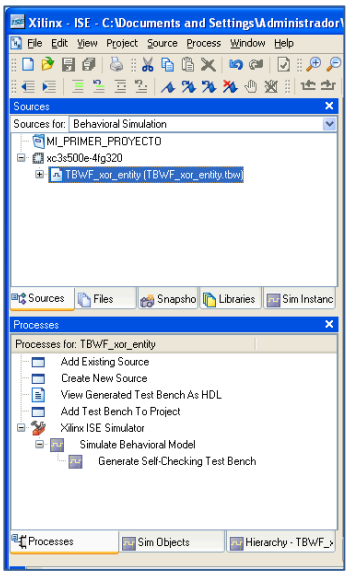
   Una vez definido el valor de las señales se debe guardar los cambios.



→ **Señales entrada**: a y b

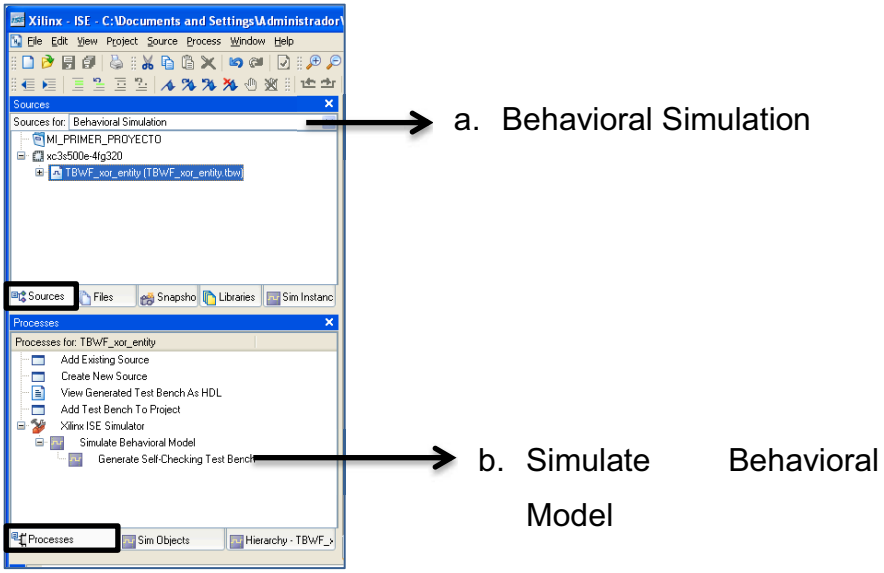
7. Para implementar la simulación, regresar a la ventana de los módulos y dar clic en el archivo (.tbw), cuyo nombre es TBWF_xor_entity.

   Es importante guardar los cambios para poder visualizar la ubicación del archivo.

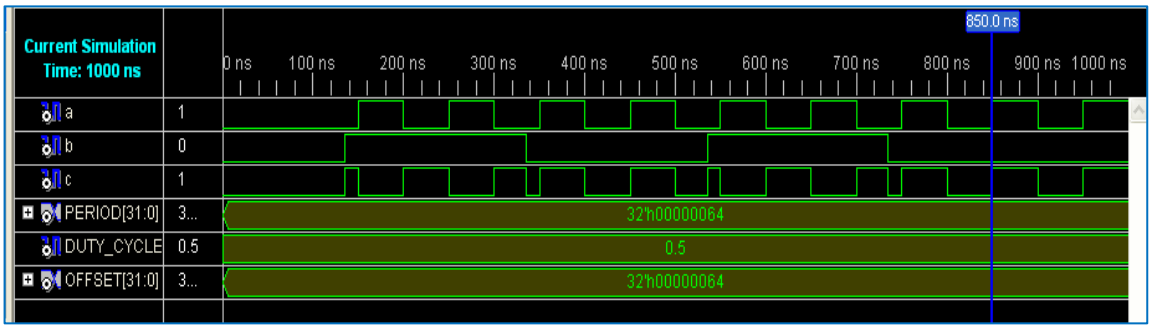


8. En la opción ***Sources for***, seleccionar:

   a. *Behavioral Simulation*.
   b. En *Processes*: dar clic →*Simulate Behavioral Model*.



9. Para finalizar se despliega la ventana, con los resultados obtenidos en la simulación, la información de la señal de salida denominada como c, dará como resultado la operación xor entre las entradas a y b.



La utilización de esta herramienta es de gran ayuda en el desarrollo de este proyecto, pero se debe recordar que solo es una ayuda en *software*, y el funcionamiento real de los diseños solo se puede comprobar con la implementación en el *hardware*.

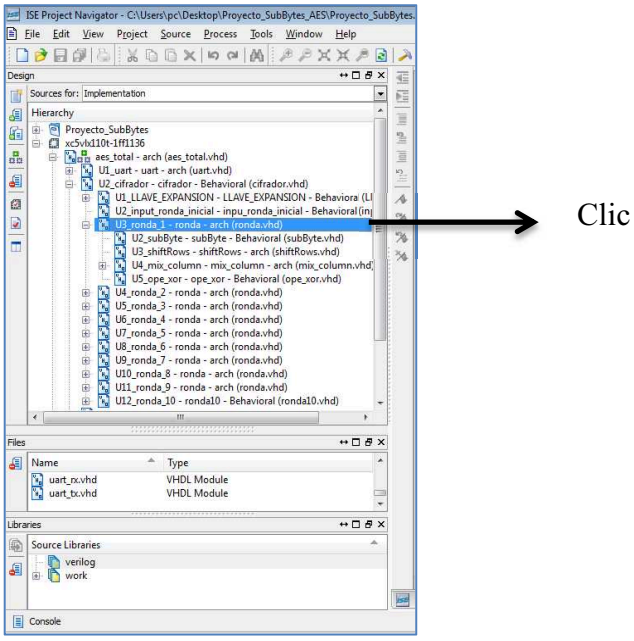
ANEXO 6

DIAGRAMA RTL

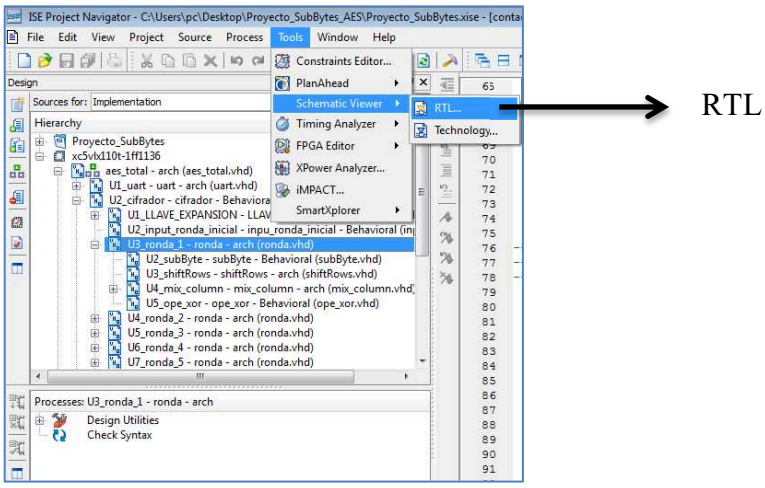
Un diagrama RTL por sus siglas en inglés *Register Transfer Level*, es una descripción del comportamiento de un diseño de transferencia de registros, que permite verificar la descripción de un chip. Este tipo de simulación también permite validar la sintaxis del código y confirmar que este funcione como se pretende que lo haga.

El diagrama RTL que se va a simular, es la ronda 1 del proceso de cifrado, que se detalla en el capítulo 3. Una vez creado el proyecto y sintetizado, se siguen los siguientes pasos:

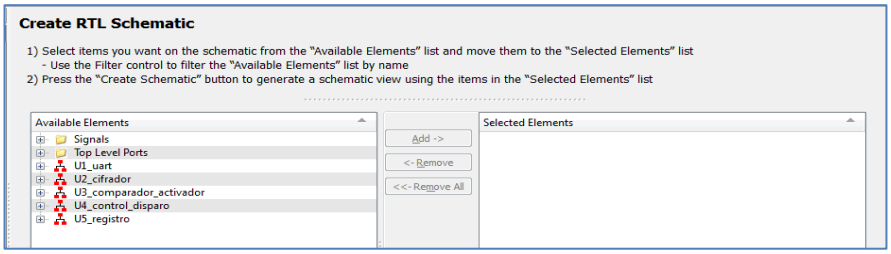
1. En la entidad ronda1, dar clic → Selecciona el requerimiento.



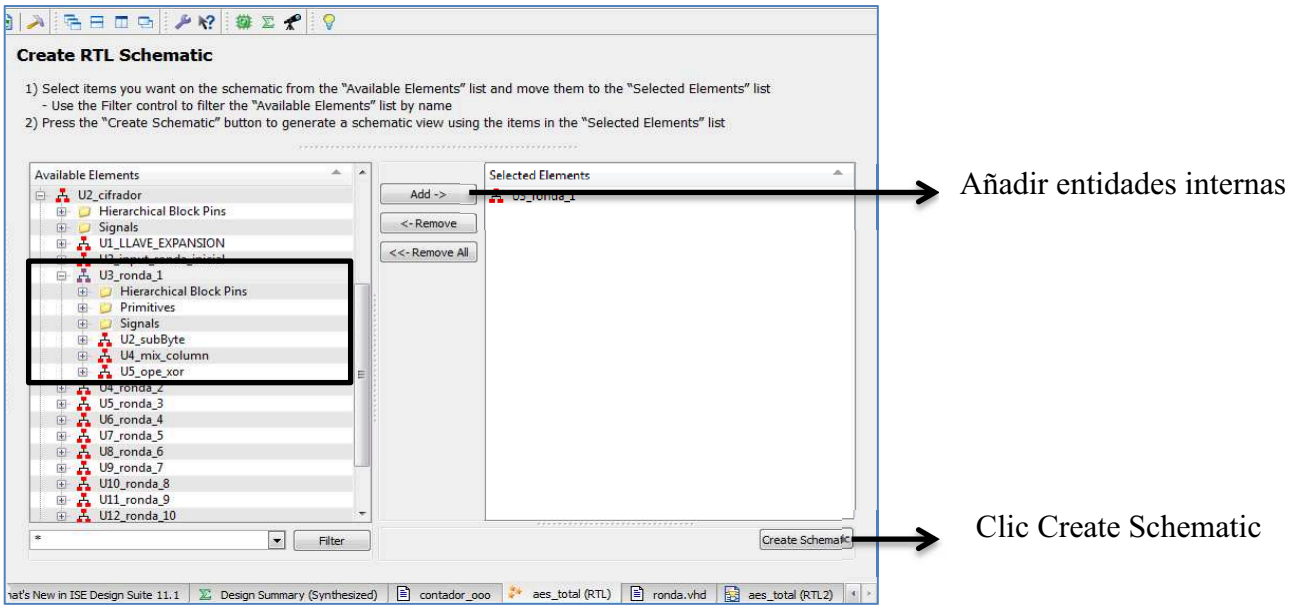
2. Seleccionar en la barra de herramientas *Tools* → *Schematic Viewer* → RTL.



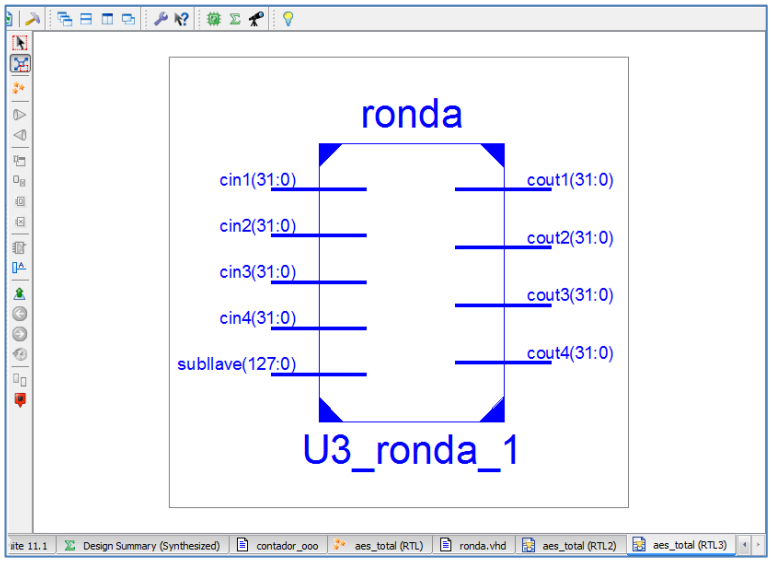
3. Se despliega el menú de diagrama RTL.



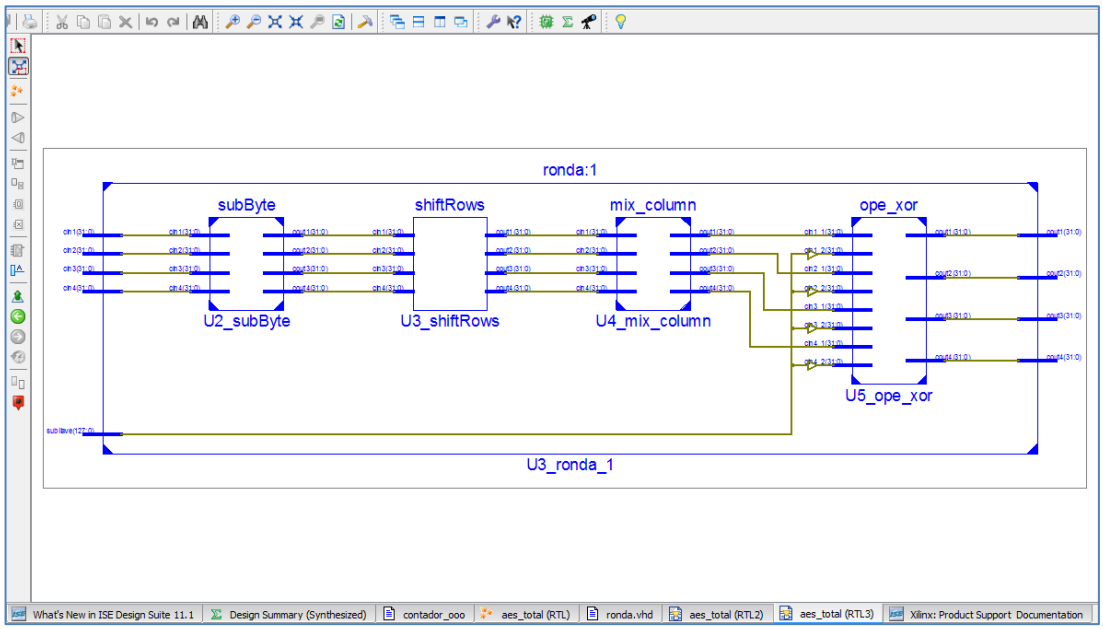
4. Seleccionar las entidades internas y dar clic → *Create Schematic.*



5. Se visualiza la entidad ronda 1 total.



6. Dando clic en la entidad total se observa la interconexión de las entidades internas.



REFERENCIAS

[A6.1] Xilinx ISE Design Suite 13 Software Manuals, Xilinx, 2011.

ANEXO 7

CÓDIGO VHDL DEL ALGORITMO DECIFRADO S-DES

(Se encuentra en digital)

ANEXO 8

CÓDIGO MATLAB DEL ALGORITMO DE CIFRADO DES

(Se encuentra en digital.)

ANEXO 9

CAMPOS FINITOS

9.1 GRUPO, ANILLO Y CAMPOS

Para comprender el significado de un Campo (***F***) se debe saber la definición de Grupo (***G***) y Anillo (***R***).

9.1.1 GRUPO

Un grupo es una estructura algebraica que consta de un conjunto (***G***), junto con una operación (·), que combina cualquier pareja de elementos (***a,b***) de G, para formar un tercer elemento ***a·b*** o ***ab***. Suele estar denotado por ***{G, ·}*** y para que se pueda calificar como un grupo, el conjunto y la operación deben satisfacer algunas condiciones llamadas axiomas de grupo establecidos en la tabla A9-1. [A9.1] [A9.2]

Condición	Definición
Clausura	$\forall a, b \in G : a \cdot b \in G$
Asociativa	$\forall a, b, c \in G : (a \cdot b) \cdot c = a \cdot (b \cdot c)$
Elemento Identidad	$\exists \boldsymbol{e} \in G, \forall a \in G : a \cdot \boldsymbol{e} = \boldsymbol{e} \cdot a = a$
Elemento Inverso	$\forall a \in G, \exists a' \in G : a \cdot a' = a' \cdot a = \boldsymbol{e}$

Tabla A9- 1. Axiomas de grupo *{G, ·}*. [A9.1]

El operador (·), es genérico y puede referirse a la adición, multiplicación, o alguna otra operación matemática.

9.1.1.1 Grupo Abeliano

Un grupo abeliano ***{G, + }*** consiste en un conjunto ***G*** y una operación definida en sus elementos, representados por : ***+ .***

$$+ : G \times G \rightarrow G : (a , b) \rightarrow a + b. \quad [A9.3]$$

Además tiene que cumplir con las condiciones de la tabla A9-2.

Condición	Definición
Clausura	$\forall a, b \in G : a + b \in G.$
Asociativa	$\forall a, b, c \in G : (a + b) + c = a + (b + c).$
Conmutativa	$\forall a, b \in G : a + b = b + a.$
Elemento Neutro	$\exists 0 \in G, \forall a \in G : a + 0 = a.$
Elementos Inversos	$\forall a \in G, \exists b \in G : a + b = 0$

Tabla A9- 2. Condiciones para que sea un grupo abeliano *{G,+}.* [A9.3]

9.1.1.1.1 Anillo

Un anillo ***(R),*** es un conjunto de elementos con dos operaciones, llamadas adición y multiplicación, cuya notación es: ***{R, +, *},*** de modo que ***{R,*}*** es un grupo conmutativo con elemento neutro. El producto ***(*)*** es asociativo y tiene la propiedad distributiva respecto a la suma.

Si el producto es conmutativo se habla de un anillo conmutativo y si el anillo posee un elemento neutro para el producto, se lo llama anillo con unidad. [A9.1] [A9.2]

Este conjunto debe de cumplir con las siguientes condiciones:

- La estructura es un ***grupo abeliano***.

- La operación ***(*)*** es cerrada y asociativa sobre ***R***. Existe un elemento neutro para ***(*)*** en ***R***.
- Las dos operaciones ***(+)*** y ***(*)*** están relacionadas por la ley distributiva.
- Un anillo es llamado conmutativo si la operación ***(*)*** es conmutativa.

9.1.1.1.2 Campo

F es un campo si y solamente si ***F*** es un anillo en el cuál la multiplicación es conmutativa y cada elemento, excepto 0, tiene un inverso multiplicativo. Es posible definir un campo ***{F,+,x},*** con respecto a la adición y la multiplicación si:

- **F** es un grupo conmutativo con respecto a la adición.
- **F** − { 0 } es un grupo conmutativo con respecto a la multiplicación.
- Se mantiene la ley distributiva de los anillos.

9.1.1.1.3 Campo finito

Un campo finito (***GF)***, o campo de Galois, es aquel que tiene un número finito de elementos. Los campos finitos más usados en criptografía son los campos: primos, binarios y óptimos extendidos.

El orden de un campo finito es el número de elementos en el campo ***n***. Un campo finito ***GF*** de orden ***q*** solamente puede existir si ***q*** es una potencia de un numero primo $q = p^n$**.** Donde ***n***, es un entero positivo.

$$q = p^n$$

Número Primo

Formas de un Campo Finito:

- GF(p)
- $GF(p^n)$
- $GF(2^n)$

9.2 ARITMÉTICA MODULAR

Dados dos números enteros positivos ***n*** y ***a***, si se divide ***a*** por ***n***, se obtiene un cociente entero ***q*** y un resto entero ***r*** que obedecen a la siguiente relación:

$$\boldsymbol{a = qn + r \quad 0 \leq r < n}$$

Dos enteros ***a*** y ***b*** se dice que son congruentes módulo ***n***, si ***(a mod n) = (b mod n).*** Esto se escribe como:

$$\boldsymbol{a \equiv b(mod\ n)}$$

Ejemplo:

Un ejemplo se describe en la figura A9-1.

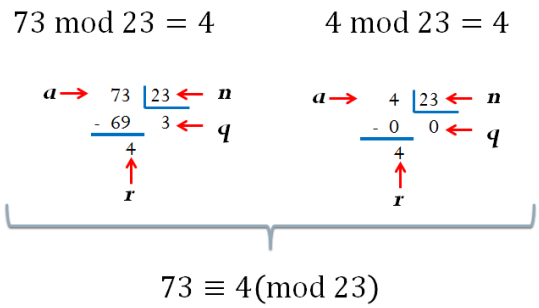


Figura A9-1. Ejemplo *a* ≡ *b(mod n)*.

Realizando la misma operación en MatLab se tiene:

```
r1 = mod(73,23)
r2 = mod(4,23)
r1 = 4
r2 = 4
```

9.2.1 PROPIEDAD DE LA CONGRUENCIA

La congruencia cumple las siguientes propiedades:

- $a \equiv b(mod\, n)$ si $n\,(a\, b)$
- $a \equiv b(mod\, n)$ implica $b \equiv a(mod\, n)$
- $a \equiv b(mod\, n)$ y $b \equiv c(mod\, n)$

  implica $a \equiv c(mod\, n)$ [A9.2]

Las propiedades de aritmética modular se indican en la tabla A9-3.

Propiedad	Expresión
Conmutativa	(w+x) mod n = (x + w) mod n (w *x* x) mod n =(x *x* w) mod n
Asociativa	[(w + x) + y] mod n = [w + (x + y)] mod n [(w ***x*** x) ***x*** y] mod n = [w ***x*** (x ***x*** y)] mod n
Distributiva	[w +(x + y)] mod n = [(w ***x*** x) +(w ***x*** y)] mod n [w +(x ***x*** y)] mod n = [(w + x) ***x*** (w + y)] mod n
Identidad	(0 + w) mod n = w mod n (1 + w) mod n = w mod n
Inverso aditivo (-w)	For each w ∈ Z_n there exists az such that w+z ≡ 0 mod n

Tabla A9-3. Propiedades aritmética modular. [A9.2]

9.3 CAMPOS FINITOS DE LA FORMA *GF(p)*

Un campo finito de la forma ***GF(p)***, es un campo cuyo orden (número de elementos en el campo), debe ser una potencia de un primo $\boldsymbol{p^n}$*;* donde ***n*** es un entero positivo. Entonces un campo finito de la forma ***GF(p)*** es cuando ***n = 1***, este campo finito posee una estructura diferente que la de los campos finitos con ***n > 1.*** [A9.2]

Para un número primo ***p***, el campo finito de orden ***p*** se define como el conjunto de enteros no negativos:

- *Zp = {0,1,2,3,…,p -1}* más las operaciones de aritmética modular módulo ***p***.

Consideraciones para que un ***GF(p)*** sea un campo finito:

- ***GF(p),*** debe ser un campo.
- Se debe demostrar la existencia de un inverso multiplicativo modulo ***p*** para cada uno de los elementos del conjunto ***Zp***, es decir debe haber un $\boldsymbol{w^{-1}}$ para todo $\boldsymbol{w_i \neq 0}$ elemento de ***Zp*** tal que $\boldsymbol{(w^{-1}) \times (w_i) = e}$, donde ***e*** es la identidad ***(1mod p)*** para ese conjunto ***Zp.*** También se puede observar la existencia de un inverso multiplicativo $\boldsymbol{w^{-1}}$ para todos los elementos ***w*** de ***Zp***, si ***w*** es un primo relativo para p, es decir al efectuar la operación multiplicación: $\boldsymbol{(w \bmod p) * (w^{-1} \bmod p)}$.
- La existencia de un elemento neutro (0), es decir que si realizamos la operación suma modulo ***p:*** $\boldsymbol{(w \bmod p) + ((-w) \bmod p)}$.

Ejemplo:

Demostrar que GF(2) es un campo finito.

- PD: ***GF(2)*** es campo:
  Entonces $\mathrm{Zp} = \{0,1\}$, **Zp** está formada de dos elementos,
  ∴ **Zp** es un grupo, anillo y campo, cumple con la primera condición.
  ∴ $\mathrm{GF}(2)$es un campo.

- PD: $\exists \mathrm{w}^{-1} \forall \mathrm{w} \in \mathrm{Zp};\ \mathrm{w} \neq 0$
  Zp= {0,1} ; p = 2, entonces aplicando operaciones de aritmética modular módulo dos.
  Multiplicando entre si todos y cada uno de los elementos de Zp.
  0 mod 2 x 0 mod 2 = 0
  0 mod 2 x 1 mod 2 = 0
  1 mod 2 x 0 mod 2 = 0
  1 mod 2 x 1 mod 2 = 1
  → 1 mod 2 x 1 mod 2 = 1
  ∴ El inverso multiplicativo de 1 mod 2, es 1.

∴ Se cumple que $\exists w^{-1} \forall w \in Zp; w \neq 0$ ya que:

0 mod 2 x 0 mod 2 = 0 nos da como resultado cero y no uno.

- PD: ∃ elemento neutro.

  0 mod 2 + 0 mod 2 = 0

  0 mod 2 + 1 mod 2 = 1

  1 mod 2 + 0 mod 2 = 1

  1 mod 2 + 1 mod 2 = 0

  → El 0 (cero) es el elemento neutro.

  ∴ ∃ elemento neutro.

  ∴ GF (2) es un campo finito de la forma GF(p).

De las operaciones suma y multiplicación realizadas anteriormente se tiene:

+	0	1
0	0	1
1	1	0

Suma

x	0	1
0	0	1
1	1	0

Multiplicación

x	-w	w^{-1}
0	0	-
1	1	1

Inversos

Se concluye que la suma en módulo 2, es equivalente a la operación XOR y la multiplicación es equivalente a la operación AND. [A9.2]

9.4 ARITMÉTICA POLINOMIAL CON COEFICIENTES ZP

Un polinomio de grado ***n***, con coeficientes enteros mayores o iguales a cero tiene la forma:

$$f(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0 = \sum_{i=0}^{n} a_i \, x^i$$

[A9.2]

Donde los a_i son los coeficientes del polinomio, y además estos coeficientes pertenecen a un campo ***F***. La división de polinomios puede representarse:

$$\frac{f(x)}{g(x)} = q(x) + \frac{r(x)}{g(x)}$$

$$f(x) = q(x)g(x) + r(x)$$

[A9.2]

Esto también puede representarse en aritmética modular como:

$$r(x) = q(x)\ mod\ g(x)$$

[A9.2]

Ejemplo:

Realizar las operaciones de aritmética modular usando los polinomios f(x), g(x) sobre GF(2) y MatLab:

$f(x) = x^6 + x^5 + x^2 + x + 1$
$g(x) = x^2 + x + 1$

- Suma

```
% Operación Suma
f = [1 1 0 0 1 1 1];
g = [0 0 0 0 1 1 1];
suma = mod(f+g,2)
suma =  1     1     0     0     0     0     0
% Respuesta en representación polinomial suma = x^6 + x^5
```

- Resta

```
% Operación Resta
f = [1 1 0 0 1 1 1];
g = [0 0 0 0 1 1 1];
resta = mod(f-g,2)
resta =  1     1     0     0     0     0     0
% Respuesta en representación polinomial resta= x^6 + x^5
```

- Multiplicación

```
% Operación Multiplicación
f = [1 1 0 0 1 1 1];
g = [0 0 0 0 1 1 1];
mult = mod(conv(f,g),2)
```

```
mult =      1     0     0     1     1     0     1     0     1
% Respuesta en representación polinomial mult= x^8 + x^5 + x^4 + x^2 + 1
```

- División

```
% Operación Multiplicación
f = [1 1 0 0 1 1 1];
g = [1 1 1];
div = mod(deconv(f,g),2)
div =    1     0     1     1     1
% Respuesta en representación polinomial mult= x^4 + x^2 + x + 1
```

El uso de MatLab, en estas operaciones es básico, pero en ejemplos posteriores, donde la resolución de los ejercicios requiere mayor complejidad y tiempo, se observará la verdadera fortaleza de esta herramienta.

9.4.1 POLINOMIO IRREDUCIBLE

Se dice que un polinomio ***f(x)*** es irreducible si y solo si ***f(x)*** no puede ser expresado como el producto de dos polinomios, un polinomio irreducible también es llamado polinomio primo.

Ejemplo:

Comprobar si el polinomio ***f(x)***, es reducible sobre ***GF(2).***

$$f(x) = x^3 + 1$$

Los factores del término independiente 1 son: -1 y +1. Al usar el factor -1; y se divide para x + 1:

```
% División f(x)/g(x).
f = [1 0 0 1];
g = [1 1];
% Almacenar cociente y  residuo en [q,r]
[q,r]=deconv(f,g);
% Convertir mod2 [q,r]
qmod = mod(q,2)
qmod = 1     1     1
rmod = mod(r,2)
rmod = 0     0     0     0
```

Se obtiene un residuo de cero y se concluye que los factores de f(x) son el cociente multiplicado g(x) como se observa a continuación:

$$x^3 + 1 = (x^2 + x + 1)(x + 1)$$
$$\therefore f(x) es\ reducible\ sobre\ GF(2)$$

Ejemplo:

Comprobar si el polinomio ***f(x)***, es irreducible sobre ***GF(2).***

$$f(x) = x^3 + x + 1$$

Los factores del término independiente 1 son: -1 y +1. Al usar el factor -1; y se divide para g(x) = x + 1:

```
% División f(x)/g(x).
f = [1 0 1 1];
g = [1 1];

% Almacenar cociente y  residuo en [q,r]
[q,r]=deconv(f,g);

% Convertir mod2 [q,r]
qmod = mod(q,2)
qmod = 1     1     0
rmod = mod(r,2)
rmod =  0     0     0     1
```

Se obtiene un residuo de uno y se concluye que f(x), no puede ser expresado como el producto de polinomios en GF(2).

$$\therefore f(x) es\ irreducible\ sobre\ GF(2)$$

9.4.2 MÁXIMO COMÚN DIVISOR DE POLINOMIOS

Se dice que el polinomio ***c(x)*** es el máximo común divisor de dos polinomios ***a(x)*** y ***b(x)*** si cumple:

- ***c(x)*** divide a los dos polinomios ***a(x)*** y ***b(x).***
- Cualquier divisor de ***a(x)*** y ***b(x)*** es un divisor de ***c(x).***

Se puede adaptar el algoritmo de Euclides para calcular el máximo común divisor de dos polinomios ***a(x)*** y ***b(x),*** de la siguiente manera:

$$\gcd[a(x), b(x)] = gcd[b(x), a(x) mod\ b(x)]$$

[A9.2]

El algoritmo de Euclides en seudo código puede ser escrito como:

$EUCLID\ [a(x), b(x)]$
1. $A(x) \leftarrow a(x);\ B(x) \leftarrow b(x)$
2. $if\ B(x) = 0\ return\ A(x) = \gcd[a(x), b(x)]$
3. $R(x) = A\ (x)\ mod\ B(x)$
4. $A(x) \leftarrow B(x)$
5. $B(x) \leftarrow R(x)$
6. goto 2

[A9.2]

Cabe señalar que el grado de ***a(x)*** es mayor que el grado de ***b(x).***

9.4 CAMPOS FINITOS DE LA FORMA *GF(2^n)*

La forma general de estos campos finitos es: $\boldsymbol{p^n}$, donde ***p*** es un número primo y ***n*** es un entero positivo. Trabajar en estos campos resulta una forma atractiva en los algoritmos de encriptación.

En lugar de adaptar las propiedades de la Aritmética Modular, lo que se hace es utilizar las propiedades de Aritmética Polinomial en la construcción un campo finito deseado.

9.4.1 ARITMÉTICA MODULAR POLINOMIAL

Considere un conjunto ***S*** de todos los polinomios de grado (n-1) o menor, cuyos coeficientes pertenecen al campo ***Zp***. Un polinomio tiene la forma:

$$f(x) = a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \cdots + a_1x + a_0 = \sum_{i=0}^{n-1} a_i x^i$$

[A9.2]

Cada a_i toma valores en el conjunto: $Z_p = \{0,1,\ldots,p-1\}$. Donde p es un número primo, por lo tanto existen número de polinomios diferentes en el conjunto S.

La aritmética modular polinomial, sigue las mismas reglas que la aritmética de polinomios con las reglas básicas del álgebra, tomando en cuenta las aclaraciones siguientes:

- Los coeficientes de los polinomios están en módulo **p**, por lo tanto se puede usar la misma aritmética que se usó en los campos finitos **Zp**.
- Si el resultado de una multiplicación, resulta en un polinomio de grado mayor que **n-1**, este polinomio debe ser reducido a un polinomio **m(x)**, donde **m(x)** es un polinomio irreducible, el residuo puede ser representado por:

$$r(x) = f(x)\ mod\ m(x)$$

[A9.2]

Para realizar operaciones de aritmética modular polinomial usaremos el campo finito y el polinomio irreducible usado por AES:

- Campo finito: GF(2^8)
- Polinomio irreducible: $m(x) = x^8 + x^4 + x^3 + x + 1$

Ejemplo:

Realizar la suma de los polinomios f(x), g(x) sobre GF(2^8) con $m(x) = x^8 + x^4 + x^3 + x + 1$, usando MatLab.

$f(x) = x^5 + x^3 + x + 1$
$g(x) = x^6 + x^2 + 1$

```
% Suma polinomial en GF(2^8) y m(x) = x^8 + x^4 + x^3 + x + 1
f=[0 1 0 1 0 1 1];
```

```
g=[1 0 0 0 1 0 1];
% Suma (f+g)
suma= mod((f+g),2)
suma =  1     1     0     1     1     1     0
```

$$f(x) + g(x)\ mod\ m(x) = x^6 + x^5 + x^3 + x^2 + x$$

Ejemplo:

Realizar la multiplicación de los polinomios f(x), g(x) sobre GF(2^8) con $m(x) = x^8 + x^4 + x^3 + x + 1$, usando MatLab.

$$f(x) = x^5 + x^3 + x + 1$$
$$g(x) = x^6 + x^2 + 1$$

```
% Multiplicación polinomial en GF(2^8) y m(x) = x^8 + x^4 + x^3 + x + 1
f=[0 1 0 1 0 1 1];
g=[1 0 0 0 1 0 1];

% Multiplicación (f x g)
mul = mod (conv(f,g),2)
mul = 0 1  0  1  0  0  1  0  0  0  1  1  1

% Expresado mul en polinomio mul(x) = x^11 + x^9 + x^7 + x^6 + x^2 + x + 1
% Reducido mul en modulo m(x)
m = [1 0 0 0 1 1 0 1 1]
[q r] = deconv(mul,m)
q = 0   1   0   1   0
r = 0   0   0   0   0  -1   0  -1  -2  -1   0   0   1

residuomod = mod(r,2)
residuomod = 0   0   0   0   0   1   0   1   0   1   0   0   1
```

$$f(x)\ x\ g(x)\ mod\ m(x) = x^7 + x^5 + x^3 + 1$$

9.5 CONSIDERACIONES COMPUTACIONALES EN *GF(2^n)*

Un polinomio f(x) en GF(2^n) :

$$f(x) = a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \cdots + a_1 x + a_0 = \sum_{i=0}^{n-1} a_i x^i$$

[A9.2]

Únicamente puede ser representado por sus coeficientes n binaria (an1 an2 . .. a0). Por lo tanto, todo polinomio en GF (2^n) Puede ser representado por n-bits.

La suma de polinomios se realiza, sumando los correspondientes coeficientes, y, en el caso de polinomios sobre Z_2, además es la operación XOR.

Así, la adición de dos polinomios f(x) y g(x) en GF (2^8) con $m(x) = x^8 + x^4 + x^3 + x + 1$, corresponde a una operación XOR bit a bit. Se puede realizar la suma como se observa en la tabla A9-4.

$f(x) = x^5 + x^3 + x + 1$
$g(x) = x^6 + x^2 + 1$

Operación	Resultado	Notación
$\boldsymbol{(x^5 + x^3 + x + 1) + (x^6 + x^2 + 1)}$	$x^6 + x^5 + x^3 + x^2 + x$	Polinomial
(0101011) ⊕ (1000101)	(0 01 1 0 1 1 1 0)	Binaria
{2B} ⊕ {45}	{6E}	Hexadecimal

Tabla A9-4. Operación suma polinomial en GF(2^8).

REFERENCIAS

[A9.1] (2012) Grupo (Matemática). [Online]. Disponible: http://es.wikipedia.org/wiki/Grupo_(matem%C3%A1tica)

[A9.2] W. Stallings, *Cryptography and Network Security Principles and Practices*. Cuarta Edición, 2005.

[A9.3] J. Daemen y V. Rijrnen, *The Design of Rijndael AES – The Advanced Encryption Standard*, Primera Edición, 2002.

[A9.4] Advanced Encryption Standard (AES), Federal Information Processing Standards Publication 197, 2001.

ANEXO 10

FUNDAMENTOS MATEMÁTICOS DE LAS FUNCIONES DE AES

10.1 CAJA – S

Para calcular los valores que comprenden la Caja-S, se toman los siguientes criterios:

- Se debe calcular el inverso multiplicativo de cada uno de los elementos del campo GF(2^8).
- Luego se aplica una transformación lineal a cada uno de los inversos multiplicativos, obtenidos en el paso anterior.

10.1.1 CÁLCULO DE INVERSO MULTIPLICATIVO DE LOS ELEMENTOS DEL CAMPO GF(2^8)

Esta operación consiste en encontrar los inversos multiplicativos de todos los elementos del campo GF(2^8). Para encontrar el inverso multiplicativo de un elemento en GF(2^8), se debe considerar el elemento identidad multiplicación que para el caso de los elementos en GF(2^8) es {01}. Por lo tanto la definición del inverso multiplicativo es la siguiente:

Para cualquier no-cero polinomio binomial b(x), de grado menor que 8, el inverso multiplicativo de b(x) se denota como $b^{-1}(x)$. Se dice que $b^{-1}(x)$ es el inverso multiplicativo si cumple con:

$$[b(x).b^{-1}(x)] mod\ m(x) = \{01\}$$

Para calcular el elemento inverso multiplicativo, se hace uso del algoritmo de Euclides extendido:

$EUCLIDES\ [m(x), b(x)]$

Paso uno:
$[A_1\ (x), A_2\ (x), A_3\ (x)] \leftarrow\ 1, 0, m(x)$

Paso dos:
$[B_1\ (x), B_2\ (x), B_3\ (x)] \leftarrow\ 0, 1, b(x)$

Paso tres:
$if\ B_3(x) = 0\ return\ A_3(x) = \gcd[m(x), b(x)]\ no\ hay\ inverso$

Paso cuatro:
$if\ B_3(x) = 1\ return\ B_3(x) = \gcd[m(x), b(x)]$

Paso cinco:
$\Longmapsto\ B_2(x) = b(x)^{-1} mod\ m(x)$
$Q(x) = quociente \frac{A_3\ (x)}{B_3\ (x)}$

Paso seis:
$[T_1\ (x), T_2\ (x), T_3\ (x)] \leftarrow A_1\ (x) xor Q(x) B_1\ (x), A_2\ (x) xor Q(x) B_2\ (x), A_3\ (x) xor Q(x) B_3\ (x)$

Paso siete:
$[A_1\ (x), A_2\ (x), A_3\ (x)] \leftarrow\ B_1\ (x), B_2\ (x), B_3\ (x)$

Paso ocho:
$[B_1\ (x), B_2\ (x), B_3\ (x)] \leftarrow\ T_1\ (x), T_2\ (x), T_3\ (x)$
Go to 2.

En donde $m(x) = x^8 + x^4 + x^3 + x + 1$, corresponde al polinomio irreducible que corresponde al estándar AES y b(x) corresponde el polinomio del cual se quiere obtener el inverso multiplicativo.

10.1.1.1 Procedimiento para calcular el inverso multiplicativo utilizando el algoritmo de Euclides extendido

Paso Uno:

En primer lugar el algoritmo nos indica que se debe asignar el valor 1, 0 y m(x) a los polinomios $[A_1\ (x), A_2\ (x), A_3\ (x)]$, respectivamente. Luego se debe asignar el valor de 0,1 y b(x) a los polinomios $[B_1\ (x), B_2\ (x), B_3\ (x)]$, respectivamente.

Paso Dos:

En este paso se debe verificar si $B_3(x) = 0$, en caso de ser cierto se retorna el valor $A_3(x)$, y se dice que no existe elemento inverso de b(x).

Paso Tres:

En el caso en el que $B_3(x) = 1$, se retorna como máximo común divisor de m(x) y b(x) el valor de $B_3(x)$ y como valor de inverso multiplicativo de b(x) el valor $B_2\ (x)$.

Paso Cuatro:

En caso de que el valor $\mathrm{B}_3(\mathrm{x}) \neq 1 \wedge \mathrm{B}_3(\mathrm{x}) \neq 0$, se debe obtener el quociente de dividir el polinomio $A_3(x)$ para el polinomio $B_3(x)$:

$$Q(x) = quociente \frac{A_3\ (x)}{B_3\ (x)}$$

Paso Cinco:

En este paso el algoritmo indica que se asigne los valores: $A_1\ (x) xor Q(x) B_1\ (x), A_2\ (x) xor Q(x) B_2\ (x), A_3\ (x) xor Q(x) B_3\ (x)$, a los polinomios tupla $[T_1\ (x), T_2\ (x), T_3\ (x)]$, respectivamente.

Paso Seis:

Se debe actualizar los valores de $A_1\ (x), A_2\ (x), A_3\ (x)$ con los valores de $B_1\ (x), B_2\ (x), B_3\ (x)$, respectivamente.

Paso Siete:

De igual forma que en el paso anterior se debe actualizar los valores de $B_1\ (x), B_2\ (x), B_3\ (x)$, con los valores de $T_1\ (x), T_2\ (x), T_3\ (x)$.

Paso Ocho:

Por último se debe ir al paso dos, luego al paso tres y verificar si $B_3(x) = 1$, en cuyo caso se retornará como valor de inverso multiplicativo de b(x) el valor $B_2\ (x)$. En caso de que $\mathrm{B}_3(\mathrm{x}) \neq 1 \wedge \mathrm{B}_3(\mathrm{x}) \neq 0$, se deberá realizar las iteraciones necesarias hasta encontrar que $B_3(x) = 1$.

El resultado de estas operaciones se observa en la tabla A10-1.

		Y														
		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**a**	**b**	**c**	**d**	**e**	**f**
X	**0**	00	01	8d	f6	cb	52	7b	d1	e8	4f	29	c0	b0	e1	e5	c7
	1	74	b4	aa	4b	99	2b	60	5f	58	3f	fd	cc	ff	40	ee	b2
	2	3a	6e	5a	f1	55	4d	a8	c9	c1	0a	98	15	30	44	a2	c2
	3	2c	45	92	6c	f3	39	66	42	f2	35	20	6f	77	bb	59	19
	4	1d	**fe**	37	67	2d	31	f5	69	a7	64	ab	13	54	25	e9	09
	5	ed	5c	05	ca	4c	24	87	Bf	18	3e	22	f0	51	ec	61	17
	6	16	5e	af	d3	49	a6	36	43	f4	47	91	df	33	93	21	3b
	7	79	b7	85	10	b5	Ba	3c	b6	70	d0	d0	06	a1	fa	81	82
	8	83	7e	7f	80	96	73	be	56	9b	9e	95	d9	f7	02	b9	a4
	9	de	6a	32	6d	d8	8ª	84	72	2a	14	9f	88	f9	dc	89	9a
	a	fb	7c	2e	c3	8f	b8	65	48	26	c8	12	4a	ce	e7	d2	62
	b	0c	e0	1f	ef	11	75	78	71	a5	8e	76	3d	bd	bc	86	57
	c	0b	28	2f	a3	da	d4	e4	0f	a9	27	53	04	1b	fc	ac	e6
	d	7a	07	ae	63	c5	Db	e2	Ea	94	8b	c4	d5	9d	f8	90	6b
	e	b1	0d	d6	eb	c6	0e	cf	Ad	08	4e	d7	e3	5d	50	1e	b3
	f	5b	23	38	34	68	46	03	8c	dd	9c	7d	a0	cd	1a	41	1c

Tabla A10-1. Inversos multiplicativos en GF(2^8).

Ejemplo:

Se realizará un ejemplo práctico para obtener el elemento inverso multiplicativo de un elemento de GF(2^8), se tomará como valor en ASCII al carácter A.

Representación hexadecimal: 41

Representación polinomial: $b(x) = x^6 + 1$

Para encontrar el elemento inverso multiplicativo se usará Matlab y el algoritmo de Euclides extendido:

$EUCLID\ [m(x), b(x)]$

1. $[A_1\ (x), A_2\ (x), A_3\ (x)] \leftarrow\ 1, 0, m(x)$
2. $[B_1\ (x), B_2\ (x), B_3\ (x)] \leftarrow\ 0, 1, b(x)$

3. $if\ B_3(x) = 0\ return\ A_3(x) = \gcd[m(x), b(x)]\ no\ hay\ inverso$
4. $if\ B_3(x) = 1\ return\ B_3(x) = \gcd[m(x), b(x)]$
5. $B_2(x) = b(x)^{-1} mod\ m(x)$
6. $Q(x) = quociente \frac{A_3\ (x)}{B_3\ (x)}$
7. $[T_1\ (x), T_2\ (x), T_3\ (x)] \leftarrow A_1\ (x) xor Q(x) B_1\ (x), A_2\ (x) xor Q(x) B_2\ (x), A_3\ (x) xor Q(x) B_3\ (x)$
8. $[A_1\ (x), A_2\ (x), A_3\ (x)] \leftarrow\ B_1\ (x), B_2\ (x), B_3\ (x)$
9. $[B_1\ (x), B_2\ (x), B_3\ (x)] \leftarrow\ T_1\ (x), T_2\ (x), T_3\ (x)$

Go to 2.

[A10-1]

Iteración 1.

1. $[A_1\ (x), A_2\ (x), A_3\ (x)] \leftarrow\ 1, 0, x^8 + x^4 + x^3 + x + 1$

$[B_1\ (x), B_2\ (x), B_3\ (x)] \leftarrow\ 0, 1, x^6 + 1$

2. $B_3(x) \neq 0$

3. $B_3(x) \neq 1$

4. $Q(x) = quociente \frac{A_3\ (x)}{B_3\ (x)} = \frac{x^8 + x^4 + x^3 + x + 1}{x^6 + 1} = x^2$
5. $[T_1\ (x), T_2\ (x), T_3\ (x)] \leftarrow 1, x^2, x^4 + x^3 + x^2 + x + 1$
6. $[A_1\ (x), A_2\ (x), A_3\ (x)] \leftarrow\ 0, 1, x^6 + 1$
7. $[B_1\ (x), B_2\ (x), B_3\ (x)] \leftarrow\ 1, x^2, x^4 + x^3 + x^2 + x + 1$
8. *Go to 2.*

Iteración 2.

2. $B_3(x) \neq 0$

3. $B_3(x) \neq 1$

4. $Q(x) = \frac{x^6 + 1}{x^4 + x^3 + x^2 + x + 1} = x^2 + x$
5. $[T_1\ (x), T_2\ (x), T_3\ (x)] \leftarrow x^2 + x,\ x^4 + x^3 + 1, x + 1$
6. $[A_1\ (x), A_2\ (x), A_3\ (x)] \leftarrow\ 1, x^2, x^4 + x^3 + x^2 + x + 1$
7. $[B_1\ (x), B_2\ (x), B_3\ (x)] \leftarrow x^2 + x\ ,\ x^4 + x^3 + 1, x + 1$
8. *Go to 2.*

Iteración 3.

2. $B_3(x) \neq 0$

3. $B_3(x) \neq 1$

4. $Q(x) = \frac{x^4 + x^3 + x^2 + x + 1}{x + 1} = x^3 + x$

5. $[T_1\ (x), T_2\ (x), T_3\ (x)] \leftarrow x^5 + x^4 + x^3 + x^2 + 1, x^7 + x^6 + x^5 + x^4 + x^3 + x^2 + x, 1$
6. $[A_1\ (x), A_2\ (x), A_3\ (x)] \leftarrow\ x^2 + x, x^4 + x^3 + 1\ , \mathrm{x} + 1$
7. $[B_1\ (x), B_2\ (x), B_3\ (x)]$

$$\leftarrow x^5 + x^4 + x^3 + x^2 + 1\ , x^7 + x^6 + x^5 + x^4 + x^3 + x^2 + x, 1$$

8 Go to 2.

Iteración 4.

2. $B_3(x) \neq 0$
3. $B_3(x) = 1$

$B_2(x) = b(x)^{-1} mod\ m(x)$
$x^7 + x^6 + x^5 + x^4 + x^3 + x^2 + x = (x^6 + 1)'\ \ mod\ x^8 + x^4 + x^3 + x + 1$
$111110001 = (01000001)'\ \ mod\ 100011011$

$$\mathbf{FE = (41)'\ \ mod\ 100011011}$$

Por lo tanto se concluye que el elemento inverso multiplicativo de 41 es FE.

10.1.2 TRANSFORMACIÓN LINEAL DE LOS ELEMENTOS INVERSOS MULTIPLICATIVOS.

La transformación lineal se realiza sobre los elementos de la tabla A10-1, y cumple la regla:

$$b_i{}' = b_i \oplus b_{(i+4)mod8} \oplus b_{(i+5)mod8} \oplus b_{(i+6)mod8} \oplus b_{(i+7)mod8} \oplus c_i$$
$$0 \leq i < 8, c = 01100011 = 63_x$$

Representación en bits

$b_0{}' = b_0 \oplus b_4 \oplus b_5 \oplus b_6 \oplus b_7 \oplus c_0$
$b_1{}' = b_1 \oplus b_5 \oplus b_6 \oplus b_7 \oplus b_0 \oplus c_1$
$b_2{}' = b_2 \oplus b_6 \oplus b_7 \oplus b_0 \oplus b_1 \oplus c_2$
$b_3{}' = b_3 \oplus b_7 \oplus b_0 \oplus b_1 \oplus b_2 \oplus c_3$
$b_4{}' = b_4 \oplus b_0 \oplus b_1 \oplus b_2 \oplus b_3 \oplus c_4$
$b_5{}' = b_5 \oplus b_1 \oplus b_2 \oplus b_3 \oplus b_4 \oplus c_5$
$b_6{}' = b_6 \oplus b_2 \oplus b_3 \oplus b_4 \oplus b_5 \oplus c_6$
$b_7{}' = b_7 \oplus b_3 \oplus b_4 \oplus b_5 \oplus b_6 \oplus c_7$

Representación matricial

$$\begin{bmatrix} b_0' \\ b_1' \\ b_2' \\ b_3' \\ b_4' \\ b_5' \\ b_6' \\ b_7' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

Ejemplo:

Se realizará un ejemplo práctico para demostrar como funciona la transformación lineal sobre los inversos multiplicativos.

Representación hexadecimal: FE
Representación binaria: $b(x) = 11111110$

$b_0' = 0 \oplus 1 \oplus 1 \oplus 1 \oplus 1 \oplus 1 = 1$
$b_1' = 1 \oplus 1 \oplus 1 \oplus 1 \oplus 0 \oplus 1 = 1$
$b_2' = 1 \oplus 1 \oplus 1 \oplus 0 \oplus 1 \oplus 0 = 0$
$b_3' = 1 \oplus 1 \oplus 0 \oplus 1 \oplus 1 \oplus 0 = 0$
$b_4' = 1 \oplus 0 \oplus 1 \oplus 1 \oplus 1 \oplus 0 = 0$
$b_5' = 1 \oplus 1 \oplus 1 \oplus 1 \oplus 1 \oplus 1 = 0$
$b_6' = 1 \oplus 1 \oplus 1 \oplus 1 \oplus 1 \oplus 1 = 0$
$b_7' = 1 \oplus 1 \oplus 1 \oplus 1 \oplus 1 \oplus 0 = 1$

$$\mathbf{b_i' = 10000011 = 83}$$

Por lo tanto se concluye que el elemento de la caja-S que corresponde al valor hexadecimal 41 es 83.

Y el resultado de encontrar todos los elementos de la caja-S se observa en la tabla A10-2.

		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	01	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	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
	4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
	5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
	6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	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	0b	db
	a	e0	32	3a	0a	49	06	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	08
	c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
	d	70	3e	b5	66	48	03	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	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Tabla A10- 2. Caja-S. [A10.2]

10.2 RCON [i/Nk]

Contiene los valores dados por $[x^{i-1}, \{00\}, \{00\}, \{00\}]$ donde (x es denotada como {02}) en el campo GF(2^8), i representa el número de ronda, en el caso de AES son 10 rondas.

$$x^{1-1} = x^0 mod(x^8 + x^4 + x^3 + x + 1) \rightarrow 00000001 \rightarrow \{01\} \rightarrow \{01\}, \{00\}, \{00\}, \{00\}$$
$$x^{2-1} = x^1 mod(x^8 + x^4 + x^3 + x + 1) \rightarrow 00000010 \rightarrow \{02\} \rightarrow \{02\}, \{00\}, \{00\}, \{00\}$$
$$x^{3-1} = x^2 mod(x^8 + x^4 + x^3 + x + 1) \rightarrow 00000100 \rightarrow \{04\} \rightarrow \{04\}, \{00\}, \{00\}, \{00\}$$
$$x^{4-1} = x^3 mod(x^8 + x^4 + x^3 + x + 1) \rightarrow 00001000 \rightarrow \{08\} \rightarrow \{08\}, \{00\}, \{00\}, \{00\}$$
$$x^{5-1} = x^4 mod x^8 + (x^4 + x^3 + x + 1) \rightarrow 00010000 \rightarrow \{10\} \rightarrow \{10\}, \{00\}, \{00\}, \{00\}$$
$$x^{6-1} = x^5 mod(x^8 + x^4 + x^3 + x + 1) \rightarrow 00100000 \rightarrow \{20\} \rightarrow \{20\}, \{00\}, \{00\}, \{00\}$$
$$x^{7-1} = x^6 mod(x^8 + x^4 + x^3 + x + 1) \rightarrow 01000000 \rightarrow \{40\} \rightarrow \{40\}, \{00\}, \{00\}, \{00\}$$
$$x^{8-1} = x^7 mod(x^8 + x^4 + x^3 + x + 1) \rightarrow 10000000 \rightarrow \{80\} \rightarrow \{80\}, \{00\}, \{00\}, \{00\}$$
$$x^{9-1} = x^8 mod(x^8 + x^4 + x^3 + x + 1) \rightarrow 00011011 \rightarrow \{1b\} \rightarrow \{1b\}, \{00\}, \{00\}, \{00\}$$
$$x^{10-1} = x^9 mod(x^8 + x^4 + x^3 + x + 1) \rightarrow 00110110 \rightarrow \{36\} \rightarrow \{36\}, \{00\}, \{00\}, \{00\}$$

10.3 MEZCLA DE COLUMNAS

Consiste en multiplicar cada columna de la matriz de entrada, por una matriz constante que se obtiene en GF(2^8) [X]/(X^2+1).

10.3.1 REPRESENTACIÓN MATRICIAL

Las columnas se consideran polinomios sobre GF(2^8), y se multiplican módulo mx= x^4 + 1, con un polinomio fijo cx = $\{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}$

Dado:

$$b(x) = b_0 + b_1x + b_2x^2 + b_3x^3$$
$$c(x) = c_0 + c_1x + c_2x^2 + c_3x^3$$
$$d(x) = b * c\ [mod(x^4 + 1)]$$

$$d(x) = b_0c_0 + b_0c_1x + b_0c_2x^2 + b_0c_3x^3 + b_1xc_0 + b_1xc_1x + b_1xc_2x^2 + b_1xc_3x^3 + b_2x^2c_0 + b_2x^2c_1x + b_2x^2c_2x^2 + b_3x^2c_3x^3 + b_3x^3c_0 + b_3x^3c_1x + b_3x^3c_2x^2 + b_3x^3c_3x^3$$

$$\begin{aligned} d(x) = \quad & b_0c_0 + b_0c_1x + b_0c_2x^2 + b_0c_3x^3 + c_0x + b_1c_1x^2 + b_1c_2x^3 + b_1c_3x^4 + \\ & b_2c_0x^2 + b_2c_1x^3 + b_2c_2x^4 + b_3c_3x^5 + b_3c_0x^3 + b_3c_1x^4 + b_3c_2x^5 + b_3c_3x^6 \end{aligned}$$

Los polinomios x^4, x^5, x^6 se reducen a modulo (x^4+1), donde:

$x^4 mod\ (x^4+1)=1$
$x^5 mod\ (x^4+1)=x$
$x^6 mod\ (x^4+1)=x^2$

$$\begin{aligned} d(x) = \quad & b_0c_0 + b_0c_1x + b_0c_2x^2 + b_0c_3x^3 + \\ & b_1c_0x + b_1c_1x^2 + b_1c_2x^3 + b_1c_3 + \\ & b_2c_0x^2 + b_2c_1x^3 + b_2c_2 + b_3c_3x + \\ & b_3c_0x^3 + b_3c_1 + b_3c_2x + b_3c_3x^2 \end{aligned}$$

Se remplaza los valores de: c(x) = {03}x^3+ {01}x^2+ {01}x+{02}

$$\begin{aligned} d(x) = \quad & b_0(02) + b_1(03) + b_2(01) + b_3(01) \\ & b_0(01) + b_1(02) + b_2(03) + b_3(01) \\ & b_0(01) + b_1(01) + b_2(02) + b_3(02) \\ & b_0(03) + b_1(01) + b_2(01) + b_3(02) \end{aligned}$$

$$\begin{bmatrix} d1 \\ d2 \\ d3 \\ d4 \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 02 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} bo \\ b1 \\ b2 \\ b3 \end{bmatrix}$$

Remplazando los valores de *d* y *b,* se tiene la representación matricial:

$$d_n = a'_n$$

$$b_n = a_n$$

$$\begin{bmatrix} a'_0 \\ a'_1 \\ a'_2 \\ a'_3 \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 02 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} ao \\ a1 \\ a2 \\ a3 \end{bmatrix}$$

El resultado de la multiplicación matricial es:

$a'_0 = (\{02\} \cdot a_0) \oplus (\{03\} \cdot a_1) \oplus (a_2) \oplus (a_3)$
$a'_1 = (a_0) \oplus (\{02\} \cdot a_1) \oplus (\{03\} \cdot a_2) \oplus (a_3)$
$a'_2 = (a_0) \oplus (a_1) \oplus (\{02\} \cdot a_2) \oplus (\{03\} \cdot a_3)$
$a'_3 = (\{03\} \cdot a_0) \oplus (a_1) \oplus (a_2) \oplus (\{02\} \cdot a_3)$

10.4 MEZCLA DE COLUMNAS INVERSA

Consiste en multiplicar cada columna de la matriz de entrada, por una columna constante que se obtiene en GF(2^8) [X]/(X^4+1) con un polinomio fijo cx = $\{0b\}x^3 + \{0d\}x^2 + \{09\}x + \{0e\}$.

Una representación gráfica se indica en la figura A10-1.

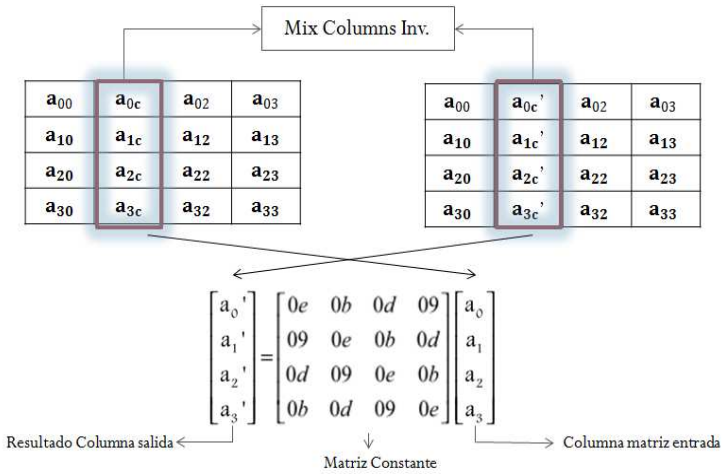


Figura A10-1. Mezcla de columnas inversa.

El resultado de la multiplicación matricial es:

$a'_0 = (\{0e\} \cdot a_0) \oplus (\{0b\} \cdot a_1) \oplus (\{0d\} \cdot a_2) \oplus (\{09\} \cdot a_3)$
$a'_1 = (\{09\} \cdot a_0) \oplus (\{0e\} \cdot a_1) \oplus (\{0b\} \cdot a_2) \oplus (\{0d\} \cdot a_3)$
$a'_2 = (\{0d\} \cdot a_0) \oplus (\{09\} \cdot a_1) \oplus (\{0e\} \cdot a_2) \oplus (\{0b\} \cdot a_3)$
$a'_3 = (\{0b\} \cdot a_0) \oplus (\{0d\} \cdot a_1) \oplus (\{09\} \cdot a_2) \oplus (\{0e\} \cdot a_3)$

REFERENCIAS

[A10.1] W. Stallings, *Cryptography and Network Security Principles and Practices*. Cuarta Edición, 2005.

[A10.2] Advanced Encryption Standard (AES), Federal Information Processing Standards Publication 197, 2001.

ANEXO 11

CÓDIGO MATLAB DEL ALGORITMO DE CIFRADO AES

(Se encuentra en digital.)

ANEXO 12

CÓDIGO VHDL DEL ALGORITMO DE CIFRADO AES

(Se encuentra en digital.)

ANEXO 13

DOCUMENTO UG247, DE ASIGNACIÓN DE LOS PUERTOS DEL DISPOSITIVO FPGA CORRESPONDIENTES A XC5VLX110T

(Se encuentra en digital.)