ESCUELA SUPERIOR POLITECNICA DEL LITORAL
FACULTAD DE INGENIERÍA EN ELECTRICIDAD Y COMPUTACIÓN

"IMPLEMENTACIÓN HARDWARE DEL ESTANDAR DE ENCRIPTACIÓN AVANZADO (AES) EN UNA FPGA"

INFORME DE PROYECTO DE GRADUACIÓN

Previo a la obtención del título de:

INGENIERO EN CIENCIAS COMPUTACIONALES ESPECIALIZACIÓN SISTEMAS TECNOLÓGICOS

PRESENTADA POR:

CELI MENDEZ JORGE ALBERTO

GUAYAQUIL – ECUADOR

2012

AGRADECIMIENTO

A Dios autor de mi existencia por permitirme culminar con éxito este sueño, a mis padres por su apoyo incondicional y desmedido, un agradecimiento especial a mis hermanos que contribuyeron para que todo esto fuera posible.

DEDICATORIA

A mis Padres

a mi madre adorada, por su dedicación, valentía y esfuerzo por sus hijos, a mi padre por sus consejos y preocupación, a mi hijo Derian David fuente de inspiración para la culminación de este trabajo, a mis hermanos, a mis hermanos políticos, a mis sobrinos, a mis familiares y amigos.

DECLARACIÓN EXPRESA

"La responsabilidad del contenido de este proyecto de graduación me corresponden exclusivamente, y el patrimonio intelectual de la misma a la ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL" (Reglamento de Graduación de la ESPOL)

_____

Jorge Alberto Celi Mendez

TRIBUNAL DE SUSTENTACION

Ing. Sara Rios O.

Subdecana de la FIEC

Ing. Ronald Ponguillo

Director del Proyecto

Dra. Katherine Chiluiza G.

Tribunal

RESUMEN

En la seguridad informática, la protección de la información tiene una gran importancia por lo cual se hace necesario el uso de técnicas que nos permitan en alguna medida asegurar que la información mantenga su integridad y confidencialidad en la transmisión y almacenamiento, esta técnicas básicas que se necesitan para proteger la información las provee la criptografía.

El presente proyecto se enfoca en el diseño de la arquitectura del algoritmo de encriptación avanzada AES-Rijndael haciendo uso de la tecnología de arreglos de puertas programables por campos (FPGA), con lenguaje de descripción de hardware (VHDL), para lo cual se utilizó una FPGA Cyclone II y la herramienta Quartus II de Altera, en la cual se sintetizó y simuló la arquitectura diseñada.

Esta implementación se centra en el proceso de cifrado, soportando bloques de de 128 bits tanto para los datos como para la clave, los datos son agrupados sobre una matriz de bytes que contiene 4 filas y 4 columnas estos bytes representan elementos de un Campo Finito GF(2^8) o Campo de Galois, las operaciones utilizadas en el algoritmo Rijndael las mismas que son llevadas a cabo en una serie de iteraciones o también llamadas rondas que son operaciones de cambios de posición de los bytes y operaciones

sobre el campo finito GF(2^8). El número de rondas dependerá del tamaño del bloque de datos y de la longitud de la clave, para el presente caso en el cual se utiliza un bloque de datos y clave de 128 bits se utilizaran 10 rondas.

Para la comprobación de proceso de encriptado se hará uso de una pequeña aplicación desarrollada en lenguaje Java la cual permitirá adquirir la información que ha sido sometida al proceso de encriptado, y enviar a la tarjeta el texto que se desea encriptar, la comunicación entre la aplicación y la tarjeta DE2 se la realizara mediante puerto serial a través de un modulo que maneja la comunicación RS-232 con una tasa de transferencia de 19200 b/s.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ABREVIATURAS

AES: Advance Encryption Standar.

Estándar de Encriptación Avanzado.

FPGA: Field Programmable Gate Array.

Arreglos de Puertas Programable por Campos.

NIST: National Institute of Standards and Technology.

Instituto Nacional de Estándares y Tecnología.

TDES: Triple Data Encryption Standar.

Triple Estándar de Encriptación de Datos.

VHDL: Very high speed integrated circuit (VHSIC) Hardware Description Language.

Lenguaje de Descripción de Hardware Para Circuitos Integrados de Muy Alta Velocidad.

RS232: Interfaz para el intercambio de datos binarios.

UART: Universal Asynchronous Receiver-Transmitter Protocol.

Protocolo Trasmisor-Receptor Asíncrono Universal.

ECB: Electronic Codebook.

CBC: Cipher Block Chaining.

CFB: Cipher Feedback.

OFB: Output Feedback.

INTRODUCCIÓN

En la actualidad en medios como el internet viaja gran cantidad de información la cual muchas veces es información de tipo sensible que solo debería ser conocida por las persona a las cuales está destinada, esto hace que cada vez sea más necesario el uso de técnicas de encriptado y desencriptado, las mismas que deberían ser implementadas en sistemas que permitan realizar estas tareas en el menor tiempo posible por cuanto existe un gran volumen de información a procesar.

El presente proyecto pretende obtener las ventajas que brinda el hardware con respecto al software en la implementación de algoritmos criptográficos, para lo cual se ha desarrollado una arquitectura en una FPGA de Altera para el proceso de cifrado usando AES, que es el actual estándar de cifrado simétrico dispuesto por el Instituto Nacional de Estándares y Tecnología (NIST) en remplazo del TDES, el AES es uno de los algoritmos más populares usados en criptografía simétrica.

AES es rápido tanto en software como en hardware, es relativamente fácil de implementar, y requiere poca memoria. Como nuevo estándar de cifrado, se está utilizando actualmente a gran escala.

CAPITULO 1

1. CONCEPTOS GENERALES DE CRIPTOGRAFIA

1.1 Criptografía clásica.

El termino criptografía proviene del griego “Kryptos”, escondido, y “Graphos”, escritura para lo cual se puede traducir literalmente como “escritura oculta” se trata de utilizar técnicas matemáticas para escribir mensajes de tal forma que éste solo pueda ser interpretado por la persona a cual está dirigido.

En la actualidad en donde el poseer el conocimiento, “información” es sinónimo de poder, la criptografía es la herramienta necesaria para proteger la información teniendo como principales características: Garantizar la integridad de la comunicación entre el emisor y receptor del mensaje (personas, organizaciones, etc.) que se llevan a cabo sobre canales inseguros y, en segundo lugar, asegurar la autenticidad de la información en donde el remitente sea quien dice ser y que el mensaje enviado no haya sido modificado en su camino.

Las técnicas criptográficas se pueden clasificar en dos según el tipo de la clave utilizada, las cuales son: Criptografía simétrica o de clave secreta y Criptografía asimétrica o de clave pública.

En el presente trabajo nos enfocaremos en la criptografía simétrica ya que el algoritmo AES se encuentra dentro de este tipo de cifrado.

1.2 Criptografía de clave simétrica.

Es el tipo de criptografía en la cual tanto el emisor como el receptor deben conocer la clave, la que les permitirá encriptar y desencriptar mensajes.

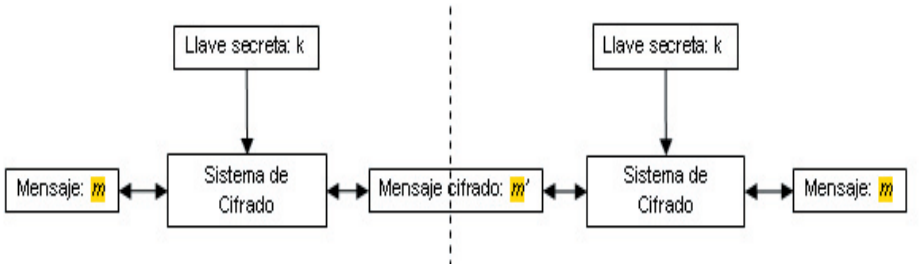


Figura 1- 1 : Criptografía Simétrica. [10]

Este tipo de criptografía garantiza la confidencialidad dado que quien posee la clave será capaz de ver el mensaje, los algoritmos de criptografía simétrica se basan en operaciones algebraicas muy sencillas, lo cual hace que el proceso de cifrado sea muy pequeño. Esta característica hace que estos algoritmos sean tomados en cuenta en aplicación en las que se requiere gran velocidad de flujo de datos.

Los algoritmos simétricos se sustentan en los conceptos de Confusión y Difusión emitidos por Claude Shannon.

Confusión tiene por objetivo evitar el descubrimiento de la clave para lo cual establece una relación lo más compleja posible entre la clave y las estadísticas del texto cifrado y se consigue mediante la sustitución de unos símbolos por otros.

Difusión tiene como objetivo establecer una relación lo más compleja posible entre el cifrado y el texto plano, lo cual se consigue mediante la permutación de bits.

1.2.1 Criptografía Simétrica por Bloques.

Este tipo de criptografía se caracteriza por agrupar el mensaje en bloques de n bits cada uno, antes de aplicar el proceso de encriptado a cada uno de los bloques con la misma clave.

Existen cuatro modos de operación en el cifrado por bloques que pueden ser aplicados a cualquier algoritmo tipo bloque, los cuales son:

- ECB: Se cifran los bloques por separado usando la misma clave

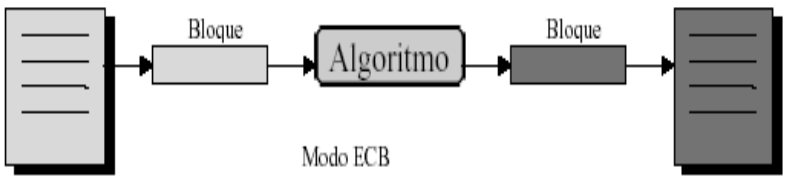


Figura 1- 2: Modo ECB. [10]

- CBC: Cada boque de entrada al algoritmo de cifrado se relaciona con operaciones XOR con la salida del bloque anterior.

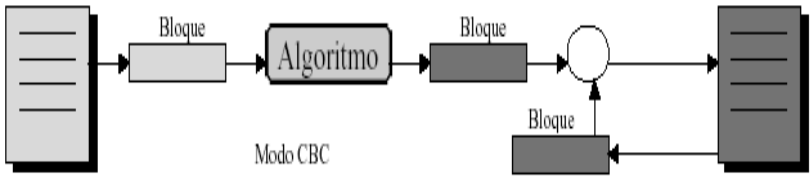


Figura 1- 3: Modo CBC. [10]

- CFB: Realiza una XOR entre caracteres o bit del texto y las salidas del algoritmo.

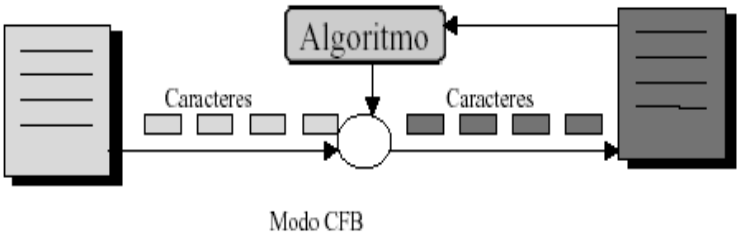


Figura 1- 4: Modo CFB. [10]

- OFB: Como el CFB, realiza una XOR entre caracteres o bit aislados del texto y las salidas del algoritmo.

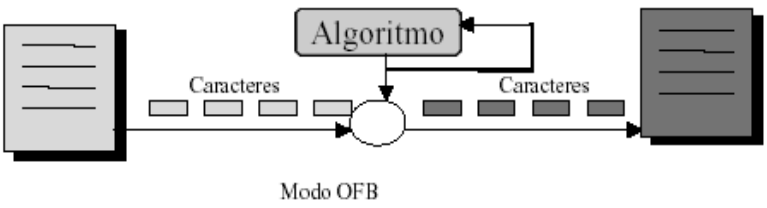


Figura 1- 5: Modo OFB. [10]

1.2.2 Criptografía Simétrica por Flujo.

Consiste en un generador de clave cuya secuencia aleatoria de salida se somete a una operación binaria (OR-exclusiva) con los bits del mensaje, para formar la secuencia de bits de salida.

Este cifrado considera el mensaje como un flujo de bit y genera a la salida el correspondiente flujo de bits resultante de la transformación en el proceso de cifrado.

1.3 Algoritmo RIJNDAEL-AES.

El algoritmo AES o también conocido como Rijndael el cual es un algoritmo de cifrado por bloques, que tiene como principal característica soportas claves de 128, 192 y 256 bits y bloques de datos de 128 bits. Este algoritmo fue aceptado como estándar, después de una convocatoria que realizó el NIST en 1997 y luego de un proceso de estandarización fue seleccionado en remplazo del su predecesor DES.

“Rijndael” es un acrónimo formado por la contracción de los nombres de los criptógrafos belgas que desarrollaron el algoritmo Joan Daemen y Vincent Rijmen.[1]

Los criterios bajos los cuales fue desarrollado el algoritmo son los siguientes:

- Resistencia a todos los ataques
- Posibilidad de ser implementado en hardware.

- Velocidad, la cual está relacionada a las complejas operaciones que realiza.
- Seguridad física, la cual es completamente alta si el algoritmo se encuentra desarrollado en un chip.
- Re-usos de componentes, dado que podrán existir ciertos componentes que sean utilizados tanto en el cifrado como en el descifrado.

CAPITULO 2

2. ESTRUCTURA DEL ALGORITMO RIJNDAEL – AES

Cada una de las operaciones del algoritmo AES se las realiza en un campo finito $GF(2^8)$, en el que los operandos son considerados polinomios dentro del campo, cada elemento del campo puede ser considerado como una cadena de 8 bits ya sea en su representación binaria o hexadecimal o como un polinomio de grado 7 o menor en el cual sus coeficientes sean {0,1}. En este caso los coeficientes del polinomio son los respectivos de la representación en binario de un elemento. Por ejemplo {03} en hexadecimal es equivalente a {0000 0011} en binario y se puede representar como el siguiente polinomio:

$$c(x) = 0 \cdot x^7 + 0 \cdot x^6 + 0 \cdot x^5 + 0 \cdot x^4 + 0 \cdot x^3 + 0 \cdot x^2 + 1 \cdot x + 1 \cdot 1 = x + 1 \quad (1)$$

La multiplicación de elementos de $GF(2^8)$ se encuentra determinada por la multiplicación de los correspondientes polinomios módulo por un polinomio irreducible de grado 8 que se denomina $m(x)$ y se presenta en la Ec.(2):

$$m(x) = x^8 + x^4 + x^3 + x + 1 \quad (2)$$

Por ejemplo para multiplicar un elemento variable $a = a7a6a5a4a3a2a1a0$ por un elemento constante {03} es equivalente a calcular

$$b(x) = b7x^7 + b6x^6 + b5x^5 + b4x^4 + b3x^3 + b2x^3 + b1x + b0 =$$

$$= (a7x^7 + a6x^6 + a5x^5 + a4x^4 + a3x^3 + a2x^2 + a1x + a0) \cdot (x + 1)$$

$$mod(x^8 + x^4 + x^3 + x + 1) \quad (3)$$

$$b(x) = (a7 + a6) \cdot x^7 + (a6 + a5) \cdot x^6 + (a5 + a4) \cdot x^5 + (a7 + a4 + a3) \cdot x^4 + (a7 + a3 + a2) \cdot x^3 + (a2 + a1) \cdot x^2 + (a7 + a1 + a0) \cdot x + (a7 + a0) \quad (4)$$

Cada bit de a producto b, puede ser representado con una función XOR de al menos tres bits, ejemplo, $b7 = (a7 + a6),\ b4 = (a4 + a3 + a7)$, etc.

En cada ronda de transformación correspondiente al algoritmo AES, se realizan operaciones sobre una matriz rectangular de bytes de dimensión 4 filas y N_b columnas, denominada *matriz de estado* en la cual se encuentran representado el texto a ser cifrado. Siendo N_b el número de columnas en función del tamaño del bloque:

$$N_b = tamaño\ del\ bloque\ utilizado\ en\ bits/32$$

De la misma forma la clave de cifrado es representada con una estructura igual a la *matriz de estado*, es decir mediante una matriz rectangular de bytes de 4 filas y N_k columnas. Siendo N_k el número de columnas en función del tamaño de la clave:

$$N_k = tamaño\ de\ la\ clave\ en\ bits/32$$

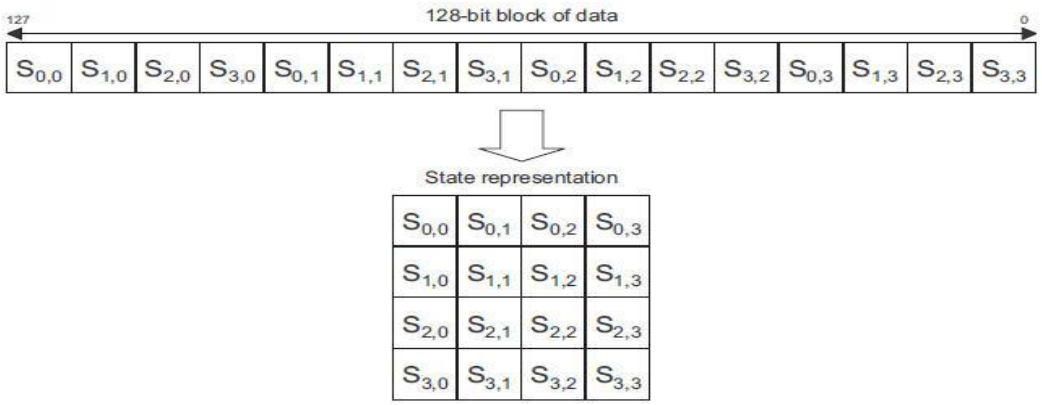


Figura 2- 1: Representación Matriz de Estado. [9]

AES realiza un cifrado iterativo, esto consiste en aplicar de forma repetida rondas de transformación a la matriz de estado, en este caso en el que bloque de datos es de 128 bits, el número de rondas N_r definido por los diseñadores es de 10 rondas, determinado por la siguiente expresión:

$$N_r = max(N_k, N_b) + 6$$

La figura 2.2 ilustra esquema del proceso de cifrado con AES.

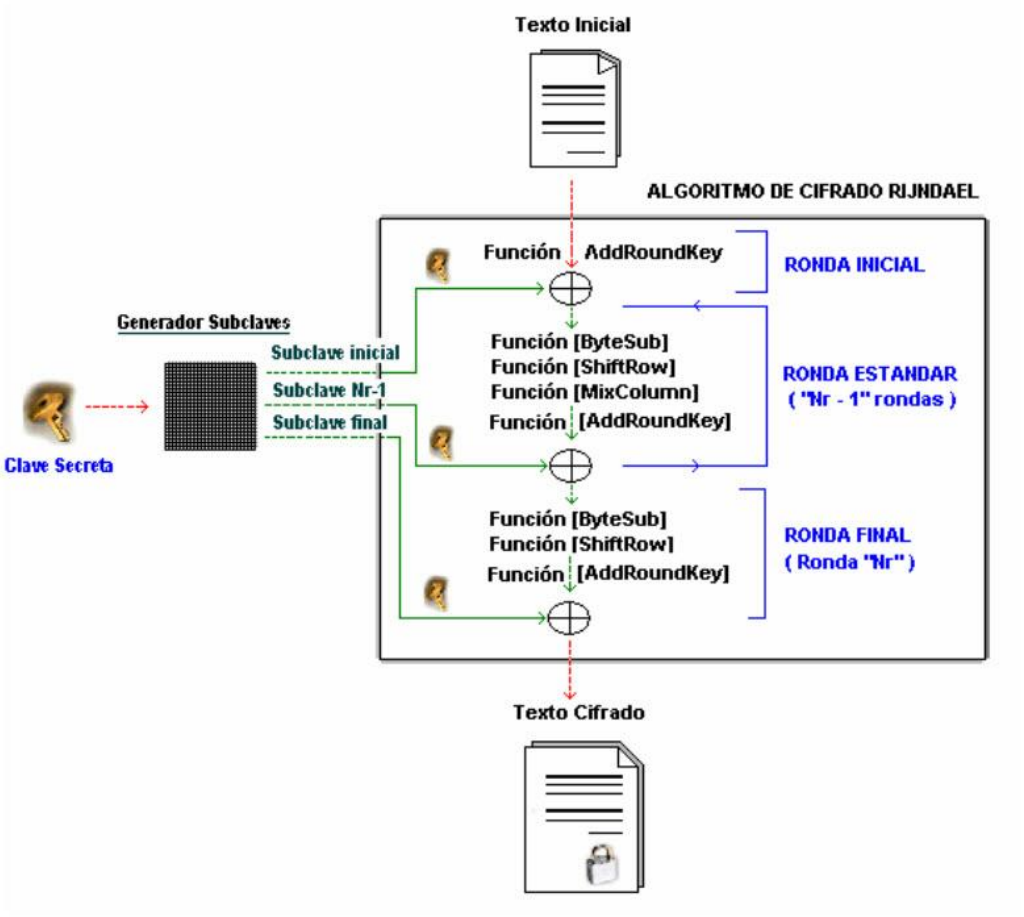


Figura 2- 2: Esquema del Algoritmo AES. [4]

2.1 Sustitución de Bytes: SubBytes

Se trata de una sustitución no lineal que se aplica de manera independiente a cada byte de la matriz de estado, generando un nuevo byte. Esta transformación consiste en la sustitución de cada byte por un nuevo byte el cual es obtenido de la matriz de sustitución llamada S-Box en donde el byte de la matriz de estado es usado como índice para obtener el nuevo byte de la matriz de sustitución.

La figura 2.2 ilustra la transformación por SubBytes a una matriz de estado.

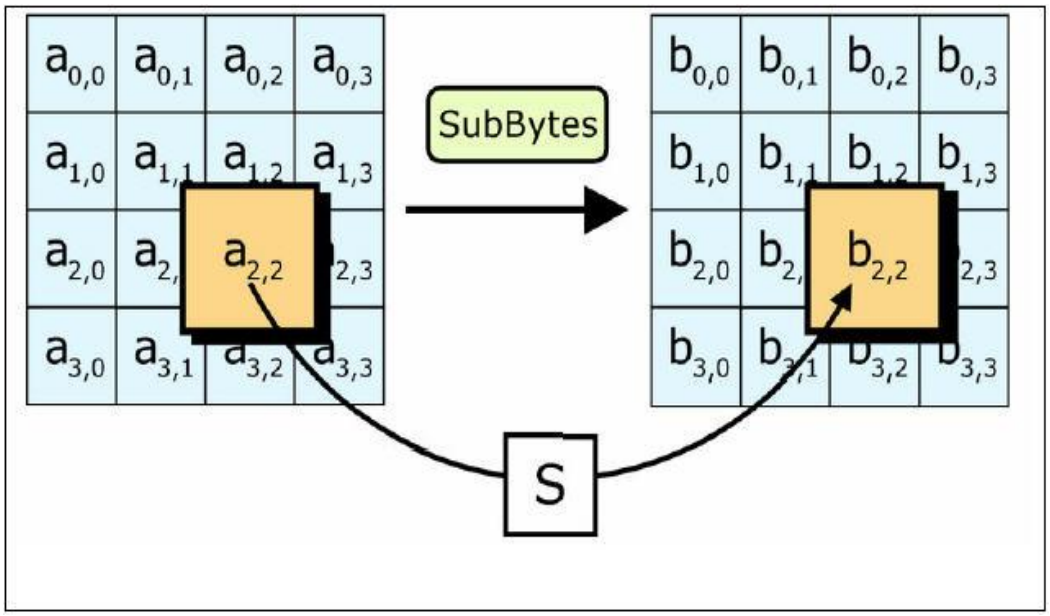


Figura 2- 3: SubBytes. [8]

En esta fase de SubBytes, cada byte en la matriz de estado es reemplazado con su entrada en una tabla de búsqueda fija de 8 bits,

S; bij = S(aij)

2.2 Corrimiento de Renglones: ShiftRow

Esta transformación consiste en rotar hacia la izquierda las filas de la matriz de estado, es decir, se hace rotar los bytes de las filas de la matriz de estado resultante de la transformación anterior (SubBytes).

La primera fila de esta matriz siempre permanece inalterada, la segunda fila se rota 1 byte, la tercera fila se rota 2 bytes, y la cuarta fila se

rota 3 bytes cuando son bloques de 128 o 192 bits. Cuando los bloques son de 256 bits estos desplazamientos son diferentes en este caso se rotaran; 0 bytes, 1 byte, 3 bytes, 4 bytes respectivamente.

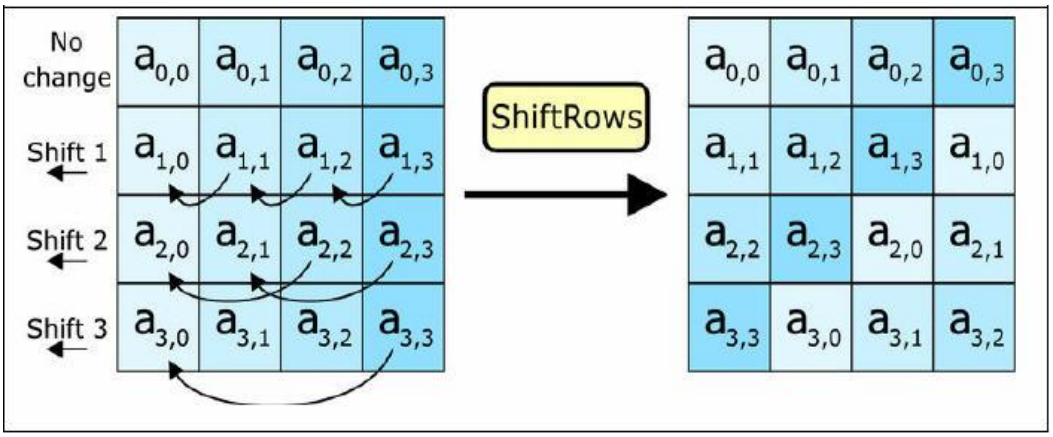


Figura 2- 4: ShiftRows. [8]

Por ejemplo, si el tamaño del bloque fuera de 128 bits ($N_b = 4$) la fila 0 no se le aplican rotación, la fila 1 se la rotaria 1 byte ($C_1 = 1$), la fila 2 se la rotaria 2 bytes $C_2 = 2$, y la fila 3 se rotaria 3 bytes $C_3 = 3$

Tamaño de bloque	C1	C2	C3
128 bits ($N_b = 4$)	1	2	3
192 bits ($N_b = 6$)	1	2	3
256 bits ($N_b = 8$)	1	3	4

Figura 2- 5: Rotaciones. [4]

2.3 Mezclado de Columnas: MixColumns

En esta función los cuatro bytes de cada columna de la matriz de estado se combinan usando una transformación lineal inversible. En esencia en esta etapa se realiza una mezcla de los bytes de las columnas.

Cada columna se trata como un polinomio cuyos coeficientes pertenecen a $GF(2^8)$, es decir son también polinomios.

La operación principal de esta función es la multiplicación las columnas de bytes modulo $x^4 + 1$ por el polinomio $c(x)$ donde matemáticamente $c(x)$ viene representando por:

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

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

Donde $s'(x)$ representa la matriz de estado resultante de esta transformación y $s(x)$ la matriz de estado entrante en este caso sería la matriz resultante de la transformación (ShiftRows), esta fórmula se la podría representar de forma matricial donde c representa el índice de la columna que se está procesando.

La grafica 2.6 se representa a cada columna de la matriz de estado es multiplicada por un polinomio constante $c(x)$

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

Figura 2- 6: Producto Matrices. [8]

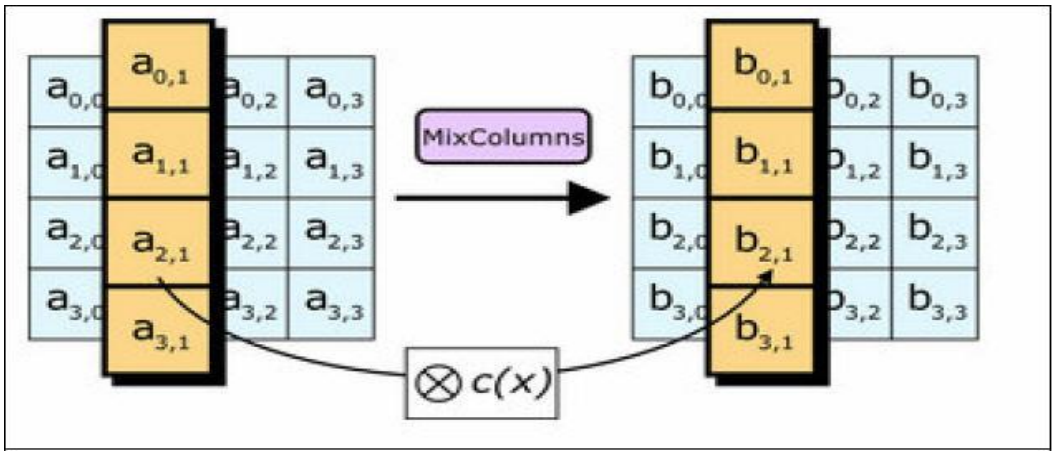


Figura 2- 7: MixColumns. [4]

2.4 Suma de la Clave (Llave): AddRoundKey

Esta transformación consiste en que cada byte de la matriz de estado que proviene de la transformación anterior (MixColumns) se combina con una clave de la ronda mediante la operación lógica XOR; cada una de estas

claves son generadas a partir de la clave original de cifrado para determinada ronda.

La matriz resultante de esta transformación, será la siguiente matriz de estado para la siguiente ronda o la matriz de salida en el caso de ser la última ronda.

El aspecto principal en que se basa esta función recae en las subclaves, dado que el algoritmo al ser desarrollado con el principio de criptografía moderna, donde se establece que la seguridad de un algoritmo solo debe depender de la clave utilizada, se utilizan diferente subclaves K_i en el proceso de cifrado para que el resultado del algoritmo dependa únicamente de una información externa al sistema en este caso la clave del usuario.

El AddRoundKey es ilustrado en la figura 2.5. En donde a_{ij} denota el byte correspondiente a la matriz de estado, mientras que k_{ij} denota a la llave de la ronda, y b_{ij} representa el byte de la matriz de estado resultante.

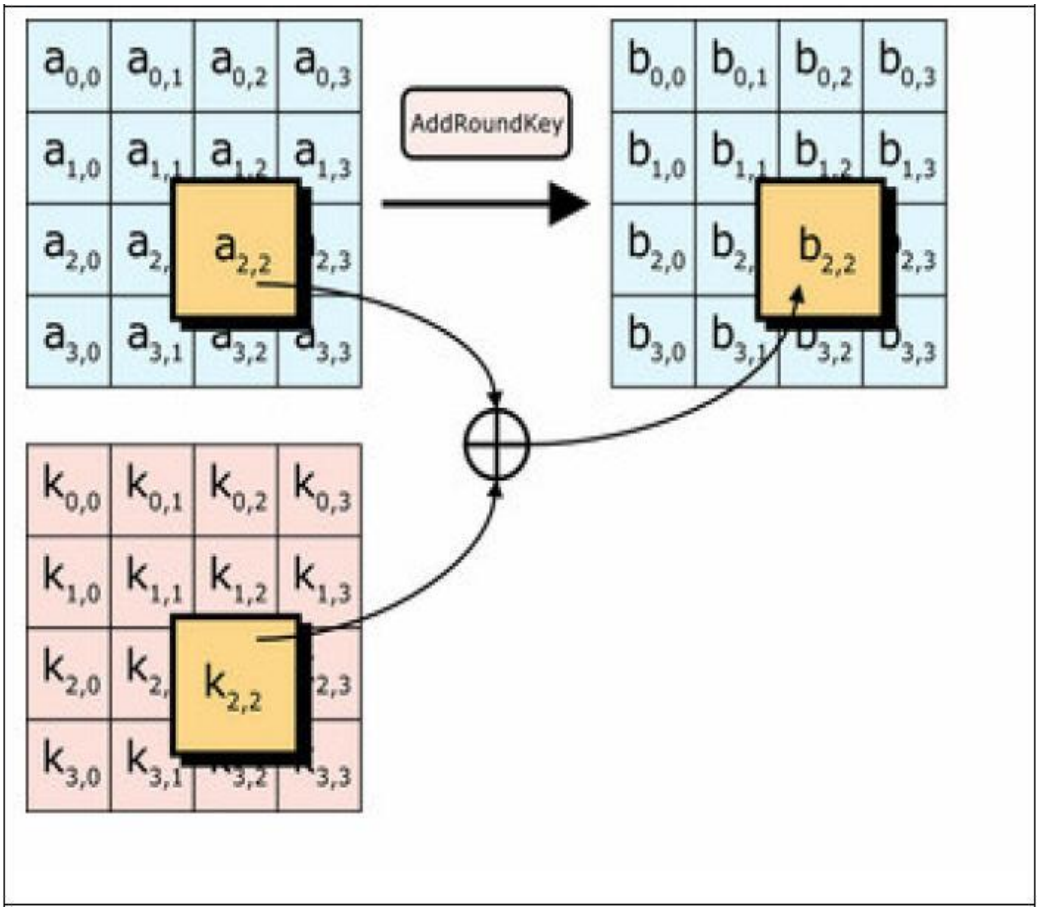


Figura 2- 8: AddRoundKey. [8]

2.5 Función de Expansión de Clave

Esta función permite generar los bytes necesarios como subclaves a partir de la clave inicial del sistema.

Esta función de expansión se puede describir como una arreglo lineal, denominado W conformado por palabras de 4 bytes y con una longitud de $N_b * (N_r + 1)$. Cada subclave tiene el mismo tamaño que la clave original.

Las primeras N_k palabras del arreglo contienen la clave inicial de cifrado dado que esta clave se mapea tal cual en el arreglo W, mientras que las demás claves se generaran a partir de las primeras N_k palabras. Teniendo en cuenta que la clave de la posición "i" se obtiene de la suma (XOR) de la clave de la posición anterior con una clave N_k posiciones anteriores, es decir $w_i = w_{i-1} \otimes w_{i-N_k}$ para subclaves con índice múltiplo de N_k, antes de realizar la adición con una clave w_{i-1} se somete a dos transformaciones, RotWord() y SubWord(), y por una adición con una constante Rcon[i].

En la función RotWord() se realiza una permutación cíclica de los bytes de subclave y que corresponde a un byte a la izquierda. Por ejemplo la subclave es $[a_0, a_1, a_2, a_3]$ despues de ejecutar la función se obtiene $[a_1, a_2, a_3, a_0]$.

La función SubWord() consiste en obtener los valores definidos en S-Box para cada uno de los cuatro bytes de la subclave.

La función Rcon genera una constante teniendo en cuenta que:

$$Rcon(j) = (R(j), 0,0,0)$$

Donde cada $R(j)$ es un elemento $GF(2^8)$ correspondiente al valor x^{j-1}

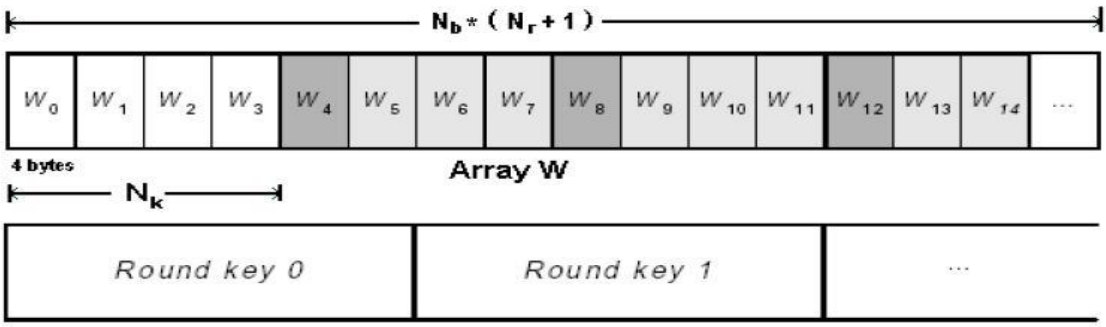


Figura 2- 9: Subclaves. [4]

CAPITULO 3

3. DISEÑO DEL MÓDULO DE ENCRIPCION AES

3.1. Generalidades.

Este capítulo se centra en el diseño del los módulos utilizados tanto para la transferencia, recepción de datos y el algoritmo AES, utilizando un FPGA de la familia Altera y el entorno de programación Quartus II, tal como se indica en el diagrama de bloques que se muestra en la Figura 3-2

Esta implementación destaca y demuestra el potencial que poseen los FPGAs en el desarrollo de aplicaciones complejas que requieren gran catidad de computo y recursos lógicos.

Como se observa en la Figura 3.1 los módulos del sistema son:

- Modulo Reloj. Este modulo provee de una señal o un pulso antirebote con lo cual se evita que al activar esta señal se reciba más de un pulso, esta es generada por la activación de un switch en la tarjeta DE2 la cual fue utilizada para la sincronización en el envió de datos desde la tarjeta hacia la PC.

En la figura 3-1 se muestra una representación de la entidad VHDL de este modulo.

Básicamente el modulo envía una señal de alto cuando se detecta un alto en la señal PB_SIN_REBOTE y un bajo en una señal interna. En la tabla 3-1 se indica la dirección, la longitud en bits y una descripción de cada uno de los puertos.

Figura 3- 1: Representación de la Entidad VHDL Reloj.

Puerto	Dirección	Longitud en bits	Descripción
PB_SIN_REBOTE	Entrada	1	Pulso para su generación sin rebote.
CLOCK	Entrada	1	Señal de reloj del modulo.
UN_PULSO	Salida	1	Señal de habilitación de los módulos de transferencia..

Tabla 3-1 : Puertos de la entidad VHDL Reloj

- Módulo Uart. Este módulo se encarga de la recepción de las cadenas de texto, en la cuales se envía el texto a cifrar y la

respectiva contraseña para la realización del proceso de encriptación. Desde la interfaz gráfica se envían en forma de caracteres ASCII la cadena a cifrar y la contraseña. Este módulo de recepción consta de tres etapas como se muestra en la figura 3-2

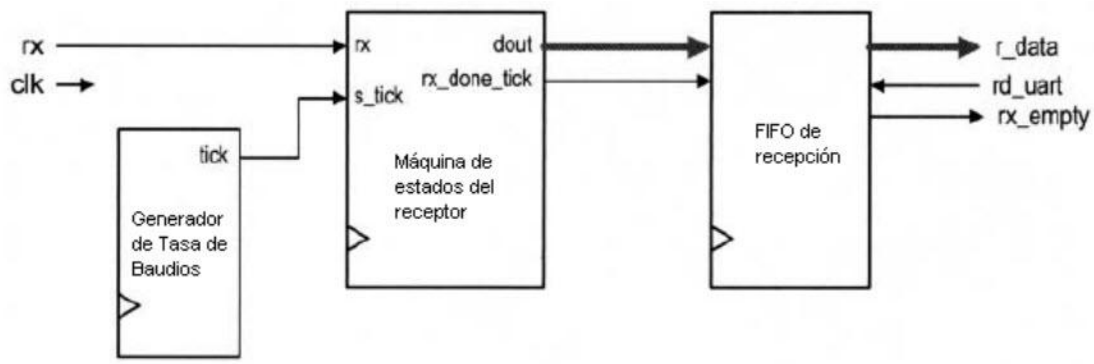


Figura 3- 2: Sistema de recepción UART [7]

3.2 Generador de Tasa de Baudios

Este bloque genera una señal s_tick la cual es la que indica el momento que debe tomar una muestra del bit recibido, se toman 16 muestras y para esto se debe poner en 1L una señal cada vez que el contador alcance el valor de 54 $\left(\frac{50x10^6}{16x57600}\right)$, en el caso que la velocidad de 57600 bps y se tenga un oscilador de 50MHz en el FPGA. Esta señal no actúa como una señal de reloj en la máquina de estado, esta es una señal de habilitación de los registros que actúan como contadores en el módulo. [7]

3.3 Receptor

El receptor es una máquina de estados que se basa en un esquema de muestreo, en el que cada bit es muestreado 16 veces, con la finalidad de obtener un punto medio. El proceso para la comunicación de N bits de datos y M bits de parada, el muestreo se realizara de la siguiente forma.

1. Se espera por la señal de entrada tenga el valor de cero el cual sería el bit de inicio y luego se inicia el contador de muestras que se incrementa cada vez que la señal s_tick del generado de baudios tiene el valor de 1L
2. Cuando el contador alcance el valor de 7, la señal de ha alcanzado el punto medio del bit de inicio y el contador se reinicia a 0.
3. Cuando el contador alcanza el valor de 15, la señal de entrada ha alcanzado el punto medio del primer bit de datos; este valor es desplazado en un registro y se reinicia el contador.
4. Se repite el paso 3, N-1 veces hasta que se recuperan todos los bits de datos.
5. Cuando se utiliza un bit de paridad, el paso 3 se repite una vez más.
6. Repetir el paso 3, M veces más, hasta obtener los bits de parada.

Como se observa en la figura 3-4 en la cual se representa el diagrama ASM, se definen 4 estados para la realización del procedimiento de muestreo: *idle, start, data, y stop.* La maquina se mantiene en estado *idle* hasta que la señal

de recepción *rx* tome el valor de cero. Los estados *start, data y stop* representan los procesos de los bits de inicio, datos y parada respectivamente de acuerdo al procedimiento de muestreo antes expuesto. En el código VHDL que implementa esta máquina de estados están presentes las constantes D_BIT, que indica el número de bits de datos y SB_TICK, que indica el número de muestras necesarias para los bits de parada, que para el caso de este diseño son 8 y 16 respectivamente.

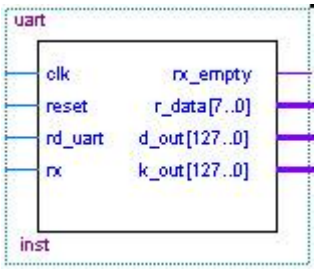


Figura 3- 3: Representación de la Entidad VHDL Uart.

Puerto	Dirección	Longitud en bits	Descripción
clk	Entrada	1	Señal de reloj del modulo.
reset	Entrada	1	Señal de reset del modulo
rd_uart	Entrada	1	Selector del modo escritura
rx	Entrada	1	Señal conectada al pin del FPGA para manejo de puerto serial
rx_empty	Salida	1	Señal que indica que una nueva cadena está disponible
r_data	Salida	8	Cadena de bits que se lee del puerto serial
d_out	Salida	128	Texto enviado para encriptar
k_out	Salida	128	Clave de encriptacion

Tabla 3-1: Puertos de la entidad VHDL Uart.

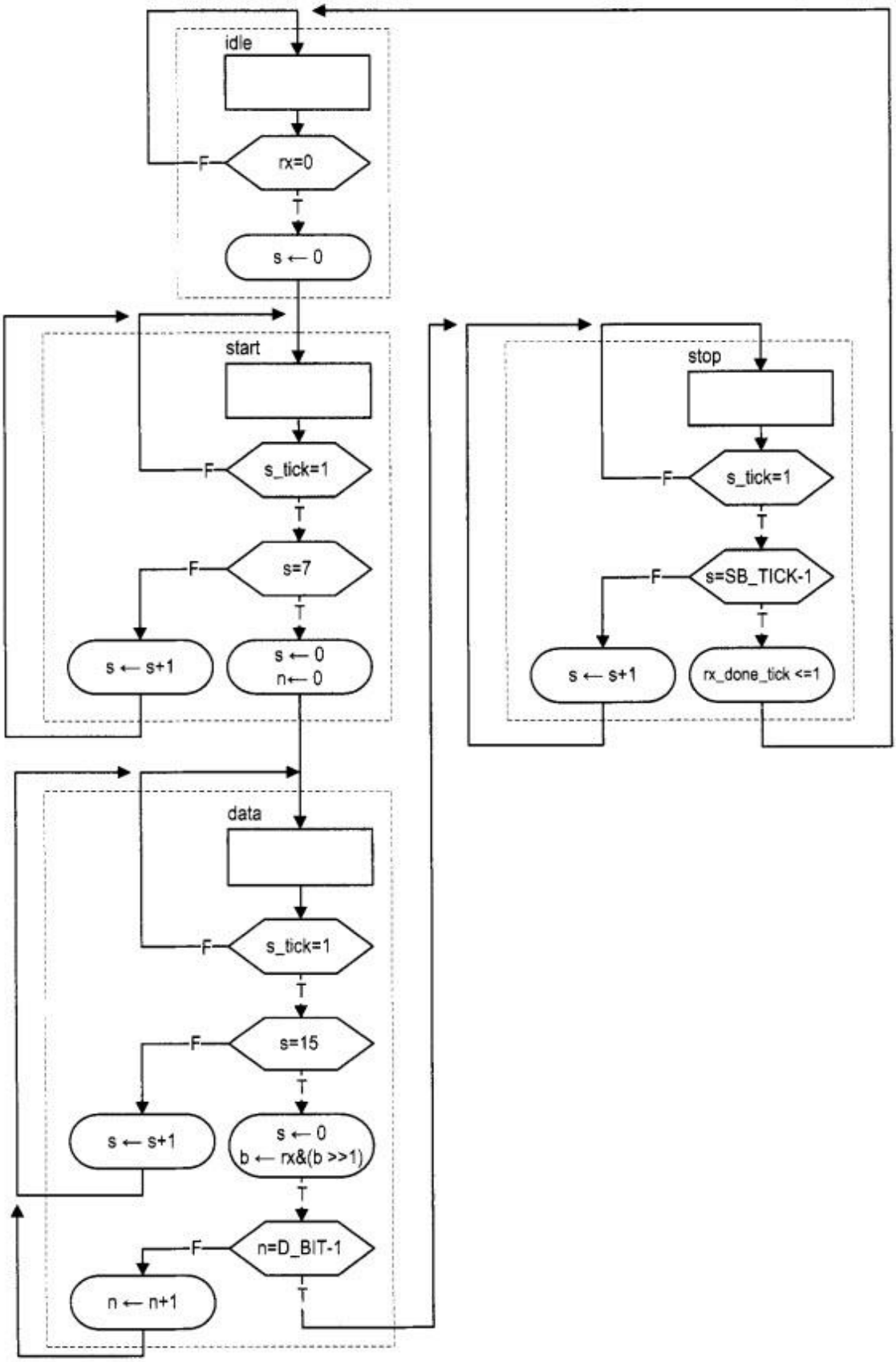


Figura 3- 4: Diagrama ASM de máquina de estados del transmisor [7].

- Modulo Aes_Encript. Módulo principal en cual se encuentra implementado en el algoritmo AES para la realización del proceso de encriptación.

  En la figura 3-5 se muestra la representación de la entidad VHDL de este modulo. Básicamente este módulo es el principal del algoritmo AES, desde este módulo se realiza el llamado a cada una de la entidades que implementan el algoritmo.

  En la tabla 3-2 se detalla la dirección, la longitud en bits y una descripción de cada uno de los puertos que la entidad VHDL de este modulo.

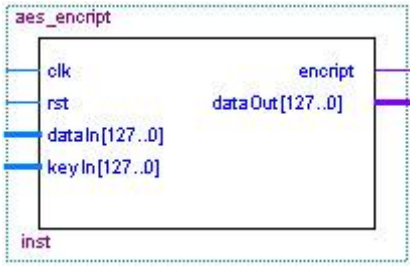


Figura 3- 5: Representación de la Entidad VHDL aes_encript.

Puerto	Dirección	Longitud en bits	Descripción
clk	Entrada	1	Señal de reloj del módulo.
rst	Entrada	1	Señal de reset del módulo
dataIn	Entrada	128	Bits que representan la cadena de datos a encriptar
keyIn	Entrada	128	Bits que representar la clave a usar en la encriptación
encript	Salida	1	Señal que indica la

			finalización de la encriptación
dataOut	Salida	128	Bits que representan el resultado de la encriptación.

Tabla 3-2: Puertos de la entidad VHDL Aes_Encript.

- Módulo Transfer_Data. En este módulo se realiza el manejo del resultado obtenido por el módulo anterior para que mediante una máquina de estados se realice el envió de este resultado al siguiente módulo que es donde se realiza el envió de esta cadena a la PC,. El objetivo de este módulo es el de enviar de byte en byte la cadena resultado de la encriptación al módulo que maneja el puerto serial para el envío de los datos.

  En la figura 3-6 Se ilustra un representación de la entidad del bloque transfer_data y en la Tabla 3-4 se indica la dirección, longitud en bits y una pequeña descripción de los puertos que posee esta entidad.

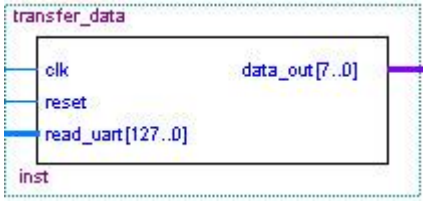


Figura 3- 6: Representación de la Entidad VHDL transfer_data.

Puerto	Dirección	Longitud en bits	Descripción
clk	Entrada	1	Señal de reloj del módulo.
reset	Entrada	1	Señal de reset del módulo
read_uart	Entrada	128	Bits que representan la cadena encriptada a ser transferida a la PC
data_out	Entrada	128	Bits que representa el byte a enviarse a la PC

Tabla 3-3: Puertos de la entidad VHDL Transfer_Data.

El proceso de transmisión de bytes empieza cuando se detecta, a través del puerto de entrada clk un valor 1L lo cual hace que se active el paso entre los estados en los cuales de define el byte que debe ser enviado.

- Módulo Uart_Send. Módulo mediante el cual se realiza el manejo del puerto serial para realizar el envió de los datos encriptados hacia la PC. Este módulo es igual al de recepción cambiando en algunas condiciones en la máquina de estados del transmisor el generador de baudios y es el mismo que se utiliza en la etapa de recepción.

3.4 Transmisor

Como se puede observar en la figura 4-7 la máquina de estados del transmisor tiene cuatro estados: idle, start, data, stop, en la descripción VHDL al igual que el caso de recepción se incluyen las contantes D_BIT la cual indica el número de bits de los datos y SB_TICK que indica el número de muestras necesarias para los bits de parada.

La forma como se lleva a cabo la transmisión es el siguiente.

1. En el estado *start* se transmite el bit de inicio con el valor de 0L a través de la señal *tx,* esto hasta que el registro *s* alcance el valor de 15, luego de eso se realiza la transición al estado *data.*
2. En el estado *data* se realiza la transmisión de los bits de datos, en el cual se asigna a cada bit del registro *b* la señal de salida *tx* durante el tiempo que tarde el registro *s* en alcanzar las 15 muestras; una vez que el este registro alcance el valor de 15, se realiza el incremento del registro n, que es el que cuenta el número de bits de datos transmitidos, y se desplaza el registro *b* una posición hacia la derecha. Cuando este registro alcanza el valor que tiene la constante D_BIT-1, se produce la transición al estado *stop.*
3. En el estado *stop* se hace la transferencia del bit de parada, para esto se asigna el valor 1L a la señal de salida *tx,* esto se realiza hasta que el registro *s* alcance el valor de la constante SB_TICK-1. Después de esto realiza la transición al estado inicial *idle.*

En la figura 3-7se muestra una representación de la entidad VHDL de este módulo y en la tabla 3-5 Se indica la dirección, la longitud en bits y una descripción de los puertos que posee la entidad VHDL del bloque Uart_Send.

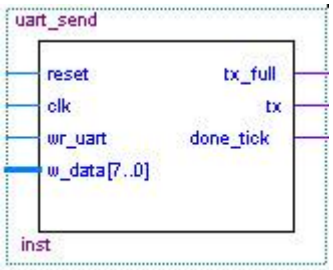


Figura 3- 7: Representación de la Entidad VHDL uart_send.

Puerto	Dirección	Longitud en bits	Descripción
reset	Entrada	1	Señal de reset del módulo
clk	Entrada	1	Señal de reloj del modulo.
wr_uart	Entrada	1	Señal que habilita la escritura en el módulo.
w_data	Entrada	8	Cadena de bits que representa parte de la cadena encriptada a enviarse a la PC
tx_full	Salida	1	Señal que representa la condición full de la memoria de transmisión
tx	Salida	1	Señal conectada al pin del FPGA para manejo de puerto

			serial.
done_tick	Salida	1	Señal que indica que el proceso de transmisión a terminado.

Tabla 3-1: Puertos de la entidad VHDL Uart_Send.

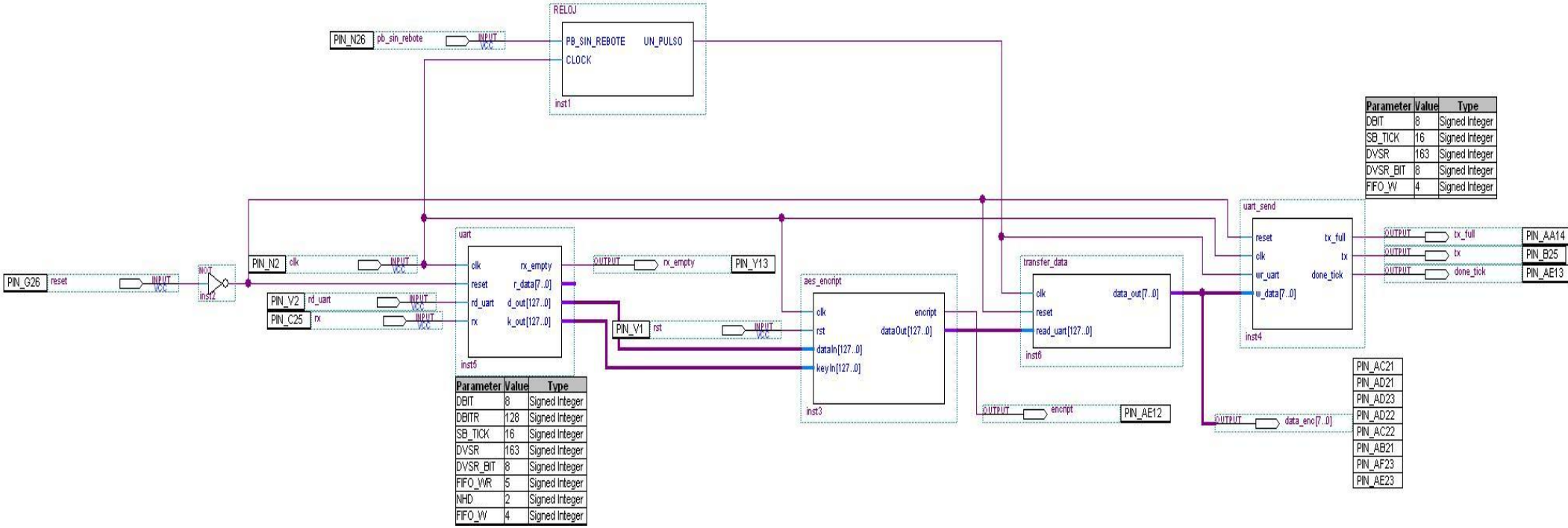


Figura 3- 8: Esquemático.

3.5 Elección de la tecnología FPGA a utilizar.

FPGA por sus siglas en ingles *Field Programmable Gate Array*. Es un dispositivo semiconductor conformado por bloques lógicos que se encuentran interconectados con diversos recursos programables para permitir la personalización de los diseños. Son en la actualidad la mejor alternativa de diseño de hardware debido a su funcionalidad, fiabilidad, miniaturización, robustez, bajo consumo y sencillez de desarrollo**,** esta tecnología ofrece muchas ventajas para el desarrollo de diseño de equipos para criptografía.

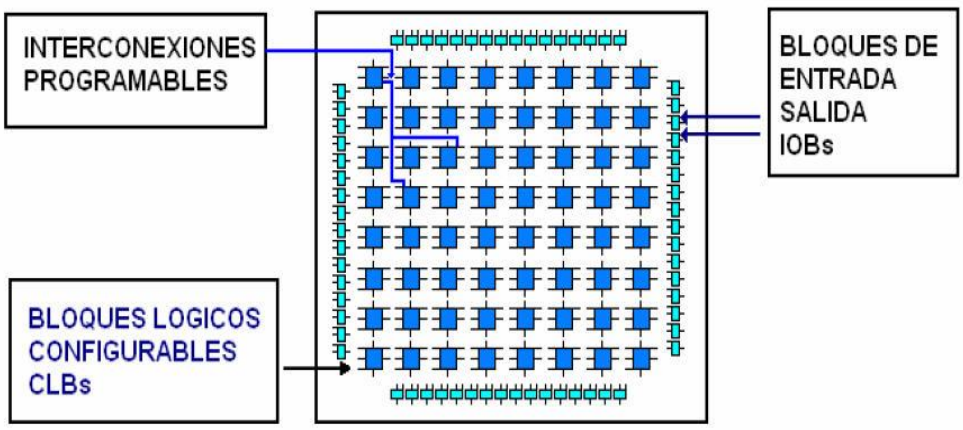


Figura 3- 9: Arquitectura básica de un FPGA.

La familia FPGA seleccionada para el presente diseño es Altera Cyclone II, chip EP2C35. En este caso el dispositivo forma parte de la “The Altera Development and Education (DE2)”, la herramienta de software que uso para la implementación del algoritmos AES-128 en VHDL es Quartus II Versión 9.1 Web Edition. Esta herramienta también nos permite realizar las simulaciones del comportamiento en tiempo de nuestros diseños.

El dispositivo cuenta con los siguientes recursos:

- Un total de 33216 elementos lógicos.
- 475 terminales de E/S
- 105 bloques de memoria RAM
- 4 bucles de fase (PLL)

3.6 Elección del Lenguaje

Se ha elegido a VHLD para la realizar la descripción del circuito, este es un lenguaje de descripción de hardware para el modelado de sistemas digitales que surgió a partir del programa de Circuitos Integrados de muy alta velocidad (VHSIC por sus siglas en ingles); actualmente es también utilizado en la simulación y la síntesis. Con este lenguaje podemos lograr minimizar la complejidad del desarrollo aplicando técnicas de partición y jerarquía.

En VHDL las entidades(componentes, circuitos) están conformadas por dos partes, la parte visible o externa que corresponde al nombre de la entidad y sus conexiones y una parte oculta o interna en la cual se encuentran algoritmos de la entidad y su implementación. Una vez definida la interfaz externa, otras entidades pueden usarla cuando se haya completado su desarrollo. Este concepto de vista externa e interna es central para una perspectiva de diseño de sistemas en VHDL.

Una entidad es definida en relación a otras, por sus conexiones y comportamiento.

Se puede explotar implementaciones externas (arquitecturas) de una de ellas sin cambiar el resto del diseño. Después de definir una entidad para un diseño, se puede reutilizar en otros diseños tanto como sea posible [8].

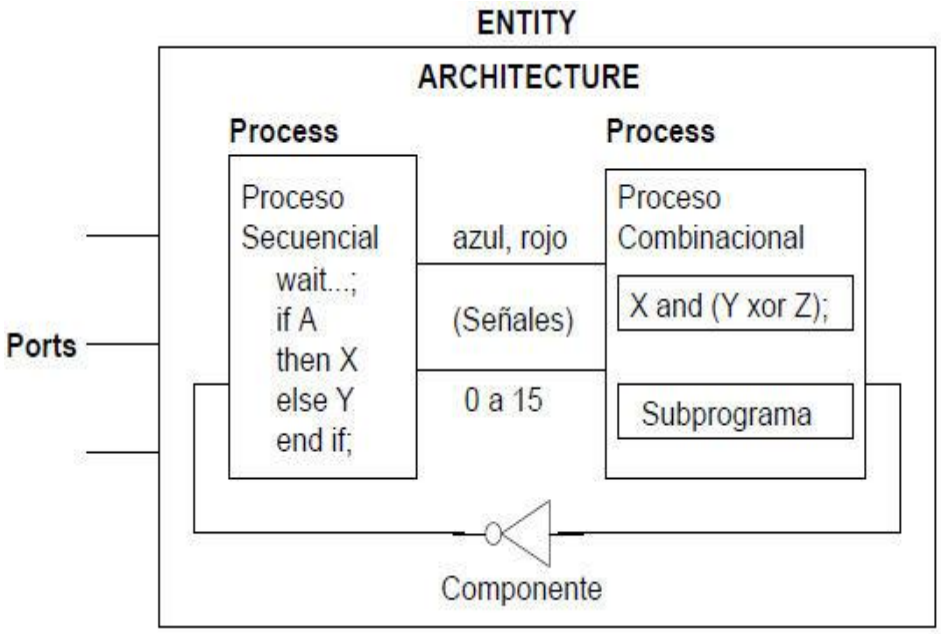


Figura 3- 10: Un modelo VHDL de hardware. [8]

En VHDL existen varias formas de diseñar un circuito las cuales son:

Comportamiento: Es posible describir un circuito digital simplemente describiendo su comportamiento funcional o algoritmo de diseño, expresado en un proceso secuencial VHDL.

Flujo de datos: Los datos son vistos como un flujo a través del diseño, Una operación es definida en términos de una colección de transformación de datos, expresado como una sentencia concurrente.

Estructural: Modelo en donde los componentes de un diseño se enumeran y se interconectan. Esta expresada por instanciación de componentes.

3.7 Estrategia Para Implementación de Operación del Algoritmo Rijndael.

Para la implementación del algoritmo Aes-Rijndael se ha procedido a determinar las funciones principales que intervienen en el algoritmo, las cuales se procedió a implementar ha manera de entidades de la aplicación para su posterior instanciación en un módulo controlador del proceso de cifrado. A continuación se detallan cada una de la entidades que intervienen en el proceso.

3.7.1. SubBytes.

Esta entidad tiene como objetivo realizar una sustitución de bytes en la cual se utiliza una función llamada sbox en donde esta hace el papel de una memoria ROM de 256 posiciones y funciona con cada byte de entrada correspondiente a la matriz de estado que se va a transformar, se toma como dirección este byte, dicha dirección contiene un valor byte el cual va a ser el byte de salida resultante y así mismo este proceso se repite hasta completar los 128 bits (16 bytes).

3.7.2. ShiftRows

En la siguiente entidad se realiza desplazamientos de los bytes de la matriz de estado generada en la entidad SubBytes, en la cual se realizan un desplazamiento en la segunda fila, que está constituida por los byte 4, byte 5, byte 6, byte 7;dos desplazamientos en la tercera fila, que está constituida por los byte 8, byte 9, byte 10, byte 11; tres desplazamientos en la cuarta fila que está constituida por los byte 12, byte 13, byte 14 y byte 15; la primera fila no sufre ningún desplazamiento.

3.7.3. MixColumns

La entidad MixColumns en el proceso de cifrado realiza la tarea de multiplicar las columnas de la matriz de estado por una columna constante de la forma:

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

La cual es sometida a un desplazamiento para la realización de la multiplicación con la siguiente columna de la matriz, para obtener el byte resultante de cada columna se realiza un operación XOR entre los bytes resultantes de la multiplicación por (02), el byte resultante de la multiplicación por (03), el byte 3 y el byte 4.

3.7.4. AddRoundKey

En esta entidad se realiza la operación XOR entre el texto a cifrar y la clave inicial (Ronda Inicial), la funcionalidad de esta entidad es utilizada también en el proceso de cifrado entre la matriz de estado que se genera en la entidad MixColumns y la subclave generada para la respectiva ronda. La matriz

generada será la nueva matriz de estado para la siguiente ronda. Para finalizar el proceso de cifrado se realiza una última instanciación de esta entidad que corresponde a la ronda final que se lleva a cabo entre la subclave final y la matriz de estado que proviene de la entidad ShifRow, hallando el dato de salida del algoritmo (texto cifrado).

3.7.5. KeySchedExpansion

En esta entidad se realiza la generación de las 10 subclaves necesarias para el proceso de cifrado, generadas a partir de la clave inicial. Se inicia a partir de la cuarta columna de la matriz de clave inicial que está constituida por los bytes 3, 7, 11 y 15, los mismos que son sometidos a un desplazamiento realizado por la función rotByte. A esta columna resultante se realiza la sustitución de bytes mediante la función sbox, luego se realizan operaciones XOR entre la columna remplazada, el valor obtenido de la constante rcon y la primera columna de la matriz obteniendo como resultado la primera subclave. Las siguientes tres subclaves se obtienen de la siguiente manera:

$$W_i = W_{i-1}\ XOR\ W_{i-4}$$

Los valores de rcon se obtienen de la siguiente manera:

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

$$RC[1] = 1;$$

$$RC[i] = x \cdot RC[i{\sim}1] = x^{i{\sim}1};$$

$$RC[1] = 1$$

$$RC[2] = x = 2$$

$$RC[3] = x^2 = 4$$

$$RC[4] = x^3 = 8$$

$$RC[5] = x^4 = 10$$

$$RC[6] = x^5 = 2$$

$$RC[7] = x^6 = 40$$

$$RC[8] = x^7 = 80$$

$$RC[9] = x^8 = x^4 + x^3 + x + 1 = 1b$$

$$RC[10] = x^9 = x^5 + x^4 + x^2 + x = 36$$

CAPITULO 4

4. IMPLEMENTACION DE PRUEBA Y RESULTADOS

4.1. Desarrollo del Software en VHDL

En la actualidad, la mayor parte de los sistemas de comunicación requieren de seguridad en la transferencia de los datos con la finalidad de mantener la privacidad del mensaje transmitido; este mensaje puede tratarse de un email o una transacción de una gran cantidad de dinero entre entidades bancarias, bajo este contexto es de suma importancia mantener la seguridad de los canales de comunicación, con estos antecedentes se ha ideado una aplicación que permite hacer uso del core de encriptación para el cifrado de una cadena de texto en formato ASCII o hexadecimal, esta cadena se someterá a un proceso de descifrado o desencriptacion que provee la aplicación.

La comunicación entre la aplicación y el core se llevó a cabo mediante el puerto serial, haciendo uso del protocolo de comunicación RS232.

La sincronización de los datos entre la tarjeta DE2 y la aplicación estára controlada mediante un core para manejo de comunicación serial.

4.1.1 Algoritmo de encriptación AES.

El proceso de encriptación realizado por el algoritmo AES consiste en la aplicación de 4 funciones matemáticas sobre la información que se desea encriptar. Estas transformaciones son aplicadas en cada una de las rondas que se son definidas según el tamaño de la clave.

Primeramente la información a cifrar es mapeada en una matriz, esta matriz inicia el proceso de encriptación en la ronda inicial que consiste en una operación or exclusiva (AddRoundKey) entre la subclave generada y la matriz, seguido de esta primera ronda, a la matriz resultante se le aplican 4 transformaciones invertibles, repitiendo este proceso Nr-1 veces, Finalmente se aplica una ronda a la matriz resultante de las Nr-1 rondas anteriores aplicando la funciones en el siguiente orden subBytes, shiftRows, addRoundKey. El resultado de esta ronda es la información encriptada deseada.

A continuación se revisan la implementación y su comportamiento a nivel de simulación de cada una de las funciones que forman parte del algoritmo de encriptación.

4.1.1.1 Entidad SubBytes

La primera función en ser implementada, es la de substitución de bytes (SubBytes), que realizar el intercambio de un byte de entrada por otro byte

predeterminado que es obtenido de la función **sbox** que modela una tabla fija de valores con 256 posiciones y valores de 8 bits, la entidad posee las siguiente entradas necesarias para el comportamiento como: **rst** que es una señal de reset, **clk** que es la señal de reloj y una matriz **($\mathbf{matrizIn_{ij}}$)** del tipo matriz_data que es del orden 4x4 en la cual cada una de la posiciones de la $\mathbf{matrizIn_{ij}}$ corresponden a un byte, cada uno de estos bytes son los parámetros de entrada a la función **sbox** obteniendo de esta manera el respectivo byte cuyo valor es definido por el algoritmo de encriptación AES. La tabla que se implementa en la función sbox es la que se muestra en la figura.

x \ y	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
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

Figura 4- 1: Sbox. [8]

Ahora se presentan los resultado obtenido en el proceso de simulación de la entidad subbytes en el cual contamos con las señales de entrada **matrizIn[0], matrizIn[1], matrizIn[2], matrizIn[3]** que representan las filas de la matriz de estado, conformada cada elemento de la misma por un byte y en la cual están representada la cadena de bytes a encriptar, en la figura 4-2 que está la gráfica de la forma de onda que representa los resultados de la simulación obtenido por el software Quartus II, se puede ver en detalle el resultado de la entidad, a la cual se aplica un señal de reloj con un ciclo de 10ns, también se puede ver la marca de 10.0ns y 30.0ns mostrando el tiempo máximo de retardo de la entidad. Los resultados que muestran los parámetros de rendimiento obtenidos por esta entidad se los puede ver en la Figura 4.3

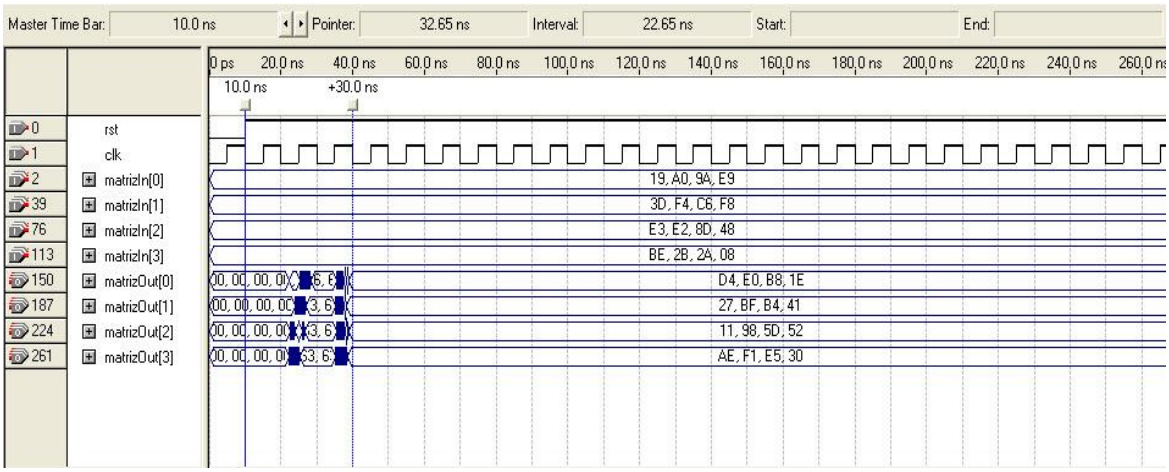


Figura 4- 2: Simulación Subbytes.

Figura 4- 3: Recursos FPGA en Subbytes.

4.1.1.2 Entidad ShiftRows

Luego de aplicarse la entidad Subbytes a la matriz de estado la siguiente función que debe aplicarse es la función ShiftRows la cual se ha implementado en la entidad que tiene como entradas las señales **rst, clk** del tipo std_logic y una matriz **($matrizIn_{ij}$)** del tipo matriz_data, esta matriz de entrada es el resultado de la aplicación de la función SubBytes a la matriz de estado. Como se puede observar en los resultado obtenidos de la simulación del proceso de la entidad, el cual es el de realizar un desplazamiento a la izquierda en los bytes de las tres últimas filas, las cuales son desplazadas cíclicamente en distintas cantidades.

La señal de reloj que se aplicó en la simulación es de 5ns por ciclo, obteniéndose un tiempo de retardo de 10ns para los resultados finales en la matriz ($matrizOut_{ij}$) en cuyas tres últimas filas se puede observar la rotación realizada.

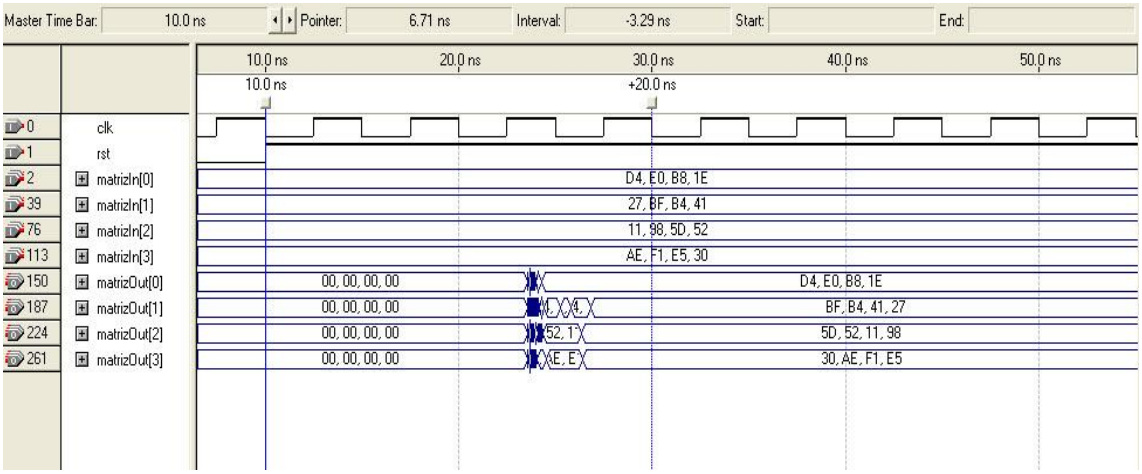


Figura 4- 4: Simulación ShiftRows.

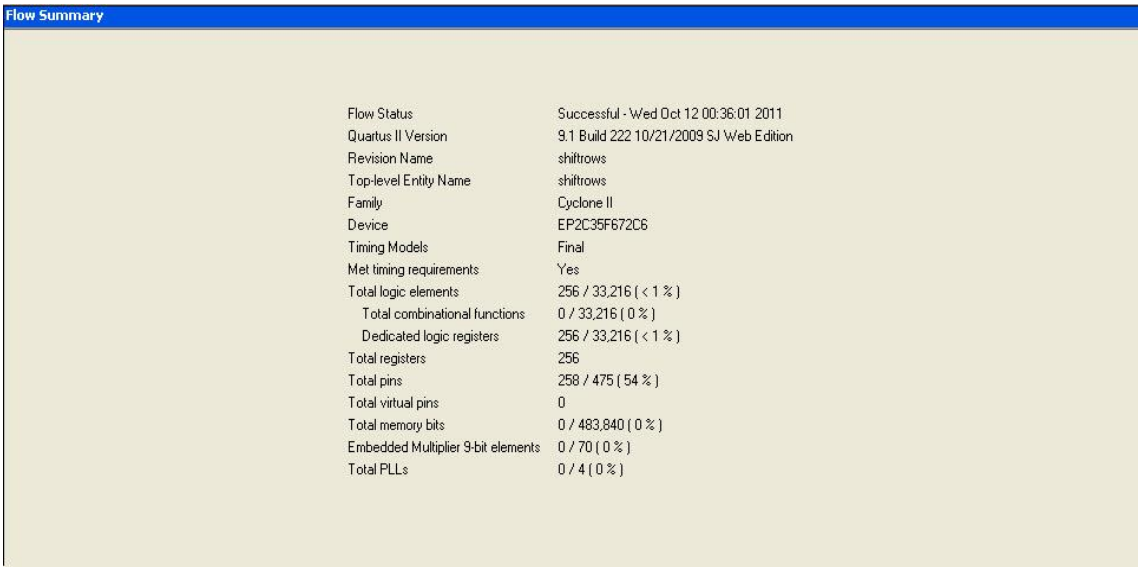


Figura 4- 5: Recursos FPGA en ShiftRows.

4.1.1.3 Entidad MixColumns

Continuando con la funciones del algoritmo la siguiente función implementada como entidad es la función MixColumns, tiene como señales de entrada a las señales **rst, clk** del tipo std_logicy una matriz **($\mathbf{matrizIn_{ij}}$)** del tipo matriz_data la cual será la matriz que se obtiene como resultado de la aplicación de la función ShiftRows, dado que en la función MixColumns se debe realizar una operación en la cual se debe operar la matriz columna por columna, donde estas columnas de bytes se consideran como polinomios de cuatro términos las cuales deben ser multiplicadas por un polinomio fijo, para este caso en esta transformación MixColumns se requiere 16 multiplicaciones por 2 y 16 multiplicaciones por 3, así como también se

requiere realizar 48 operaciones XOR. El funcionamiento de la entidad ha sido simulado y el resultado se presenta en la figura 4-6 En esta figura las señales **matrizIn[0]** a **matrizIn[4]** son simplemente la representación de la matriz de estado antes de la aplicación de **MixColunms()**. Las señales **matrizOut[0]** a **matrizOut[4]** en su conjunto representan el resultado obtenido, también se puede observar el retardo inicial en la obtención de los bytes de la operación.

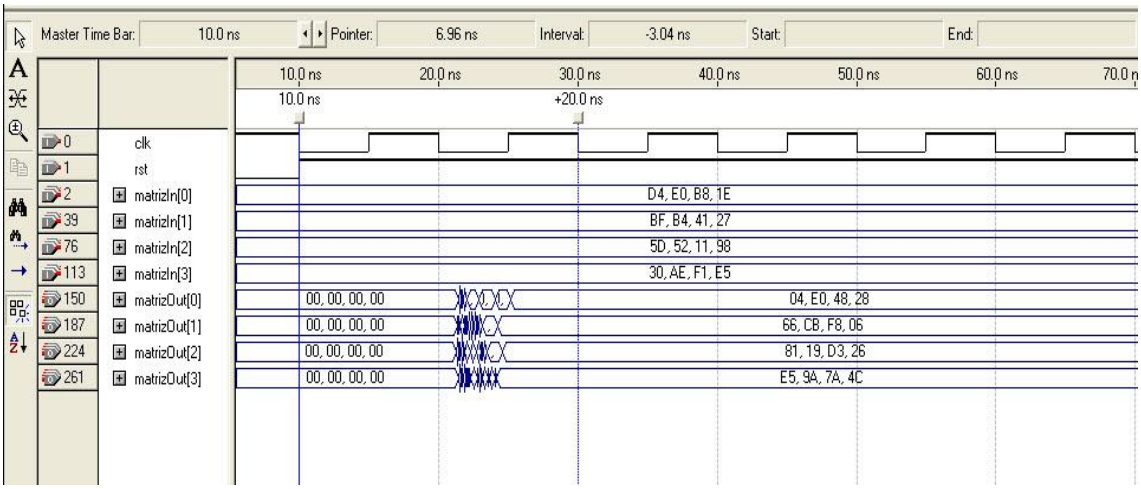


Figura 4- 6: Simulacion Mixcolumns.

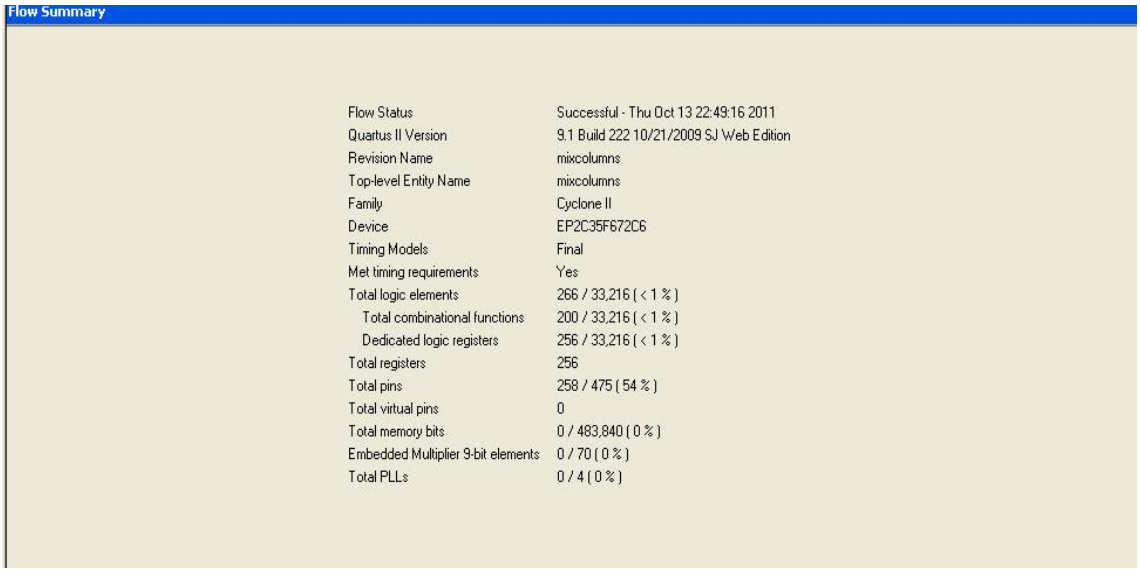


Figura 4- 7: Recursos FPGA en Mixcolumns.

4.1.1.4 Entidad AddRoundKey

El siguiente paso en el algoritmo Rijndael es la función AddRoundkey la misma que es implementada como entidad en la cual contamos con las señales fijas de **rst** y **clk** del tipo sdt_logic y para esta entidad se hace uso la matriz de estado obtenida del proceso de la entidad MixColumns la cual está representada por la **martrizIn** del tipo matriz_data y las subclaves obtenidas a partir de la clave inicial y que se generan en la función KeySchedExpasion. Esta clave está representada en las señales w0 a w3 las cuales son del tipo std_logic_vector(31 downto 0). El proceso de esta entidad es muy sencillo dado que consiste en la adición tipo XOR de cada unos de los bytes en las posiciones de la matrizIn con los correspondiente bytes de la subclave. Para la realización de la simulación del comportamiento de ésta entidad trabajó con la primera subclave obtenida de la clave de cifrado, se puede apreciar en

el gráfico el resultado obtenido del proceso el cual está representado en la señal matrizOut[0] a matrizOut[3], así mismo se puede apreciar el retardo con el cual se obtienen los resultados esperados, la señal de reloj aplicada para ésta simulación tiene un periodo de 10.0ns con lo que se determina que la entidad obtiene los resultado después de 2 periodo de reloj.

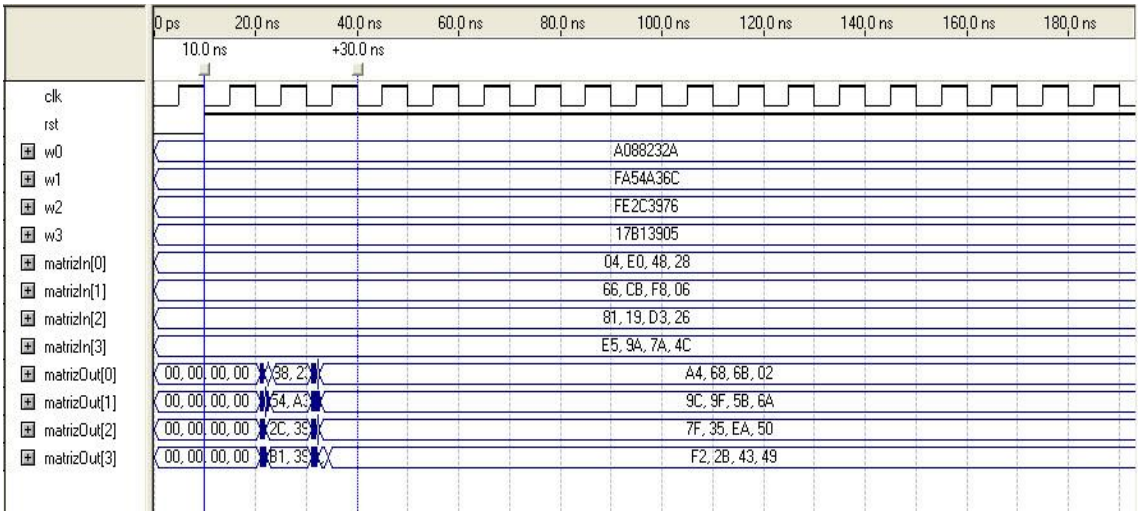
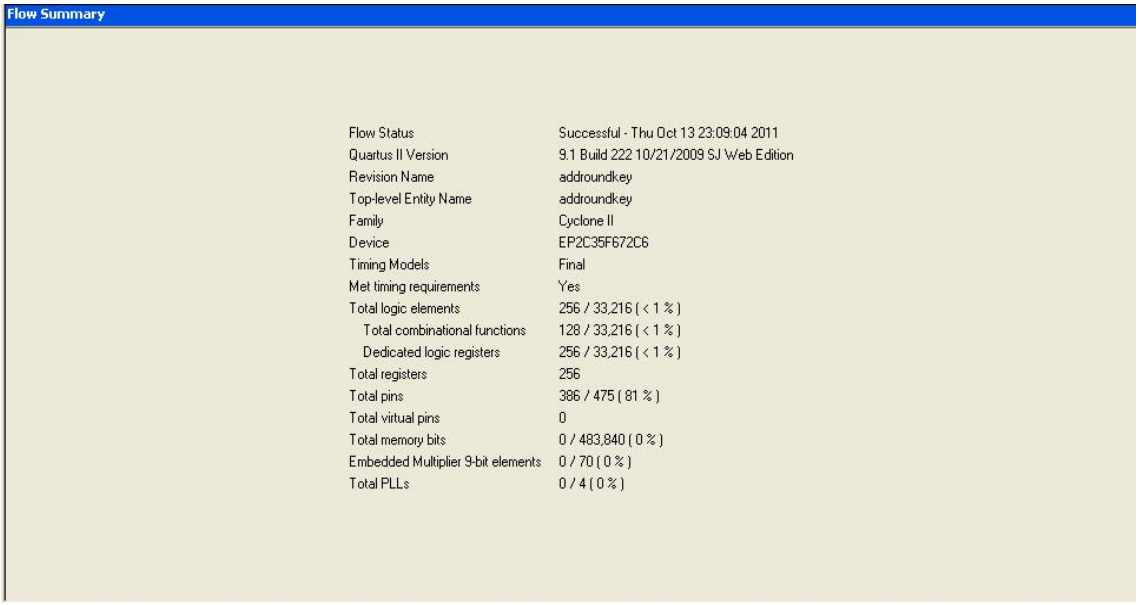


Figura 4- 8: Simulación AddRoundKey.



Flow Summary

Flow Status	Successful - Thu Oct 13 23:09:04 2011
Quartus II Version	9.1 Build 222 10/21/2009 SJ Web Edition
Revision Name	addroundkey
Top-level Entity Name	addroundkey
Family	Cyclone II
Device	EP2C35F672C6
Timing Models	Final
Met timing requirements	Yes
Total logic elements	256 / 33,216 (< 1 %)
Total combinational functions	128 / 33,216 (< 1 %)
Dedicated logic registers	256 / 33,216 (< 1 %)
Total registers	256
Total pins	386 / 475 (81 %)
Total virtual pins	0
Total memory bits	0 / 483,840 (0 %)
Embedded Multiplier 9-bit elements	0 / 70 (0 %)
Total PLLs	0 / 4 (0 %)

Figura 4- 9: Recursos de FPGA en AddRoundKey.

4.1.1.5 Entidad KeySchedExpansion

La presente entidad como ya se ha explicado anteriormente realiza la expansión de la clave original, generando el número necesario de claves de 32 bits para satisfacer las necesidades de todas las claves del proceso. Como se puede ver en la figura 4-10 la entidad cuenta con las entradas de las señales **rst** y **clk** del tipo std_logic y de la clave original **keyIn** del tipo std_logic_vector (127 downto 0). Para la presente longitud de clave que es de 128 bits la cantidad de subclaves que tienen que generarse es de 43 subclaves, para llevar a cabo este proceso la entidad hace uso de la funciones subByte que tiene como entrada un std_logic_vector (31 downto 0) en donde se divide este vector en palabras de 1 byte y se les aplica la caja sbox a cada uno de los bytes para obtener una palabra de salida, de la misma forma se hace uso de la función rotByte que tiene como entrada un std_logic_vector (31 downto 0) que forma la palabra $[a_0, a_1, a_2, a_3]$ y a la cual se procede a realizar una rotación de los bytes retornando $[a_1, a_2, a_3, a_0]$. En la gráfica podemos observar las primera 10 subclaves generadas y el respectivo tiempo de retardo para la generación de estos resultados. En la figura 4-11 podemos observar las estadísticas del comportamiento de la entidad KeySchedExpansion.

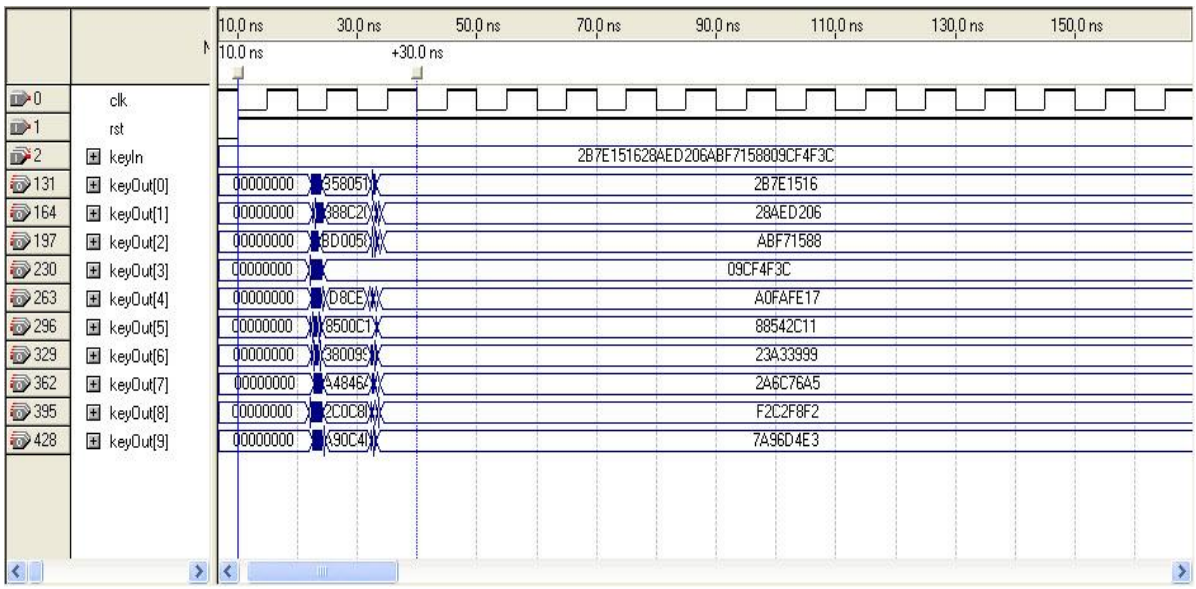


Figura 4- 10: Simulación KeySchedExpansion.

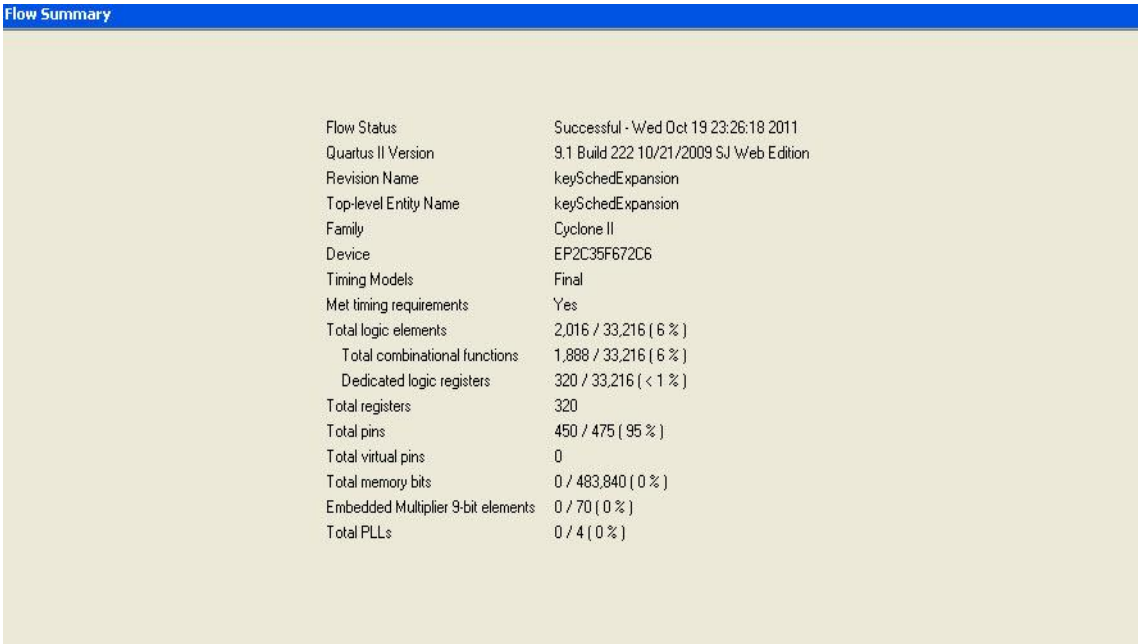


Figura 4- 11: Recursos de FPGA en KeySchedExpansion.

4.1.2 Envió de datos

El envío de los datos encriptados desde la tarjeta hacia la aplicación en la PC se realiza mediante el transmisor que es parte del circuito (UART) Universal Asynchronous Receiver Transmitter, que es el circuito lógico necesario para la comunicación entre interfaces seriales, el cual es aplicado en conjunto con el estándar (EIA) Electronic Industries Alliance RS-232 en el que especifica las características eléctricas, mecánicas y funcionales para la comunicación de datos entre dos equipos. El transmisor es esencialmente un registro de desplazamiento en donde se cargan los datos en paralelo y son desplazados bit a bit a una tasa de transmisión específica. La trasmisión inicia con un bit el cual es '0' y es conocido con el bit de inicio, seguido por los bits de datos los cuales pueden ser 6, 7 o 8 bits, más un bit opcional que es conocido como el bit de paridad que es utilizado para la detección de errores. Para la paridad impar este bit es seteado a '0' cuando en los bits de datos existe un número impar de bit '1'; para la paridad par este bit es seteado a '1' cuando existe un número par de bits '1'. Antes de realizarse la transmisión el trasmisor y receptor deben estar de acuerdo en una serie de parámetros tales como, la tasa de transferencias (número de bits por segundo), número de bits de datos, bits de parada, uso de paridad. Para el presente proyecto se utilizó una tasa de transferencia de 19200 bits por segundo, 8 bits de datos, 1 bit de parada y sin bit de paridad.

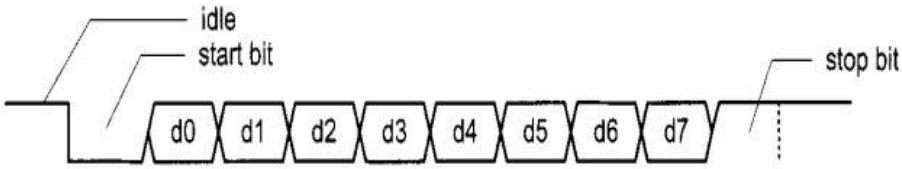


Figura 4- 12: Transmisión de bytes [7].

4.1.3 Recepción de datos

La recepción de los datos en la tarjeta DE2 está controlada mediante una FSM (Finite State Machine) cuyos principales estados son **start**, **data**, y **stop** en donde se representan los procesos de bit de inicio, bits de datos y bit de parada. En la entidad contenedora de esta FSM se han definido dos constantes DBIT constante que indica el número de bits de datos y SB_TICK constante que indica el número de pulsos necesarios para detectar el bit de parada. Este valor puede ser 16, 24, y 32 para 1, 1.5 y 2 bits de parada respectivamente. s_tick es la señal habilitadora la misma que es obtenida desde un circuito en el cual se genera la tasa de muestreo de los bits. Existen dos registros contadores representados por **s** y **n**, el registro s almacena el número de pulsos muestreados y cuenta hasta 7 en el estado **start**, hasta 15 en el estado **data** y hasta SB_TICK en el estado **stop**. El registro **n** almacena el número de bits de datos recibidos.

El estado data, los bits recuperados están dentro de un registro representado por **b** y la señal rx_done_tick es activada en un ciclo de reloj después que el proceso de recepción es completado.

4.2 Desarrollo del Software en Java

Dado que el lenguaje Java nos da la posibilidad de tener una amplia gama de recursos y librerías para facilitarnos el desarrollo del software se ha procedido a desarrollar bajo este lenguaje la parte correspondiente a desencriptación y la comunicación con la tarjeta DE2 mediante vía serial para el envío de los datos a encriptar y la recepción de los datos encriptados.

4.2.1 Algoritmo de descifrado AES

Para la implementación de la desencriptación, se utilizaron los APIs Java para encriptación definido en dos paquetes:

- javax.crypto.Cipher
- javax.crypto.spec.SecretKeySpec

Que en este caso nos permiten obtener una instancia de los objetos:

- Cipher c = Cipher.getInstance("AES")
- SecretKeySpec sks = new SecretKeySpec(key, "AES")

Con los cuales se puede realizar la invocación de la función init en modo de encriptación:

- c.init(Cipher.DECRYPT_MODE, sks)

Para luego obtener el resultado de la encriptación en un tipo de dato byte:

- cipherText = c.doFinal(cipherText)

4.2.2 Envío de datos

Para realizar el envío de los datos desde el PC a la tarjeta DE2 se utilizó el API de java definido en el paquete:

- javax.comm

Primeramente se definen las variables que se utilizan para controlar la comunicación.

```
Enumeration ports;
CommPortIdentifier pID;
OutputStream outStream;
SerialPort serPort;
```

Luego se procede a obtener un identificador del puerto serial que se utilizar para el envío de los datos mediante la sentencia:

```
pID = CommPortIdentifier.getPortIdentifier("COM1");
```

Una vez obtenido este identificador procedemos a realizar la apertura de este puerto en donde se indica los parámetros bajo los cuales se realiza la comunicación serial:

```
serPort = (SerialPort)pID.open("Writer",2000);
outStream = serPort.getOutputStream();
```

```
serPort.setSerialPortParams(19200,
                            SerialPort.DATABITS_8,
                            SerialPort.STOPBITS_1,
                            SerialPort.PARITY_NONE);
```

Establecida la comunicación serial entre la PC y la tarjeta se procede a realizar el envío o la escritura de los datos en el puerto serial mediante la sentencia:

```
outStream.write(Character.toString(messageChar).getBytes());
```

4.2.3 Recepción de datos.

Al igual como en el envío de datos para realizar la recepción de los datos enviados desde la tarjeta DE2 se utilizó el API de java definido en el paquete:

- javax.comm

  Definimos las variables necesarias para el control de la recepción de los datos:

```
Enumeration ports;
CommPortIdentifier pID;
InputStream inStream;
SerialPort serialPort;
```

Procedemos a obtener un identificador del puerto mediante el cual se realiza la lectura de los datos:

```
pID = CommPortIdentifier.getPortIdentifier("COM1");
```

Obtenido el identificador procedemos a realizar la lectura de lo que se ha escrito en el puerto mediante el control de un hilo que estará censando el momento en el que se produzca un evento:

```
SerialPortEvent.DATA_AVAILABLE
```

Para luego proceder a obtener los datos que han sido escritos en el puerto estableciendo la conexión con los parámetros previamente configurados para llevar a cabo la comunicación serial:

```
serialPort = (SerialPort)pID.open("Reader", 2000);

inStream = serialPort.getInputStream();

serialPort.addEventListener(this);

serialPort.notifyOnDataAvailable(true);

serialPort.setSerialPortParams(19200,

                    SerialPort.DATABITS_8,

                    SerialPort.PARITY_NONE,

                    SerialPort.STOPBITS_1);
```

La lectura de los datos escritos en el puerto serial se lleva a cabo mediante la sentencia:

```
int numBytes = inStream.read(readBuffer);
```

4.3. Protocolo RS-232

Figura 4- 13: Interfaz RS-232.

Es una interfaz para el intercambio serial de datos entre un DTE (Equipo Terminal de Datos) y un DCE (Equipo de Comunicación de Datos)

Todas las normas RS-232 cumplen con los siguientes niveles de voltaje:

- Un 1 lógico es un voltaje comprendido entre -5v y -15v en el transmisor y entre -3v y -25v en el receptor.
- Un 0 lógico es un voltaje comprendido entre +5v y +15v en el transmisor y entre +3v y +25v en el receptor.

En el estándar no se hace referencia al tipo de conector que debe usarse. Sin embargo los conectores más comunes son el DB-25 (25 pines) y el DB-9

(9 pines). El conector hembra debe estar asociado con el DCE y el macho en el DTE.

Cada pin puede ser de entrada o de salida, teniendo una función específica cada uno de ellos. Las más importantes son:

Pin	Función
TXD	Transmitir Datos
RXD	Recibir Datos
DTR	Terminal de datos listo
DSR	Equipo de datos listo
RTS	Solicitud de envío
CTS	Libre para envío
DCD	Detección de Portadora

Tabla 4-1 Pines de un conector serial DB-9.

Las señales TXD, DTR y RTS son de salida, mientras que RDX, DSR, CTS y DCD son de entrada. La masa de referencia para todas las señales es SG (Señal de tierra).

Pin en DB -25	Pin en DB - 9	Señal	Descripción	E/S
1	1	-	Masa chasis	-
2	3	TXD	Transmit Data	S
3	2	RXD	Receive Data	E
4	7	RTS	Request To Send	S
5	8	CTS	Clear To Send	E
6	6	DSR	Data Set Ready	E
7	5	SG	Signal Ground	-
8	1	CD/DCD	(Data) Carried Detect	E
15	-	TXC(*)	Transmit Clock	S

17	-	RXC(*)	Receive Clock	E
20	4	DTR	Data Terminal Ready	S
22	9	RI	Ring Indicator	E
24	-	RTXC(*)	Transmit/Receive Clock	S

Tabla 4-2: Señales seriales en los conectores DB-25 y DB-9.

(*)=Normalmente no conectados en el DB-25

4.3. Pruebas y Resultados

Para la realización de la pruebas se hace uso de una aplicación la cual permite la comunicación entre el PC y la tarjeta, estas pruebas se realizaron con diferente tipos de datos tanto para el texto como para la clave, se comprobó que para la dos formas de encriptación enviando los datos en formato hexadecimal y ascii el algoritmo funciona correctamente.

Prueba 1: Cadena de hexadecimales para el texto y para la clave

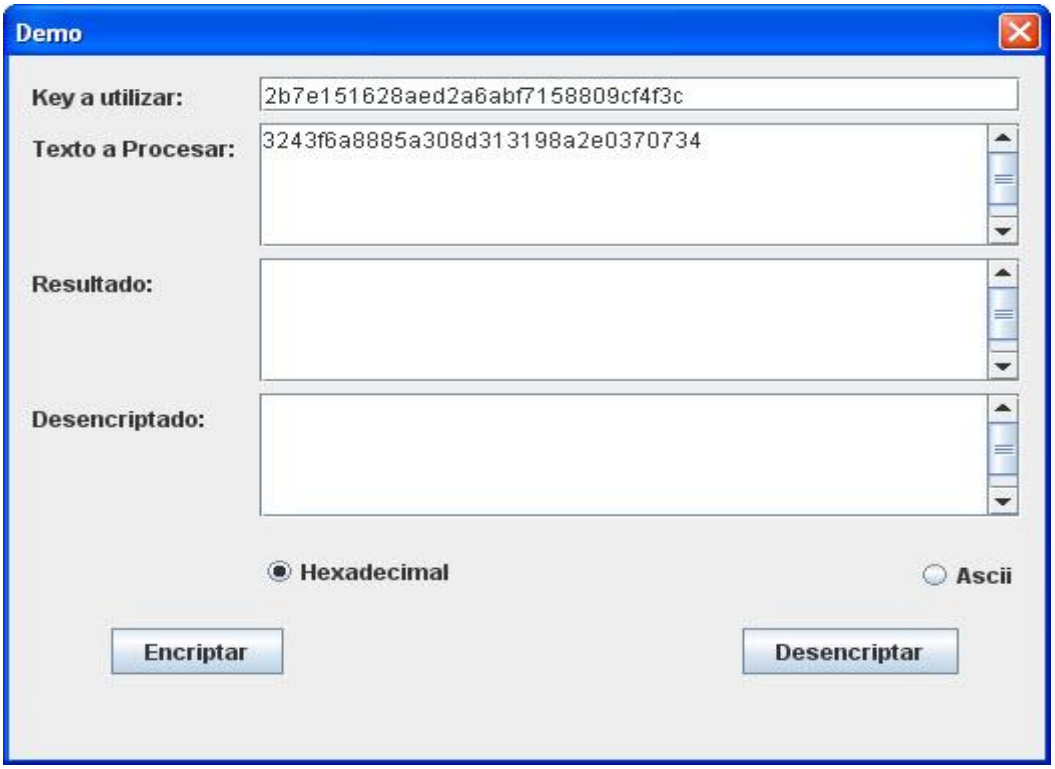


Figura 4- 14: Prueba 1

Resultado:

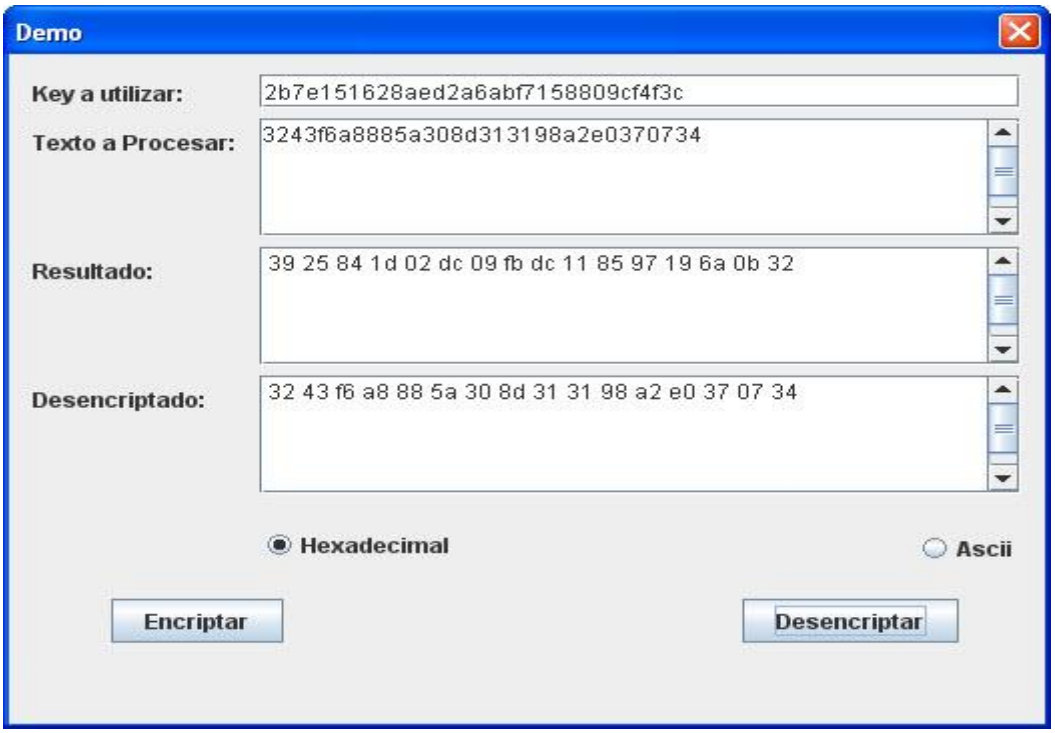


Figura 4- 15 : Resultado Prueba 1

Prueba 2: Cadena de números tanto para la clave como para el texto a encriptar.

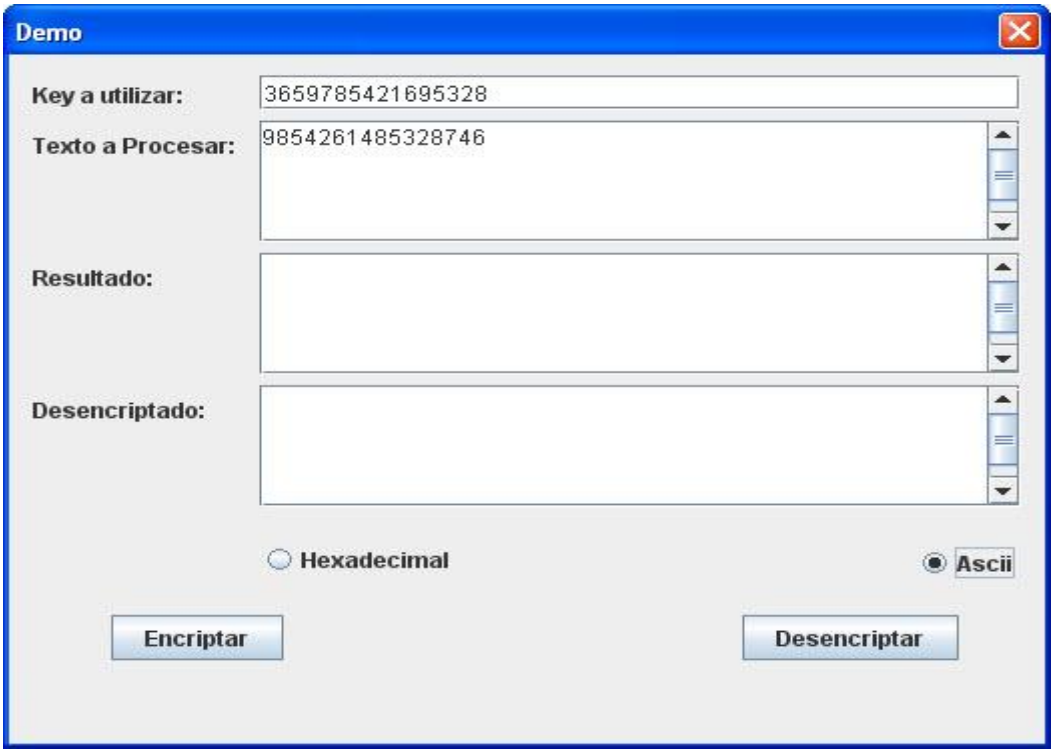


Figura 4- 16: Prueba 2

Resultado:

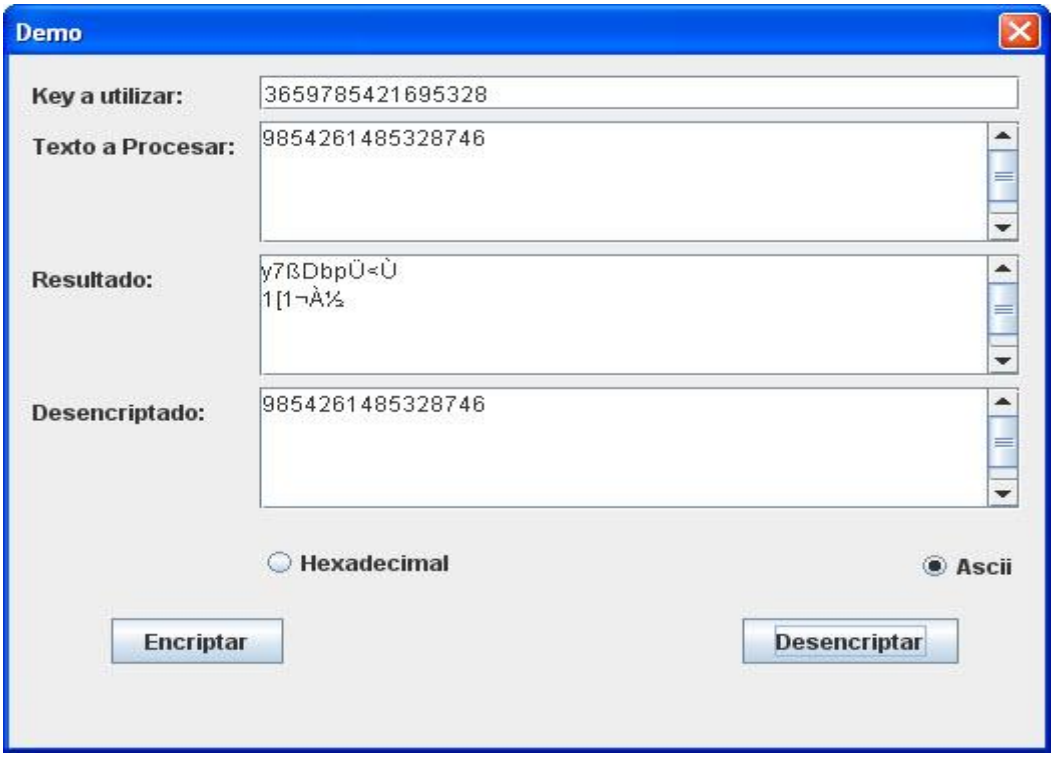


Figura 4- 17: Resultado Prueba 2

Prueba 3: Letras minúsculas tanto para la clave como para el texto a encriptar.

Figura 4- 18: Prueba 3

Resultado:

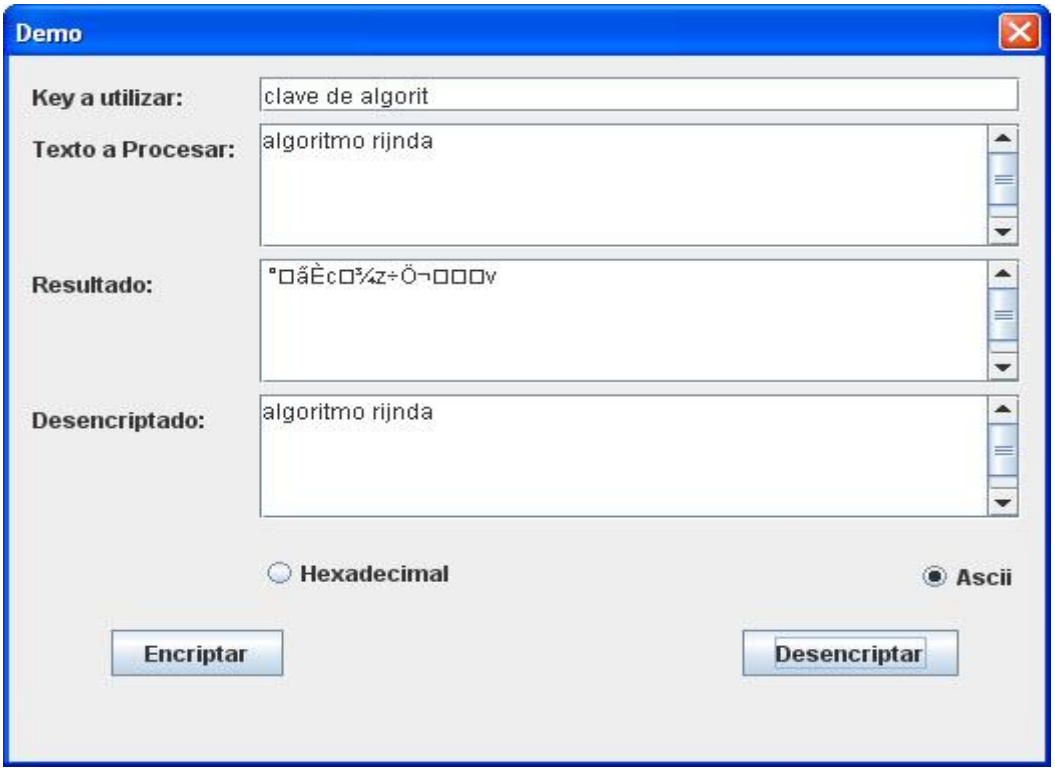


Figura 4- 19: Resultado prueba 3

Prueba 4: Letras mayúsculas tanto para la clave como para el texto a cifrar.

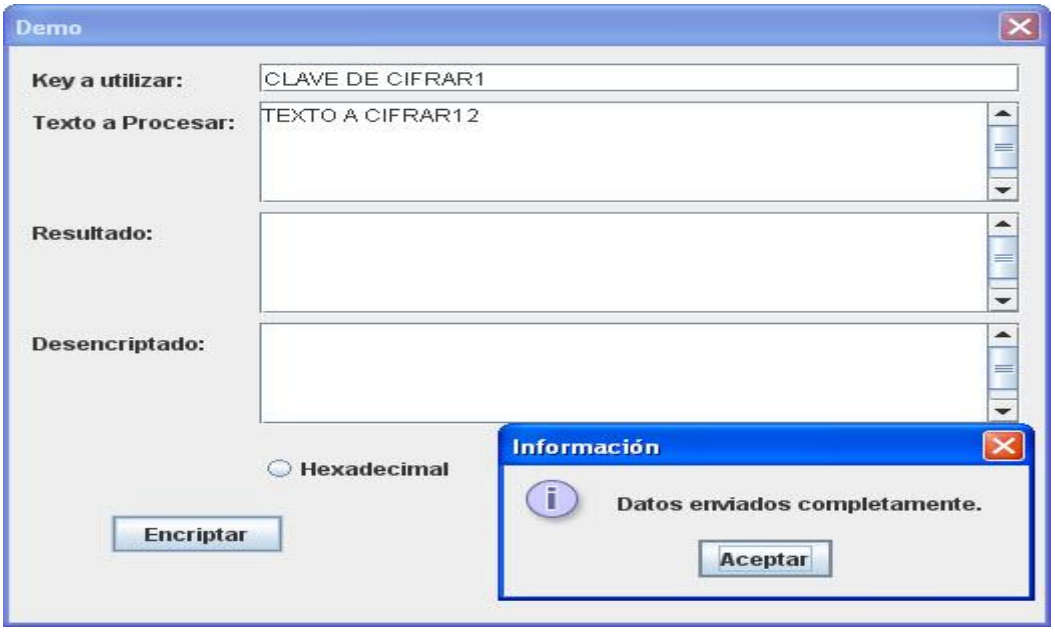


Figura 4- 20: Prueba 4

Resultado:

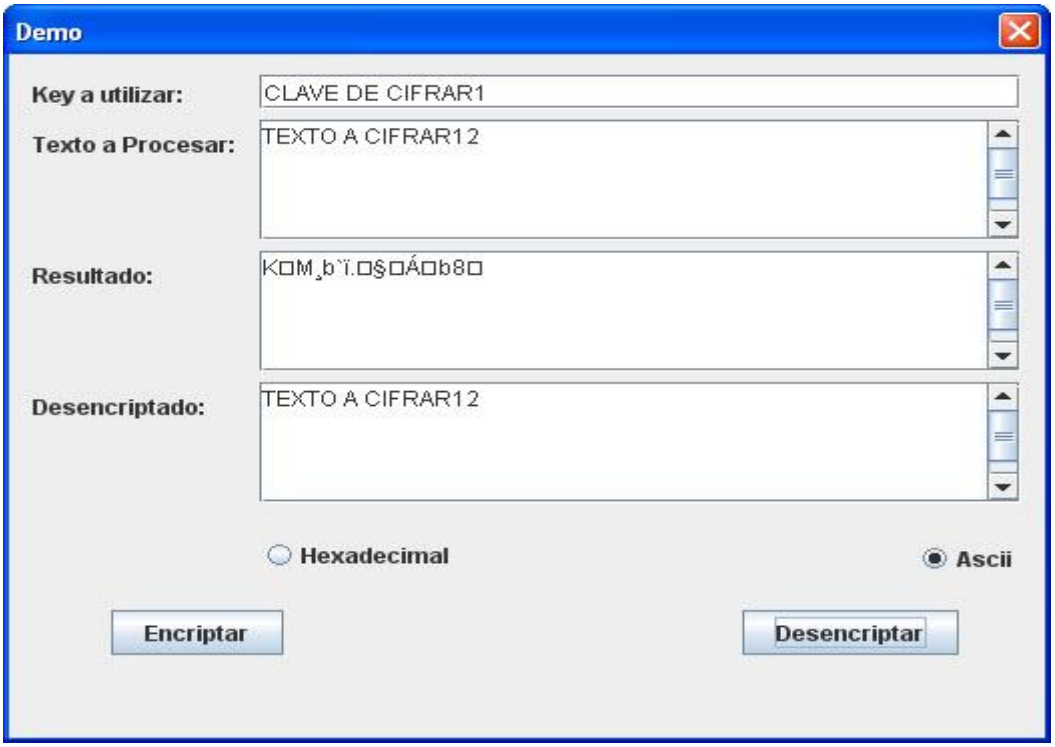


Figura 4- 21: Resultado prueba 4

Prueba 5: Combinación de letras, numero y caracteres especiales tanto para la clavo como el texto a encriptar.

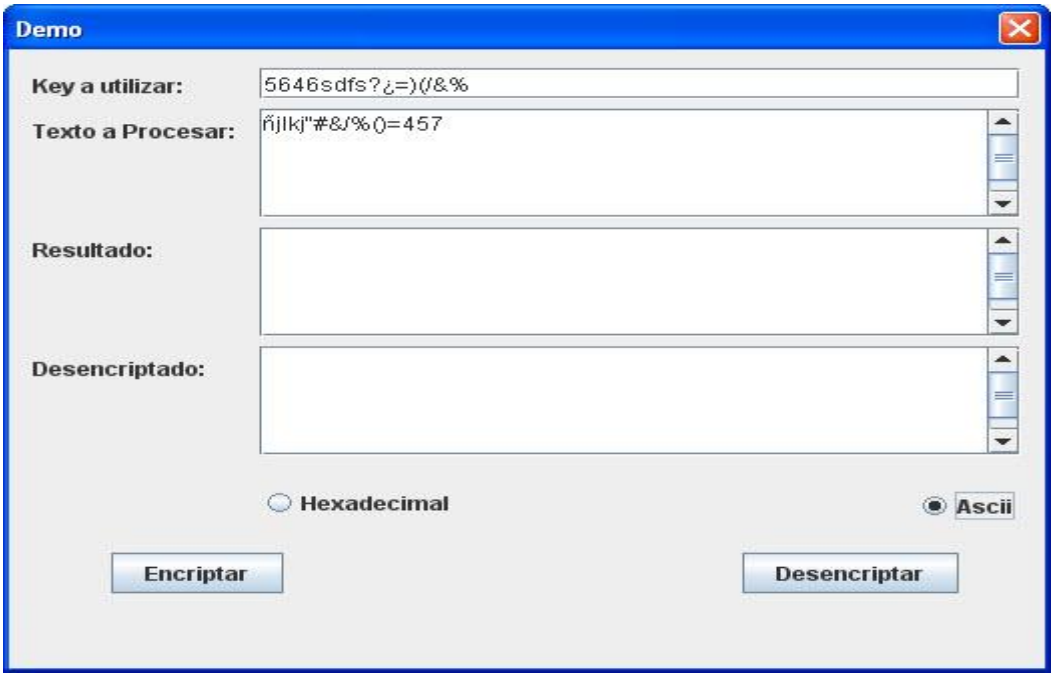


Figura 4- 22: Prueba 5

Resultado:

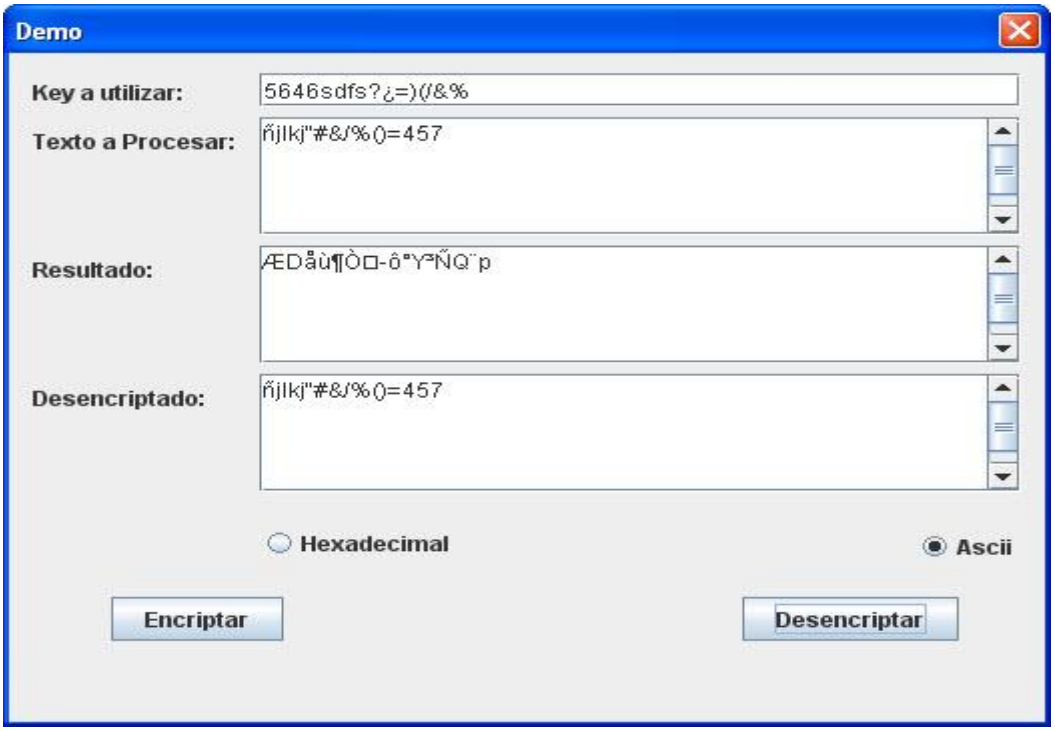


Figura 4- 23: Resultado prueba 5

Pruebas	Texto a encriptar	Porcentaje de Efectifividad de Encriptacion en FPGA
Prueba 1	3243f6a8885a308d313198a2e0370734	100%
Prueba 2	9854261485328746	100%
Prueba 3	Algoritmo rijnda	100%
Prueba 4	TEXTO A CIFRAR12	100%
Prueba 5	njlkj"#&/%()457	100%

Tabla 4-3: Efectividad de Encriptación en FPGA.

CONCLUSIONES

1. El algoritmo AES es unos de los algoritmos más utilizados por su gran seguridad y estabilidad, debido que en la actualidad intentar quebrantar el algoritmo que tenga como clave 128, 192 o 256 bits de clave por fuerza bruta resulta impracticable computacionalmente.

2. Se comprendió la estructura y el funcionamiento del algoritmo AES lo cual permitió que se realice una síntesis correcta y una implementación exitosa de este algoritmo usando la tecnología FGPA y el lenguaje programación VHDL.

3. Analizando el algoritmo de desencriptación AES para realizar su implementación en hardware, se observo que es necesario mucho más código y ciclos a implementar en comparación con otros sistemas de cifrado, a pesar de que para este algoritmo se reutiliza parcialmente el circuito que implementa la encriptación.

4. En la implementación de cada una de una de la funciones que forman parte del algoritmo se logra que se utilicen la menor cantidad de elementos lógicos y bits de memoria de la FPGA, dándose esta utilización mínima de recursos se da la posibilidad de que este algoritmo sea

implementado en dispositivos de menores recursos de esta u otra familia de FPGA.

5. Al no estar implementando la desencriptación a nivel de FPGA se está logrando saturar menos los recursos del FPGA, de forma que se produce un mejor rendimiento en el proceso de encriptación.

RECOMENDACIONES

1. Para futuros trabajos se recomienda realizar la investigación considerando implementar la parte del algoritmo que realiza la desencriptación en hardware.

2. Ampliar el campo de la investigación para la implementación en hardware de los sistemas de cifrado, con el fin de abastecer la demanda de soluciones con rendimiento económico, ya que se cree que será la nueva tendencia de las soluciones de seguridad.

3. Al utilizar VHDL para modelar el funcionamiento del circuito, se debe tener en cuenta no exceder en alto nivel con el que se lo diseña de manera que se obtenga un diseño eficiente y optimo, debido a que se puede provocar que el circuito no esté descrito de forma correcta, lo cual da paso a la realización de operaciones incorrectas o la generación de una mayor cantidad de elementos lógicos al momento de sintetizar el diseño.

4. Para futuros trabajos se recomienda realizar la implementación del algoritmos de encriptación AES aumentando el tamaño de la clave a 192 y 256 bits, para lo cual se debe tomar en cuenta que para el caso de tamaños de clave de 256 bits se realiza un proceso más complejo para la

generación de las subclaves, debiéndose realizar cambios únicamente el bloque SubKey (Generación de subclaves)

ANEXOS

Entidad subbytes

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use work.aes_pack.all;

entity subbytes is
      port(
            clk         : in std_logic;
            rst         : in std_logic;
            matrizIn    : in matriz_data;
            matrizOut   : out matriz_data
            );
end entity subbytes;
------------------------------------------------------------------
---- Architecture para subbytes
------------------------------------------------------------------
--
architecture behave of subbytes is
------------------------------------------------------------------
--
-- Funcion subbytes
------------------------------------------------------------------
--
Function Sub_Bytes (signal matriz:matriz_data) Return matriz_data
is
            Variable S:matriz_data;
            Begin
            For i in 0 to 3 Loop
                  For j in 0 to 3 Loop
                        S(j)(i):= sbox(matriz(j)(i));
                  End Loop;
            End Loop;
            Return S;
      End Sub_Bytes;
------------------------------------------------------------------
--
-- Señales
------------------------------------------------------------------
--
      signal sIn : matriz_data;
      signal sOut: matriz_data;
begin
```

```
------------------------------------------------------------------
--
-- Implementación de subbytes
------------------------------------------------------------------
--
      procIn : process (clk, rst) is
      begin
            if rst = '0' then
                  sIn <= (others => (others => x"00"));
            elsif rising_edge(clk) then
                  sIn <= matrizIn;
            end if;
      end process procIn;
      procOut : process (clk, rst) is
      begin
            if rst = '0'  then
                  matrizOut <= (others => (others => x"00"));
            elsif rising_edge(clk) then
                  matrizOut <= Sub_Bytes(sIn);
            end if;
      end process procOut;
end architecture behave;
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use work.aes_pack.all;
```

Entidad shiftrows

```
entity shiftrows is
      port(
            clk         : in std_logic;
            rst         : in std_logic;
            matrizIn    : in matriz_data;
            matrizOut   : out matriz_data);
end entity shiftrows;
------------------------------------------------------------------
--
-- Architecture para shiftrows
------------------------------------------------------------------
--
architecture behave of shiftrows is
------------------------------------------------------------------
--
-- Tipo de datos
------------------------------------------------------------------
--
Type Vector is array (0 to 3)of std_logic_vector(0 to 7);
------------------------------------------------------------------
--
```

```
-- Función para Shift_Rows
--------------------------------------------------------------------
--
Function Shift_Rows (signal matriz:matriz_data) Return
matriz_data is
      Variable S:matriz_data;
      Variable i,j,n:Integer;
      Variable V:Vector;
      Begin
      n:=4;
      S:=matriz;
      For i in 0 to 3 Loop
            For j in 0 to 3 Loop
                  V(j):= S(i)((j+n+i)mod 4);
            End Loop;
            For j in 0 to 3 Loop
                  S(i)(j):= V(j);
            End Loop;
      End Loop;
      Return S;
End Shift_Rows;
--------------------------------------------------------------------
--
-- Signals
--------------------------------------------------------------------
--    signal sIn : matriz_data;
signal sOutTemp : matriz_data;
signal sOut: matriz_data;
begin




--------------------------------------------------------------------
--
-- Implementación de shiftrows
--------------------------------------------------------------------
--
      procIn : process (clk, rst) is
      begin
            if rst = '0' then
                  sIn <= (others => (others => x"00"));
            elsif rising_edge(clk) then
                  sIn <= matrizIn;
            end if;
      end process procIn;

      procOut : process (clk, rst) is
      begin
            if rst = '0'  then
                  matrizOut <= (others => (others => x"00"));
```

```
            elsif rising_edge(clk) then
                  matrizOut <= Shift_Rows(sIn);
            end if;
      end process procOut;
end architecture behave;
```

Entidad mixcolumns

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use work.aes_pack.all;

entity mixcolumns is
      port(
            clk         : in std_logic;
            rst         : in std_logic;
            matrizIn    : in matriz_data;
            matrizOut   : out matriz_data);
end entity mixcolumns;
-------------------------------------------------------------------
--
-- Architecture para mixcolumns
-------------------------------------------------------------------
--
architecture behave of mixcolumns is
-------------------------------------------------------------------
--
-- Funcion Mix_Columns
-------------------------------------------------------------------
--
 Function Mix_Columns (signal matriz:matriz_data) Return
matriz_data is
      Variable S:matriz_data;
      Begin
      For i in 0 to 3 Loop
            For j in 0 to 3 Loop
                  S(i)(j):= mult2(matriz(i)(j)) xor
mult3(matriz((i+1) mod 4)(j)) xor matriz((i+2) mod 4)(j) xor
matriz((i+3) mod 4)(j);
            End Loop;
      End Loop;
      Return S;
End Mix_Columns;


-------------------------------------------------------------------
--
-- Señales
```

```
------------------------------------------------------------------
--
      signal s : matriz_data;
      signal result: matriz_data;
begin
------------------------------------------------------------------
---- Implementación de mixcolumns
------------------------------------------------------------------
--
      procInOut : process(clk) is
      begin
            if rising_edge(clk) then
                  s <= matrizIn;
            end if;
      end process procInOut;
      procOut : process (clk) is
      begin
            if rising_edge(clk) then
                  matrizOut <= Mix_Columns(s);
            end if;
      end process procOut;
end architecture behave;
```

Entidad addrounkey

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use work.aes_pack.all;

entity addroundkey is
      port(
            clk          : in std_logic;
            rst          : in std_logic;
            matrizKey    : in matriz_data;
            matrizIn     : in matriz_data;
            matrizOut    : out matriz_data);
end entity addroundkey;
------------------------------------------------------------------
--
-- Architecture para addroundkey
------------------------------------------------------------------
--
architecture behave of addroundkey is
------------------------------------------------------------------
--
--Funcion Round_Key
------------------------------------------------------------------
--
```

```
Function Round_Key (Signal matriz,Key:matriz_data) return
matriz_data is
      Variable S: matriz_data;
      Begin
            For i in 0 to 3 Loop
                  For j in 0 to 3 Loop
                        S(i)(j):= matriz(i)(j) xor Key(i)(j);
                  End Loop;
            End Loop;
      return S;
End Round_Key;
------------------------------------------------------------------
--
-- Signals
------------------------------------------------------------------
--
      signal sIn : matriz_data;
      signal sOut: matriz_data;
      signal key : matriz_data;
begin
------------------------------------------------------------------
--
-- Implementación de addroundkey
------------------------------------------------------------------
--
      procIn : process (clk, rst) is
      begin
            if rst = '0' then
                  sIn    <= (others => (others => x"00"));
            elsif rising_edge(clk) then
                  sIn    <= matrizIn;
                  key    <= matrizKey;
            end if;
      end process procIn;

      procOut : process (clk, rst) is
      begin
            if rst = '0'  then
                  matrizOut <= (others => (others => x"00"));
            elsif rising_edge(clk) then
                  matrizOut <= Round_Key(sIn,key);
            end if;
      end process procOut;
end architecture behave;
```

Package aes_pack

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
```

```
package aes_pack is

type state_array is array (0 to 3) of std_logic_vector(7 downto
0);
type matriz_data is array (0 to 3) of state_array;
type array_data is array (1 to 10) of matriz_data;
subtype slv_8 is std_logic_vector(7 downto 0);
type keyschedType is array (0 to 43) of std_logic_vector(31
downto 0);
type RconType is array (0 to 9) of std_logic_vector(31 downto 0);

      type state_machine is
(st1,st2,st3,st4,st5,st6,st7,st8,st9,st10,st11,st12,st13,st14,st1
5,st16,st17,st18,st19,st20,st21,st22,st23,st24,st25,st26,st27,st2
8,st29,st30,st31,st32);

    type machine is
(st1,st2,st3,st4,st5,st6,st7,st8,st9,st10,st11,st12,st13,st14,st1
5,st16,st17);



      constant Rcon : RconType:= (x"01000000",
                                  x"02000000",
                                  x"04000000",
                                  x"08000000",
                                  x"10000000",
                                  x"20000000",
                                  x"40000000",
                                  x"80000000",
                                  x"1b000000",
                                  x"36000000"
                                  );

      function sbox(dir: std_logic_vector(7 downto 0)) return
std_logic_vector;

      function rotByte (I : std_logic_vector(31 downto 0)) return
std_logic_vector;

      function subByte (I : std_logic_vector(31 downto 0))  return
std_logic_vector;

      function mult2 (I : std_logic_vector(7 downto 0)) return
std_logic_vector;

      function mult3 (I : std_logic_vector(7 downto 0)) return
std_logic_vector;
```

```
      function rotVector (v : std_logic_vector(7 downto 0)) return
std_logic_vector;

      function rotarVector (v : std_logic_vector(7 downto 0))
return std_logic_vector;



end aes_pack;

package body aes_pack is --======== Fin de package header =======-
-

function sbox(dir: std_logic_vector(7 downto 0)) return
std_logic_vector is
variable data: bit_vector(7 downto 0);
variable data_stdlogic: std_logic_vector(7 downto 0);

begin
case dir is
  when "00000000" => data := X"63"; --0
  when "00000001" => data := X"7C"; --1
  when "00000010" => data := X"77"; --2
  when "00000011" => data := X"7B"; --3
  when "00000100" => data := X"F2"; --4
  when "00000101" => data := X"6B"; --5
  when "00000110" => data := X"6F"; --6
  when "00000111" => data := X"C5"; --7
  when "00001000" => data := X"30"; --8
  when "00001001" => data := X"01"; --9
  when "00001010" => data := X"67"; --10
  when "00001011" => data := X"2B"; --11
  when "00001100" => data := X"FE"; --12
  when "00001101" => data := X"D7"; --13
  when "00001110" => data := X"AB"; --14
  when "00001111" => data := X"76"; --15
  when "00010000" => data := X"CA"; --16
  when "00010001" => data := X"82"; --17
  when "00010010" => data := X"C9"; --18
  when "00010011" => data := X"7D"; --19
  when "00010100" => data := X"FA"; --20
  when "00010101" => data := X"59"; --21
  when "00010110" => data := X"47"; --22
  when "00010111" => data := X"F0"; --23
  when "00011000" => data := X"AD"; --24
  when "00011001" => data := X"D4"; --25
  when "00011010" => data := X"A2"; --26
  when "00011011" => data := X"AF"; --27
  when "00011100" => data := X"9C"; --28
  when "00011101" => data := X"A4"; --29
  when "00011110" => data := X"72"; --30
```

```
when "00011111" => data := X"C0"; --31
when "00100000" => data := X"B7"; --32
when "00100001" => data := X"FD"; --33
when "00100010" => data := X"93"; --34
when "00100011" => data := X"26"; --35
when "00100100" => data := X"36"; --36
when "00100101" => data := X"3F"; --37
when "00100110" => data := X"F7"; --38
when "00100111" => data := X"CC"; --39
when "00101000" => data := X"34"; --40
when "00101001" => data := X"A5"; --41
when "00101010" => data := X"E5"; --42
when "00101011" => data := X"F1"; --43
when "00101100" => data := X"71"; --44
when "00101101" => data := X"D8"; --45
when "00101110" => data := X"31"; --46
when "00101111" => data := X"15"; --47
when "00110000" => data := X"04"; --48
when "00110001" => data := X"C7"; --49
when "00110010" => data := X"23"; --50
when "00110011" => data := X"C3"; --51
when "00110100" => data := X"18"; --52
when "00110101" => data := X"96"; --53
when "00110110" => data := X"05"; --54
when "00110111" => data := X"9A"; --55
when "00111000" => data := X"07"; --56
when "00111001" => data := X"12"; --57
when "00111010" => data := X"80"; --58
when "00111011" => data := X"E2"; --59
when "00111100" => data := X"EB"; --60
when "00111101" => data := X"27"; --61
when "00111110" => data := X"B2"; --62
when "00111111" => data := X"75"; --63
when "01000000" => data := X"09"; --64
when "01000001" => data := X"83"; --65
when "01000010" => data := X"2C"; --66
when "01000011" => data := X"1A"; --67
when "01000100" => data := X"1B"; --68
when "01000101" => data := X"6E"; --69
when "01000110" => data := X"5A"; --70
when "01000111" => data := X"A0"; --71
when "01001000" => data := X"52"; --72
when "01001001" => data := X"3B"; --73
when "01001010" => data := X"D6"; --74
when "01001011" => data := X"B3"; --75
when "01001100" => data := X"29"; --76
when "01001101" => data := X"E3"; --77
when "01001110" => data := X"2F"; --78
when "01001111" => data := X"84"; --79
when "01010000" => data := X"53"; --80
when "01010001" => data := X"D1"; --81
```

```
when "01010010" => data := X"00"; --82
when "01010011" => data := X"ED"; --83
when "01010100" => data := X"20"; --84
when "01010101" => data := X"FC"; --85
when "01010110" => data := X"B1"; --86
when "01010111" => data := X"5B"; --87
when "01011000" => data := X"6A"; --88
when "01011001" => data := X"CB"; --89
when "01011010" => data := X"BE"; --90
when "01011011" => data := X"39"; --91
when "01011100" => data := X"4A"; --92
when "01011101" => data := X"4C"; --93
when "01011110" => data := X"58"; --94
when "01011111" => data := X"CF"; --95
when "01100000" => data := X"D0"; --96
when "01100001" => data := X"EF"; --97
when "01100010" => data := X"AA"; --98
when "01100011" => data := X"FB"; --99
when "01100100" => data := X"43"; --100
when "01100101" => data := X"4D"; --101
when "01100110" => data := X"33"; --102
when "01100111" => data := X"85"; --103
when "01101000" => data := X"45"; --104
when "01101001" => data := X"F9"; --105
when "01101010" => data := X"02"; --106
when "01101011" => data := X"7F"; --107
when "01101100" => data := X"50"; --108
when "01101101" => data := X"3C"; --109
when "01101110" => data := X"9F"; --110
when "01101111" => data := X"A8"; --111
when "01110000" => data := X"51"; --112
when "01110001" => data := X"A3"; --113
when "01110010" => data := X"40"; --114
when "01110011" => data := X"8F"; --115
when "01110100" => data := X"92"; --116
when "01110101" => data := X"9D"; --117
when "01110110" => data := X"38"; --118
when "01110111" => data := X"F5"; --119
when "01111000" => data := X"BC"; --120
when "01111001" => data := X"B6"; --121
when "01111010" => data := X"DA"; --122
when "01111011" => data := X"21"; --123
when "01111100" => data := X"10"; --124
when "01111101" => data := X"FF"; --125
when "01111110" => data := X"F3"; --126
when "01111111" => data := X"D2"; --127
when "10000000" => data := X"CD"; --128
when "10000001" => data := X"0C"; --129
when "10000010" => data := X"13"; --130
when "10000011" => data := X"EC"; --131
when "10000100" => data := X"5F"; --132
```

```
when "10000101" => data := X"97"; --133
when "10000110" => data := X"44"; --134
when "10000111" => data := X"17"; --135
when "10001000" => data := X"C4"; --136
when "10001001" => data := X"A7"; --137
when "10001010" => data := X"7E"; --138
when "10001011" => data := X"3D"; --139
when "10001100" => data := X"64"; --140
when "10001101" => data := X"5D"; --141
when "10001110" => data := X"19"; --142
when "10001111" => data := X"73"; --143
when "10010000" => data := X"60"; --144
when "10010001" => data := X"81"; --145
when "10010010" => data := X"4F"; --146
when "10010011" => data := X"DC"; --147
when "10010100" => data := X"22"; --148
when "10010101" => data := X"2A"; --149
when "10010110" => data := X"90"; --150
when "10010111" => data := X"88"; --151
when "10011000" => data := X"46"; --152
when "10011001" => data := X"EE"; --153
when "10011010" => data := X"B8"; --154
when "10011011" => data := X"14"; --155
when "10011100" => data := X"DE"; --156
when "10011101" => data := X"5E"; --157
when "10011110" => data := X"0B"; --158
when "10011111" => data := X"DB"; --159
when "10100000" => data := X"E0"; --160
when "10100001" => data := X"32"; --161
when "10100010" => data := X"3A"; --162
when "10100011" => data := X"0A"; --163
when "10100100" => data := X"49"; --164
when "10100101" => data := X"06"; --165
when "10100110" => data := X"24"; --166
when "10100111" => data := X"5C"; --167
when "10101000" => data := X"C2"; --168
when "10101001" => data := X"D3"; --169
when "10101010" => data := X"AC"; --170
when "10101011" => data := X"62"; --171
when "10101100" => data := X"91"; --172
when "10101101" => data := X"95"; --173
when "10101110" => data := X"E4"; --174
when "10101111" => data := X"79"; --175
when "10110000" => data := X"E7"; --176
when "10110001" => data := X"C8"; --177
when "10110010" => data := X"37"; --178
when "10110011" => data := X"6D"; --179
when "10110100" => data := X"8D"; --180
when "10110101" => data := X"D5"; --181
when "10110110" => data := X"4E"; --182
when "10110111" => data := X"A9"; --183
```

```
when "10111000" => data := X"6C"; --184
when "10111001" => data := X"56"; --185
when "10111010" => data := X"F4"; --186
when "10111011" => data := X"EA"; --187
when "10111100" => data := X"65"; --188
when "10111101" => data := X"7A"; --189
when "10111110" => data := X"AE"; --190
when "10111111" => data := X"08"; --191
when "11000000" => data := X"BA"; --192
when "11000001" => data := X"78"; --193
when "11000010" => data := X"25"; --194
when "11000011" => data := X"2E"; --195
when "11000100" => data := X"1C"; --196
when "11000101" => data := X"A6"; --197
when "11000110" => data := X"B4"; --198
when "11000111" => data := X"C6"; --199
when "11001000" => data := X"E8"; --200
when "11001001" => data := X"DD"; --201
when "11001010" => data := X"74"; --202
when "11001011" => data := X"1F"; --203
when "11001100" => data := X"4B"; --204
when "11001101" => data := X"BD"; --205
when "11001110" => data := X"8B"; --206
when "11001111" => data := X"8A"; --207
when "11010000" => data := X"70"; --208
when "11010001" => data := X"3E"; --209
when "11010010" => data := X"B5"; --210
when "11010011" => data := X"66"; --211
when "11010100" => data := X"48"; --212
when "11010101" => data := X"03"; --213
when "11010110" => data := X"F6"; --214
when "11010111" => data := X"0E"; --215
when "11011000" => data := X"61"; --216
when "11011001" => data := X"35"; --217
when "11011010" => data := X"57"; --218
when "11011011" => data := X"B9"; --219
when "11011100" => data := X"86"; --220
when "11011101" => data := X"C1"; --221
when "11011110" => data := X"1D"; --222
when "11011111" => data := X"9E"; --223
when "11100000" => data := X"E1"; --224
when "11100001" => data := X"F8"; --225
when "11100010" => data := X"98"; --226
when "11100011" => data := X"11"; --227
when "11100100" => data := X"69"; --228
when "11100101" => data := X"D9"; --229
when "11100110" => data := X"8E"; --230
when "11100111" => data := X"94"; --231
when "11101000" => data := X"9B"; --232
when "11101001" => data := X"1E"; --233
when "11101010" => data := X"87"; --234
```

```
  when "11101011" => data := X"E9"; --235
  when "11101100" => data := X"CE"; --236
  when "11101101" => data := X"55"; --237
  when "11101110" => data := X"28"; --238
  when "11101111" => data := X"DF"; --239
  when "11110000" => data := X"8C"; --240
  when "11110001" => data := X"A1"; --241
  when "11110010" => data := X"89"; --242
  when "11110011" => data := X"0D"; --243
  when "11110100" => data := X"BF"; --244
  when "11110101" => data := X"E6"; --245
  when "11110110" => data := X"42"; --246
  when "11110111" => data := X"68"; --247
  when "11111000" => data := X"41"; --248
  when "11111001" => data := X"99"; --249
  when "11111010" => data := X"2D"; --250
  when "11111011" => data := X"0F"; --251
  when "11111100" => data := X"B0"; --252
  when "11111101" => data := X"54"; --253
  when "11111110" => data := X"BB"; --254
  when "11111111" => data := X"16"; --255
  when others => null;
end case;
data_stdlogic := to_StdLogicVector(data);
return data_stdlogic;
end function sbox;


 function rotByte (I : std_logic_vector(31 downto 0)) return
std_logic_vector is
      variable result : std_logic_vector(31 downto 0);
    begin
      result(31 downto 24)  := I(23 downto 16);
      result(23 downto 16)  := I(15 downto 8);
      result(15 downto 8)   := I(7 downto 0);
      result(7 downto 0)    := I(31 downto 24);
    return result;
 end rotByte;

function subByte (I : std_logic_vector(31 downto 0))  return
std_logic_vector is
      variable result : std_logic_vector(31 downto 0);
    begin
      result(7 downto 0)  := sbox(I(7 downto 0));
      result(15 downto 8) := sbox(I(15 downto 8));
      result(23 downto 16):= sbox(I(23 downto 16));
      result(31 downto 24):= sbox(I(31 downto 24));
    return result;
end subByte;
```

```
function mult2 (I : std_logic_vector(7 downto 0))return
std_logic_vector is
      variable  result : std_logic_vector(7 downto 0);
      variable  temp   : std_logic_vector(7 downto 0);
    begin
       result := (I(6),I(5),I(4),I(3) xor I(7),I(2) xor
I(7),I(1),I(0) xor I(7),I(7));
      return result;
end mult2;

function mult3 (I : std_logic_vector(7 downto 0)) return
std_logic_vector is
      variable result : std_logic_vector(7 downto 0);
    begin
      result := mult2(I) xor I;
      return result;
end mult3;

function rotVector (v : std_logic_vector(7 downto 0)) return
std_logic_vector is
      variable result : std_logic_vector(7 downto 0);
    begin
           for j in 0 to 7 loop
               if j < 4 then
                      result(j+4) := v(j);
                 else
                      result(j-4) := v(j);
               end if;
           end loop;
      return result;
end rotVector;

function rotarVector (v : std_logic_vector(7 downto 0)) return
std_logic_vector is
      variable result : std_logic_vector(7 downto 0);
      variable i : integer := 7;
    begin
           for j in 0 to 7 loop
                 result(i) := v(j);
                 i:=i-1;
           end loop;
      return result;
end rotarVector;

end package body aes_pack;--====== Fin de package body======--
```

BIBLIOGRAFÍA

[1] Oriol Baratech Soler, “Sistema de mensajería instantánea punto apunto mediante cifrado por intercambio de clave de sesión”, Disponible en: http://openaccess.uoc.edu/webapps/o2/bitstream/10609/912/1/38408tfc.pdf, 2006

[2] Kenney David, “Energy Efficiency Analysis and Implementation of AES on an FPGA”, Disponible en: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.148.2354, 2008

[3] Hui Qin, Tsutomu Sasao, Yukihiro Iguchi. “An FPGA Design of AES Encryption Circuit with 128-bit keys”, Disponible en: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.60.6770, 2005

[4] Muñoz Muñoz Alfonso, “Algoritmo Criptografico Rijndael”, Disponible en: http://www.kriptopolis.org/seguridad-europea-para-estados-unidos-rijndael , 2004

[5] AES 128 Encryption/Decryption, http://cegt201.bradley.edu/projects/proj2005/aes128/detailedfunction.html, fecha de consulta Septiembre-2011.

[6] AES Algoritmo (Rijndael) Information, http://csrc.nist.gov/archive/aes/rijndael/wsdindex.html, fecha de consulta Septiembre-2011

[7] Pong P. Chu FPGA Prototyping by VHDL Examples Xilinx Spartan 3 Version, Editorial Jhon Wiley & Sons, 2008

[8] Guzmán Palomino, Romero Zamora Abelardo, Manuel Solbes Ángel Bosch, Alfonso, “Diseño e implementación de algoritmos criptográficos sobre FPGA”, Disponible en: http://eprints.ucm.es/8911/, 2005

[9] K. Gaj, P. Chodowiec, “FPGA and ASIC Implementations of AES”, Disponible en: http://teal.gmu.edu/courses/ECE746/project/S08_Project_resources.htm , 2001

[10] García Lobo Belén M., “Algoritmos Criptográficos”, Disponible en: http://www.uv.es/~montanan/redes/trabajos/algoritmos-criptograficos.ppt, 2002