Universidad Autónoma Metropolitana Unidad Azcapotzalco

División de Ciencias Básicas e Ingeniería

Ingeniería en Computación

Proyecto Terminal de Ingeniería en Computación

Implementación del algoritmo de cifrado AES mediante la metodología Hardware-Software

Proyecto que presenta:

Rogelio Vargas Márquez

Para obtener el titulo de:

Ingeniero en Computación

Asesor del Proyecto:

M. en C. Oscar Alvarado Nava

México, D.F. **Diciembre de 2010**

Resumen

Rijndael es un método criptográfico que ha cobrado una importancia significativa, ya que ha reemplazado a DES, y se ha convertido en un estándar (AES).

AES es el nuevo estándar de cifrado simétrico dispuesto por el NIST, después de un periodo de competencia entre 15 algoritmos sometidos. El 2 de Octubre de 2000 fue designado el algoritmo Rijndael como AES, el estándar reemplazó de TDES, para ser usado en los próximos 20 años.

El siguiente documento surge como parte del proyecto del diseño e implementación de Rijndael mediante la metodología de Codiseño Hardware-Software. La implementación del Hardware se realizo mediante lenguaje VHDL y la parte correspondiente a el Software se realizo en lenguaje C, para posteriormente aplicarlo en un FPGA *XUP Virtex 2 Pro* con la finalidad de agilizar el proceso cifrado de datos.

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

1.1. Motivaciones

Desde que el hombre comenzó a comunicarse con los demás surgió la necesidad de que algunos de sus mensajes sólo fueran conocidos por aquellas personas a las cuales estaban destinados. La necesidad de enviar mensajes de forma que sólo fueran entendidos por los destinatarios hizo que se crearan sistemas de cifrado. Con el tiempo, los sistemas criptográficos fueron avanzando en complejidad, debido a que la necesidad de privacidad y seguridad se han vuelto vitales.

FPGA es el acrónimo de *Field Programmable Gate Array* (Matriz de puertas programable por un usuario en el 'campo' de una aplicación). Se trata de dispositivos electrónicos digitales programables de muy alta densidad.

AES (*Advanced Encryption Standar*) es el nuevo estándar de criptografía simétrica adoptado en el FIPS-197 [1] (*Federal Information Processing Standards*). El algoritmo Rijndael fue elegido principalmente por garantizar seguridad, que significa ser inmune a los ataques conocidos, tener un diseño simple, y poder ser implementado en la mayoría de los escenarios posibles, desde dispositivos con recursos limitados, como *smart cards*, hasta procesadores paralelos. El tiempo a permitido que AES sea adaptado poco a poco, desde los protocolos más usados como SSL, hasta las aplicaciones más especializadas.

La criptografía consiste en que el emisor toma el mensaje que quiere enviar (mensaje en claro) y lo procesa mediante un algoritmo matemático y una clave o llave (secuencia secreta de caracteres) para crear un texto cifrado único. Este texto cifrado viaja por medio del canal de comunicación establecido y llega al destinatario que lo convierte en el mensaje original, utilizando el mismo algoritmo (inverso) y la clave. La clave debe estar disponible en el transmisor y en el receptor simultáneamente durante la comunicación.

La finalidad de la criptografía es, en primer lugar, garantizar el secreto en la comunicación entre dos entidades (personas, organizaciones, etc.) y, en segundo lugar, asegurar que la información que se envía es auténtica en un doble sentido: que el remitente sea realmente quien dice ser y que el contenido del mensaje enviado, habitualmente denominado criptograma, no haya sido modificado en su tránsito.

La Metodología de Codiseño Hardware-Software (MCHS) puede definirse como el diseño en conjunto de hardware y software, esto es, el desarrollo de sistemas heterogéneos, con el objetivo de distribuir una aplicación entre dos o más particiones hardware y software creando un sistema capaz de acelerar una aplicación con los mínimos costos de diseño, desarrollo y prueba posibles. La inclusión de la parte hardware tiene como objetivo principal conseguir acelerar la ejecución de la aplicación. El mantenimiento de partes de la aplicación en software, ejecutado por un microprocesador de propósito general, se debe a que estas partes no son críticas desde el punto de vista de velocidad de ejecución.

1.2. Objetivos

En este reporte, se presenta la implementación de un aplicación para el cifrado de datos mediante el algoritmo de cifrado *AES* , la implementación de la misma se realizará mediante lenguaje VHDL (Hardware) y lenguaje C (Software). El objetivo primordial de este proyecto es la implementación del algoritmo para lograr un mejor desempeño, lo cual disminuirá el tiempo de cifrado respecto a una aplicación puramente en Software.

Para alcanzar el objetivo principal del proyecto, se dividió en los siguientes objetivos particulares:

- **Analizar el algoritmo Rijndael (AES).** Consiste en entender las etapas del proceso que involucra cifrar la información mediante este algoritmo, para posteriormente diseñar e implementar el Hardware y Software correspondientes.

- **Implementación del algoritmo en lenguaje C.** Una vez entendido en su totalidad el algoritmo se implementara en lenguaje C con el fin ver cuales son las etapas de mayor densidad de trabajo, para después realizar pruebas y comparaciones con la implementación en Hardware.

- **Diseño e Implementación de los módulos para el cifrado de datos.** Describe el proceso que se encarga de cifrar una matriz de datos (4x4 bytes para el case de esta implementación). En base a lo anterior describir un algoritmo para llevar acabo dicho proceso y realizar la implementación del mismo en VHDL.

- **Diseño e Implementación del módulo de expansión de claves.** Se encargara de expandir la *Key* (Clave o Llave) original en 11 Subclaves que se utilizan dentro de cada etapa del cifrado.

- **Validación y Pruebas.** Una vez que los módulos son integrados, se verificaron las salidas con ayuda del software que implemento anteriormente, junto con el *Rijndael-Inspector-v1.1-esp*, para verificar la veracidad de las salida obtenida.

1.3. Organización del Reporte

El Reporte está organizado de la siguiente manera: En el Capítulo 2 se presenta una introducción al algoritmo *Rijndael*[1]. Se lleva a cabo en el mismo capítulo una descripción de cada etapa del proceso de cifrado, enumerando los pasos a seguir para llevarlo a cabo. También se muestra como es que se realiza el proceso de la expansión de la *Key*.

En el Capítulo 3 se menciona el análisis de la implementación en Software, se hace mención de la información obtenida de este análisis la cual se ocupo para la implementación en Hardware.

El Capítulo 4 presenta el diseño y la implementación en hardware del algoritmo específico de AES (Rijndael), además se muestran los diagramas y explicación de los circuitos propuestos.

Finalmente las conclusiones se presentan en el capítulo **??** junto con el análisis de la implementaciósan de la simulación en lenguaje VHDL y el trabajo futuro que se puede realizar basado en este proyecto.

Capítulo 2

Proceso de Cifrado

La criptografía es la técnica, bien sea aplicada al arte o la ciencia, que altera las representaciones lingüísticas de un mensaje. En esencia trata de enmascarar las representaciones caligráficas de una lengua, de forma discreta. Si bien, el área de estudio científico que se encarga de ello es la Criptología.

Para ello existen distintos métodos, en donde el más común es el cifrado. Esta técnica enmascara las referencias originales de la lengua por un método de conversión gobernado por un algoritmo que permita el proceso inverso o descifrado de la información. El uso de esta u otras técnicas, permite un intercambio de mensajes que sólo puedan ser leídos por los destinatarios.

El descifrado es el proceso inverso que recupera el texto plano a partir del criptograma y la clave. El protocolo criptográfico especifica los detalles de cómo se utilizan los algoritmos y las claves (y otras operaciones primitivas) para conseguir el efecto deseado. El conjunto de protocolos, algoritmos de cifrado, procesos de gestión de claves y actuaciones de los usuarios, es lo que constituyen en conjunto un criptosistema, que es con lo que el usuario final trabaja e interactúa.

Existen diversos tipos de Criptografía entre los que destacan:

- Criptografía simétrica o convencional.
- Criptografía asimétrica o de clave pública.
- Criptografía de curva elíptica.
- Criptografía híbrida.

2.1. Criptografía simétrica

La criptografía simétrica es un método criptográfico en el cual se usa una misma clave para cifrar y descifrar mensajes. Las dos partes que se comunican han de ponerse de acuerdo de antemano sobre la clave a usar. Una vez ambas tienen acceso a esta clave, el remitente cifra un mensaje usándola, lo envía al destinatario, y éste lo descifra con la misma.

Un buen sistema de cifrado pone toda la seguridad en la clave y ninguna en el algoritmo. En otras palabras, no debería ser de ninguna ayuda para un atacante conocer el algoritmo que se está usando. Sólo si el atacante obtuviera la clave, le serviría conocer el algoritmo. Los algoritmos de cifrado ampliamente utilizados tienen estas propiedades (por ejemplo: GnuPG en sistemas GNU). Dado que toda la seguridad está en la clave, es importante que sea muy difícil adivinar el tipo de clave. Esto quiere decir que el abanico de claves posibles, o sea, el espacio de posibilidades de claves, debe ser amplio.

Actualmente, los ordenadores pueden descifrar claves con extrema rapidez, y ésta es la razón por la cual el tamaño de la clave es importante en los criptosistemas modernos. El algoritmo de cifrado DES usa una clave de 56 bits, lo que significa que hay 2 elevado a 56 claves posibles (72.057.594.037.927.936 claves). Esto representa un número muy alto de claves, pero un ordenador genérico puede comprobar el conjunto posible de claves en cuestión de días. Una máquina especializada puede hacerlo en horas. Algoritmos de cifrado de diseño más reciente como 3DES, Blowfish e IDEA usan claves de 128 bits, lo que significa que existen 2 elevado a 128 claves posibles. Esto equivale a muchísimas más claves, y aun en el caso de que todas las máquinas del planeta estuvieran cooperando, tardarían más tiempo en encontrar la clave que la edad del universo.

El principal problema con los sistemas de cifrado simétrico no está ligado a su seguridad, sino al intercambio de claves. Una vez que el remitente y el destinatario hayan intercambiado las claves pueden usarlas para comunicarse con seguridad, pero ¿qué canal de comunicación que sea seguro han usado para transmitirse las claves? Sería mucho más fácil para un atacante intentar interceptar una clave que probar las posibles combinaciones del espacio de claves. Otro problema es el número de claves que se necesitan. Si tenemos un número n de personas que necesitan comunicarse entre sí, se necesitan n/2 claves para cada pareja de personas que tengan que comunicarse de modo privado. Esto puede funcionar con un grupo reducido de personas, pero sería imposible llevarlo a cabo con grupos más grandes.

2.2. Descripción del algoritmo Rijndael (AES)

El algoritmo fue desarrollado por dos criptólogos belgas, Joan Daemen y Vincent Rijmen, y enviado al proceso de selección AES bajo el nombre Rijndael", un *portmanteau* empaquetado de los nombres de los inventores.

En criptografía, *Advanced Encryption Standard* (AES), también conocido como Rijndael, es un esquema de cifrado por bloque adoptado como un estándar de encriptación por el gobierno de los Estados Unidos, y se espera que sea usado en el mundo entero, como también analizado exhaustivamente, como fue el caso de su predecesor, el Estándar de Encriptación de Datos (DES). Fue adoptado por el Instituto Nacional de Estándares y Tecnología (NIST) como un FIPS (PUB 197) en noviembre del 2001 después de 5 años del proceso de estandarización. Estrictamente hablando, AES no es precisamente Rijndael ya que Rijndael permite un mayor rango de tamaño de bloque y clave; AES tiene un tamaño de bloque fijo de 128 bits y tamaños de llave de 128, 192 o 256 bits, mientras que Rijndael puede ser especificado por una clave que sea múltiplo de 32 bits, con un mínimo de 128 bits y un máximo de 256 bits. La mayoría de los cálculos del algoritmo AES se hacen en un campo finito determinado. En la figura 2.1 describe el funcionamiento general de las etapas principales del proceso de cifrado de los datos.

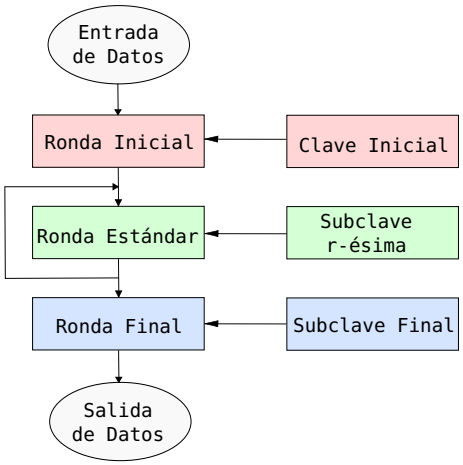

Figura 2.1: Proceso general de cifrado mediante AES.

Recordemos que AES trabaja con bloques de datos de 128 bits y longitudes de claves de 128, 192, 256 bit según el FIPS-197[1]. Además AES tiene 10, 12 o 14 vueltas respectivamente, cada vuelta de AES consiste en la aplicación de una ronda estándar, que consiste de 4 transformaciones básicas, la última ronda es especial y consiste de 3 operaciones básicas, añadiendo siempre una ronda inicial. Por otro lado tenemos el programa de claves o extensión de la clave. Las transformaciones básicas son AddRoundKey, SubByte, ShiftRows, MixColumns, y por último de Key Schedule.

Para esta implementación se ocupara un bloque de datos de 128 bits, con clave de 128 bits, por lo cual serán 10 rondas las que se realizaran.

AES interpreta la entrada de datos como una matriz de 4x4 bytes, esta matriz es la entrada del algoritmo AES y va cambiando en cada una de las rondas, y se llamara Matriz de Estado.

a_{00}	a_{01}	a_{02}	a_{03}
a_{10}	a_{11}	a_{12}	a_{13}
a_{20}	a_{21}	a_{22}	a_{23}
a_{30}	a_{31}	a_{32}	a_{33}

Figura 2.2: Matriz de Estado.

2.2.1. AddRoundKey

En esta transformación se toma la Matriz de Estado y se hace un XOR con la Matriz de Claves correspondiente a cada ronda. Este proceso se realiza individualmente con cada elemento de las matrices, obteniendo una nueva Matriz Estado que contiene el XOR realizado.

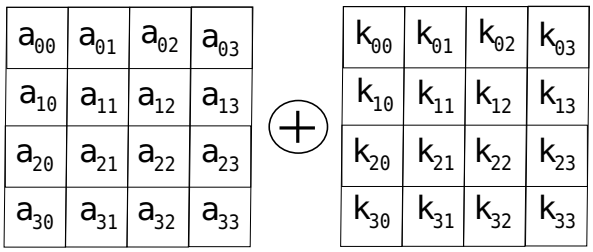

Figura 2.3: XOR entre Matriz Estado y Matriz de Clave

2.2.2. SubByte

En esta ronda se sustituye cada elemento (byte) de la Matriz de Estado por un elemento de la Matriz S-Box AES. Este procedimiento se hace dividiendo el byte de la Matriz Estado en dos partes, los primeros 4 bits indicaran la Fila y los siguientes 4 la columna, el elemento que corresponde a esos valores sera el valor que se sustituye dentro de la matriz $A[ij]$.

Como ejemplo se tomara el elemento $A[23] = S[26]$, el procedimiento se puede ver en la figura 2.4. Cabe aclarar que el procedimiento y análisis matemático para obtener la Matriz S-Box AES se explica con mayor detalle en el FIPS-197 [1].

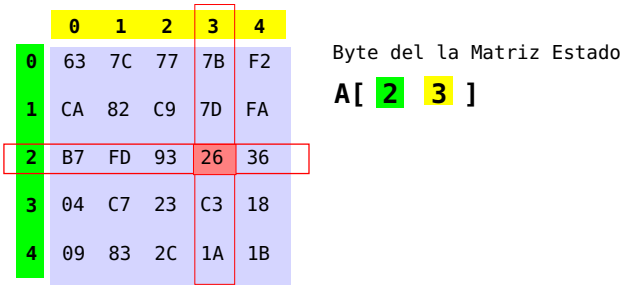

Figura 2.4: Ejemplo de la sustitución de un elemento de $A[ij]$ por uno de $S[ij]$

2.2.3. ShiftRow

La transformación ShifRow se aplica sobre la Matriz de Estado aplicando corrimientos de bytes sobre cada una de las filas de la matriz, obteniendo una nueva Matriz Base con los bytes recorridos. Los corrimientos son circulares hacia la izquierda.

El número de bytes a recorrer depende de la fila de la matriz, estos quedan: 0 bytes para la primer fila, 1 bytes para la segunda fila, 2 bytes para la tercera fila y 3 bytes para la cuarta fila.

En la Figura 2.6 se puede observar un diagrama que describe como se hacen los corrimientos para cada fila.

2.2.4. MixColumns

A cada columna de la matriz $A[ij]$ la multiplica por una columna constante en $GF(2)[x]/(x^4+1)$. MixColumns toma cada columna A, y la manda a otra columna A', que se obtiene al multiplicar A por un polinomio constante

$$c(x) \ni GF(2^8)[x]/(x^4+1), c(x) = 3x^3 + 1x^2 + 1x + 2$$

entonces $A' = A \cdot c(x)$. Este producto puede representarse por la siguiente matriz.

$$\begin{bmatrix} a'_0 \\ a'_1 \\ a'_2 \\ a'_3 \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

Cabe aclarar que este proceso se realiza sobre todas las columnas de la Matriz Estado.

Las transformaciones explicadas anteriormente se aplican durante 9 rondas intermedias y en la ultima ronda se aplican todas las transformaciones a excepción de MixColumns.

	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	F5	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 2.5: Matriz S-Box AES.

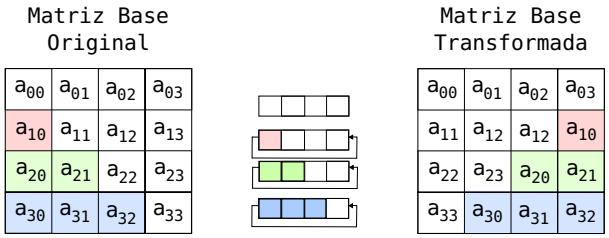

Figura 2.6: Proceso elaborado en la etapa ShiftRow.

2.2.5. Key Schedule - Cálculo de Subclaves.

Este proceso consiste en expandir al Clave inicial en 11 subclaves parciales que se utilizan en la ronda inicial, las 9 principales y la ronda final.

La clave expandida puede verse como una matriz de 44 columnas enumeradas del 0 al 43, donde las primeras 4 columnas son la clave original y cada grupo de 4 columnas representa la Subclave de la ronda correspondiente, como se muestra en la siguiente figura.

El cálculo de cada Subclave se hace de al siguiente manera:

1. Las palabras que ocupan una posición con múltiplo de 4 ($W_4, W_8, ..., W_{40}$) se calculan:

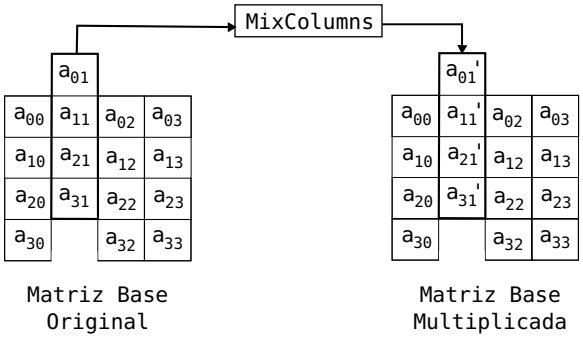

Figura 2.7: Proceso de multiplicación en la etapa MixColumns.

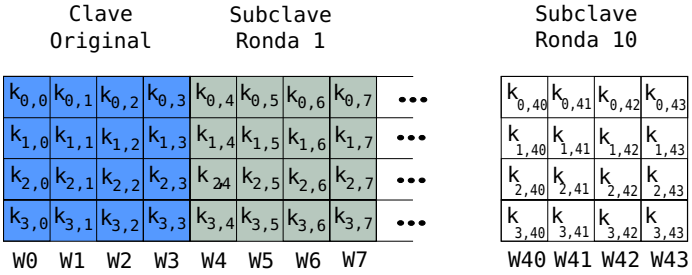

Figura 2.8: División de Columnas de las Subclaves en cada ronda.

a) Aplicando **RotWord** y **SubBytes** a la palabra anterior W_{i-1}

b) Aplicando **XOR** con el resultado obtenido del paso anterior y la palabra de 4 posiciones antes W_{i-4} más una constante de Ronda **Rcon**.

2. Las restantes palabras de 32 bits W_i se calculan haciendo XOR con la palabra anterior W_{i-1} y de cuatro posiciones antes W_{i-4}.

Para ejemplificar esta descripción tomaremos la Clave de Cifrado del ejemplo descrito en el FIPS-197 como vector de prueba para el caso 128bits.

CK = 2B 7E 15 16 28 AE D2 A6 AB F7 15 88 09 CF 4F 3C

Donde quedan las palabras:

W_0 = 2B 7E 15 16 $\quad$ W_1 = 28 AE D2 A6

W_2= AB F7 15 88 $\quad$ W_3 = 09 CF 4F 3C

Este proceso se ilustra en la figura 2.9.

.

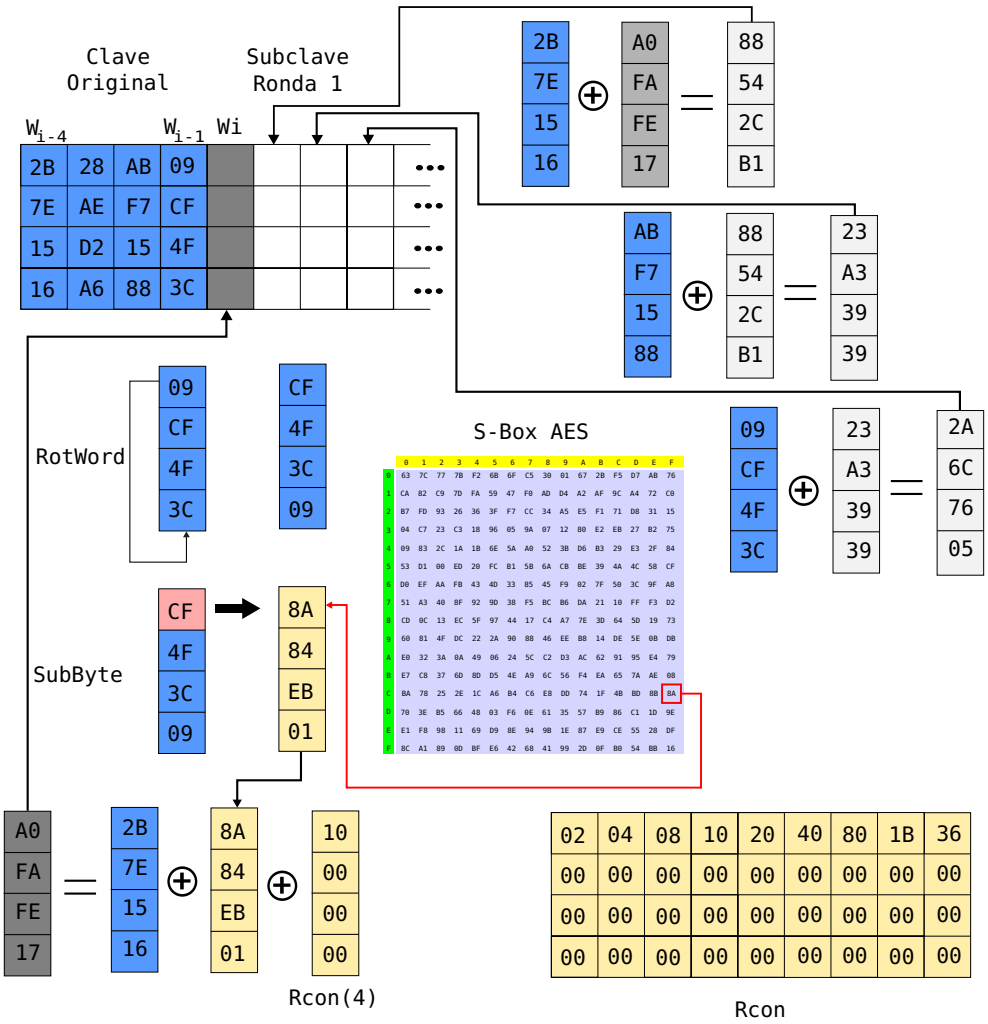

Figura 2.9: Ejemplo del proceso para el calculo de Subclaves

2.3. Algoritmos de la norma FIPS-197

Los siguientes algoritmos son descritos en la norma FIPS-197 [1], los cuales son los que realizan el proceso de cifrado y el proceso de expansión de las subclaves.

Donde:

- *Mi*: Matriz In - Es la Matriz Estado de entrada (Datos iniciales)
- *K* : Key - Es la Clave de cifrado expandida (Clave inicial y las 10 Subclaves de cada ronda)
- *Mo*: Matriz Out - Es la Matriz Cifrada.
- *Ms*: Matriz State - Es la matriz de estado.

Algoritmo 1 Proceso de Cifrado

Entrada: byte $Mi[4 * Nb]$, word $K[Nb * (Nr + 1)]$
Salida: byte $Mo[4 * Nb]$

```
byte Ms[4, Nb]
Ms ⇐ Mi
AddRoundKey(state, K[0, Nb-1])
for ronda = 1 to Nr − 1 do
  SubBytes(Ms)
  ShiftRows(Ms)
  MixColumns(Ms)
  AddRoundKey(Ms, K[ronda*Nb, (ronda+1)*Nb-1])
end for
SubBytes(Ms)
ShiftRows(Ms)
AddRoundKey(Ms, w[Nr*Nb, (Nr+1)*Nb-1])
Mo ⇐ Ms
```

Figura 2.10: Proceso de Cifrado

- Nb: Número de palabras(columnas) para las matrices, en este caso 4.
- Nr: Número de rondas para el proceso de cifrado en este caso 10.

Algoritmo 2 Proceso de Expansión de Clave

Entrada: byte $key[4 * Nk]$, Nk
Salida: word $K[Nb * (Nr + 1)]$

```
word temp
i ⇐ 0
while i < Nk do
    K[i] = word(key[4*i], key[4*i+1], key[4*i+2], key[4*i+3])
    i = i + 1
end while
i = Nk
while i < Nb*(Nr+1) do
    temp = w[i-1]
    if i mod Nk = 0 then
        temp = SubWord(RotWord(temp)) ⊕ Rcon[i/Nk]
    else if Nk > 6 and i mod Nk = 4 then
        temp = SubWord(temp)
    end if
    w[i] = w[i-Nk] ⊕ temp
    i ⇐ i + 1
end while
```

Figura 2.11: Proceso de Expansión de Clave

Donde:

- key: Es la Clave de Cifrado inicial.
- K: Key - Es la Clave de cifrado expandida (Clave inicial y las 10 Subclaves de cada ronda)
- Nr: Número de rondas para el proceso de cifrado en este caso 10.
- Nk: Largo de la Clave (número de columnas en la clave) para nuestro caso 4.

De manera general los anteriores algoritmos pueden usarse para describir el procedimiento general para el cifrado de datos, esto logra verse en la figura 2.12.

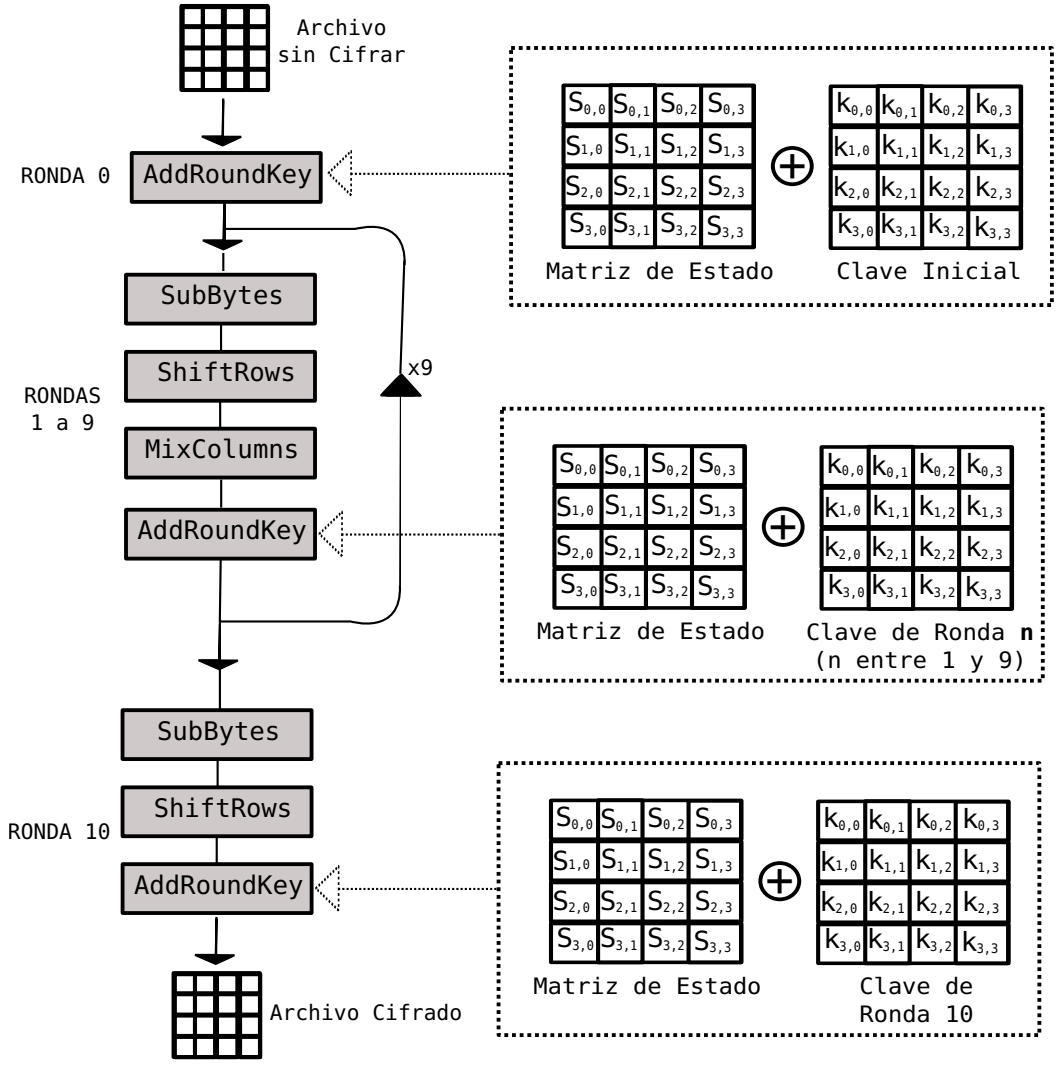

Figura 2.12: Proceso completo para el cifrado de Datos

Capítulo 3

Análisis de la versión en software

El primer paso para implantar el algoritmo de cifrado AES (Rijndael) en un sistema Hardware-Software, consiste en hacer un análisis del algoritmo en software que nos pueda proporcionar información sobre el costo de tiempo, la cantidad de operaciones y el trabajo realizado por cada una de las partes que componen el algoritmo. Este análisis tiene como objetivo detectar las partes críticas del programa, es decir las partes que consumen la mayor cantidad de tiempo de ejecución.

Los factores que se consideraron para identificar las secciones críticas para ser implantadas en Hardware son:

- Las operaciones más repetitivas del código.
- Las operaciones que operan a nivel de bits.
- Las operaciones que no son adecuadas para un procesador como son corrimientos y rotaciones.

3.1. Implementación del Algoritmo

El código fuente usado para este análisis proviene de *AES - Advanced Encryption Standard*[14]. El programa tiene la característica de usar un algoritmo que es más eficiente para software al evitar realizar las operaciones descritas en el Capítulo 2 en la sección correspondiente a *MixColumns*, esto es conveniente ya que puede se puede hacer una mejor comparación en la eficiencia entre el Hardware y el Software.

Las diferencias entre esta implementación y la norma FIPS-197 [1] son:

- La implementación convierte la entrada de 128 bits(dieciséis datos tipo char) en una matriz de 4x4 bytes. Mientras que en la norma AES se trabaja con un solo arreglo 128 bits.

- La implementación convierte la clave de entrada de 128 bits(dieciséis datos tipo char) en una matriz de 4x4 bytes, donde al expandirla la deja en un arreglo de tres dimensiones de 11x4x4 donde el primer índice solo sirve para indicar la columna W_k haciendo de esta manera más sencillo el manejo de los índices con las rondas. Mientras que en la norma AES se trabaja con un solo arreglo de 128 bits a la entrada y al momento de expandirla queda en un solo arreglo para los 1048 bits.

- La S-Box esta declarada en un arreglo de 256 elementos de 8 bits. Mientras que en la norma no tiene un parecido inmediato con la S-Box.

- Las operaciones que se realizan en la sección *MixColumns* al parecer fueron realizadas con anterioridad y los resultados son contenidos en un arreglo de 256 elementos de 8 bits cada uno, donde cada elemento corresponde al resultado de la solución de cada polinomio descrito dentro de la norma. Mientras que en la norma están implícitas en la función que se encarga de esta sección.

- Cabe mencionar que se hicieron diversos cambios al código fuente para hacer comparaciones entre las salidas obtenidas por esta implementación, el *Rijndael-Inspector* y la implementación en Hardware.

Esta versión en software esta escrita en Lenguaje C y fue ejecutada en una plataforma con sistema operativo Ubuntu v10.1. Los algoritmo 2.10 y 2.11 presentan la estructura más general para el algoritmo completo.

3.2. Análisis de las operaciones realizadas

El algoritmo utilizado en la implementación en software para el cifrado de un bloque de 128 bits esta descrito en el pseudocódigo de la figura 3.1.

Se puede notar que que la estructura general de este algoritmo es muy similar a la estructura del algoritmo para el proceso de cifrado mencionada en el capítulo 2, las diferencias consisten en las entradas que requiere cada función, las variables auxiliares que se utilizan para realizar cada proceso y las funciones adicionales que se utilizan dentro de cada etapa del proceso de cifrado.

Internamente el proceso realizado por cada función se describe a continuación con la finalidad de ver la carga de trabajo que realiza cada una de ellas y de notar las operaciones a nivel de bits que se realizan.

- Substitution: Hace el recorrido completo de la matriz $a[4][4]$ sustituyendo cada elemento por su correspondiente de la matriz S-Box.

- ShiftRows: Para hacer los corrimientos de bytes necesarios ocupa una matriz temporal donde almacena el corrimiento hecho hecho en cada fila ($a[i][(j + shifts[((BC - 4) >> 1)][i][d])\,\%BC]$)

Algoritmo 3 Proceso de Cifrado en Software

Entrada: unsigned char $a[4][4]$, $k[4][4]$
Salida: unsigned char $a[4][4]$

```
KeySched(k[4][4],rk[10 + 1][4][4])
int BC ⇐4
int ROUNDS⇐10
KeyAddition(a,rk[0],BC)
for r = 1 to ROUNDS do
  Substitution(a,S,BC)
  ShiftRows(a,0,BC)
  MixColumns(a,BC)
  KeyAddition(a,rk[r],BC)
end for
Substitution(a,S,BC)
ShiftRow(a,0,BC)
KeyAddition(a,rk[ROUNDS],BC)
```

Figura 3.1: Pseudocódigo del proceso de Cifrado

- MixColumns: Hace uso de 3 funciones más para realizar el proceso descrito en el capítulo 2, además de utilizar 2 matrices de 256 elemento de 8 bits cada uno para almacenar el resultado de la multiplicación $GF(2^8)$ base 3.
- KeyAddition: Realiza la operación XOR entre cada elemento de la matriz $a[4][4]$ y $rk[4][4]$.
- RijndaelKeySched: Hace el mismo procedimiento descrito en el algoritmo para la expansión del claves descrito en el capítulo 2, salvo que la salida es un arreglo de tres dimensiones.

Dentro de cada función se realizan diversas operaciones, en la tabla 3.1 se puede observar la cantidad trabajo realizado por cada función dentro del proceso de cifrado de un bloque de 128 bits. Cabe aclarar que los valores obtenidos para las siguientes pruebas son valores promedio de las repeticiones que realizaron de cada una.

Con el análisis del trabajo que realiza cada función se dividió este algoritmo en 4 etapas, con la finalidad de ver la carga en cada una de ellas y ver en cuales se debe buscar una implementación eficiente en hardware, quedando el algoritmo de la figura 3.2. Haciendo un análisis similar al realizado para el trabajo realizado en la ejecución por cada función, se realizo otro análisis pero esta ver para estimar el trabajo que se realiza en cada etapa del proceso de cifrado, los resultados puede verse en la tabla 3.2.

Nombre de la función	Apariciones en todo el proceso	Trabajo realizado en la ejecución(%)
Substitution	10	12.64
ShiftRows	10	19.27
MixColumns	10	28.90
KeyAddition	11	13.95
KeySched	1	25.24
Total	42	100.00

Cuadro 3.1: Trabajo realizado por cada función

Etapa	Trabajo realizado en la ejecución(%)
Expansión de Clave	25.24
Ronda Inicial	1.13
Rondas	69.63
Ronda Final	4.00
Total	100.00

Cuadro 3.2: Trabajo realizado en cada etapa

Con los porcentajes obtenidos de las pruebas del trabajo realizado y que son presentados en las tablas 3.1 y 3.2 puede establecerse la fracción que del código que es susceptible a implementar en hardware.

Se podría realizar un análisis del tiempo usado por las operaciones lógicas dentro del proceso, pero con la información adquirida se llegó a las siguientes conclusiones de esta etapa:

- Son 5 los circuitos que se diseñaron e implementaron, los cuales serán explicados en el capítulo 4.
- Se noto que las etapas que más trabajo realizan en la ejecución son *MixColumns* y *KeyShedExpansion*, se busco una manera de implementarlos de una mejor manera en hardware ya que aquí se realizan gran número de operaciones lógicas.
- Se evito ocupar tantos recursos de memoria temporal para variables y tablas de datos, ya que en hardware significa más uso de memorias RAM y ROM lo cual hace más robusto el circuito.

Algoritmo 4 Proceso de cifrado en Etapas

Entrada: byte Kin[4][4], A[4][4], K_Nr[44][4]
Salida: byte A[4][4]

Etapa 1: Expansión de Clave
K-Nr[44][4] ← KeySchedExpansion(Kin[4][4])

Etapa 2: Ronda inicial
KeyAddition(a,rk[0],BC)

Etapa 3: Rondas
for $r = 1$ to 9 **do**
  SubBytes(A)
  ShiftRows(A)
  MixColumns(A)
  AddRK(K-Nr[r][4],A)
end for

Etapa 4: Ronda Final
SubBytes(A)
ShiftRows(A)
AddRK(K-Nr[*ronda*][4],A)

Figura 3.2: Algoritmo del proceso divido en etapas

Capítulo 4

Codiseño Hardware-Software

El objetivo de las técnicas de Codiseño Hardware-Software (CHS), desarrolladas a lo largo de la década de los 90's, ha sido distribuir una aplicación entre dos o más particiones hardware y software.

El mantenimiento de partes de la aplicación en software, ejecutado por un microprocesador de propósito general (CPU, Central Processing Unit), se debe a que estas partes no son críticas desde el punto de vista de la velocidad de ejecución, y a que la implementación mixta hardware-software resulta más barata que una implementación totalmente en hardware.

4.1. Generalidades del diseño

De acuerdo al estudio realizado en el capítulo anterior, se determinó que existen 5 partes a implementar en hardware. Cada una corresponde a una función del proceso de cifrado *KeyShedExpansion, SubBytes, ShiftRows, MixColumns y AddRoundKey* , además se agregan 2 circuitos que corresponden a la etapa de *Rondas* y*Ronda final*.

4.2. Dispositivo y Herramientas de diseño

Para poder iniciar la implantación fue necesario seleccionar un dispositivo específico. El dispositivo proporcionado por el asesor fue un XC2VP30 de la familia XUP Virtex-II Pro un FPGA de Xilinx las características de este dispositivo pueden verse en la tabla 4.1. El software utilizado para la compilación y simulación del diseño en VHDL fue el ISE y EDK 8.2i de Xilinx, el cual es el indicado por que estamos trabajando con un FPGA de Xinlix.

Dispositivo 2vp30ff896-7			
Celdas lógicas	30,816	Multipliers	136 de 18-bit
Bloque RAM	2,448 Kb	Procesadores	2 PowerPC
SDRAM DIMM	Acepta RAM de 2Gbytes	Puertos	PS/2 y RS-232

Cuadro 4.1: Características principales dispositivo XC2VP30

4.3. Ciclo de diseño de circuitos en FPGA's

El ciclo general del diseño para este trabajo consistió en los siguientes pasos:

- Proyección de los patrones elegidos.
  Básicamente consistió la migración de software a hardware las operaciones lógicas necesarias.

- Optimización de la arquitectura.
  Aquí se buscaron posibles mejoras a los circuitos, ya sea para obtener un mejor rendimiento o para reducir el tamaño de los mismos.

- Implementación de los bloques en VHDL.
  Se utilizo un método de diseño Top-Down, de modo que se capturo la idea usando un alto nivel de abstracción y después se fue incrementando el nivel del detalle de acuerdo al circuito propuesto.

- Simulación de cada uno de los circuitos.
  Una vez implementados los circuitos en VHDL se simularon para ver como se comportaban, observar que las salidas corresponden a las reales y ver el tiempo en ciclos de reloj que toma cada uno.

4.4. Señales utilizadas

Las señales son las lineas que transportan información dentro del circuito y pueden tener diferentes rangos dependiendo de donde fueron utilizadas, a continuación se mencionan las más importantes dentro de la simulación del circuito.

- **Clk**: es la señal de reloj requerida que sincroniza a los elementos del circuito.

- **Enable**: es la señal que habilita a cada parte del circuito, cuando vale 0 el circuito se reinicia poniendo todo los valores a 0, cuando vale uno el circuito comienza a funcionar.

- **Ain**: es la entrada que trae la matriz de datos a cifrar.

- **Aout**: es la matriz cifrada final.

- **sigA**: es la señal de transferencia de la matriz de estado entre los diversos elementos del circuito.

- **SigKout**$[i]$: son once señales cada una lleva la subclave de cifrado de la ronda correspondiente.

4.5. Diseño e implementación de los circuitos

La migración a hardware de los patrones de las secuencias que realizan el proceso de cifrado en software en su mayoría fue un paso casi directo, la secuencia que se realiza en software puede implementarse de manera análoga en hardware, permitiendo así la conversión de cada parte en un circuito digital de manera fácil. En las siguientes subsecciones se muestran los diagramas de los circuitos diseñados, la explicación del funcionamiento de cada uno de ellos y además se incluye la simulación del circuito completo.

4.5.1. Circuito keyExpansion

La función *KeyShedExpansion* del software fue una de las que presento mayor trabajo de la migración, esto se debe a que es un proceso secuencial que tiene dependencia con resultados previos para el calculo de las columnas posteriores. Para su implementación en hardware se utilizo una variable que almacena cada columna W de las subclaves de cada ronda, en la figura 4.1 puede observarse esta variable o memoria temporal resaltado en un color diferente.

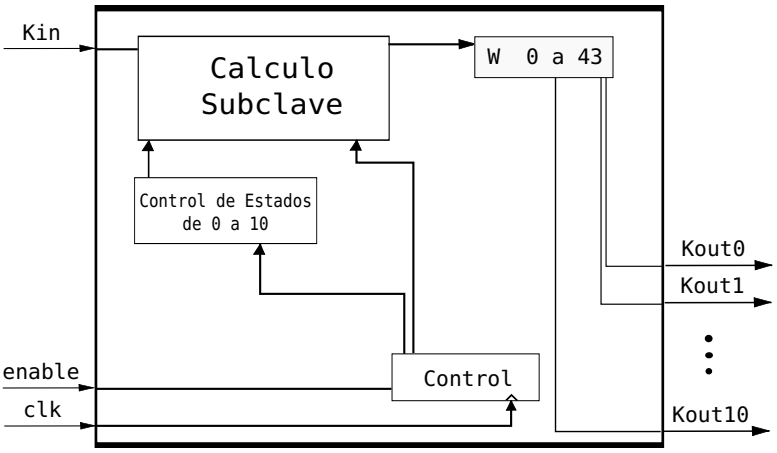

Figura 4.1: Diagrama del circuito KeyShedExpansion

A continuación se describe el funcionamiento del circuito. Esencialmente es una máquina de estados (once estados) donde en cada estado se calcula la subclave correspondiente a una ronda especifica del proceso de cifrado, las funciones auxiliares

para este circuito son:

- **SubWord**: sustituye un elemento de la subclave por el correspondiente de la matriz S-Box.
- **RotWord**: Rota la columna según los requerido en el proceso de expansión de clave.
- **Rcon**: Realiza la operación lógica XOR entre la columna correspondiente y la matriz Rcon

El diagrama de la máquina de estados y las señales que se activan para el intercambio de los estados puede verse en la figura 4.2.

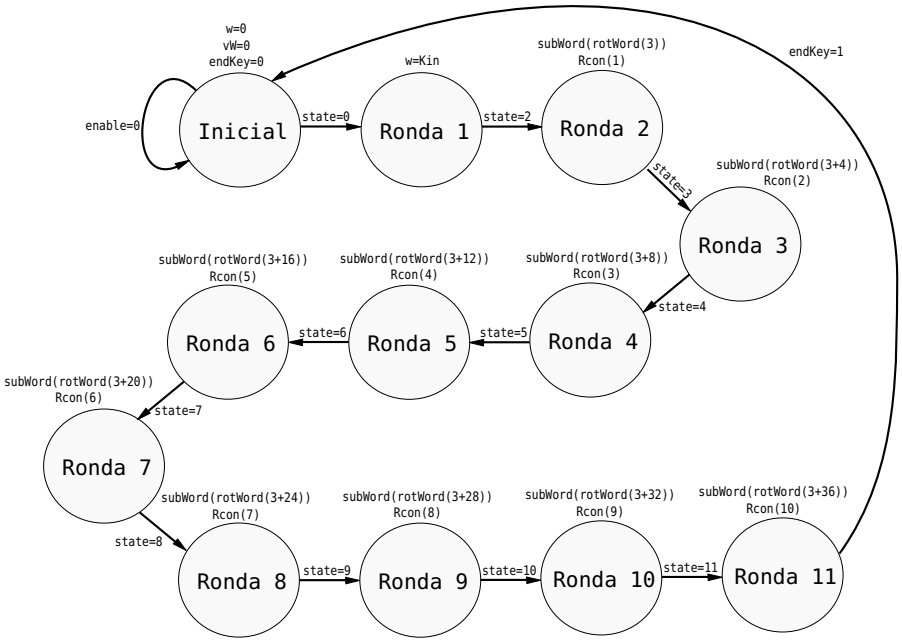

Figura 4.2: Diagrama de estados para la keyExpansion

El detalle del funcionamiento de como se realiza este proceso esta explicado en el capítulo 2. Este circuito corresponde a la Etapa 1 del algoritmo 3.2.

4.5.2. Circuito addRK

La función *AddRoundKey* o *KeyAddition* de la parte del software realiza la operación **XOR** entre la matriz de estado A y la clave o subclave *key* correspondiente de la ronda. El diagrama puede observarse en la figura 4.3

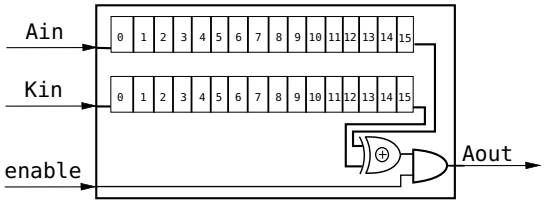

Figura 4.3: Diagrama del circuito addRK

4.5.3. Circuito mixColumns

La función *MixColumns* de la parte del software represento otro buen reto para el diseño en hardware, en la etapa de documentación se encontró un documento [15] que explicaba como realizar las multiplicaciones en el campo de Galois a nivel de bits, usando corrimientos y operaciones lógicas, lo cual es ideal al momento de hacer la importación de software a hardware. El diagrama puede observarse en la figura 4.4.

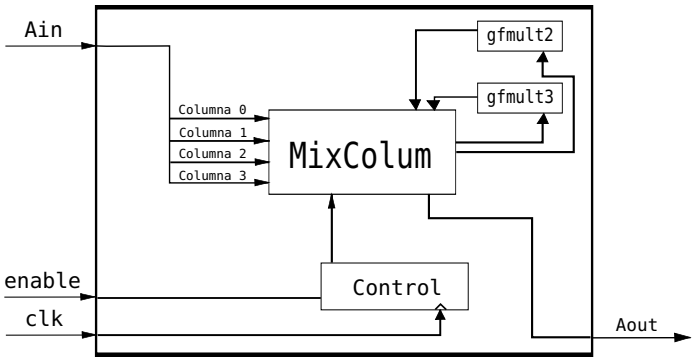

Figura 4.4: Diagrama del circuito mixColumns

El funcionamiento del circuito consiste en recibir la matriz A que sera transformada por la función *mixColumns*, en la sección *mixColumn* se realiza las operaciones lógicas que realizan el proceso ayudado de las funciones *gfmult2* y *gfmult3*.

4.5.4. Circuito shiftRow

La función *shiftRows* de la parte de software consiste en diversos corrimientos en filas de la matriz A, esto en hardware se logro con una asignación directa ala salida correspondiente ya que cada 8 bits de nuestra salida representa un elemento de la matriz de ingreso, todo esto puede verse en la figura 4.5 donde cada casilla de los arreglos representa una salida o entrada de 8 bits y corresponde a un elemento de la matriz estado A.

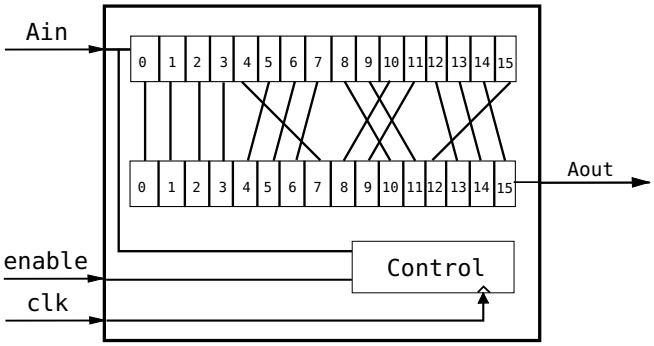

Figura 4.5: Diagrama del circuito shiftRow

4.5.5. Circuito subByte

La función *subByte* o *subtitutionByte* intercambia los elementos de la matriz A por los correspondientes de la matriz *S-BOX* según el procedimiento explicado en el capítulo 2, esto puede verse en el diagrama 4.6

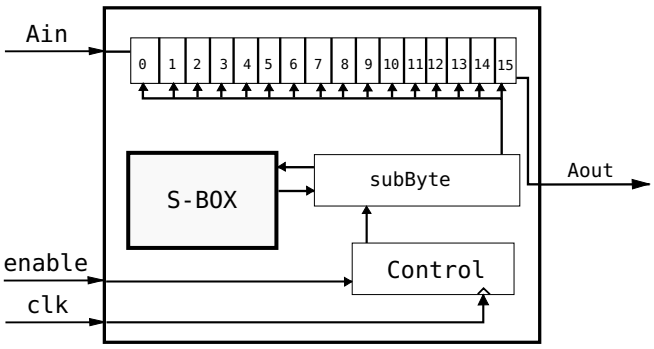

Figura 4.6: Diagrama del circuito subByte

El bloque *subByte* del circuito es el encargado de buscar el elemento en la memoria **ROM S-Box** que corresponde al elemento enviado desde la matriz *Ain* de entrada.

4.5.6. Circuito completo para AES

Después de construir los elementos básicos para la implementación se procedió a crear un circuito más que es el encargado de realizar las rondas iniciales. Este circuito únicamente es la unión de las 4 funciones intermedias que se realizan en cada ronda, se tomo esta decisión ya que por medio de VHDL se puede repetir este elemento 9 veces y cada parte obtenida se encargara de una ronda intermedia. Se hizo algo similar con la ronda final solo que para esta no se utilizo la función *mixColumns* ya que no es requerida para esta etapa. El diagrama del circuito completo puede observarse en la figura 4.7.

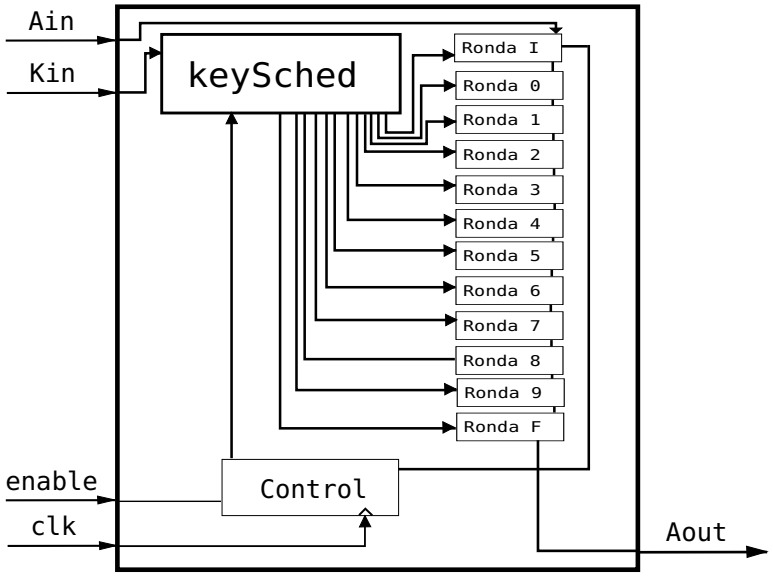

Figura 4.7: Diagrama del circuito subByte

Los bloques **Ronda N** contienen dentro los cuatro elementos básicos diseñados con anterioridad, que corresponde a cada función del proceso de cifrado *mixColumns, addRoundKey, shiftRows* y *subByte*. El bloque de Control incluido en todos los circuitos tiene la función de esperar que el circuito se habilite por medio de la señal *enable*, además de sincronizar todos los circuitos durante el proceso, en este caso funciona con flancos de subida del reloj.

También es importante mencionar que las funciones internamente se aplican en diferentes elementos algunas en columnas y otras en filas, por lo que el manejo de índices requirió especial atención ya que si algún rango no correspondía el cifrado se realizaba pero con resultados no esperados.

4.6. Simulación y Pruebas

En la siguiente sección se muestran imágenes de algunas pruebas realizadas, en las figuras siguientes se podrán observar las señales *testX* las cuales solo sirven para poder revisar como se comporta el circuito en cada etapa del proceso del cifrado, por que el circuito final como se observa en el diagrama 4.7 consta de 4 entradas y 1 salida. Los diagramas presentados son en onda de muestra, el ciclo de reloj usado es de 100ns y la simulación tiene una duración de 10,000ns.

La entrada usada es la misma que la segunda de muestra en *Rijndale-Inspector*, se obtuvieron los mismos resultados tanto con el inspector, software y el hardware con lo cual se confirmo el funcionamiento eficiente del hardware. En la figura 4.8 se observa la simulación completa hasta obtener la matriz final cifrada que esta en la señal *Aout*.

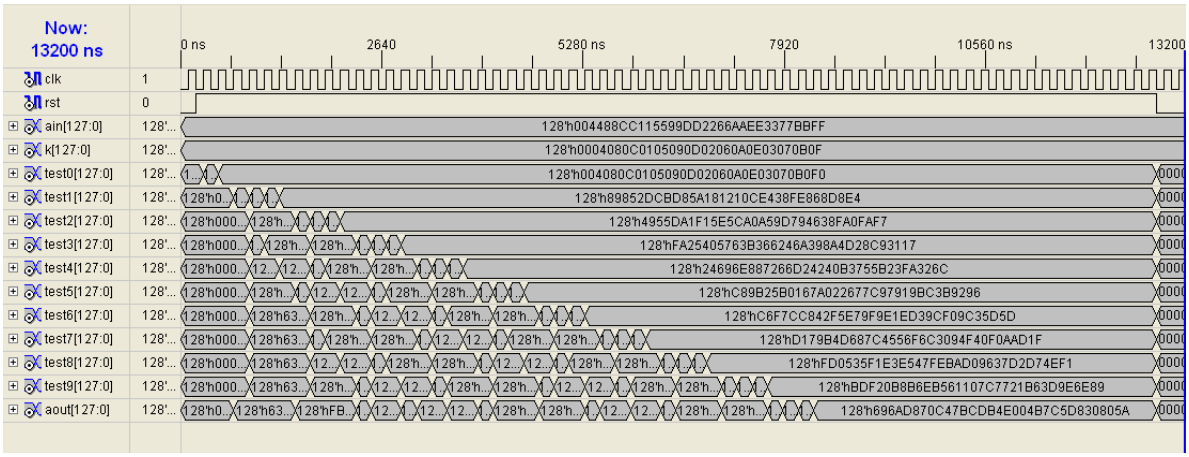

Figura 4.8: Simulación completa.

En el diagrama resaltan una especie de escalones que se dan a lo largo del proceso de cifrado en todas las señales, estos corresponden a cada ronda del proceso de cifrado. En la figura 4.9 se observa el proceso que se realizo entre dos estados intermedios, se puede observar como se el circuito obtiene el resultado solo hasta después de 8000ns, por lo cual deducimos que toma 80 ciclos de reloj realizar el proceso de cifrado por medio del hardware actual.

Se realizaron más pruebas obteniendo resultados exitosos en todas ellas. Los recursos ocupados por el FPGA al momento de generar el hardware pueden observarse en la tabla 4.2.

.

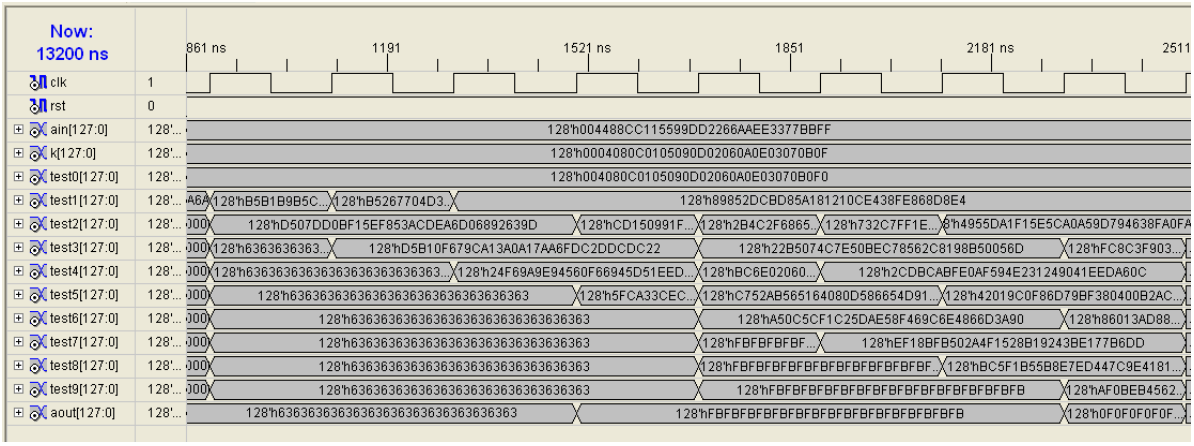

Figura 4.9: Simulación intermedia de dos rondas.

Dispositivo : 2vp30ff896-7		
Elemento	Usado	%
Slices	11639 out of 13696	84
Slice Flip Flops	9932 out of 27392	36
4 input LUTs	15276 out of 27392	55
Number used as logic	14742	
Number used as Shift registers	58	
Number used as RAMs	476	
IOs	121	
Bonded IOBs	121 out of 556	21
IOB Flip Flops	316	
BRAMs	8 out of 136	5
GCLKs	4 out of 16	25
PPC405s	2 out of 2	100
DCMs	2 out of 8	25

Cuadro 4.2: Elementos utilizados por el FPGA

Apéndice A

Código VHDL de los circuitos desarrollados

A.1. Código del circuito AES

A continuación se muestra en código del circuito AES.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity AES is
   Port ( clk : std_logic;
                rst : std_logic;
                Ain : in  STD_LOGIC_VECTOR (127 downto 0);
                K : in  STD_LOGIC_VECTOR (127 downto 0);
                Test0 : out  STD_LOGIC_VECTOR (127 downto 0);
                Test1 : out  STD_LOGIC_VECTOR (127 downto 0);
                Test2 : out  STD_LOGIC_VECTOR (127 downto 0);
                Test3 : out  STD_LOGIC_VECTOR (127 downto 0);
                Test4 : out  STD_LOGIC_VECTOR (127 downto 0);
                Test5 : out  STD_LOGIC_VECTOR (127 downto 0);
                Test6 : out  STD_LOGIC_VECTOR (127 downto 0);
                Test7 : out  STD_LOGIC_VECTOR (127 downto 0);
                Test8 : out  STD_LOGIC_VECTOR (127 downto 0);
                Test9 : out  STD_LOGIC_VECTOR (127 downto 0);
                Aout : out  STD_LOGIC_VECTOR (127 downto 0));

end AES;
```

```
architecture Behavioral of AES is

component keyExpansion is
   port( clk : in std_logic;
            enable: in std_logic;
            Kin : in STD_LOGIC_VECTOR (127 downto 0);
            Kout0 : out STD_LOGIC_VECTOR (127 downto 0);
            Kout1 : out STD_LOGIC_VECTOR (127 downto 0);
            Kout2 : out STD_LOGIC_VECTOR (127 downto 0);
            Kout3 : out STD_LOGIC_VECTOR (127 downto 0);
            Kout4 : out STD_LOGIC_VECTOR (127 downto 0);
            Kout5 : out STD_LOGIC_VECTOR (127 downto 0);
            Kout6 : out STD_LOGIC_VECTOR (127 downto 0);
            Kout7 : out STD_LOGIC_VECTOR (127 downto 0);
            Kout8 : out STD_LOGIC_VECTOR (127 downto 0);
            Kout9 : out STD_LOGIC_VECTOR (127 downto 0);
            Kout10 : out STD_LOGIC_VECTOR (127 downto 0)
   );
end component;

component ronda is
   port( clk : in  STD_LOGIC;
            enable : in  STD_LOGIC;
            Ain : in  STD_LOGIC_VECTOR (127 downto 0);
            Kin : in  STD_LOGIC_VECTOR (127 downto 0);
            Aout : out  STD_LOGIC_VECTOR (127 downto 0));
end component;

component ronda_final is
Port ( clk : in  STD_LOGIC;
          enable : in  STD_LOGIC;
          Ain : in  STD_LOGIC_VECTOR (127 downto 0);
          Kin : in  STD_LOGIC_VECTOR (127 downto 0);
          Aout : out  STD_LOGIC_VECTOR (127 downto 0));
end component;

component addRK is
   Port( clk in STD_LOGIC;
               enable: in std_logic;
               Ain : in STD_LOGIC_VECTOR (127 downto 0);
               K : in STD_LOGIC_VECTOR (127 downto 0);
               Aout : out STD_LOGIC_VECTOR (127 downto 0));
end component;
```

```
type typeKout is array (0 to 10) of std_logic_vector(127 downto 0);
type sigAAux is array (1 to 10) of std_logic_vector(127 downto 0);

signal Afinal:std_logic_vector (127 downto 0);
signal sigKout : typeKout ;
signal sigA :sigAAux;

begin
  --Primer Etapa - Calculo de Llaves
  keyExpansion_1:keyExpansion
  port map(
     clk => clk,
     enable => rst,
     Kin => K,
     Kout0 => sigKout(0),
     Kout1 => sigKout(1),
     Kout2 => sigKout(2),
     Kout3 => sigKout(3),
     Kout4 => sigKout(4),
     Kout5 => sigKout(5),
     Kout6 => sigKout(6),
     Kout7 => sigKout(7),
     Kout8 => sigKout(8),
     Kout9 => sigKout(9),
     Kout10 => sigKout(10));
  --Segund Etapa - Ronda inicial
  addRK_1: addRK
  port map (
    clk => clk,
    enable => rst,
    Ain => Ain,
          K => sigKout(0),
    Aout => sigA(1));
  --Tercer Etapa - Rondas de la 1 a la 9

  genRounds: for Nr in 1 to 9 generate
  ronda_Nr: ronda
     port map (
        clk => clk,
        enable => rst,
        Ain => sigA(Nr),
        Kin => sigKout(Nr),
```

```
            Aout => sigA(Nr+1));
    end generate genRounds;
    -- Cuarta Etapa - Ronda Final
    ronda_11: ronda_final
        port map (
            clk => clk,
            enable => rst,
            Ain => sigA(10),
            Kin => sigKout(10),
            Aout => Afinal);

Test0<=sigA(1);
Test1<=sigA(2);
Test2<=sigA(3);
Test3<=sigA(4);
Test4<=sigA(5);
Test5<=sigA(6);
Test6<=sigA(7);
Test7<=sigA(8);
Test8<=sigA(9);
Test9<=sigA(10);
Aout<=Afinal;

end Behavioral;
```

A.2. Código del circuito Ronda

A continuación se muestra en código del circuito Ronda.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity ronda is
   Port ( clk : in  STD_LOGIC;
          enable : in  STD_LOGIC;
          Ain : in  STD_LOGIC_VECTOR (127 downto 0);
          Kin : in  STD_LOGIC_VECTOR (127 downto 0);
          Aout : out  STD_LOGIC_VECTOR (127 downto 0));
end ronda;

architecture Behavioral of ronda is

component addRK is
   Port (clk: in STD_LOGIC;
               enable: in std_logic;
               Ain : in  STD_LOGIC_VECTOR (127 downto 0);
               K : in  STD_LOGIC_VECTOR (127 downto 0);
               Aout : out  STD_LOGIC_VECTOR (127 downto 0));
end component;

component mixColums is
   Port (clk : in STD_LOGIC;
               enable: in std_logic;
               Ain : in  STD_LOGIC_VECTOR (127 downto 0);
         Aout : out  STD_LOGIC_VECTOR (127 downto 0));
end component;

component shiftRow is
   Port (clk : in STD_LOGIC;
               enable: in std_logic;
               Ain : in  STD_LOGIC_VECTOR (127 downto 0);
         Aout : out  STD_LOGIC_VECTOR (127 downto 0));
end component;

component subByte is
   Port (clk(: in STD_LOGIC;
                  enable: in std_logic;
```

```
                Ain : in  STD_LOGIC_VECTOR (127 downto 0);
        Aout : out  STD_LOGIC_VECTOR (127 downto 0));
end component;

signal outSubByte : std_logic_vector(127 downto 0);
signal outshiftRow : std_logic_vector(127 downto 0);
signal outmixColums : std_logic_vector(127 downto 0);

begin

    subByte_1: subByte
    port map (
      clk => clk,
      enable => enable,
      Ain => Ain,
      Aout => outSubByte);

    shiftRow_1: shiftRow
    port map (
      clk => clk,
      enable => enable,
      Ain => outSubByte,
      Aout => outshiftRow);

    mixColumns_1: mixColums
    port map (
      clk => clk,
      enable => enable,
      Ain => outshiftRow,
      Aout => outmixColums);

    addRK_1: addRK
    port map (
      clk => clk,
      enable => enable,
      Ain => outmixColums,
           K => Kin,
      Aout => Aout);
end Behavioral;
```

A.3. Código del circuito Ronda Final

A continuación se muestra en código del circuito ronda final.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity ronda_final is
    Port ( clk : in  STD_LOGIC;
           enable : in  STD_LOGIC;
           Ain : in  STD_LOGIC_VECTOR (127 downto 0);
           Kin : in  STD_LOGIC_VECTOR (127 downto 0);
           Aout : out  STD_LOGIC_VECTOR (127 downto 0));
end ronda_final;

architecture Behavioral of ronda_final is

component addRK is
    Port( clk: in STD_LOGIC;
                enable: in std_logic;
                Ain : in  STD_LOGIC_VECTOR (127 downto 0);
                K : in  STD_LOGIC_VECTOR (127 downto 0);
                Aout : out  STD_LOGIC_VECTOR (127 downto 0));
end component;

component shiftRow is
    Port( clk(: in STD_LOGIC;
                enable: in std_logic;
                Ain : in  STD_LOGIC_VECTOR (127 downto 0);
          Aout : out  STD_LOGIC_VECTOR (127 downto 0));
end component;

component subByte is
    Port( clk(: in STD_LOGIC;
                enable: in std_logic;
                Ain : in  STD_LOGIC_VECTOR (127 downto 0);
          Aout : out  STD_LOGIC_VECTOR (127 downto 0));
end component;

signal outSubByte : std_logic_vector(127 downto 0);
signal outshiftRow : std_logic_vector(127 downto 0);
signal outmixColums : std_logic_vector(127 downto 0);
```

```
begin

    subByte_1: subByte
    port map (
      clk => clk,
      enable => enable,
      Ain => Ain,
      Aout => outSubByte);

    shiftRow_1: shiftRow
    port map (
      clk => clk,
      enable => enable,
      Ain => outSubByte,
      Aout => outshiftRow);

    addRK_1: addRK
    port map (
      clk => clk,
      enable => enable,
      Ain => outshiftRow,
      Aout => Aout);

end Behavioral;
```

A.4. Código del circuito KeyExpansion

A continuación se muestra en código del circuito keySchedExpansion.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use work.AES_package.ALL;

entity keyExpansion is
    Port ( clk: in std_logic;
           enable: in std_logic;
           Kin : in  STD_LOGIC_VECTOR (127 downto 0);
           endKey : out  STD_LOGIC;
           Kout0 : out  STD_LOGIC_VECTOR (127 downto 0);
           Kout1 : out  STD_LOGIC_VECTOR (127 downto 0);
           Kout2 : out  STD_LOGIC_VECTOR (127 downto 0);
           Kout3 : out  STD_LOGIC_VECTOR (127 downto 0);
           Kout4 : out  STD_LOGIC_VECTOR (127 downto 0);
           Kout5 : out  STD_LOGIC_VECTOR (127 downto 0);
           Kout6 : out  STD_LOGIC_VECTOR (127 downto 0);
           Kout7 : out  STD_LOGIC_VECTOR (127 downto 0);
           Kout8 : out  STD_LOGIC_VECTOR (127 downto 0);
           Kout9 : out  STD_LOGIC_VECTOR (127 downto 0);
           Kout10 : out  STD_LOGIC_VECTOR (127 downto 0)
         );
end keyExpansion;

architecture Behavioral of keyExpansion is
signal w:keysched_type;
constant Rcon:Rcon_Type:= (
                           x"01000000",
                           x"02000000",
                           x"04000000",
                           x"08000000",
                           x"10000000",
                           x"20000000",
                           x"40000000",
                           x"80000000",
                           x"1b000000",
                           x"36000000",
                           x"00000000");
```

```
begin
process(clk,enable)
variable vW:keysched_type;
variable state:integer range 0 to 10;

begin
   if enable = '0' then
      state := 0;
      w <= (others=>x"00000000");
      vW := (others=>x"00000000");
      endKey<='0';
   elsif(clk'event and clk = '1') then
      case state is
         when 0 =>
         w(0)<=Kin(127 downto 120)& Kin(95 downto 88)
         & Kin(63 downto 56)& Kin( 31 downto 24);
         w(1)<=Kin(119 downto 112)& Kin(87 downto 80)
         & Kin(55 downto 48)& Kin( 23 downto 16);
         w(2)<=Kin(111 downto 104)& Kin(79 downto 72)
         & Kin(47 downto 40)& Kin( 15 downto 8);
         w(3)<=Kin(103 downto 96)& Kin(71 downto 64)
         & Kin(39 downto 32)& Kin(  7 downto 0);
         state:=1;
         when 1 =>
            vW(4) := (subWord(rotWord(w(3))) xor Rcon(1)) xor w(0);
            w(4)  <= vW(4);
            w(5)  <= vW(4) xor w(1);
            w(6)  <= (vW(4) xor w(1)) xor w(2);
            w(7)  <= ((vW(4) xor w(1)) xor w(2)) xor w(3);
            state := 2;
         when 2 =>
            vW(4+4) := (subWord(rotWord(w(3+4))) xor Rcon(2)) xor w(0+4);
            w(4+4) <= vW(4+4);
            w(5+4) <= vW(4+4) xor w(1+4);
            w(6+4) <= (vW(4+4) xor w(1+4)) xor w(2+4);
            w(7+4) <= ((vW(4+4) xor w(1+4)) xor w(2+4)) xor w(3+4);
            state := 3;
         when 3 =>
            vW(4+8) := (subWord(rotWord(w(3+8))) xor Rcon(3)) xor w(0+8);
            w(4+8) <= vW(4+8);
            w(5+8) <= vW(4+8) xor w(1+8);
            w(6+8) <= (vW(4+8) xor w(1+8)) xor w(2+8);
            w(7+8) <= ((vW(4+8) xor w(1+8)) xor w(2+8)) xor w(3+8);
```

```
        state := 4;
    when 4 =>
        vW(4+12) := (subWord(rotWord(w(3+12))) xor Rcon(4)) xor w(0+12);
        w(4+12)  <= vW(4+12);
        w(5+12)  <= vW(4+12) xor w(1+12);
        w(6+12)  <= (vW(4+12) xor w(1+12)) xor w(2+12);
        w(7+12)  <= ((vW(4+12) xor w(1+12)) xor w(2+12)) xor w(3+12);
        state := 5;
    when 5 =>
        vW(4+16) := (subWord(rotWord(w(3+16))) xor Rcon(5)) xor w(0+16);
        w(4+16)  <= vW(4+16);
        w(5+16)  <= vW(4+16) xor w(1+16);
        w(6+16)  <= (vW(4+16) xor w(1+16)) xor w(2+16);
        w(7+16)  <= ((vW(4+16) xor w(1+16)) xor w(2+16)) xor w(3+16);
        state := 6;
    when 6 =>
        vW(4+20) := (subWord(rotWord(w(3+20))) xor Rcon(6)) xor w(0+20);
        w(4+20)  <= vW(4+20);
        w(5+20)  <= vW(4+20) xor w(1+20);
        w(6+20)  <= (vW(4+20) xor w(1+20)) xor w(2+20);
        w(7+20)  <= ((vW(4+20) xor w(1+20)) xor w(2+20)) xor w(3+20);
        state := 7;
    when 7 =>
        vW(4+24) := (subWord(rotWord(w(3+24))) xor Rcon(7)) xor w(0+24);
        w(4+24)  <= vW(4+24);
        w(5+24)  <= vW(4+24) xor w(1+24);
        w(6+24)  <= (vW(4+24) xor w(1+24)) xor w(2+24);
        w(7+24)  <= ((vW(4+24) xor w(1+24)) xor w(2+24)) xor w(3+24);
        state := 8;
    when 8 =>
        vW(4+28) := (subWord(rotWord(w(3+28))) xor Rcon(8)) xor w(0+28);
        w(4+28)  <= vW(4+28);
        w(5+28)  <= vW(4+28) xor w(1+28);
        w(6+28)  <= (vW(4+28) xor w(1+28)) xor w(2+28);
        w(7+28)  <= ((vW(4+28) xor w(1+28)) xor w(2+28)) xor w(3+28);
        state := 9;
    when 9 =>
        vW(4+32) := (subWord(rotWord(w(3+32))) xor Rcon(9)) xor w(0+32);
        w(4+32)  <= vW(4+32);
        w(5+32)  <= vW(4+32) xor w(1+32);
        w(6+32)  <= (vW(4+32) xor w(1+32)) xor w(2+32);
        w(7+32)  <= ((vW(4+32) xor w(1+32)) xor w(2+32)) xor w(3+32);
        state := 10;
```

```
            when 10 =>
                vW(4+36) := (subWord(rotWord(w(3+36))) xor Rcon(10)) xor w(0+36);
                w(4+36)  <= vW(4+36);
                w(5+36)  <= vW(4+36) xor w(1+36);
                w(6+36)  <= (vW(4+36) xor w(1+36)) xor w(2+36);
                w(7+36)  <= ((vW(4+36) xor w(1+36)) xor w(2+36)) xor w(3+36);
                endKey<='1';
            when others =>
        end case;
    end if;

end process;

Kout0 <= w(0)&w(1)&w(2)&w(3);--0
Kout1 <= w(4)&w(5)&w(6)&w(7);--1
Kout2 <= w(8)&w(9)&w(10)&w(11);--2
Kout3 <= w(12)&w(13)&w(14)&w(15);--3
Kout4 <= w(16)&w(17)&w(18)&w(19);--4
Kout5 <= w(20)&w(21)&w(22)&w(23);--5
Kout6 <= w(24)&w(25)&w(26)&w(27);--6
Kout7 <= w(28)&w(29)&w(30)&w(31);--7
Kout8 <= w(32)&w(33)&w(34)&w(35);--8
Kout7 <= w(28)&w(29)&w(30)&w(31);--9
Kout8 <= w(32)&w(33)&w(34)&w(35);--10
Kout9 <= w(36)&w(37)&w(38)&w(39);--11
Kout10 <= w(40)&w(41)&w(42)&w(43);--12
end Behavioral;
```

A.5. Código del circuito AddRoundKey

A continuación se muestra en código del circuito AddRoundKey.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity addRK is
    Port( clk: in STD_LOGIC;
          enable: in std_logic;
          Ain : in  STD_LOGIC_VECTOR (127 downto 0);
          K : in  STD_LOGIC_VECTOR (127 downto 0);
          Aout : out  STD_LOGIC_VECTOR (127 downto 0));
end addRK;

architecture Behavioral of addRK is
begin
    process (clk,enable)
    begin
        if enable = '0' then
            Aout<=(others=>'0');
        elsif(clk'event and clk = '1') then
            Aout(127 downto 120)<=Ain(127 downto 120) XOR K(127 downto 120);
            Aout(119 downto 112)<=Ain(119 downto 112) XOR K(95 downto 88);
            Aout(111 downto 104)<=Ain(111 downto 104) XOR K(63 downto 56);
            Aout(103 downto 96)<=Ain(103 downto 96) XOR K(31 downto 24);
            Aout( 95 downto 88)<=Ain( 95 downto 88) XOR K(119 downto 112);
            Aout( 87 downto 80)<=Ain( 87 downto 80) XOR K(87 downto 80);
            Aout( 79 downto 72)<=Ain( 79 downto 72) XOR K(55 downto 48);
            Aout( 71 downto 64)<=Ain( 71 downto 64) XOR K(23 downto 16);
            Aout( 63 downto 56)<=Ain( 63 downto 56) XOR K(111 downto 104);
            Aout( 55 downto 48)<=Ain( 55 downto 48) XOR K(79 downto 72);
            Aout( 47 downto 40)<=Ain( 47 downto 40) XOR K(47 downto 40);
            Aout( 39 downto 32)<=Ain( 39 downto 32) XOR K(15 downto 8);
            Aout( 31 downto 24)<=Ain( 31 downto 24) XOR K(103 downto 96);
            Aout( 23 downto 16)<=Ain( 23 downto 16) XOR K(71 downto 64);
            Aout( 15 downto 8)<=Ain( 15 downto 8) XOR K(39 downto 32);
            Aout( 7 downto 0)<=Ain( 7 downto 0) XOR K(7 downto 0);
        end if;
    end process;
end Behavioral;
```

A.6. Código del circuito ShiftRow

A continuación se muestra en código del circuito ShiftRow.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity shiftRow is
    Port( clk : in STD_LOGIC;
          enable: in STD_LOGIC;
          Ain : in  STD_LOGIC_VECTOR (127 downto 0);
          Aout : out  STD_LOGIC_VECTOR (127 downto 0));
end shiftRow;

architecture Behavioral of shiftRow is
begin
process(clk,enable)
begin
    if enable = '0' then
        Aout<=(others=>'0');
    elsif(clk'event and clk = '1') then
        Aout(127 downto 120)<=Ain(127 downto 120);
        Aout(119 downto 112)<=Ain(119 downto 112);
        Aout(111 downto 104)<=Ain(111 downto 104);
        Aout(103 downto 96)<=Ain(103 downto 96);
        Aout( 95 downto 88)<=Ain(87 downto 80);
        Aout( 87 downto 80)<=Ain(79 downto 72);
        Aout( 79 downto 72)<=Ain(71 downto 64);
        Aout( 71 downto 64)<=Ain(95 downto 88);
        Aout( 63 downto 56)<=Ain(47 downto 40);
        Aout( 55 downto 48)<=Ain(39 downto 32);
        Aout( 47 downto 40)<=Ain(63 downto 56);
        Aout( 39 downto 32)<=Ain(55 downto 48);
        Aout( 31 downto 24)<=Ain(7 downto 0);
        Aout( 23 downto 16)<=Ain(31 downto 24);
        Aout( 15 downto 8)<=Ain(23 downto 16);
        Aout( 7 downto 0)<=Ain(15 downto 8);
    end if;
    end process;
end Behavioral;
```

A.7. Código del circuito subByte

A continuación se muestra en código del circuito subBytes.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use work.AES_package.ALL;

entity subByte is
   Port ( clk : in  STD_LOGIC;
                    enable : in  STD_LOGIC;
                    Ain : in  STD_LOGIC_VECTOR (127 downto 0);
          Aout : out  STD_LOGIC_VECTOR (127 downto 0));
end subByte;

architecture Behavioral of subByte is
begin
process(clk,enable)
begin
   if enable = '0' then
      Aout<=(others=>'0');
   elsif(clk'event and clk = '1') then
      Aout(127 downto 120)<=SBOX(conv_integer(Ain(127 downto 120)));
      Aout(119 downto 112)<=SBOX(conv_integer(Ain(119 downto 112)));
      Aout(111 downto 104)<=SBOX(conv_integer(Ain(111 downto 104)));
      Aout(103 downto 96)<=SBOX(conv_integer(Ain(103 downto 96)));
      Aout( 95 downto 88)<=SBOX(conv_integer(Ain( 95 downto 88)));
      Aout( 87 downto 80)<=SBOX(conv_integer(Ain( 87 downto 80)));
      Aout( 79 downto 72)<=SBOX(conv_integer(Ain( 79 downto 72)));
      Aout( 71 downto 64)<=SBOX(conv_integer(Ain( 71 downto 64)));
      Aout( 63 downto 56)<=SBOX(conv_integer(Ain( 63 downto 56)));
      Aout( 55 downto 48)<=SBOX(conv_integer(Ain( 55 downto 48)));
      Aout( 47 downto 40)<=SBOX(conv_integer(Ain( 47 downto 40)));
      Aout( 39 downto 32)<=SBOX(conv_integer(Ain( 39 downto 32)));
      Aout( 31 downto 24)<=SBOX(conv_integer(Ain( 31 downto 24)));
      Aout( 23 downto 16)<=SBOX(conv_integer(Ain( 23 downto 16)));
      Aout( 15 downto 8) <=SBOX(conv_integer(Ain( 15 downto 8)));
      Aout( 7 downto 0) <=SBOX(conv_integer(Ain( 7 downto 0)));
      end if;
   end process;
end Behavioral;
```

A.8. Código del circuito MixColumns

A continuación se muestra en código del circuito mixColumns.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use work.AES_package.ALL;

entity mixColums is
   Port ( clk : in STD_LOGIC;
                enable: in std_logic;
                Ain : in  STD_LOGIC_VECTOR (127 downto 0);
          Aout : out  STD_LOGIC_VECTOR (127 downto 0));
end mixColums;

architecture Behavioral of mixColums is

begin
process (clk,enable)
begin
if enable = '0' then
   Aout<=(others=>'0');
elsif(clk'event and clk = '1') then
   --Columna 0
   Aout(127 downto 120)<=gfmult2(Ain(127 downto 120)) xor gfmult3(Ain(95 downto  88))
   xor Ain(63 downto 56) xor Ain(31 downto 24);
   Aout( 95 downto 88)<=Ain(127 downto 120) xor gfmult2(Ain(95 downto 88))
   xor gfmult3(Ain(63 downto 56)) xor Ain(31 downto 24);
   Aout( 63 downto 56)<=Ain(127 downto 120) xor Ain(95 downto 88)
   xor gfmult2(Ain(63 downto  56)) xor gfmult3(Ain(31 downto 24));
   Aout( 31 downto 24)<=gfmult3(Ain(127 downto 120)) xor Ain(95 downto 88)
   xor Ain(63 downto  56) xor gfmult2(Ain(31 downto 24));
   --Columna 1
   Aout(119 downto 112)<=gfmult2(Ain(119 downto 112)) xor gfmult3(Ain(87 downto  80))
   xor Ain(55 downto 48) xor Ain(23 downto 16);
   Aout( 87 downto 80)<=Ain(119 downto 112) xor gfmult2(Ain(87 downto 80))
   xor gfmult3(Ain(55 downto 48)) xor Ain(23 downto 16);
   Aout( 55 downto 48)<=Ain(119 downto 112) xor Ain(87 downto  80)
   xor gfmult2(Ain(55 downto 48)) xor gfmult3(Ain(23 downto 16));
   Aout( 23 downto 16)<=gfmult3(Ain(119 downto 112)) xor Ain(87 downto 80)
   xor Ain(55 downto 48) xor gfmult2(Ain(23 downto 16));
   --Columna 2
```

```
    Aout(111 downto 104)<=gfmult2(Ain(111 downto 104)) xor gfmult3(Ain(79 downto  72))
    xor Ain(47 downto 40) xor Ain(15 downto 8);
    Aout( 79 downto 72)<=Ain(111 downto 104) xor gfmult2(Ain(79 downto  72))
    xor gfmult3(Ain(47 downto 40)) xor Ain(15 downto 8);
    Aout( 47 downto 40)<=Ain(111 downto 104) xor Ain(79 downto  72)
    xor gfmult2(Ain(47 downto 40)) xor gfmult3(Ain(15 downto 8));
    Aout( 15 downto 8)<=gfmult3(Ain(111 downto 104)) xor Ain(79 downto 72)
    xor Ain(47 downto 40) xor gfmult2(Ain(15 downto 8));
    -- Columna 3
    Aout(103 downto 96)<=gfmult2(Ain(103 downto 96)) xor gfmult3(Ain(71 downto 64))
    xor Ain(39 downto 32) xor Ain(7 downto 0);
    Aout( 71 downto 64)<=Ain(103 downto 96) xor gfmult2(Ain(71 downto 64))
    xor gfmult3(Ain(39 downto 32)) xor Ain(7 downto 0);
    Aout( 39 downto 32)<=Ain(103 downto 96) xor Ain(71 downto 64)
    xor gfmult2(Ain(39 downto 32)) xor gfmult3(Ain(7 downto 0));
    Aout(  7 downto 0)<=gfmult3(Ain(103 downto 96)) xor Ain(71 downto 64)
    xor Ain(39 downto 32) xor gfmult2(Ain(7 downto 0));
end if;
end process;
end Behavioral;
```

A.9. Código del paquete AES package

A continuación se muestra en código del circuito AES package.

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

package AES_package is

    type keysched_type is array (0 to 43) of std_logic_vector(31 downto 0);
    type RconType is array (0 to 11) of std_logic_vector(31 downto 0);
    type SBOX_matriz is array ( 0 TO 255 ) of std_logic_vector(7 downto 0);
    constant SBOX:SBOX_matriz:=(
        x"63",x"7C",x"77",x"7B",x"F2",x"6B",x"6F",x"C5",
        x"30",x"01",x"67",x"2B",x"FE",x"D7",x"AB",x"76",
        x"CA",x"82",x"C9",x"7D",x"FA",x"59",x"47",x"F0",
        x"AD",x"D4",x"A2",x"AF",x"9C",x"A4",x"72",x"C0",
        x"B7",x"FD",x"93",x"26",x"36",x"3F",x"F7",x"CC",
        x"34",x"A5",x"E5",x"F1",x"71",x"D8",x"31",x"15",
        x"04",x"C7",x"23",x"C3",x"18",x"96",x"05",x"9A",
        x"07",x"12",x"80",x"E2",x"EB",x"27",x"B2",x"75",
        x"09",x"83",x"2C",x"1A",x"1B",x"6E",x"5A",x"A0",
        x"52",x"3B",x"D6",x"B3",x"29",x"E3",x"2F",x"84",
        x"53",x"D1",x"00",x"ED",x"20",x"FC",x"B1",x"5B",
        x"6A",x"CB",x"BE",x"39",x"4A",x"4C",x"58",x"CF",
        x"D0",x"EF",x"AA",x"FB",x"43",x"4D",x"33",x"85",
        x"45",x"F9",x"02",x"7F",x"50",x"3C",x"9F",x"A8",
        x"51",x"A3",x"40",x"8F",x"92",x"9D",x"38",x"F5",
        x"BC",x"B6",x"DA",x"21",x"10",x"FF",x"F3",x"D2",
        x"CD",x"0C",x"13",x"EC",x"5F",x"97",x"44",x"17",
        x"C4",x"A7",x"7E",x"3D",x"64",x"5D",x"19",x"73",
        x"60",x"81",x"4F",x"DC",x"22",x"2A",x"90",x"88",
        x"46",x"EE",x"B8",x"14",x"DE",x"5E",x"0B",x"DB",
        x"E0",x"32",x"3A",x"0A",x"49",x"06",x"24",x"5C",
        x"C2",x"D3",x"AC",x"62",x"91",x"95",x"E4",x"79",
        x"E7",x"C8",x"37",x"6D",x"8D",x"D5",x"4E",x"A9",
        x"6C",x"56",x"F4",x"EA",x"65",x"7A",x"AE",x"08",
        x"BA",x"78",x"25",x"2E",x"1C",x"A6",x"B4",x"C6",
        x"E8",x"DD",x"74",x"1F",x"4B",x"BD",x"8B",x"8A",
        x"70",x"3E",x"B5",x"66",x"48",x"03",x"F6",x"0E",
        x"61",x"35",x"57",x"B9",x"86",x"C1",x"1D",x"9E",
        x"E1",x"F8",x"98",x"11",x"69",x"D9",x"8E",x"94",
        x"9B",x"1E",x"87",x"E9",x"CE",x"55",x"28",x"DF",
```

```
        x"8C",x"A1",x"89",x"0D",x"BF",x"E6",x"42",x"68",
        x"41",x"99",x"2D",x"0F",x"B0",x"54",x"BB",x"16"
    );  signal w:keysched_type;

    --Funciones
    function gfmult2 (I : std_logic_vector(7 downto 0)) return std_logic_vector;
    function gfmult3 ( I : std_logic_vector(7 downto 0)) return std_logic_vector;
    function rotWord ( I : std_logic_vector(31 downto 0)) return std_logic_vector;
    function subWord ( I : std_logic_vector(31 downto 0)) return std_logic_vector;
    function subBytesLUT (I : std_logic_vector(7 downto 0)) return std_logic_vector;

end AES_package;


package body AES_package is

    function gfmult2 (I : std_logic_vector(7 downto 0)) return std_logic_vector is
            variable  resultado : std_logic_vector(7 downto 0);
            begin
                resultado := (I(6 downto 0) & '0') xor
                (x"1B" and ("000" & I(7)& I(7) & "0" & I(7)& I(7)));
                return resultado;
        end gfmult2;

    function gfmult3 (I : std_logic_vector(7 downto 0)) return std_logic_vector is
        variable resultado : std_logic_vector(7 downto 0);
        begin
            resultado := gfmult2(I) xor I;
            return resultado;
    end gfmult3;

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

    function subBytesLUT (I : std_logic_vector(7 downto 0)) return std_logic_vector is
  variable resultado : std_logic_vector(7 downto 0);
```

```
  constant SBOX_aux : SBOX_matriz := SBOX;
  begin
       resultado := SBOX_aux(conv_integer(I));
     return resultado;
   end subBytesLUT;

   function subWord (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):= subBytesLUT(I(7 downto 0));
       result(15 downto 8):= subBytesLUT(I(15 downto 8));
       result(23 downto 16):= subBytesLUT(I(23 downto 16));
       result(31 downto 24):= subBytesLUT(I(31 downto 24));
     return result;
   end subWord;

end AES_package;
```

Bibliografía

[1] "FIPS-197": Advanced Encryption Standard
`http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf`.Consultada en febrero del 2010.

[2] Advanced Encryption Standard.
`http://home.arcor.de/arne.schlosser/dls/Rijndael.pdf`. Consultada en febrero del 2010.

[3] XILINX
`http://www.xilinx.com/`. Consultada en Noviembre del 2010.

[4] GDHL Home page.
`http://ghdl.free.fr/`. Consultada en febrero del 2010.

[5] GTKWave.
`http://gtkwave.sourceforge.net/`. Consultada en febrero del 2010.

[6] The GNU Compiler Collection
`http://gcc.gnu.org/`. Consultada en febrero del 2010.

[7] Michael Welshenbach, *"Cryptography in C and C++"* , Apress, 2001. ISBN 1-893115-95-X.

[8] Bruce Schneier, *"Applied Cryptography: Protocols, Algorithms and Source Code in C"*, 2da Ed. John Wiley and Sons, 1996. ISBN 0-471-11709-9.

[9] Tanenbaum, Andrew S., *"Redes de Computadoras"*, 3ra Ed. Prentice Hall, 1997. ISBN 968-880-958-6.

[10] Oscar Alvarado Nava, *"Implementación en FPGAs de Algoritmos de Compresión-Descompresión para Dispositivos Móviles "*, Tesis de Maestría,Centro de Investigación y de Estudios Avanzados del Instituto Politecnico Nacional, Febrero 2007.

[11] Emanuel López Trejo, *"Implementación eficiente en FPGA del Modo CCM usando AES"*. Tesis de Maestría, Centro de Investigación y de Estudios Avanzados del Instituto Politecnico Nacional, Septiembre 2005.

[12] Mizael Sánchez Santiago, *"Implementación en hardware-software del algoritmo criptográfico DES"*. Tesis de Maestría, Centro de Investigación y de Estudios Avanzados del Instituto Politecnico Nacional, Julio 2003.

[13] J.-P. Deschamps, A. Guzmán Sacristán, J. I. Martínez Torre, B. Romero, *"Procesador para aplicaciones Criptográficas"*. Universidad Rey Juan Carlos, Madrid, España.
`http://www.iberchip.org/VIII/docs/sesm3-2_3/deschamps001.pdf`. Consultada en febrero del 2010.

[14] José de Jesús Angel Angel, *"AES - Advanced Encryption Standard"*, Versión 2005.

[15] Kit Choy Xintong, *"Understanding AES Mix-Columns Transformation Calculation"*, University of Wollongong.