Encriptación de imágenes en FPGA utilizando el algoritmo de Rijndael

Laura Milena Daza Parra

CC. 1018425645

Trabajo de Grado para optar al título de Ingeniero Electrónico

juan carlos giraldo carvajal

Ingeniero electrónico MSc.

director

Pontificia Universidad Javeriana

Facultad de Ingeniería

Bogotá, Colombia

2012

Índice general

Abstract

This work presents the design of a crypto-processor based on Rijndael cipher and its implementation in a programmable device FPGA Cyclone II manufactured by ALTERA using the hardware description language VHDL. The digital system performs both the encryption and decryption of 128 x 128 pixels images downloaded from MATLAB, using a serial port RS232, with a transmission rate of 14400 bauds. It was used the methodology proposed for digital design courses at Pontificia Universidad Javeriana. This Rijndael cipher implementation goes beyond the encryption of a simple block (16 bytes) compared to previous works, because we have demonstrated that Rijndael cipher might be used successfully for encryption greater amount of data.

En este trabajo se presenta el diseño de un criptoprocesador basado en el algoritmo de Rijndael y su implementación en una FPGA Cyclone II de ALTERA con lenguaje de descripción VHDL. Este sistema digital encripta y desencripta imágenes de máximo 128 x 128 pixels caragadas desde MATLAB utilzando puerto serial RS232 a una tasa de transferencia de 14400 baudios. Se utilizó la metodología propuesta para los cursos de diseño digital en la Pontificia Universidad Javeriana. Esta implementación va más allá que anteriores trabajos donde solo se procesa un bloque (16 bytes) ya que se demuestra que este algoritmo puede ser utilizado exitosamente para encriptar grandes cantidades de datos.

Acknowledgements

I cannot finish this work without saying how grateful I´m with my family. I would like to thank my parents, they play an important role in my life. To them I dedicate this work. Without their love, affection, motivation and support this thesis and culmination of my degree would not have been possible. Thanks to Prof. Juan Carlos Giraldo for his guidance in this entire process, for the trust and support that he gave me. Last but not the least I would like to thank my friends, thank you very much to you all.

Introducción

La criptografía, derivada del griego *kryptos* (ocultar) y *grafos* (escribir), es el arte o ciencia de cifrar y descifrar información utilizando técnicas que hagan posible el intercambio de mensajes de manera que sólo puedan ser leídos por las personas a quienes va dirigidos. En este proceso se toma el mensaje a encriptar y mediante pasos organizados, es modificado dependiendo de una palabra clave o llave conocida solo por los destinatarios.

En la actualidad, la criptografía es aplicada en diferentes campos. Desde comunicaciones hasta seguridad informática, la electrónica y la criptografía han ido de la mano en búsqueda de sistemas seguros, rápidos y eficientes. Muchas han sido las propuestas para establecer estándares de encriptación y durante muchos años, uno de ellos, Data Encryption Standard (DES), dominó el área de la criptografía de clave simétrica[1]. El avance tecnológico en lo que respecta a la velocidad de procesamiento de los datos puso a DES en una situación de vulnerabilidad debido al tamaño relativamente corto de su clave. Esto motivó su reemplazo como estándar [1].

A principios de este siglo, el National Institute of Standards and Technology (NIST) comenzó a trabajar en la elaboración de un nuevo estándar conocido como Advanced Encryption Standard (AES) o algoritmo de Rijndael, desarrollado por Vincent Rijmen y Joan Daeman [2]. Una de las mayores diferencias frente a sus antecesores es la ausencia de una estructura tipo Feistel[2]. En su

[1]La criptografía simétrica es un método criptográfico en el cual se usa una misma clave para cifrar y descifrar mensajes

[2]La estructura tipo Feistel utiliza una única función, esto facilita su implementación en hardware al contar con solo un algoritmo pero lo hace vulnerable a ataques

lugar se adoptó una forma de ronda compuesta de tres transformaciones uniformes e invertibles que garantizan linealidad y difusión uniforme del conjunto de datos. La combinación de seguridad, eficiencia, flexibilidad y fácil implementación hace de él una excelente opción de ser usado como algoritmo estándar utilizado en la tecnología actual y del futuro [3].

Con los rápidos avances en materia de intercambio de información electrónica, la seguridad se ha convertido en un punto crítico tanto en almacenaje como en transmisión de dicha información. Ahora no solo intercambiamos textos sino también planos o esquemáticos de procesos que deben ser protegidos contra accesos no autorizados. Por dicha razón, se implementó un codificador de 128 bit utilizando el algoritmo de Rijndael para encriptar y des encriptar imágenes en una FPGA. Una de las principales motivaciones del proyecto es disminuir el tiempo de ejecución del algoritmo en software aprovechando la flexibilidad de dicha plataforma para implementar la ejecución de tareas en paralelo. El uso del lenguaje VHDL permitirá solucionar el problema ya que la arquitectura podrá describirse en función de bloques, procesos y componentes externos, facilitando la estructura de procedimientos y funciones.

Este informe se divide en dos partes principales. En la primera se hace una explicación del algoritmo de Rijndael, mostrando las funciones que se deben realizar y un ejemplo de matriz encriptada. La segunda parte aborda la metodología de diseño realizada, allí se hace la definición de los módulos funcionales y jerarquías.

Capítulo 1

Descripción del algoritmo de Rijndael

El algoritmo Rijndael es un cifrador de bloque iterativo con una longitud variable, en esta aplicación se utiliza de 128 bits por lo tanto el número de rondas se fijará a 10. Fue ideado por Joan Daemen y Vincent Rijmen y enviado al National Institute of Standards and Technology (NIST) como propuesta para el nuevo algoritmo de cifrado estándar, Advanced Encryption Standard (AES)[2].

Este cifrador toma un bloque de información por vez, conocido como *PlainText*, y lo transforma en otro bloque de la misma longitud, llamado *CipherText*. Se utiliza un tercer bloque llamado *CipherKey* o clave. El cifrador fue diseñado teniendo en cuenta tres criterios: un máximo de resistencia frente a ataques conocidos, sencillez en el diseño y que su implementación pueda realizarse de manera compacta teniendo en cuenta a la vez características de velocidad y adaptabilidad a diferentes plataformas[4].

El algoritmo se puede dividir en dos partes: la transformación del mensaje por un lado o *EncryptionProcess*, y la transformación de la clave o *KeySchedule* de donde se obtiene una *sub-clave* para cada ronda. Tanto para la encriptación de datos como para el proceso inverso, el algoritmo utiliza

cuatro transformaciones o sus inversas. Todas las transformaciones son orientadas al byte y se conocen como: *AddRoundKey*, *SubBytes*, *ShiftRows* y *MixColumns*.

1.1. *EncryptionProcess*

El proceso de encriptación está compuesto por cuatro transformaciones uniformes e invertibles, 9 rondas principales y una ronda final como se ve en la figura 1.1, este mezclado lineal garantiza difusión sobre múltiples rondas.

A continuación se muestra un ejemplo del proceso de enciptación con las siguientes matrices de prueba:

$$\underbrace{\begin{array}{|c|c|c|c|}\hline 32 & 88 & 31 & E0 \\ \hline 43 & 5A & 31 & 37 \\ \hline F6 & 30 & 98 & 07 \\ \hline A8 & 8D & A2 & 34 \\ \hline\end{array}}_{\text{PlainText}} \quad \underbrace{\begin{array}{|c|c|c|c|}\hline 2B & 28 & AB & 09 \\ \hline 7E & AE & F7 & CF \\ \hline 15 & D2 & 15 & 4F \\ \hline 16 & A6 & 88 & 3C \\ \hline\end{array}}_{\text{CipherKey}} \tag{1.1}$$

1.1.1. *AddRoundKey*

La *sub-clave* se combina con la matriz *PlainText*. En cada ronda se obtiene una *sub-clave* de la clave principal utilizando el *KeySchedule*. La *sub-clave* se combina cada byte de la matriz *PlainText* mediante la operación XOR.

$$\begin{array}{|c|c|c|c|}\hline 32 & 88 & 31 & E0 \\ \hline 43 & 5A & 31 & 37 \\ \hline F6 & 30 & 98 & 07 \\ \hline A8 & 8D & A2 & 34 \\ \hline\end{array} \oplus \begin{array}{|c|c|c|c|}\hline 2B & 28 & AB & 09 \\ \hline 7E & AE & F7 & CF \\ \hline 15 & D2 & 15 & 4F \\ \hline 16 & A6 & 88 & 3C \\ \hline\end{array} = \begin{array}{|c|c|c|c|}\hline 19 & A0 & 9A & E9 \\ \hline 3D & F4 & C6 & F8 \\ \hline E3 & E2 & 8D & 48 \\ \hline BE & 2B & 2A & 08 \\ \hline\end{array} \tag{1.2}$$

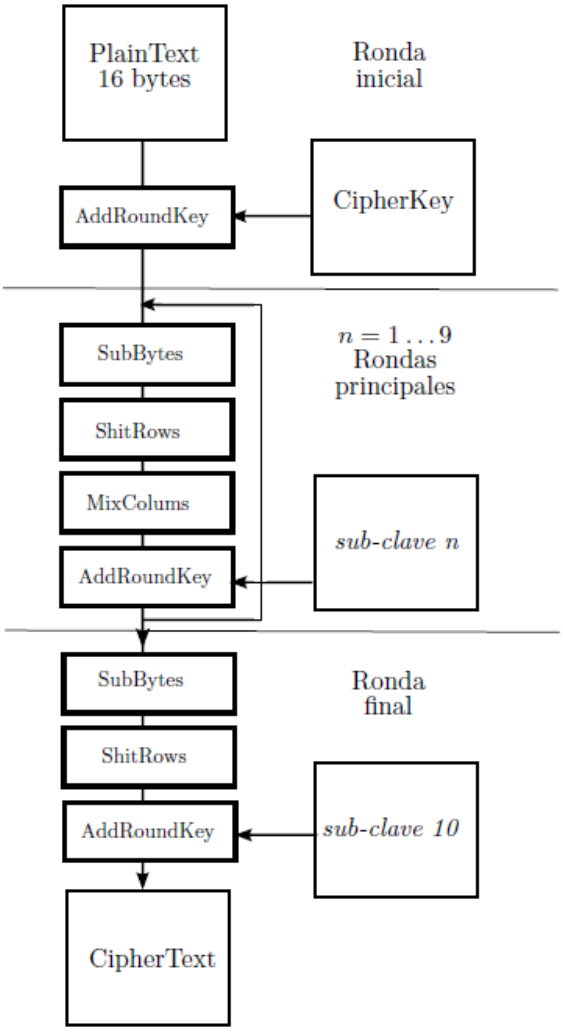


Figura 1.1: Proceso de encriptación

1.1.2. *SubBytes*

En esta transformación invertible se hace una sustitución a cada byte en la matriz. Usando una tabla de sustitución denominada *SBox* se provee no linealidad al cifrado evitando puntos estables.

$$\begin{array}{|c|c|c|c|} \hline 19 & A0 & 9A & E9 \\ \hline 3D & F4 & C6 & F8 \\ \hline E3 & E2 & 8D & 48 \\ \hline BE & 2B & 2A & 08 \\ \hline \end{array} \Rightarrow \begin{array}{|c|c|c|c|} \hline D4 & E0 & B8 & 1E \\ \hline 27 & BF & B4 & 41 \\ \hline 11 & 98 & 5D & 52 \\ \hline AE & F1 & E5 & 30 \\ \hline \end{array} \tag{1.3}$$

1.1.3. *ShiftRows*

En esta etapa se operan las filas de la matriz rotando de manera cíclica los bytes. Los elementos de la primera fila quedan en la misma posición. Cada byte de la segunda fila es rotado una posición a la izquierda. De manera similar, la tercera y cuarta fila rota 2 y 3 bytes respectivamente.

$$\begin{array}{|c|c|c|c|} \hline D4 & E0 & B8 & 1E \\ \hline 27 & BF & B4 & 41 \\ \hline 11 & 98 & 5D & 52 \\ \hline AE & F1 & E5 & 30 \\ \hline \end{array} \Rightarrow \begin{array}{|c|c|c|c|} \hline D4 & E0 & B8 & 1E \\ \hline BF & B4 & 41 & 27 \\ \hline 5D & 52 & 11 & 98 \\ \hline 30 & AE & F1 & E5 \\ \hline \end{array} \tag{1.4}$$

1.1.4. *MixColumns*

Los cuatros bytes de cada columna de la matriz se combinan usando una transformación lineal invertible. Esta función toma los cuatro bytes de cada columna y los multiplica por una determinada matriz.

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \bullet \begin{array}{|c|} \hline D4 \\ \hline BF \\ \hline 5D \\ \hline 30 \\ \hline \end{array} \Rightarrow \begin{array}{|c|} \hline 04 \\ \hline 66 \\ \hline 81 \\ \hline E5 \\ \hline \end{array} \tag{1.5}$$

$D4$	$E0$	$B8$	$1E$		04	$E0$	48	28
BF	$B4$	41	27	$\Rightarrow$	66	CB	$F8$	06
$5D$	52	11	98		81	19	$D3$	26
30	AE	$F1$	$E5$		$E5$	$9A$	$7A$	$4C$

(1.6)

1.2. *KeySchedule*

Para obtener la *sub-clave* necesaria para cada ronda del proceso, se hace una expanción del bloque *CipherKey* obteniendo once *sub-claves* parciales. La clave expandida puede verse como una matriz de 44 columnas y 176 bytes en total. En las primeras cuatro columnas se coloca la matriz *CipherText* dada inicialmente.

w_{i-4}			w_{i-1}	w_i											
2B	28	AB	09												
7E	AE	F7	CF									$\cdots$			
15	D2	15	4F												
16	A6	88	3C												

(1.7)

Las columnas que ocupan una posición múltiplo de 4 (w_4, w_8,..., w_{40}) se calculan aplicando la transformación *RotWord* haciendo un desplazamiento de una posición hacia arriba y posteriormente aplicando la transformación *SubBytes* explicada en la sección anterior con la columna w_{i-1}.

w_{i-1}		
CF		8A
4F	$\Rightarrow$	84
3C		EB
09		01

(1.8)

Luego se hace la operación XOR entre el resultado anterior, la palabra de 4 posiciones antes w_{i-4} y una constante de la matriz fija *Rcon*.

01	02	04	08	10	20	40	80	1B	36
00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00

⇢ Matriz fija Rcon (1.9)

w_{i-4}			$Rcon_1$		w_i
2B	⊕	8A	01	⇒	0A
7E		84	00		FA
15		EB	00		FE
16		01	00		17

(1.10)

Las otras plabras de 32 bits w_i se calculan haciendo la operación XOR en la palabra anterior w_{i-1} con la palabra de cuatro posiciones antes w_{i-4}.

	w_{i-4}			w_{i-1}											
2B	28	AB	09	A0											
7E	AE	F7	CF	FA								...			
15	D2	15	4F	FE											
16	A6	88	3C	17											

(1.11)

w_{i-4}		w_{i-1}		
28		A0		88
AE	$\bigoplus$	FA	$\Rightarrow$	54
D2		FE		2C
A6		17		B1

(1.12)

2B	28	AB	09	A0	88	23	2A	F2	7A	59	73		D0	C9	E1	B6
7E	AE	F7	CF	FA	54	A3	6C	C2	96	35	59	$\cdots$	14	EE	3F	63
15	D2	15	4F	FE	2C	39	76	95	B9	80	F6		F9	25	0C	0C
16	A6	88	3C	17	B1	39	05	F2	43	7A	7F		A8	89	C8	A6

(1.13)

1.3. *DecryptionProcess*

En el proceso de desencriptado, se utilizan las mismas funciones que en *EncryptionProcess* en su versión inversa. Al igual que el proceso de encriptación, se cuenta con 9 rondas principales y una ronda final como se ve en la figura 1.2. En cada ronda se utiliza una *sub-clave* que se obtiene el proceso de *KeySchedule*.

1.3.1. *InvSubBytes*

Al igual que con la transformación *SubBytes* se logra una sustitución no lineal a cada byte en la matriz. En este caso se utiliza otra tabla de sustitución denominada *InvSBox*.

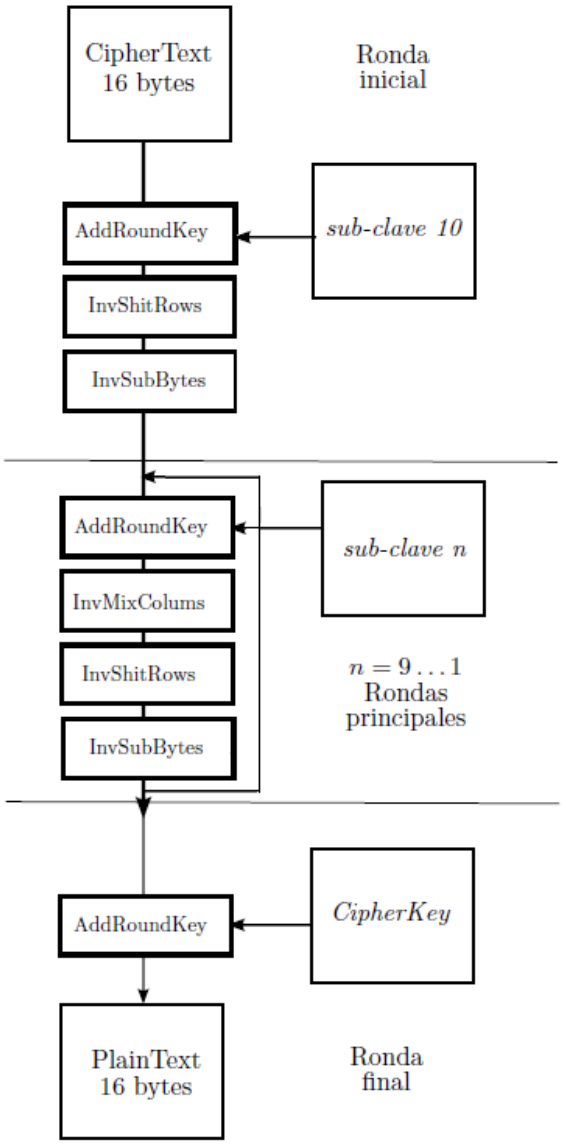


Figura 1.2: Proceso de desencriptación

1.3.2. *InvShiftRows*

A diferencia que en el proceso de encriptado, en esta etapa los elementos de la matriz rotan hacia la derecha. Se conserva el número de posciones que se debe desplazar para cada fila.

1.3.3. *InvMixColumns*

Los cuatros bytes de cada columna de la matriz se combinan usando una transformación lineal invertible. Esta función toma los cuatro bytes de cada columna y los multiplica por una determinada matriz diferente a la del proceso de encriptado.

$$\begin{bmatrix} 14 & 11 & 13 & 09 \\ 09 & 14 & 11 & 13 \\ 13 & 09 & 14 & 11 \\ 11 & 13 & 09 & 14 \end{bmatrix} \dashrightarrow \text{Matriz para función InvMixColumns} \tag{1.14}$$

Capítulo 2

Descripción del proyecto

El proyecto Rijndael consta de un computador y una tarjeta de desarrollo cyclone II. El usuario del proyecto accede a la interfaz gráfica implementada en MATLAB como se ve en la figura 2.1, allí ingresa la clave, la imagen y selecciona el proceso que desea ejectuar (Encriptar o desencriptar).

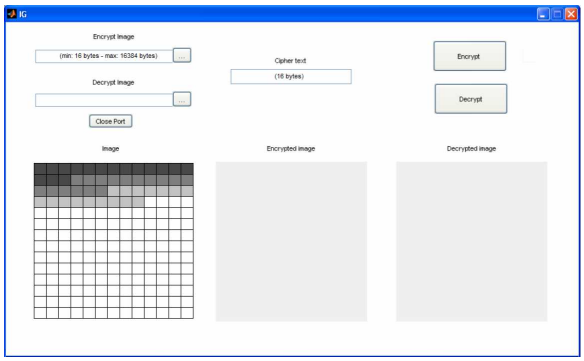
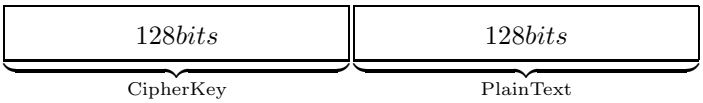


Figura 2.1: Interfaz gráfica

La imagen resultante del proceso se mostrará en la intrefaz gráfica. El proyecto sólo permite imágenes máximo de 128 x 128 pixeles. La transmisión de datos se hace por puerto RS232 a una tasa de transferencia de 14400 baudios, 1 bit de parada, sin bit de pariedad. Desde MATLAB se garantiza que el tamaño de la imagen sea múltiplo de 16, esto se logra agregando filas y columnas en blanco hasta lograr que se cumpla esta condición. Aunque se pueden recibir imágenes a color, estas se convierten a escala de grises. La imagen desencriptada no se puede recuperar a color. Para

encriptar la imagen en su totalidad, se envían partes cada una de 16 bytes junto con la clave a la FPGA, una vez encriptada se envía la siguiente matriz de la imagen. La segmentación se hace como se ve en la imagen 2.1.



(2.1)

2.1. Descripción del sistema

El proyecto está compuesto por 3 módulos principales. Cada módulo tiene una tarea asignada. El módulo **RX** está encargado de establecer la comunicación asíncrona entre el computador y la FPGA. Toma los datos seriales enviados desde MATLAB y los entrega de forma paralela (7 bits). El módulo **SystemRijndael** es el encargado de ejecutar el algoritmo de encriptación y desencriptación, va a la frecuencia del reloj interno de la FPGA de 50MHz. El módulo **TX** establece la comunicación asíncrona entre la FPGA y el computador. Toma los datos de 7 bits entregados por el módulo SystemRijndael y los entrega de forma serial. El diagrama top del sistema se observa en la figura 2.2.

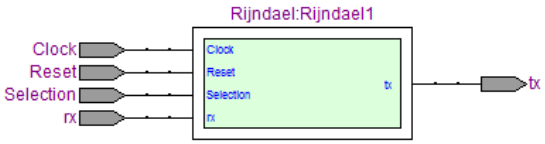


Figura 2.2: Diagrama top del sistema

2.1.1. Diagrama de bloques del sistema

En la figura 2.3 se pueden ver los bloques principales que comprenden el sistema, junto con sus interfaces de conexión.

2.1.2. Interfaces del sistema

En la tabla 2.1 se muestra un listado de las interfaces del sistema.

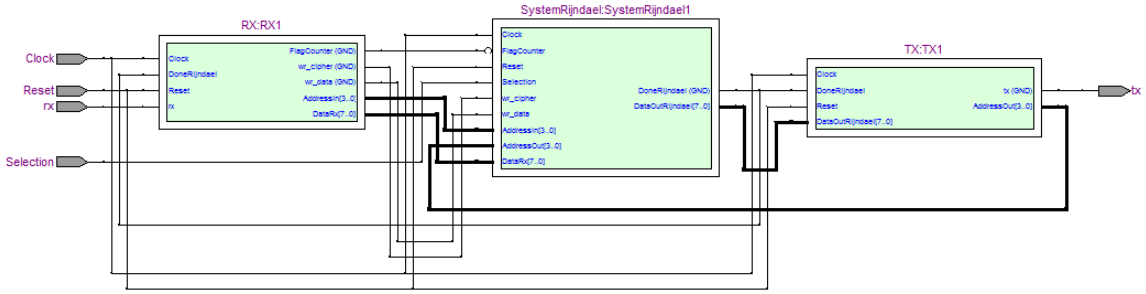


Figura 2.3: Diagrama de bloques del sistema

Cuadro 2.1: Interfaces del sistema

Nombre	Tipo	Descripción
Clock	Entrada	Reloj del sistema. Frecuencia de 50 MHz.
Reset	Entrada	Reset general de la tarjeta.
Selection	Entrada	Senal para seleccionar el proceso a ejecutar. Si está en alto se está encriptando. Se utiliza un *switch* de la tarjeta para hacer esta selección.
rx	Entrada	Senal serial, contiene los vectores de clave e imagen enviados desde MATLAB a la FPGA.
tx	Salida	Senal serial, contiene el vector de imagen procesada la cual se envía desde la FPGA a MATLAB para ser mostrada en la interfaz gráfica.

2.1.3. Diagrama de tiempos del sistema

En la figura 2.4 se observa el diagrama de tiempos del sistema donde se observa el comportamiento dinámico del mismo. Se puede observar que el tiempo de transmisión de los 32 bytes correspondientes a clave y matriz de datos es de aproximadamente 24ms; la transmisión de la matriz resultante es de 12ms, teniendo en total unos 36ms necesarios para enviar los datos y obtener su resultado. Como podemos ver, la señal Selection se encuentra en alto lo que indica que se está encriptando.

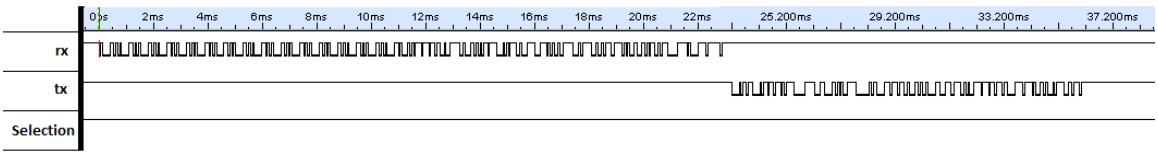


Figura 2.4: Diagrama de tiempos del sistema

2.2. Descripción del módulo RX

Este módulo es el encargado de recibir las tramas enviadas desde MATLAB. Se encarga de leer los datos y separar los bits de clave de los de imagen. Se reciben los datos en forma serial.

2.2.1. Diagrama de bloques del módulo RX

En la figura 2.5 se pueden ver los bloques principales que comprenden el sistema, junto con sus interfaces de conexión.

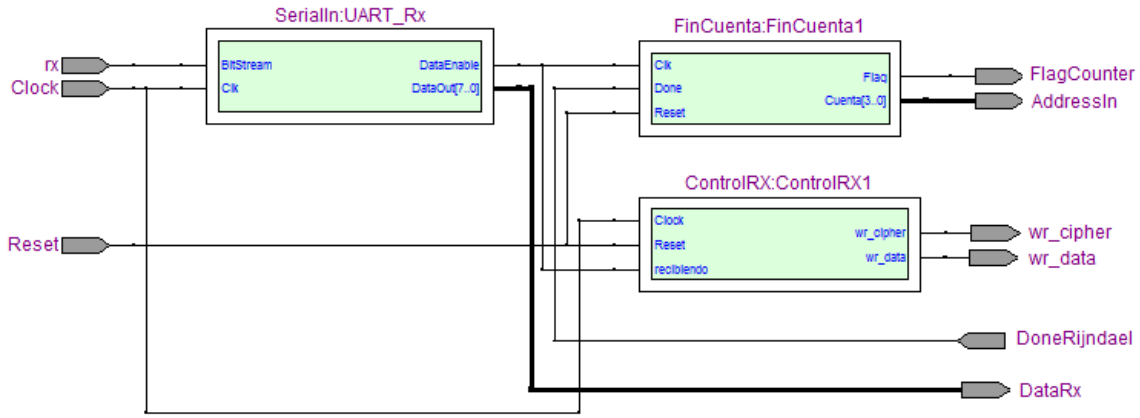


Figura 2.5: Diagrama de bloques del módulo RX

SerialIn

En este bloque se genera una señal de muestreo cuya frecuencia es 16 veces el número de baudios de la tasa de transferencia. Para no crear un nuevo reloj y violar el principio de diseño síncrono, la señal de muestreo se logra haciendo un conteo de pulsos y habilitando la señal de *BaudClk* para lograr el sincronismo del sistema.

SerialIn también cuenta con una máquina de estados que funciona con la señal *BaudClk*. Se cuentan con 3 estados: bsStart, bsStartWait y bsBitCount. La máquina se mantiene en **bsStart** hasta que detecte el bit de inicio de la trama pasando a **bsStartWait** donde se detecta cuando la trama baja a cero siendo este el bit que indica el inicio de los datos que se deben procesar. Una

vez se detecta este bit, se pasa al estado **bsBitCount** donde se ingresan los datos recibidos en un registro de corrimiento *ShiftedIn* cada pulso de la señal *BaudClk*. La señal de salida de este registro es posteriormente asignada a la salida *DataRx* del bloque.

FinCuenta

En este bloque, cada vez que se recibe un dato byte válido, se hace un conteo de bytes recibidos. Cuando se reciben 32 bytes (equivalentes a clave e imagen), se pone en alto la señal *FlagCounter*. La señal *AddressIn* aumenta en 1 cada vez que se recibe un dato válido, este contador llega hasta 31.

ControlRX

ControlRX es una máquina de estados de 34 pasos. El primero (0) es un estado de espera del cual sale cuando detecta el primer pulso de la señal *DataEnable*, indicando que se recibe el primer dato de la trama. Los primeros 16 pasos (1 - 16) se reciben los datos de la clave y entre los pasos 17 y 32 los bytes de la imagen. Cuando se encuentra recibiendo la clave, la señal *wr_cipher* se encuentra en alto. Posteriormente, cuando se recibe la imagen, la señal *wr_data* se pone en 1.

2.2.2. Interfaces del módulo RX

En la tabla 2.2 se muestra un listado de las interfaces del sistema.

Nombre	Tipo	Descripción
Clock	Entrada	Reloj del sistema. Frecuencia de 50 MHz.
Reset	Entrada	Reset general de la tarjeta.
rx	Entrada	Señal serial, contiene los vectores de clave e imagen enviados desde MATLAB a la FPGA.
DoneRijndael	Entrada	Señal que indica que el algoritmo de Rijndael se ha terminado de ejecutar. Dura en alto un ciclo de reloj de la tarjeta, cuando esto sucede la señal AddressIn se coloca en 1111
FlagCounter	Salida	Señal de un bit, indica que se terminó de recibir los 32 bytes correspondientes a la clave y la imagen.
AddressIn	Salida	Señal de 4 bits, da la dirección de memoria donde se debe ir guardando el byte recibido.
wr_data	Salida	Señal de un bit, es el *write enable* de la memoria donde se debe guardar la imagen.
wr_cipher	Salida	Señal de un bit, es el *write enable* de la memoria donde se debe guardar la clave.
DataRx	Salida	Señal de 8 bits, es el byte que se va recibiendo de MATLAB.

Cuadro 2.2: Interfaces del módulo RX

2.2.3. Diagrama de tiempos del módulo RX

Suponiendo la clave y la imagen mostradas en la figura 2.6 veamos como sería el diagrama de tiempos del sistema

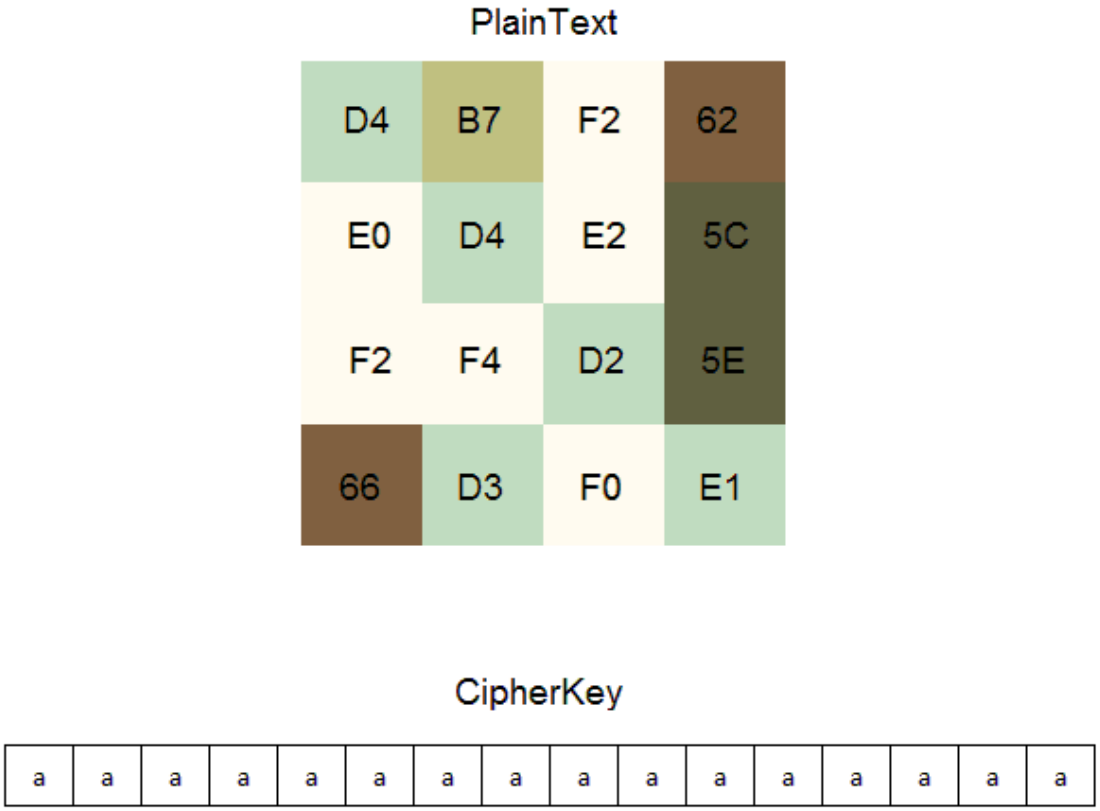


Figura 2.6: Imagen y clave de ejemplo para diagrama de tiempos

Una vez en blanco y negro, cada byte de la imagen es expresado en hexa como se muesta en la figura. La clave es una secuencia de 16 caracteres, todos iguales; la letra **a** expresada en hexa equivale a **61**. En la figura 2.7 se observa el diagrama de tiempos del módulo RX. Podemos ver que mientras se reciben los 16 caracteres de la clave, la señal wr_cipher se encuentra en alto indicando que se debe ir guardando los datos en la memoria MemCipherRAM en la dirección entregada por la señal AddressIn. Una vez recibidos estos bytes, la señal wr_cipher y se habilita la escritura en la memoria MemImageRAM donde se guardan los datos de la imagen. Una vez recibidos los 32 bytes

se pone en alto la señal FlagCounter indicando que los datos se encuentran almacenados y están listos para ser procesados.

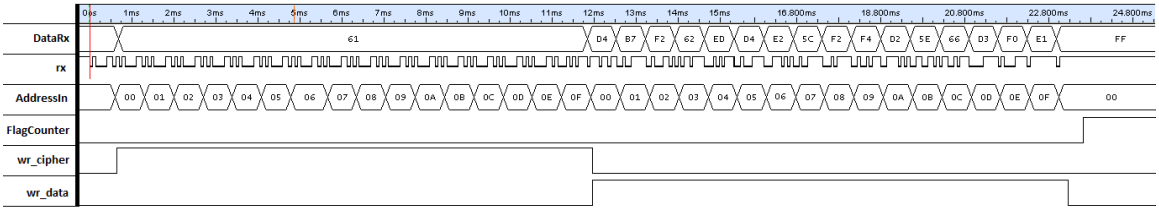


Figura 2.7: Diagrama de tiempos del módulo RX

2.3. Descripción del módulo TX

Este módulo es el encargado de recibir los 16 bytes de la imagen resultante, los convierte a forma serial y los envía a la velocidad de transmisión con que se configuró el puerto.

2.3.1. Diagrama de bloques del módulo TX

En la figura 2.8 se pueden ver los bloques principales que comprenden el sistema, junto con sus interfaces de conexión.

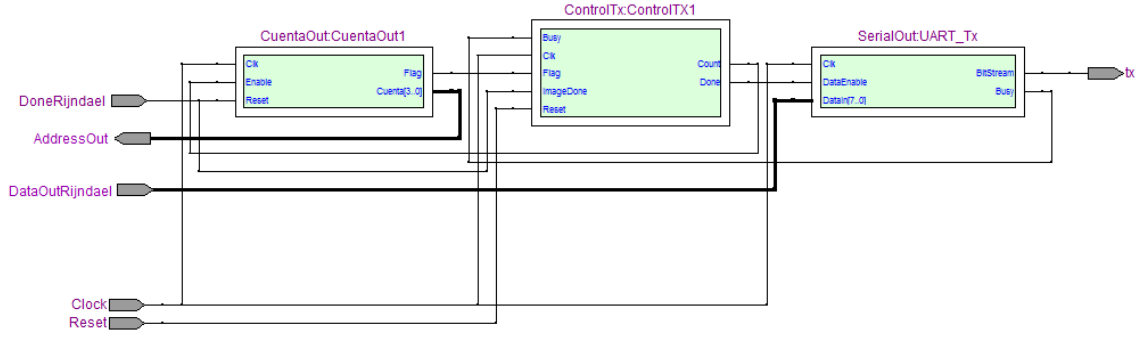


Figura 2.8: Diagrama de bloques del módulo TX

CuentaOut

Este bloque se habilita una vez se tenga la señal *DoneRijndael* que le indica que el proceso ha terminado y la imagen esta lista para ser enviada a la interfaz gráfica. Es un contador que cambia de valor cada vez que recibe la señal *Enable* del bloque ControlTx. Su salida, *Cuenta*, indica en qué dirección de la memoria MemImageOut se debe buscar el byte a enviar. El contador comienza en 0 y una vez toma el valor 16, la señal *Flag* se pone en alto y hace reset del bloque.

ControlTX

ControlRX es una máquina de estados de 6 pasos. El primero (0) es un estado de espera del cual sale cuando detecta la señal *ImageDone*. Este bloque espera a que a señal *busy* entregada por el bloque SerialOut esté en bajo para habilitar el conteo en el bloque CuentaOut. Cuando se envían los 16 bytes de la imagen procesada, se pone en alto la señal *Done* y deja este bloque en su estado inicial de espera.

SerialOut

En este bloque también se genera una señal de muestreo *BaudClk* para lograr el sincronismo con el computador. En el regitro *DataToOutput* de 10 posiciones se organizan los datos de la siguiente forma:

1	**Byte a enviar**	**0**

El bit menos significativo se coloca en 0 para indicar el bit de inicio de transmisión. El MSB se coloca en 1 siendo este el bit de parada. Cada pulso de la señal *BaudClk* se hace un corrimiento en el registro *DataToOutput* cuya salida será la señal *tx*. Mientras se tengan datos en el registro de desplazamiento, la señal *Busy* se encontrará en alto.

2.3.2. Interfaces del módulo TX

En la tabla 2.3 se muestra un listado de las interfaces del sistema.

Nombre	Tipo	Descripción
Clock	Entrada	Reloj del sistema. Frecuencia de 50 MHz.
Reset	Entrada	Reset general de la tarjeta.
DoneRijndael	Entrada	Señal que indica que el algoritmo de Rijndael se ha terminado de ejecutar. Dura en alto un ciclo de reloj de la tarjeta, cuando esto sucede la señal AddressOut se coloca en 1111
DataOutRijndael	Entrada	Señal de 8 bits. Se envía la imagen resulante del proceso de encriptación de desencriptación.
tx	Salida	Señal serial, contiene la imagen resultante desde la FPGA a MATLAB.
AddressOut	Salida	Señal de 4 bits, da la dirección de memoria de donde se debe consultar el byte de la imagen resultante para ser enviado.

Cuadro 2.3: Interfaces del módulo TX

2.3.3. Diagrama de tiempos del módulo TX

En la figura 2.9 se observa el diagrama de tiempos del módulo TX. La señal AddressOut entrega la dirección que se debe buscar en la memoria MemImageOut el byte que se debe enviar (DataOutRijndael) el cual entre en forma paralela al módulo SerialOut y sale serial en la señal *tx*.

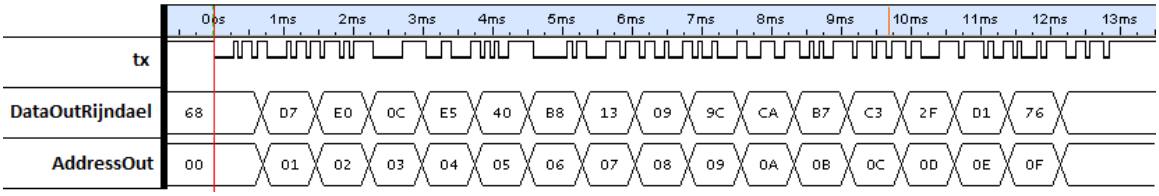


Figura 2.9: Diagrama de tiempos del módulo TX

2.4. Descripción del módulo SystemRijndael

Este módulo es el encargado de realizar el proceso de encriptación y desencriptación. Trabaja con el reloj interno de la tarjeta de 50 MHz. Internamente está conformado de 5 sub-bloques: Memorias, contadores , control, bloques de función y combinacionales.

2.4.1. Descripción de Memorias

El módulo SystemRijndael tiene 12 memorias internas. En la tabla 2.5 se muestran las cuatro memorias tipo RAM donde se puede ver el nombre de la señal de *Address*, *data*, *wren* y *q*. En la tabla 2.4 se tiene el listado de las ocho memorias tipo ROM, allí se tienen los nombres de la señales

para *Address* y *q*, estas memorias se cargan con archivos .hex. El tamaño de las memorias está dado en número de palabras de 8 bits.

Nombre	**Tamaño**	***Address***	***q***	**Descripción**
Rcon	16	SAddressRcon	SDataRcon	Se tienen los bytes correspondientes a la matriz fija *Rcon* utilizada en el proceso *KeySchedule*.
SBox	256	SMuxSBox	SDataSBox	Se tienen los bytes correspondientes a la tabla de sustitución *SBox* utilizada en la transformación *SubBytes*.
Mod2	256	SMod2	DataMod2	Se tienen los bytes necesarios para realizar la multiplicación de la matriz por **02** en la transformación *MixColumns*.
Mod3	256	SMod3	DataMod3	Se tienen los bytes necesarios para realizar la multiplicación de la matriz por **03** en la transformación *MixColumns*.
Mod9	256	SMod9	DataMod9	Se tienen los bytes necesarios para realizar la multiplicación de la matriz por **09** en la transformación *InvMixColumns*.
Mod11	256	SMod11	DataMod11	Se tienen los bytes necesarios para realizar la multiplicación de la matriz por **11** en la transformación *InvMixColumns*.
Mod13	256	SMod13	DataMod13	Se tienen los bytes necesarios para realizar la multiplicación de la matriz por **13** en la transformación *InvMixColumns*.
Mod14	256	SMod14	DataMod14	Se tienen los bytes necesarios para realizar la multiplicación de la matriz por **14** en la transformación *InvMixColumns*.
InvSBox	256	SInInvSBox	SDataInvSBox	Se tienen los bytes correspondientes a la tabla de sustitución *InvSBox* utilizada en la transformación *InvSubBytes*.

Cuadro 2.4: Memorias internas tipo ROM

2.4.2. Descripción de Contadores

El módulo SystemRijndael tiene 6 contadores internos como se ve en la tabla 2.6. También se muestra el valor máximo hasta el que cuenta, con qué señal se hace el conteo, con cual se deja en el valor inicial y su señal de salida. La señal reset es la entregada por el submódulo **Bloques de función**.

2.4.3. Descripción de Control

El submódulo Control entrega los pasos de control para los diferentes bloques. Habilita los bloques de función y habilita la escritura en las memorias RAM. Este submódulo está en su estado de reposo hasta que recibe una señal de encriptar o desencriptar, una vez se completa el algoritmo activa una señal de finalización.

Nombre	**Tamaño**	***Address***	***data***	***wren***	***q***	**Descripción**
MemImageRAM	16	AddressIn or SAddressMC	DataIn	wr_data	SQMemIP	Se guardan los bytes de imagen a procesar.
MemCipherRAM	16	AddressIn or SAddressMC	DataIn	wr_cipher	SDataMC	Se guardan los bytes de clave.
MemKeySchedule	176	SAddressMKS	SDataMKS	SWrenMKS	SQMemKS	Se guardan los bytes de las *sub-claves* resultantes del proceso *KeySchedule*.
MemImageOut	16	AddressOut or AddressIma	SOutImage	SWrenOut or WR-Decrypt	OutImage	Se guardan los bytes de imagen resultante.

Cuadro 2.5: Memorias internas tipo RAM

Nombre	**Conteo**	*cnt_en*	*sclr*	*q*	**Descripción**
CountAddressMemCipher	0 - 15	SCountMC	SResetMC or not reset	SAddressMC	Hace el recorrido de la memoria *MemCipherRAM*.
CountAddressMemKeySchedule	0 - 175	SCountMKS	SResetMKS or not reset	SAddMKS	Cuenta las posiciones de la memoria *MemKeySchedule*. Entrega la dirección cuando se está escribiendo en memoria.
CountRcon	0 - 9	SCountRcon	SResetRcon or not reset	SAddressRcon	Hace el recorrido de la memoria *Rcon*.
Rounds	0 - 10	SCountRounds	SResetRounds or not reset	SRounds	Lleva el conteo de cuantas rondas se han hecho.
CountImageDecrypt	0 - 15	SCountCDI	SResetCDI or not reset	SCountSum	Este conteo se utiliza cuando se está leyendo de la memoria *MemKeySchedule*.
CountMemOut	0 - 15	SCountOut	SResetOut or not reset	AddressOut	Hace el recorrido de la memoria *MemImageOut* para escribir la matriz resultante

Cuadro 2.6: Contadores utilizados en módulo SystemRijndael

Interfaces del submódulo Control

En la tabla 2.7 se encuentra la descripción de cada una de las señales del submódulo. El tamaño está dado en número de bits.

Nombre	Tamaño	Tipo	Descripción
Clock	1	Entrada	Reloj del sistema. Frecuencia de 50 MHz.
reset	1	Entrada	Señal de reset entregada por el submódulo Bloques de función.
Encrypt	1	Entrada	Señal que indica que se va a encriptar la imagen.
Decrypt	1	Entrada	Señal que indica que se va a desencriptar la imagen.
SAddressMKS	8	Entrada	Es la dirección en la que se encuentre la memoria *MemKeySchedule*.
SAddressRcon	4	Entrada	Es la dirección en la que se encuentre la memoria *Rcon*.
SRounds	4	Entrada	Señal que indica cuantas rondas se han hecho.
MuxAddressSBox	1	Salida	Señal que entra a el submódulo Bloques de función. Hace la selección entre 2 señales para tener la dirección de la memoria *SBox* dependiendo de cúal transformación esté activa.
S16Mult	5	Salida	Señal que entra a el submódulo Bloques de función. Se utiliza para calcular la dirección de la memoria *MemKeySchedule* cuando se está leyendo.
EnableSumMKS	1	Salida	Señal que entra a el submódulo Bloques de función. Hace la selección entre 2 señales para tener la dirección de la memoria *MemKeySchedule*.
SResetMC	1	Salida	Señal de clear del contador *CountAddressMemCipher*.
SResetMKS	1	Salida	Señal de clear del contador *CountAddressMemKeySchedule*.
SResetRcon	1	Salida	Señal de clear del contador *CountRcon*.
SResetRounds	1	Salida	Señal de clear del contador *Rounds*.
SCountRounds	1	Salida	Señal de cnt_en del contador *Rounds*.
SCountMC	1	Salida	Señal de cnt_en del contador *CountAddressMemCipher*.
SCountMKS	1	Salida	Señal de cnt_en del contador *CountAddressMemKeySchedule*.
SCountRcon	1	Salida	Señal de cnt_en del contador *CountRcon*.
SWrenMKS	1	Salida	Señal de wren de la memoria *MemKeySchedule*.
SWrenOut	1	Salida	Una de las dos señales de wren de la memoria *MemImageOut*. Está en alto cuando se está encriptando.
SRound1	1	Salida	Señal que se encuentra en alto hasta que se haya calculado la primera *sub-clave* resultante del proceso *KeySchedule*.
SResetCDI	1	Salida	Señal de clear del contador *CountImageDecrypt*.
SCountCDI	1	Salida	Señal de cnt_en del contador *CountImageDecrypt*.
SEnable264	1	Salida	Señal que se encuentra en alto mientras se esté en la primera o segunda ronda.
SEnable26	1	Salida	Señal que se encuentra en alto mientras se esté en la primera ronda.
SCounterRoundsEnd	1	Salida	Señal que se encuentra en alto hasta que se completen las 10 rondas del algoritmo.
ImageDone	1	Salida	Señal que indica que el algoritmo se ha completado. Se pone en alto durante un ciclo de reloj.
WRDecrypt	1	Salida	Una de las dos señales de wren de la memoria *MemImageOut*. Está en alto cuando se está desencriptando.
SResetOut	1	Salida	Señal de clear del contador *CountMemOut*.
SCountOut	1	Salida	Señal de cnt_en del contador *CountMemOut*.
C_out_C	366	Salida	Son las señales de control de todo el sistema.

Cuadro 2.7: Interfaces del submódulo Control

AHPL del submódulo Control

Con el fin de estructurar y describir de forma más detallada la dinámica del sistema, se realiza una descripción en lenguaje AHPL (A Hardware Programming Language)[5]. La máquina de control de este proyecto tiene 366 estados. A continuación se explica en qué estados se activan las transformaciones del algoritmo y sus condiciones de salto:

Estados		**Condición de salto**
0.	Iddle	(start x 1) + (not start x 0)
1...90	KeySchedule	(encrypt x 91) + (decrypt x 230)
91...96	AddRoundKey	(not CounterRoundsEnd x 97) + (CounterRoundsEnd x 195)
97...98	SubBytes	→ 99
99...129	ShiftRows	(not CounterRoundsEnd x 130) + (CounterRoundsEnd x 91)
130...194	MixColumns	(not CounterRoundsEnd x 91) + (CounterRoundsEnd x 195)
195...227	SWrenOut	→ 228
228.	ImageDone	→ 0.
230...237	AddRoundKey	((not SEnable26 and not CounterRoundsEnd) x 299)
		+ (SEnable26 x 238) + (CounterRoundsEnd x 365)
238...267	InvShiftRows	→ 268
268...298	InvSubBytes	→ 230
299...364	InvMixColumns	(not CounterRoundsEnd x 238) + (CounterRoundsEnd x 230)
365.	ImageDone	→ 0.

2.4.4. Descripción de Bloques de función

El módulo SystemRijndael tiene 8 bloques de función: KeySchedule, AddRoundKey, ShiftRows, MixColumns, Round1, InvShiftRows, InvMixColumns y SignalStart. Las transformaciones *InvSubBytes* y *SubBytes* no se encuentran como bloques de función; ya que estas son sustituciones en tablas, no requieren un bloque dedicado.

KeySchedule

En este bloque se lleva acabo la transformación *KeySchedule*. En la tabla 2.8 se muestra un listado de las interfaces del bloque KeySchedule. El tamaño está dado en número de bits.

Nombre	Tamaño	Tipo	Descripción
Clock	1	Entrada	Reloj del sistema. Frecuencia de 50 MHz.
Reset	1	Entrada	Reset general de la tarjeta.
C_out_C	366	Entrada	Son las señales de control de todo el sistema.
SDataMC	8	Entrada	Señal de salida de la memoria *MemCipherRAM*.
SDataRcon	8	Entrada	Señal de salida de la memoria *Rcon*.
SDataSBox	8	Entrada	Señal de salida de la memoria *SBox*.
SRound1	1	Entrada	Señal que se encuentra en alto hasta que se haya calculado la primera *sub-clave* resultante del proceso *KeySchedule*.
SDataMKS	8	Salida	Byte resultante de la transformación *KeySchedule*, se almacena en la memoria *MemKeySchedule*.
SAddressSBox	8	Salida	Señal de dirección de la memoria *SBox*.

Cuadro 2.8: Interfaces del bloque KeySchedule

Para ver el funcionamiento de este bloque se mostrará el proceso para la siguiente clave:

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

Paso 1: En un registro temporal **A** se va guardando el byte consultado en la memoria MemCipherRAM, este registro tiene un tamaño de 128 bits. En la segunda fila se muestra la posición de cada bit. Una vez se tiene el registro **A** completa se comienza el envío a la memoria MemKeySchedule.

0	**0**	**1**	**0**	**1**	**0**	**1**	**1**	**0**	**0**	**1**	**0**	**1**	**0**	**0**	**0**	**1**	**0**	**1**	**0**	**1**	**0**	**1**	**1**	...
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	...

0-7	8-15	16-23	24-31
32-39	40-47	48-55	56-63
64-71	72-79	80-87	88-95
96-103	104-111	112-119	120-127

Cuadro 2.9: Registro A

Paso 2: Para implementar la función RotWord, se tiene otro registro **B** de 32 bits donde se guardan los bits correspondientes a la cuarta columna de la clave. En la segunda fila se muestra las

posiciones del registro **A** que se guardan en el registro **B**.

CF	**4F**	**3C**	**09**
56 - 63	88 - 95	120 - 127	24 - 31

Cuadro 2.10: Registro B

Paso 3: La señal *SAddressSBox* toma el valor de los primeros 8 bits del registro **B** y realiza la sustitución en la memoria. Este reemplazo se efectua para los 8 bytes del registro **B**. El resultado de este proceso se guarda en un registro **C** de 32 bits de tamaño.

8A	**84**	**EB**	**01**
0 - 7	**8 - 15**	**16 - 23**	**24 - 31**

Cuadro 2.11: Registro C

Paso 4: Como se vio en la sección 1.2, para calcular las columnas w_4 se consulta el valor en la memoria Rcon correspondiente a la ronda que se esté realizando. Se cuenta con otro registro **D** de 32 bits donde se guarda el valor entregado por esta memoria en las primeras 8 posiciones y las demás se cargan con 0.

SDataRcon	**00**	**00**	**00**
0 - 7	**8 - 15**	**16 - 23**	**24 - 31**

Cuadro 2.12: Registro D

Paso 5: Para reutilizar el registro **A**, se carga con los resultados de las operaciones de la tabla 2.13. Una vez realizadas estas operaciones se comienza el envío de la subclave resultante para guardarla en la memoria MenKeySchedule. Cuando se termina este envío se vuelve al paso 1 de la transformación. Este proceso se repite hasta que se obtienen las 10 subclaves necesarias para el algoritmo.

A(0-7)	A(0-7) $\bigoplus$ C(0-7) $\bigoplus$ D(0-7)
A(32-39)	A(32-39) $\bigoplus$ C(8-15) $\bigoplus$ D(8-15)
A(64-71)	A(64-71) $\bigoplus$ C(16-23) $\bigoplus$ D(16-23)
A(96-103)	A(96-103) $\bigoplus$ C(24-31) $\bigoplus$ D(24-31)
A(8-15)	A(0-7) $\bigoplus$ A(8-15)
A(40-47)	A(32-39) $\bigoplus$ A(40-47)
A(72-79)	A(64-71) $\bigoplus$ A(72-79)
A(104-111)	A(96-103) $\bigoplus$ A(104-111)
A(16-23)	A(8-15) $\bigoplus$ A(16-23)
A(48-55)	A(40-47) $\bigoplus$ A(48-55)
A(80-87)	A(72-79) $\bigoplus$ A(80-87)
A(112-119)	A(104-111) $\bigoplus$ A(112-119)
A(24-31)	A(16-23) $\bigoplus$ A(24-31)
A(56-63)	A(48-55) $\bigoplus$ A(56-63)
A(88-95)	A(80-87) $\bigoplus$ A(88-95)
A(120-127)	A(112-119) $\bigoplus$ A(120-127)

Cuadro 2.13: Operaciones de la transformación KeySchedule

En la figura 2.10 se observan las 10 *sub-claves* de salida necesarias para el algoritmo. Este proceso dura aproximadamente 11.52 μS y se realiza para cada matriz de 16 bytes de la imagen.

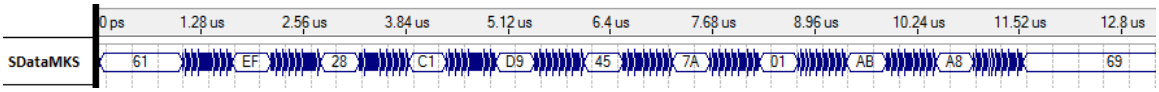


Figura 2.10: Diagrama de tiempos del bloque KeySchedule

AddRoundKey

En este bloque se lleva acabo la transformación *AddRoundKey*. Se hace la operación XOR entre la matriz a encriptar y la *sub-clave* correspondiente a la ronda. Cuenta con 3 registros temporales

de 8 bits cada uno: **A**, **B** y **C**, donde se guarda el byte de matriz, byte de clave y resultado de operación, respectivamente. Cada pulso de reloj se realiza la transformación para cada byte de la matriz.

ShiftRows

Una vez se obtiene un byte de la transformación AddRoundKey, se consulta el valor equivalente en la memoria SBox el cual es guardado en un registro temporal interno de este bloque, este registro es de un tamaño de 128 bits y es organizado de la siguiente forma para lograr el desplazamiento necesario de la transformación, en la tabla 2.14 se muestra como se van guardando los bytes dependiendo del orden en que son obtenidos:

1	2	3	4
6	7	8	5
11	12	9	10
16	13	14	15

Cuadro 2.14: Organización de bytes en registro temporal

MixColumns

En este bloque se lleva acabo la transformación *MixColumns*. Este bloque recibe datos del bloque ShiftRows. Esta transfomación es realizada mediante tablas de sustitución, se utilizan las memorias *Mod2* y *Mod3*. En el caso de la matriz (2.2):

$$\begin{array}{|c|c|c|c|} \hline D4 & E0 & B8 & 1E \\ \hline BF & B4 & 41 & 27 \\ \hline 5D & 52 & 11 & 98 \\ \hline 30 & AE & F1 & E5 \\ \hline \end{array} \tag{2.2}$$

la transformación se realizaría de la siguiente manera, para calcular el primer byte resultante se hace la operación XOR entre los valores consultados en las tablas de sustitución:

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \bullet \begin{array}{|c|} \hline D4 \\ \hline BF \\ \hline 5D \\ \hline 30 \\ \hline \end{array} \tag{2.3}$$

$$mod2(D4) = B3, mod3(BF) = DA, mod1(5D) = 5D, mod1(30) = 30 \tag{2.4}$$

$$\Rightarrow B3 \bigotimes DA \bigotimes 5D \bigotimes 30 = \mathbf{04} \tag{2.5}$$

Para encontrar el byte de la posición 4 el proceso es así:

$$mod1(D4) = D4, mod2(BF) = 65, mod3(5D) = E7, mod1(30) = 30 \tag{2.6}$$

$$\Rightarrow D4 \bigotimes 65 \bigotimes E7 \bigotimes 30 = \mathbf{66} \tag{2.7}$$

$$\begin{array}{|c|c|c|c|} \hline D4 & E0 & B8 & 1E \\ \hline BF & B4 & 41 & 27 \\ \hline 5D & 52 & 11 & 98 \\ \hline 30 & AE & F1 & E5 \\ \hline \end{array} \Rightarrow \begin{array}{|c|c|c|c|} \hline \mathbf{04} & E0 & 48 & 28 \\ \hline \mathbf{66} & CB & F8 & 06 \\ \hline 81 & 19 & D3 & 26 \\ \hline E5 & 9A & 7A & 4C \\ \hline \end{array} \tag{2.8}$$

Round1

En este bloque se guarda temporalmente la matriz resultante de la primera ronda. Los 128 bits son almacenados para ser ahora el registro de donde se deben tomar los datos de entrada para las demás rondas.

InvShiftRows

Al igual que en el proceso de encriptación, una vez se obtiene un byte de la transformación AddRoundKey, se consulta el valor equivalente en la memoria InvSBox el cual es guardado en un

registro temporal interno de este bloque, este registro es de un tamaño de 128 bits y es organizado de la siguiente forma para lograr el desplazamiento necesario de la transformación, en la tabla 2.15 se muestra como se van guardando los bytes dependiendo del orden en que son obtenidos:

1	2	3	4
5	6	7	8
11	12	9	10
14	15	16	13

Cuadro 2.15: Organización de bytes en registro temporal

InvMixcolumns

Al igual que en la transformación *MixColumns*, el resultado de la transformación se obtiene consultando los valores correspondientes en las memorias, en este caso se utilizan las memorias *Mod9*, *Mod11*, *Mod13* y *Mod14*. Se hace la sustitución y posteriormente se realiza la operación XOR.

$$\begin{bmatrix} 14 & 11 & 13 & 09 \\ 09 & 14 & 11 & 13 \\ 13 & 09 & 14 & 11 \\ 11 & 13 & 09 & 14 \end{bmatrix} \bullet \begin{bmatrix} D4 \\ BF \\ 5D \\ 30 \end{bmatrix} \tag{2.9}$$

$$mod14(D4) = 34, mod11(BF) = 0D, mod13(5D) = EC, mod09(30) = AB \tag{2.10}$$

$$\Rightarrow 34 \bigotimes 0D \bigotimes EC \bigotimes AB = \mathbf{7E} \tag{2.11}$$

SignalStart

En este bloque se generan las señales de reset e inicio del sistema. Cuando se recibe una señal de fin de recepción de datos (FlagCounter), se genera una señal que se mantiene en alto durante 5 ciclos de reloj para iniciar la máquina de control. El sistema cuenta con una señal de reset general,

generada por un pulsador integrado en la tarjeta, una vez este se pone en bajo, el bloque SignalStart genera la señal de reset. Cuando se termina de encriptar una matriz de 16 bytes, este bloque toma la señal ImageDone y genera un reset para los bloques internos del módulo SystemRijndael.

Capítulo 3

Resultados

En este cápítulo se presentan los resultados del diseño del sistema. En la tabla 3.1 se observa el reporte de recursos utilizados en el proyecto tales como cantidad de pines de entrada y salida, bits de memoria y número de celdas lógicas utilizadas. De allí se puede resaltar la diferencia entre el número de bits de memoria y el número de elementos lógicos destinados al proyecto, esto se debe a que las transformaciones fueron implementadas con tablas de sustitución las cuales se encontraban almacenadas en memorias ROM internas a la FPGA.

Device	**Total pins**	**Memory bits**	**%**	**Logic elements**	**%**
EP2C20F484C7	5	18,304	8 %	1,391	13 %

Cuadro 3.1: Reporte de recursos

En la tabla 3.2 se observa el consumo de recursos para los 3 módulos principales del sistema: RX, SystemRijndael y TX. Como es de esperarse el módulo SystemRijndael presenta el mayor número de recursos siendo el submódulo de control el que más consume y lo sigue el módulo KeySchedule.

Una de las variables que puede medir la calidad de un circuito es el *throughput*, medida con la cual se determina la cantidad de bits cifrados por el circuito en una unidad de tiempo. En la tabla 3.3 se presenta el valor de throughput desde 2 puntos de vista: el solo procesamiento de la matriz incluyendo el proceso de generación de *sub-claves* y la velocidad total de procesamiento incluyendo

Modulo	**Memory bits**	**Logic elements**
RX	0	88
SystemRijndael	18,304	1256
TX	0	46

Cuadro 3.2: Reporte de recursos por módulos

la trasmisión de datos desde MATLAB. Se puede esperar que el cuello de botella en el sistema es la transmisión de datos la cual limita bastante la velocidad de procesamiento.

Procesamiento	**Ciclos de reloj**	**No. bits**	***throughput***
Algoritmo completo	1651	256	7.75 Mbps
Transmisión y algoritmo	1800000	256	7.11 Kbps

Cuadro 3.3: Reporte de *throughput*

En las imagen 3.1 se puede observar el resultado final del sistema. Se realizaron pruebas con claves aleatorias e imágenes de varios tamaños para verificar el funcionamiento del criptoprocesador.

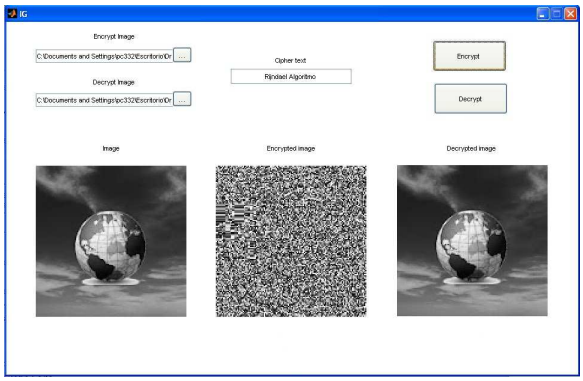


Figura 3.1: Algoritmo de Rijndael aplicado a una imagen de 16384 bytes

Capítulo 4

Conclusiones

Por último se muestran algunos trabajos reportados y sus desempeños para comparar con los logrados en esta aplicación. En la tabla 4.1 se observan dichos parámetros. La principal característica que se puede resaltar es que, a pesar de lograr mayores velocidades, estas aplicaciones lo logran con un mayor número de recursos. En la referencia [6] se pueden observar resultados similares a los presentados en este trabajo; aunque se tiene un menor número de recursos se procesa a una menor velocidad. En la última fila se tienen los resultados obtenidos en este trabajo, se tomó en cuenta el número de pines utilizados en el módulo SystemRijndael para poder compararlo con los otros trabajos.

Autor	**Total pins**	**Memory bits**	**Logic elements**	***throughput***
[7]	386	41088	1584	637.25 Mbps
[8]	391	40960	4805	320 Mbps
[9]	-	40960	21631	80 Mbps
[6]	103	4480	460	6.8 Mbps
✓	28	18304	1256	7.75 Mbps

Cuadro 4.1: Comparación de desempeños

Podemos concluir que sí es posible la utilización del algoritmo de Rijndael para el procesamiento

de grandes cantidades de información, siendo esta una de las iniciativas de realizar este proyecto.

Varios aspectos se deben rescatar de este trabajo: la reutilización de componentes como contadores y registros temporales logra una gran disminución de recursos requeridos, el cual puede ser un aspecto bastante determinante a la hora de implementar. También cabe resaltar que en este trabajo se plantea una solución integral, ya que se cuenta con el sistema de transmsión, recepción, encriptación y desencriptación, dichas opciones no se encontraron en otras aplicaciones, generalemente se tienen dos sistemas diferentes. Con estos resultados podemos ver la relación entre recursos y *throughput*, como se quería comprobar inicialmente.

El trabajo futuro debería enfocarse en mejorar la velocidad del sistema. El proceso *KeySchedule* se ejecuta y posteriormente el algoritmo, como se observa en el AHPL de control; si se logran paralelizar dichos procesos se mejoraría en gran medida el *throughput* del sistema. Como se pudo observar, uno de los principales problemas del proyecto es la transmisión de datos. Se podrían buscar otros medios de envío que mejoren este problema.

La idea de desarrollar este proyecto era lograr un prototipo de sistema de encriptación de información. Posteriormente se podría desarrollar un software dedicado para la aplicación en un programa que no requiera licencia.

Bibliografía

[1] M. C. Liberatori, *Desarrollo de encriptado AES en FPGA.* Universidad Nacional de la Plata, 2006.

[2] V. Rijmen, J. Daemen, *The design of AES-The AdvanceEncryption Standard.* Springer-Verlag, 2002.

[3] F. H. M. Ali, *A Faster Version of Rijndael Cryptographic Algorithm Using Cyclic Shift and Bit Wise Operations.* International Journal of Cryptology Research, 2009.

[4] J. Weaver, Nicholas. Wawrzynek, *High Performance, Compact AES Implementations in Xilinx FPGAs.* U.C. Berkeley BRASS group, 2002.

[5] F. Hill and G. Peterson, *Digital Systems: Hardware Organization and Design.* 1987.

[6] A. Palomino Guzmán and A. Romero Zamora, *Diseño e implementación de algoritmos criptográficos sobre FPGA. Altera Flex 10K.* 2006.

[7] G. Jácome-Calderón and J. Velasco-Medina, *IMPLEMENTACIÓN EN HARDWARE DEL ALGORITMO RIJNDAEL. FPGA Xilinx Virtex.*

[8] M. B. de Barcelos and A. F. Panato, *UM IP DE CRIPTOGRAFIA PADRÃO RIJNDAEL PARA PROJETOS EM FPGA. FPGA Xilinx Altera.*

[9] R. Manteena, *A VHDL Implemetation of the Advanced Encryption Standard-Rijndael Algorithm. Altera Max+plus II.*

[10] A. R. P. Dubey1, *Efficient Rijndael Encryption Implementation with Composite Field Arithmetic*. IBM India research group, 2002.

[11] C. Peacock, *Interfacing the Standard Parallel Port*. 1998.

[12] A. Muñoz, *SEGURIDAD EUROPEA PARA EEUU. Algoritmo Criptográfico Rijndael*. 2004.

[13] P. P. Chu, *FPGA PROTOTYPING BY VHDL EXAMPLES Xilinx SpartanTM-3 Version*. 2008.

[14] ALTERA, *Cyclone II FPGA Starter Development Kit User Guide*. 2006.

[15] B. Leperchey and C. Hymans, *FPGA implementation of the Rijndael algorithm*. 2000.