Trabajo de Fin de Grado en Ingeniería Electrónica

Circuitos Integrados Reconfigurables de Bajo Consumo

Diseño e implementación del algoritmo de cifrado estándar AES

Autor: Álvaro Pérez Mayo
Tutor: Inés Juliana del Campo Hagelstrom

Índice

Introducción y Objetivos

El objetivo principal del presente trabajo es el diseño, utilizando técnicas de bajo consumo, del algoritmo de cifrado estándar AES (Advanced Encryption Standard) [1] y su implementación sobre dispositivos reconfigurables. Como especificaciones secundarias se propondrá un diseño en alta velocidad y se estudiarán los recursos utilizados, todo ello dentro las limitaciones y la arquitectura del dispositivo.

Actualmente existen distintas tecnologías para la implementación de circuitos integrados digitales. Los dispositivos reconfigurables o dispositivos lógicos programables (PLD) tienen gran éxito en la actualidad ya que cualquier circuito de aplicación específica puede ser implementado en un PLD, siempre y cuando disponga de los recursos necesarios [2]. En particular, debido a su versatilidad y flexibilidad las FPGA (Field Programmable Gate Array) hacen de ellas unos dispositivos muy utilizados y apropiados para este tipo de tareas. Además son fácilmente programables mediante el uso de VHDL [3].

Sin embargo, uno de los principales problemas de las FPA es su alto consumo, aunque los fabricantes han conseguido importantes mejoras en los últimos años [2]. El consumo se divide en una componente estática y otra dinámica. El consumo estático depende de las corrientes internas de fugas, que no pueden ser cambiadas por el diseñador, mientras que la componente dinámica depende de los cambios que se producen en las señales y la frecuencia de los mismos. Por tanto, el diseñador debe tener en cuenta el problema del consumo desde las fases iniciales del diseño de un circuito digital.

El algoritmo AES, también conocido como Rijndael, es un esquema de cifrado por bloques adoptado como estándar de cifrado desarrollado por dos criptólogos belgas, Joan Daemen y Vincent Rijmen. Se transformó en un estándar efectivo el 26 de mayo de 2002 y desde 2006 el AES es uno de los algoritmos más populares usados en criptografía simétrica.

En 1997, el Instituto Nacional de Normas y Tecnología de EE.UU. (NIST) decidió realizar un concurso público para escoger un nuevo algoritmo de cifrado capaz de proteger información ya que el algoritmo DES (Data Encryption Standard) que se usaba hasta entonces estaba quedando obsoleto. Este algoritmo se denominó Advanced Encryption Standard (AES). En septiembre de ese año se hizo la convocatoria formal. En esta convocatoria se indicaban varias condiciones para los algoritmos que se presentaran:

- Ser de dominio público, disponible para todo el mundo.
- Ser un algoritmo de cifrado simétrico y soportar bloques de, como mínimo, 128 bits.
- Las claves de cifrado podrían ser de 128, 192 y 256 bits.
- Ser implementable tanto en hardware como en software.

Los algoritmos que cumpliesen los requisitos anteriores serían evaluados según los siguientes factores:

- Seguridad
- Eficiencia computacional
- Requisitos de memoria
- Implementable en hardware y software
- Simplicidad del diseño
- Flexibilidad

El algoritmo Rijndael ganó el concurso tras varias rondas donde se analizaban y comparaban los distintos algoritmos. En noviembre de 2001 se publicó FIPS 197 (Federal Information Processing Standards) donde el algoritmo Rijndael se asumía oficialmente como AES [4].

Entonces los objetivos de este trabajo se pueden resumir de la siguiente manera:

- Utilizar métodos de diseño que permitan reducir el consumo de un circuito integrado digital.
- Diseñar e implementar mediante dispositivos programables el algoritmo de cifrado estándar AES.
- Tener en consideración la velocidad, el consumo y los recursos utilizados.
- Conocer detalladamente la arquitectura interna de una de las familias más utilizadas de FPGA.
- Adquirir destreza en el uso de herramientas de diseño y tecnologías actuales de circuitos integrados programables.
- Adquirir soltura en la modelización de circuitos y sistemas digitales usando un lenguaje de descripción del hardware como es VHDL.

1- Circuitos integrados

El primer circuito integrado fue desarrollado en 1959 por el ingeniero Jack Kilby (1923-2005) pocos meses después de haber sido contratado por la firma Texas Instruments. Se trataba de un dispositivo de germanio que integraba seis transistores en una misma base semiconductora para formar un oscilador de rotación de fase. En el año 2000 Kilby fue galardonado con el Premio Nobel de Física por la enorme contribución de su invento al desarrollo de la tecnología. Por circuito integrado se entiende una pequeña pastilla de semiconductor de unos pocos milímetros cuadrados de área en la que se encuentra un gran número de transistores que está protegida dentro de un encapsulado de plástico o cerámica [5]. Un ejemplo de circuito integrado se encuentra en la figura 1.

Al mismo tiempo que Jack Kilby, pero de forma independiente, Robert Noyce desarrolló su propio circuito integrado, que patentó unos seis meses después. Además resolvió algunos problemas prácticos que poseía el circuito de Kilby, como el de la interconexión de todos los componentes; al simplificar la estructura del chip mediante la adición del metal en una capa final y la eliminación de algunas de las conexiones, el circuito integrado se hizo más adecuado para la producción en masa. Además de ser uno de los pioneros del circuito integrado, Robert Noyce también fue uno de los cofundadores de Intel [6], [7].

Los avances que hicieron posible el circuito integrado han sido, fundamentalmente, los desarrollos en la fabricación de dispositivos semiconductores a mediados del siglo XX y los descubrimientos experimentales que mostraron que estos dispositivos podían remplazar las funciones de las válvulas o tubos de vacío, que se volvieron rápidamente obsoletos al no poder competir con el pequeño tamaño, el consumo de energía moderado, los tiempos de conmutación mínimos, la fiabilidad, la capacidad de producción en masa y la versatilidad de los circuitos integrados.

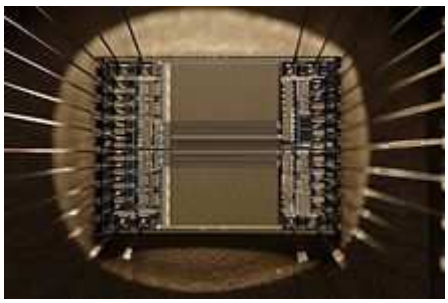


Fig. 1. Circuito integrado correspondiente a una memoria ROM

1.1– Escalas de integración y ley de Moore

Los circuitos integrados van evolucionando: se fabrican en tamaños cada vez más pequeños, con mejores características y prestaciones, mejoran su eficiencia y su eficacia, permitiendo así que mayor cantidad de elementos sean integrados en un mismo chip.

Atendiendo al nivel de integración los circuitos integrados se pueden clasificar en:

SSI (Small Scale Integration) escala pequeña: de 10 a 100 transistores

MSI (Medium Scale Integration) escala media: 101 a 1.000 transistores

LSI (Large Scale Integration) escala grande: 1.001 a 10.000 transistores

VLSI (Very Large Scale Integration) escala muy grande: 10.001 a 100.000 transistores

ULSI (Ultra Large Scale Integration) escala ultra grande: 100.001 a 1.000.000 transistores

GLSI (Giga Large Scale Integration) escala gigante: más de un millón de transistores

La evolución de los circuitos integrados ha seguido la Ley de Moore, que expresa que aproximadamente cada 2 años se duplica el número de transistores en un circuito integrado (Figura 2). Se trata de una ley empírica, formulada por el cofundador de Intel, Gordon E. Moore el 19 de abril de 1965, cuyo cumplimiento se ha podido constatar hasta hoy con pequeños ajustes [7].

En 1965 Gordon Moore afirmó que la tecnología tenía futuro, que el número de transistores por unidad de superficie en circuitos integrados se duplicaba cada año y que la tendencia continuaría durante las siguientes dos décadas. Más tarde, en 1975, modificó su propia ley al corroborar que el ritmo bajaría, y que la capacidad de integración se duplicaría aproximadamente cada 24 meses. Esta progresión de crecimiento exponencial, duplicar la capacidad de los circuitos integrados cada dos años, es lo que se considera la Ley de Moore.

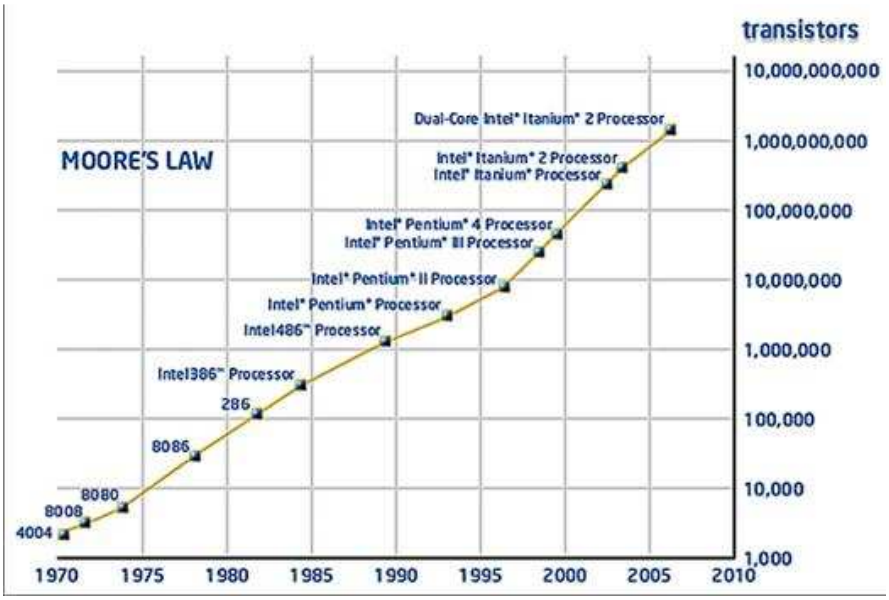


Fig. 2. Aumento de la integración en las últimas décadas

Para hacerse una idea más visual de la escala de integración se muestra la figura 3 obtenida de la web de Intel en la que se hace la comparación entre transistores y personas [7]. La comparación consiste en imaginar que si las personas fueran transistores, debido al desarrollo tecnológico, en la actualidad toda la población de China entraría en un teatro que tenía una capacidad original de 2300 personas.

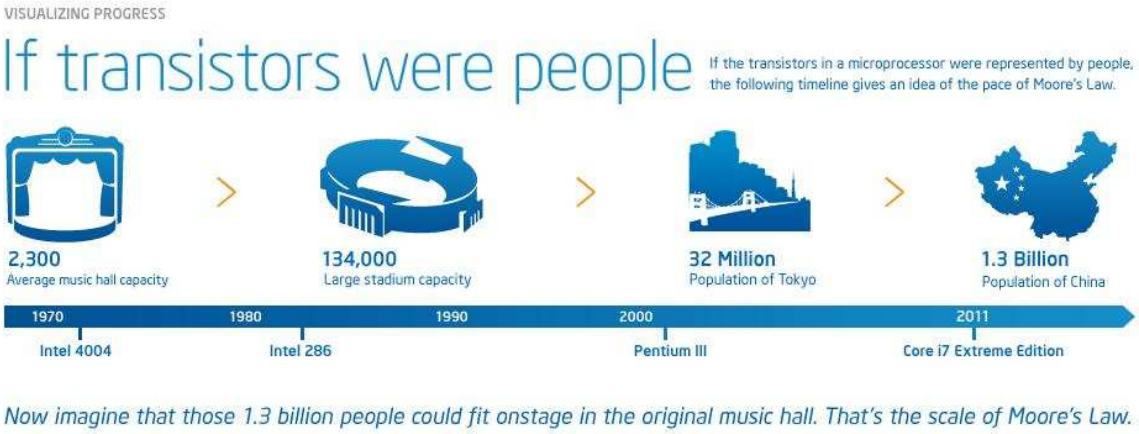


Fig. 3. Comparación personas-transistores si los transistores en un microprocesador fuesen personas

2- Dispositivos lógicos programables

2.1- Aspectos generales

Los circuitos integrados tuvieron dos vías de desarrollo, una dedicada al diseño de componentes estándar y otra dedicada al desarrollo de circuitos integrados de aplicación específica (ASIC) [8]. Por componentes estándar se entiende que son los elementos básicos de los circuitos digitales como son las puertas lógicas, los biestables, las unidades aritmético-lógicas… mientras que los ASIC son circuitos diseños para realizar una función muy específica.

Dentro del diseño de los ASIC se distinguen también dos tipos de diseño que se denominan Full-Custom y Semi-Custom. En el primero el diseño se hace totalmente a medida de la aplicación y en el otro la libertad de diseño no es total. Para el Semi-Custom hay tres tipos de dispositivos: Gate Arrays, Standar Cells y dispositivos de lógica programable.

Los Gate Arrays o matrices de puertas predifundidas consisten en una estructura regular de transistores prediseñada que se personaliza en la última fase del diseño mediante conexionado. Las Standard Cells o celdas estándar precaracterizadas son conjuntos de celdas prediseñadas con librerías de funciones para cada tecnología. Los dispositivos de lógica programable (PLD) están formados por una estructura prefabricada que se configura desde el exterior del dispositivo o chip por el propio usuario.

La estructura básica del PLD [8] permite realizar cualquier circuito combinacional mediante una estructura interna compuesta por una matriz de puertas AND seguida de una matriz de puertas OR. Si se añaden elementos de memoria se pueden implementar circuitos secuenciales. Dependiendo de qué matriz sea programable se tiene un tipo de estructura distinto. Si la matriz programable es la matriz OR solamente es de tipo PROM (Programmable Read Only Memory), si es la matriz AND es de tipo PAL (Programmable Array Logic) y si ambas son programables es de tipo PLA (Programmable Logic Array).

Si el número de puertas equivalentes (número de puertas NAND equivalentes que se podrían programar en un dispositivo) del dispositivo es menor de 200 se dice que se trata de un PLD simple (SPLD) y por el contrario si es mayor de 200 se trata de un PLD complejo (CPLD). Además si un PLD va acompañado de un microprocesador en el mismo circuito integrado se dice que se tiene un SoPC (System on Programmable Chip).

Dentro de los PLD existe otro subconjunto de dispositivos denominados FPGA (Field Programmable Gate Array). La arquitectura de una FPGA (Figura 4) consiste en una matriz de celdas lógicas que se comunican entre sí y con las celdas de entrada/salida (I/O) a través de canales de comunicación tanto horizontales como verticales. Los canales se conectan mediante elementos programables y en general las celdas lógicas son más simples que un SPLD. En general están formadas por multiplexores, tablas look-up (LUTs) y elementos de memoria (flip-flop).

La enorme libertad disponible en la interconexion de dichos bloques confiere a las FPGA una gran flexibilidad. En la mayoría de las FPGA se pueden encontrar funciones de alto nivel (como

sumadores y multiplicadores) embebidas en la propia matriz de interconexiones, así como bloques de memoria [9].

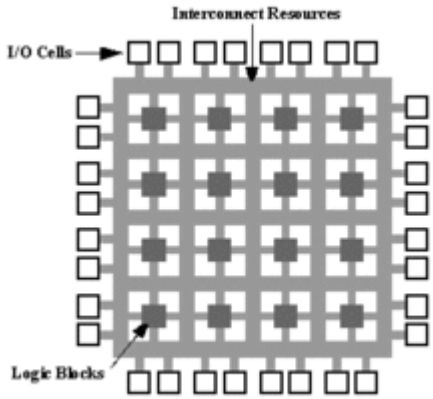


Fig. 4. Arquitectura genérica de una FPGA

Para hacer la programación de dispositivo el diseñador cuenta con la ayuda de entornos de desarrollo especializados en el diseño de sistemas a implementarse en una FPGA [3], [10]. Un diseño puede ser capturado ya sea como esquemático, o haciendo uso de un lenguaje de programación con capacidad para modelizar el paralelismo o concurrencia característico de los sistemas digitales [8]. Estos lenguajes de programación especiales son conocidos como HDL o Hardware Description Language (lenguajes de descripción de hardware). Los HDLs más utilizados son: VHDL y Verilog.

2.2- Familia Virtex-5

Una de las familias de PLD más utilizadas por los diseñadores en los últimos años ha sido la familia Virtex-5 del fabricante Xilinx [2]. La familia de FPGA Virtex-5 está construida con tecnología de 65 nanómetros y necesita una alimentación entre 1.2V y 3.3V dependiendo del dispositivo. Las Virtex-5 se basan en la arquitectura ASMBL (Advanced Silicon Modular Block) con una interconexión diagonal que proporciona rutas más rápidas. Cada celda lógica, aproximadamente hay hasta 330.000 celdas lógicas en los dispositivos de la familia Virtex-5, consta de cuatro LUTs de 6 entradas independientes que también se puede configurar como dos LUTs de 5 entradas compartidas (Figura 5).

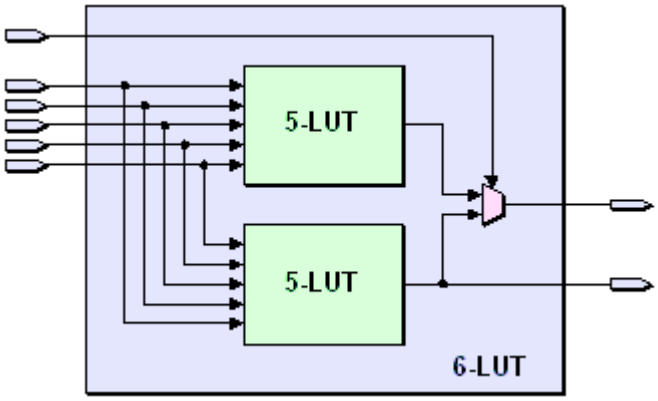


Fig. 5. LUT de 6 entradas de la familia Virtex-5

Las estructuras de bloques de RAM que aparecen en la familia Virtex-5 tienen una capacidad de 36 Kbit y operan a 550 MHz. También cuenta con bloques para la gestión del reloj (CMT - Clock Management Tiles), compuestos por 2 bloques DCM (Digital Clock Manager) y un bucle de enganche de fase (PLL -Phase-Locked Loop). Además lleva bloques DSP (Digital Signal Processing) que fundamentalmente los forman un multiplicador de 25x18 bits en complemento a dos y un acumulador de 48 bits capaces de funcionar a una frecuencia de 550 MHz y en lo que se refiere a entrada y salida de señales esta familia de dispositivos ofrece hasta 1,200 pines de I/O de propósito general que se presentan como 30 bancos de 40 pines.

En este trabajo el dispositivo que se va a utilizar es la FPGA de propósito general modelo xc5v1x50t de la familia Virtex-5 que tiene las siguientes características: 28,800 bloques lógicos, 60 bloques de memoria RAM, 48 bloques DSP, 6 CMT y 480 pines I/O [2].

2.3-Métodos de diseño para la optimización del consumo energético en FPGA

Una de las desventajas de las FPGA frente a otras tecnologías de circuitos integrados es su alto consumo. Por este motivo es importante minimizar este aspecto durante la fase de diseño. El consumo de energía posee dos componentes; la componente estática debida a las corrientes de fuga de los transistores y la componente dinámica que en general, en la tecnología CMOS, está relacionada con la carga y descarga de las capacidades parásitas. La potencia disipada (P) en estos elementos es proporcional a la capacidad (C), al cuadrado del voltaje (V), a la frecuencia (f) y al número de nodos activos (n) como se muestra en la ecuación 1 [11].

$$P = n \times V^2 \times C \times f \qquad (1)$$

Aunque la mayor parte del consumo se debe a la componente estática se puede tratar de minimizar la contribución de la componente dinámica mediante distintas técnicas [12].

2.3.1- Control del reloj

Una forma de reducir el consumo de energía en circuitos síncronos es deshabilitar el reloj en las partes del circuito en las que no se necesita o que no es necesario que este activo en todas las etapas. Para ello se puede añadir una señal de habilitación (enable) o un multiplexor en lugar de conectar directamente el reloj de activación periódica. En la figura 6 se puede ver un circuito en el que la parte encerrada por la línea punteada se puede conectar o desconectar del reloj principal utilizando una puerta AND y la señal de clock enable.

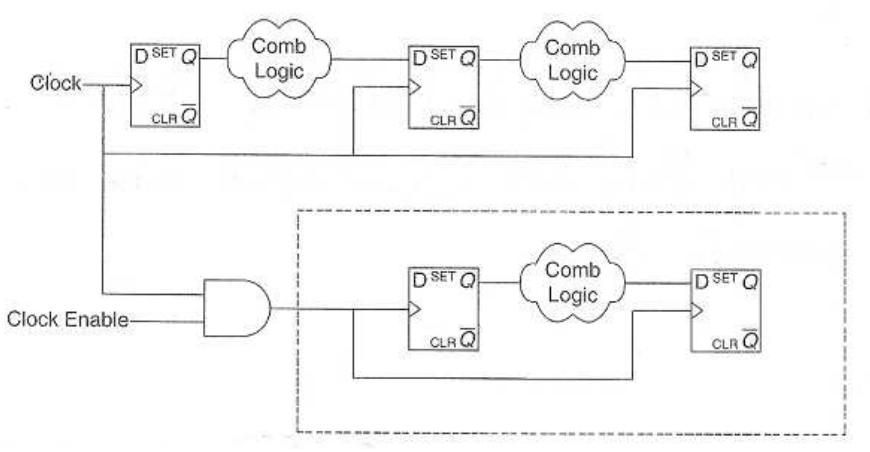


Fig. 6. Circuito síncrono con señal clock enable simple

De esta forma la parte del circuito encerrada por la línea punteada solo trabaja cuando es necesario por lo que se reduce la frecuencia promedio del reloj, de modo que se disipa menos energía de acuerdo con la ecuación (1), pero la implementación y el análisis temporal son más complejos.

Otro factor a tener en cuenta es la importancia del retardo del reloj en el diseño de lógica secuencial ya que puede darse el caso de que este retardo haga funcionar de forma incorrecta al dispositivo. Un ejemplo de ello se puede ver en la figura 7 donde si el retardo producido por los elementos lógicos definido como dL es menor que el retardo del reloj dC puede ocurrir que la señal propagada por el segundo flip-flop llegue antes al tercer flip-flop que el flanco activo del reloj. Entonces cuando llegue el flanco del reloj se volverá a propagar la misma señal en lugar de la que se tenía que haber propagado.

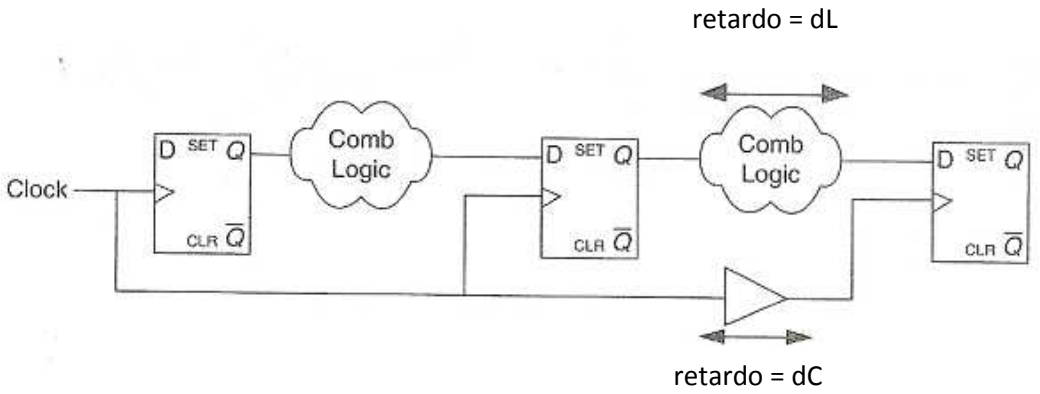


Fig. 7. Retardo de reloj en un circuito digital

Además puede ocurrir que no se tengan en cuenta estor retardos al realizar una simulación funcional pero más adelante, a la hora de implementar el diseño, pueden provocar que el circuito no realice correctamente su función.

2.3.2-Control de las entradas

En tecnología CMOS, gracias al uso de transistores de canal N y canal P y a las redes duales que forman, en los estados estables no existe corriente de drenador de forma que no se disipa energía. Idealmente los transistores que forman el circuito deberían pasar de ON a OFF de forma instantánea de tal manera que no existe corriente de drenador. Sin embargo en la realidad esas transiciones no son instantáneas por lo que hay corrientes de drenador que disipan energía. Por ello se trata de minimizar las subidas y las bajadas de las señales de entrada.

2.3.3-Reducción del voltaje

La reducción de la alimentación reduce bastante el consumo ya que este es proporcional al cuadrado del corriente y ésta como se puede ver en la ecuación (1) es proporcional al voltaje. Los fabricantes si tratan de que sus dispositivos funcionen con la menor alimentación posible pero desde el punto de vista de los usuarios no es una opción conveniente debido a que entonces el rendimiento del dispositivo se ve afectado negativamente.

2.3.4-Flip-flops de doble flanco

La corriente disipada es proporcional a la frecuencia, según la ecuación (1), por ello hay que maximizar la eficiencia de cada ciclo. Este tipo de flip-flops están diseñados para activarse con los dos flancos del reloj, tanto el ascendente como el descendente, en lugar de solo activarse con uno como en la mayoría de los flip-flops. Esto permite al dispositivo trabajar con una frecuencia que es la mitad de la que tendría que usar sin este tipo de flip-flop y así reducir la energía disipada. Si no están disponibles se pueden añadir flip-flops redundantes para emular el efecto del doble flanco pero esta solución suele ser contraproducente.

3- Algoritmo AES

3.1-Preliminares matemáticos

La seguridad del cifrado Rijndael se basa en el desorden provocado en el contenido del texto original al aplicarle una serie de permutaciones y otras operaciones matemáticas. Algunas de estas operaciones se realizan a nivel de byte (8 bits), representado mediante el campo de Galois GF(2^8) y otras a nivel de palabras de cuatro bytes [13],[14]. Para la comprensión del algoritmo es necesario conocer cada una de estas operaciones, para lo cual se deberá introducir una serie de conceptos matemáticos antes de iniciar el estudio del algoritmo propiamente dicho.

3.1.1- Cuerpos Finitos. GF(2^8)

El campo GF(2^8) se trata de un campo finito en el que los elementos serán representados como polinomios de grado 7 y con coeficientes binarios, esto es, en {0,1}.

Un byte b se compone de 8 bits que representamos como b7, b6, b5, b4, b3, b2, b1, b0 donde b7 representa el bit de mayor peso y b0 al de menor. Así podemos representar el byte como un polinomio cuyos coeficientes son los bj con j=0…7 y donde estos bj pueden tomar los valores 0 ó 1 como en la expresión (2):

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

Por ejemplo, un byte que represente el valor hexadecimal '57' (en binario 01010111) se corresponde con el polinomio

$$0x^7 + 1x^6 + 0x^5 + 1x^4 + 0x^3 + 1x^2 + 1x^1 + 1 = x^6 + x^4 + x^2 + x + 1$$

El cuerpo finito GF(2^8) tiene dos operaciones internas suma y multiplicación (+ y • respectivamente) tal que si a ϵ GF(2^8) y b ϵ GF(2^8) :

- a+b ϵ GF(2^8)
- a+'0'=a
- a•b ϵ GF(2^8)
- a•'1'=a

A continuación se muestra cómo se realizan las operaciones

3.1.2- Suma

La suma de dos elementos del campo GF(2^8) es la suma de dos polinomios, por lo que el resultado será otro polinomio. La suma de los coeficientes se corresponde con una suma modulo 2 término a término. Se puede comprobar que esta suma se corresponde con una operación EXOR (denotada por $\oplus$) entre los coeficientes de los polinomios.

Por ejemplo:

('57' +'83') = $(x^6 + x^4 + x^2 + x + 1) + (x^7 + x + 1) = x^7 + x^6 + x^4 + x^2$

Se puede comprobar que el conjunto de los polinomios de grado menor o igual que 7 y con coeficientes pertenecientes a {0,1} forman un grupo conmutativo con la suma, es decir, es una operación interna, que cumple la propiedad asociativa, conmutativa, tiene elemento neutro y tiene simétrico. Debido a la existencia de simétrico podemos referirnos a la operación resta, ya que se puede definir la resta de a y b, donde a y b son polinomios, como la suma de a con el simétrico de b.

3.1.3- Multiplicación

La multiplicación empleada en el algoritmo Rijndael se refiere realmente a la multiplicación de dos elementos del conjunto GF(2^8), es decir polinomios de grado menor o igual que 7 y con coeficientes en {0,1} pero cuyo resultado se expresa modulo m(x) definido en la expresión (3)

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

Nótese que m(x) se puede representar en hexadecimal con el valor '11B' y se puede comprobar que es un polinomio irreducible. El propósito de realizar la multiplicación módulo m(x) es con el fin de que el resultado obtenido en la operación siga siendo un polinomio de grado menor que 8, por lo que la operación seguiría siendo a nivel de byte.

Por ejemplo, la operación multiplicación de los valores hexadecimales '57' y '83'sería:

$(x^6 + x^4 + x^2 + x +1) \bullet (x^7 + x + 1) =$

$=(x^6 + x^4 + x^2 + x + 1) + (x^7 + x^5 + x^3 + x^2 + x) + (x^{13} + x^{11} + x^9 + x^8 + x^7) =$

$= x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1$

Como se puede apreciar este polinomio es de grado mayor que 8 por lo que no pertenece a GF(2^8) y así la operación no se realiza a nivel de byte. Para remediar esto y conseguir que la multiplicación siga siendo una operación interna en GF(2^8) se expresa el resultado obtenido modulo m(x)

$(x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1) \bmod m(x)$

$= x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1 \bmod (x^8 + x^4 + x^3 + x + 1)$

$= x^7 + x^6 + 1$

Un caso destacado en cuanto a la multiplicación de polinomios en GF(2^8) es la multiplicación de un polinomio b(x) de grado 7 por el polinomio a(x)= x

$a(x) \bullet b(x) = x \bullet (b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x^1 + b_0)$

$= b_7x^8 + b_6x^7 + b_5x^6 + b_4x^5 + b_3x^4 + b_2x^3 + b_1x^2 + b_0x$

Como se puede apreciar, este polinomio no pertenece a GF(2^8) ya que su grado es 8. Para que la operación sea interna en GF(2^8) se divide entre m(x) como antes. Si b7=0 el resultado es el mismo polinomio. Si b7=1, m(x) debe anular el valor de x^8.

En general, para utilizar este tipo de multiplicación se suele definir una función denominada habitualmente "xtime" que simplifica la multiplicación de un polinomio por potencias de x, este hecho es gracias a que la función "xtime" se puede ejecutar de forma reiterativa. La función "xtime" consiste en aplicar un desplazamiento a la izquierda del valor que representa el polinomio y una operación EXOR con el valor '11B' cuando el resultado de la multiplicación debe ser reducido.

3.2-Especificación del algoritmo

Estrictamente hablando, AES es un caso particular Rijndael, ya que Rijndael permite un mayor rango de tamaño de bloques y longitud de claves. AES tiene un tamaño de bloque fijo de 128 bits y tamaños de llave de 128, 192 o 256 bits (4 filas y Nb columnas, Nb=4, 6 y 8 respectivamente), 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 (4 filas y Nk columnas con Nk desde 4 hasta 8). AES opera sobre una matriz de 4×4 bytes, llamada matriz de estado [4].

La estructura del algoritmo Rijndael esta formado por un conjunto de rondas, entendiendo por rondas un conjunto de reiteraciones de 4 funciones matemáticas diferentes e invertibles denominadas habitualmente SubBytes, ShiftRows, MixColumns y AddRoundKey que se explican en los siguientes apartados..

El número de rondas que se producen viene dado por los tamaños de los bloques y claves mediante la expresión (4).

$$Nr=\max(Nb,Nk)+6 \qquad (4)$$

3.2.1-Descripción del cifrado

El proceso de cifrado consiste en la aplicación de las 4 funciones matemáticas invertibles sobre la información que se desea cifrar. Estas transformaciones se realizan dependiendo de la ronda en la que se encuentre el cifrado. Gráficamente se puede ver en la figura 8:

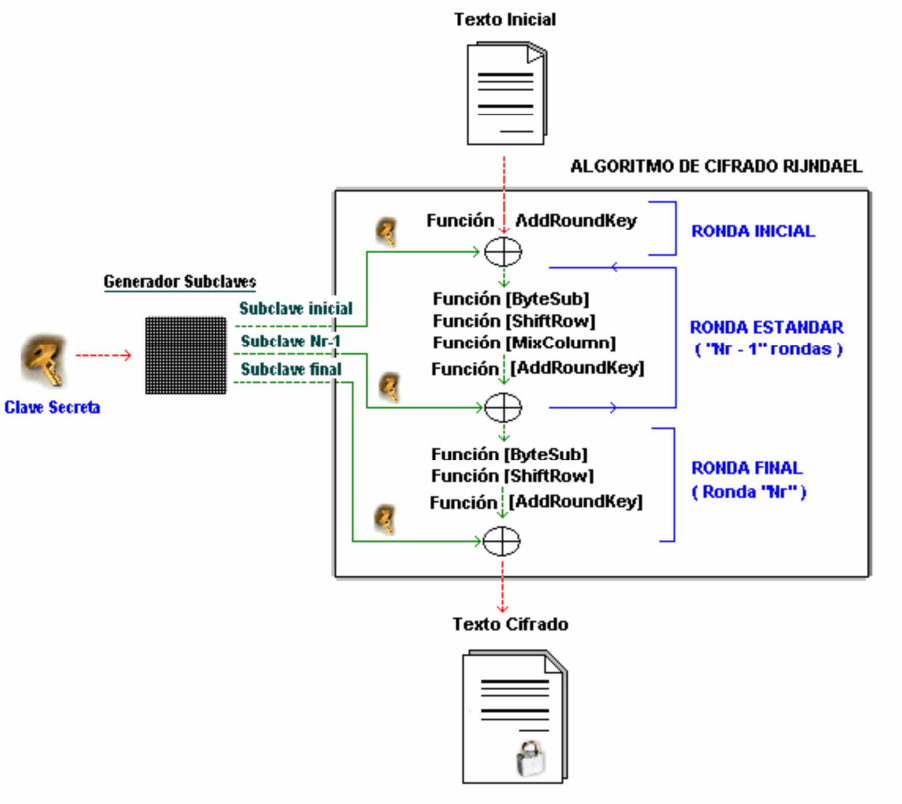


Fig. 8. Esquema de cifrado

La ronda inicial consiste en un AddRoundKey entre la matriz de estado y la subclave generada. A continuación, a la matriz de estado resultante se le aplican las 4 operaciones Nr-1 veces en lo que se denomina ronda estándar. El cifrado acaba con un a ronda final en en la que no se aplica la etapa MixColumns.

3.2.2- SubBytes

En la etapa SubBytes cada byte en la matriz es sustituido usando la caja-S de Rijndael de 8 bits (Figura 9). Esta operación provee la no linealidad en el cifrado. La caja-S utilizada proviene de la función inversa alrededor del GF(2^8).

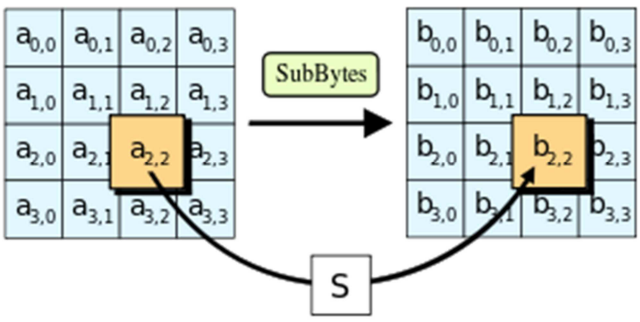


Fig. 9. Etapa SubBytes

Utilizando la caja-S se calculan todos los valores para las posibles entradas dando lugar a una tabla de sustitución denominada S-Box útil para el proceso de cifrado. Seguidamente se tiene en la Tabla I la sustitución mediante la S-Box para un byte genérico 'xy' en hexadecimal [13].

Tabla I Tabla de sustitución de un byte genérico 'xy'

hex		y															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
	1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
	2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
	3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
	4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
	5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
	6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
	7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
	8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
	9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
	a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
	b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
	c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
	d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
	e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
	f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

3.2.3- ShiftRows

El paso ShiftRows opera en las filas de la matriz de estado, rota de manera cíclica los bytes en cada fila por un determinado offset. En AES con Nb=4 y Nb=6, la primera fila queda en la misma posición. Cada byte de la segunda fila es rotado una posición a la izquierda. De manera similar, la tercera y cuarta filas son rotadas por los offsets de dos y tres respectivamente. De esta manera, cada columna de la matriz de estado resultante del paso ShiftRows está compuesta por bytes de cada columna de la matriz de estado inicial. Para Nb=8 la primera fila no rota, la segunda tiene offset de 1, la tercera de 3 y la cuarta de 4. En la figura 10 se muestra la transformación en el caso Nb=4 [15].

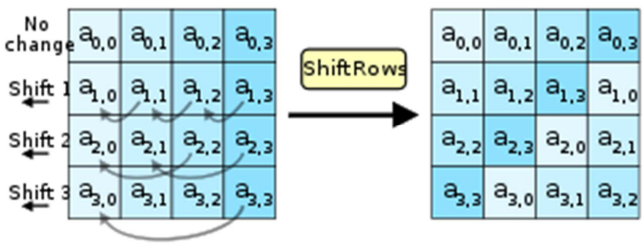


Fig. 10. Etapa ShiftRows

3.2.4- MixColumns

La transformación MixColumns actúa sobre los bytes de una misma columna de la matriz de estado. En esencia esta función permite una mezcla de los bytes de las columnas. Esta transformación considera las columnas de bytes como polinomios cuyos coeficientes pertenecen a GF(2^8), es decir , son también polinomios.

La función MixColumns consiste en multiplicar las columnas de bytes módulo x^4+1 por el polinomio c(x) [1] como se ve en la figura 11. c(x) toma el valor de la expresión (5):

$$c(x)='03'x^3+'01'x^2+'01'x+'02' \quad (5)$$

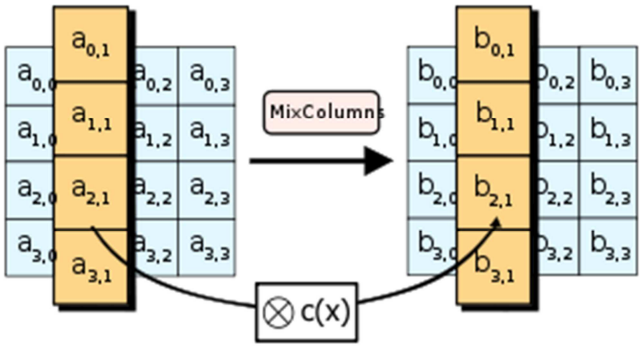


Fig. 11. Etapa MixColumns

Este polinomio c(x) es invertible y la función se puede representar algebraicamente como se muestra en la expresión (6).

$$s'(x)=c(x)\bullet s(x) \quad (6)$$

Donde s'(x) representa la matriz de estado resultante de la transformación y s(x) la matriz de estado entrante. La formula (6) queda mejor expresada en forma matricial como se muestra en la expresión (7).

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix} \quad (7)$$

Desarrollando la matriz de la expresión (7), se observa claramente como cada byte nuevo de la matriz de estado es una combinación de varios bytes de las distintas filas que forman una columna específica:

$$
\begin{aligned}
s'_{0,c} &= (\{02\} \bullet s_{0,c}) \oplus (\{03\} \bullet s_{1,c}) \oplus s_{2,c} \oplus s_{3,c} \\
s'_{1,c} &= s_{0,c} \oplus (\{02\} \bullet s_{1,c}) \oplus (\{03\} \bullet s_{2,c}) \oplus s_{3,c} \\
s'_{2,c} &= s_{0,c} \oplus s_{1,c} \oplus (\{02\} \bullet s_{2,c}) \oplus (\{03\} \bullet s_{3,c}) \\
s'_{3,c} &= (\{03\} \bullet s_{0,c}) \oplus s_{1,c} \oplus s_{2,c} \oplus (\{02\} \bullet s_{3,c})
\end{aligned}
\qquad (8)
$$

3.2.5- AddRoundKey

Consiste en una operación EXOR entre los elementos de la matriz del estado y los elementos de la matriz de la subclave .Esta subclave es el resultado de aplicar el proceso de selección sobre la expansión de la clave principal.

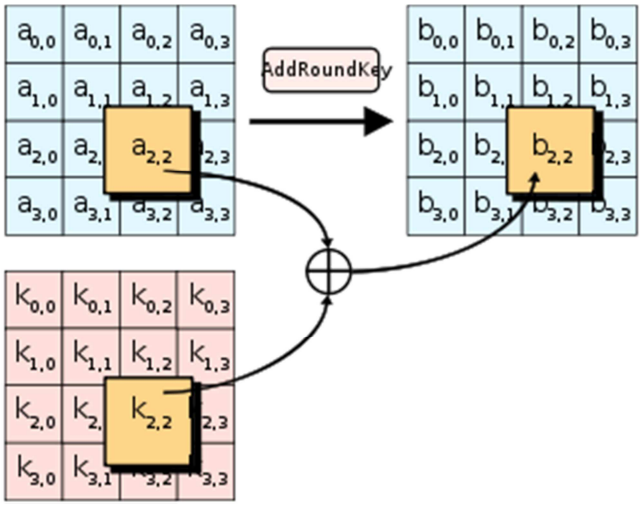


Fig. 12. Etapa AddRoundKey

3.2.5.1- Selección de clave

La función selección de clave toma consecutivamente, de la secuencia obtenida por la función de expansión de clave, bytes que va asignado a cada subclave hasta formar bloques del mismo tamaño que la matriz de estado [13].

3.2.5.2- Expansión de clave

La función expansión de clave permite generar bytes útiles como subclaves a partir de la clave del sistema. Esta función de expansión se puede describir como un array lineal donde las primeras Nk palabras son las de la clave inicial [13].

3.2.6- Proceso de Descifrado

Las operaciones SubBytes, ShiftRows, MixColumns y AddRoundKey poseen funciones inversas que se suelen denominar InvSubBytes, InvShiftRows, InvMixColumns e InvAddRoundKey. Además se usa la misma función de expansión de clave pero en el proceso de descifrado la función de selección de clave es al revés. Así las rondas se ejecutarían de la siguiente manera:

- Ronda inicial: InvAddRoundKey, InvShiftRows e InvSubBytes
- Ronda normal: InvAddRoundKey, InvShiftRows, InvMixColumns e InvSubBytes
- Ronda final: InvMixColumns

4-Diseño e Implementación

En este apartado se procede a realizar el diseño y la implementación del algoritmo AES mediante la FPGA descrita en el apartado 2.2. Para ello es necesario trabajar con el lenguaje de descripción del harware VHDL [16], [3]. VHDL es el acrónimo que representa la combinación de VHSIC (Very High Speed Integrated Circuit) y HDL (Hardware Description Language). Es un lenguaje definido por el IEEE (Institute of Electrical and Electronics Engineers) usado por ingenieros para describir circuitos digitales. Otros métodos para diseñar circuitos son la captura de esquemas (con herramientas CAD) y los diagramas de bloques, pero éstos no son prácticos en diseños complejos.

En VHDL existen distintas formas de describir un mismo circuito de tal forma que es el diseñador el que tiene que elegir cual de ellas le conviene más para el circuito que está diseñando. Las formas básicas de describir un circuito son funcional, de flujo de datos y estructural. En la forma funcional lo que se hace es describir mediante sentencias cual es el comportamiento del circuito. Mediante el flujo de datos se describe el circuito a nivel de transferencia de registro (RTL) y mediante el tipo estructural se describe el circuito con instancias de componentes que forman un diseño de jerarquía superior. La estructura del código consta de llamadas a librerías, una entidad (ENTITY) que consiste en la declaración de las entradas y salidas de un módulo mientras que la arquitectura (ARCHITECTURE) que es la descripción detallada de la estructura interna del módulo o de su comportamiento.

El entorno de diseño que se va a utilizar es el entorno ISE Design Suite 13.2 de la compañía Xilinx [2]. ISE (Integrated Software Environment) es una herramienta para la síntesis y el análisis de diseños HDL que permite al desarrollador compilar los diseños, realizar análisis temporales, examinar diagramas RTL, simular reacciones frente a distintos estímulos y configurar dispositivos. La versión ISE utilizada es la Web Edition que es gratuita pero proporciona un número limitado de dispositivos del fabricante. Suelen quedar fuera de este tipo de versiones los dispositivos de alta gama o que han salido recientemente al mercado. Sin embargo, el dispositivo que se usa en este trabajo de propósito general esta disponible en esta versión.

4.1- Arquitectura propuesta

Teniendo en cuenta lo anterior, lo explicado en el apartado 3 y las técnicas de reducción de consumo se propone una arquitectura reflejada en el diagrama de bloques de la figura 13. El diagrama consiste en una serie de módulos conectados entre si de forma que reproduzcan fielmente el algoritmo AES.

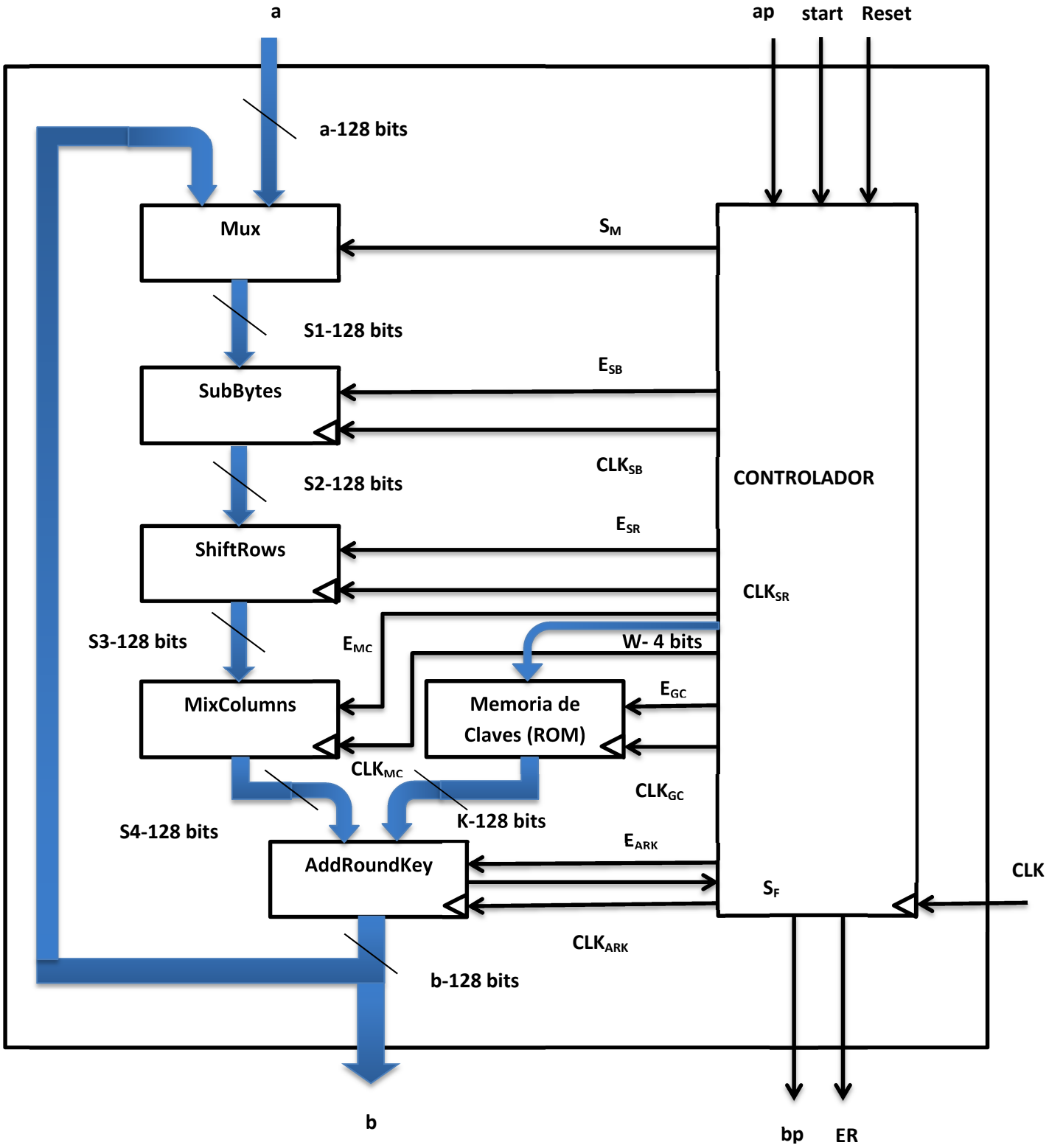


Fig. 13. Diagrama de boques representativa de la arquitectura propuesta

Ahora se procede a explicar las señales y los bloques que aparecen en el diagrama de bloques.

Señales de entrada:

- a: Datos a encriptar(introducidos por filas)
- ap: Datos a encriptar preparados
- start: Comenzar el algoritmo
- Reset: Reset asíncrono
- CLK: Reloj externo

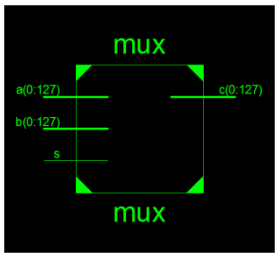
Señales de salida:

- b: Datos encriptados
- bp: Datos encriptados correctamente
- ER: señal de error

Señales internas:

- S1, S2, S3 y S4: Son estados intermedios de la matriz de estado.
- S_M: Señal de selección del multiplexor.
- E_{SB}, CLK_{SB}, E_{SR}, CLK_{SR}, E_{MC}, CLK_{MC}, E_{GC}, CLK_{GC}, E_{ARK} y CLK_{ARK}: Son los habilitadores (enable) y relojes de los módulos SubBytes, ShifRows, MixColumns, Memoria de Claves y AddRoundKey respectivamente.
- S_F: Es una señal de fin, indica que se ha completado una ronda.
- W: Es la señal de selección de la subclave dentro de la Memoria de Claves.
- K: Es una subclave.

Módulos:



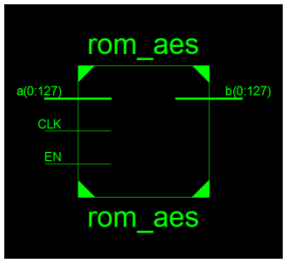
Módulo: Mux

Señales de entrada: a, b y s

Señales de salida: c

Funcionalidad: Es un multiplexor con el que se elije entre las señales de entrada a y b en función de la señal de selección s.

Fig. 14. Módulo Mux



Módulo: SubBytes

Señales de entrada: a, CLK y EN

Señales de salida: b

Funcionalidad: Este módulo realiza la sustitución descrita en la etapa homónima, punto 3.2.2 Para ello se ha hecho uso de una memoria ROM (Figura 15). Además, cuando recibe un flanco ascendente de reloj y el bloque no está habilitado, entonces carga a la señal de

Fig. 15. Módulo SubBytes

salida lo que tiene como señal de entrada en este caso b. Lo mismo ocurre para los módulos ShiftRows, MixColumns y AddRoundKey que cargan en la salida correspondiente.

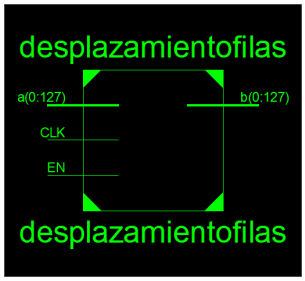


Fig. 16. Módulo ShiftRows

Módulo: ShiftRows

Señales de entrada: a, CLK y EN

Señales de salida: b

Funcionalidad: Este bloque efectúa la operación ShiftRows descrita en el apartado 3.2.3. Este módulo se diferencia sustancialmente de los otros módulos porque necesita tres flancos ascendentes para terminar de realizar su función.

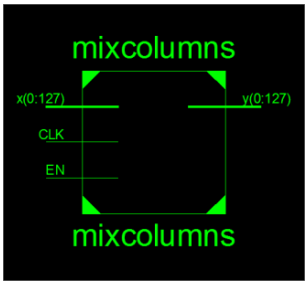


Fig. 17. Módulo MixColumns

Módulo: MixColumns

Señales de entrada: x, CLK y EN

Señales de salida: y

Funcionalidad: Realiza la función comentada en el apartado 3.2.4 en un periodo de reloj.

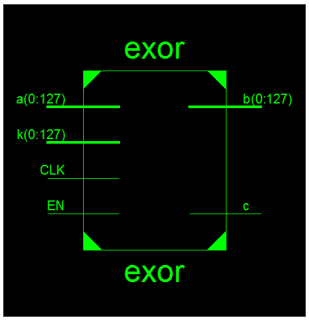


Fig. 18. Módulo AddRoundKey

Módulo: AddRoundKey

Señales de entrada: a, k, CLK y EN

Señales de salida: b y c

Funcionalidad: Es un EXOR entre la matriz de estado intermedia (a) y la subclave (k) correspondiente a la ronda en cuestión. Este módulo además genera una señal de salida c que se activa una vez se ha realizado la operación y el habilitador está activo y se desactiva cuando el habilitador se desactiva.

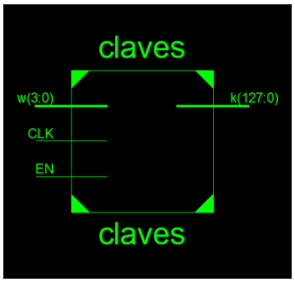


Fig. 19. Módulo Memoria de claves

Módulo: Memoria de claves

Señales de entrada: w, CLK y EN

Señales de salida: k

Funcionalidad: Es una memoria ROM en la que se almacenan las subclaves necesarias que se seleccionan mediante la señal W.

Controlador: Es el responsable de que el resto de módulos reciba las señales precisas como para realizar el algoritmos correctamente. En realidad el controlador está constituido por un contador y por una máquina de estados tipo Moore. En lo que respecta al contador, la señal SF que llega desde el módulo AddRoundKey actúa como reloj de modo que lleva la cuenta de las rondas que se han producido(c en la figura 20). El contador genera dos señales, una es la señal de selección W que es el número de ronda y la otra es una señal z que va a la maquina de estados. Esta señal vale '0' cuando el número de rondas es menor o igual que Nr-2 y '1' cuando es mayor. Además le llega una señal de Reset generada por la máquina de estados. De todas formas, en la figura 21 se muestra una simulación funcional para poder ver como genera las señales el contador.

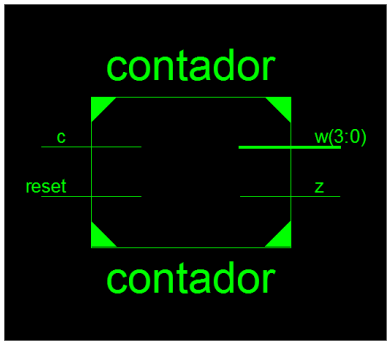


Fig. 20. Contador del Controlador

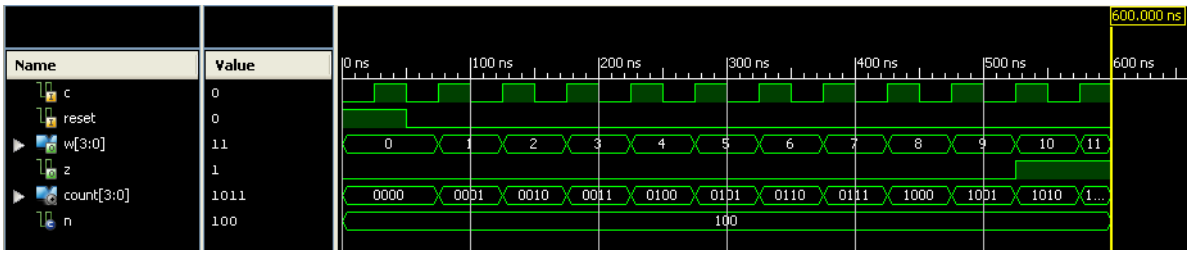


Fig. 21. Simulación funcional del contador

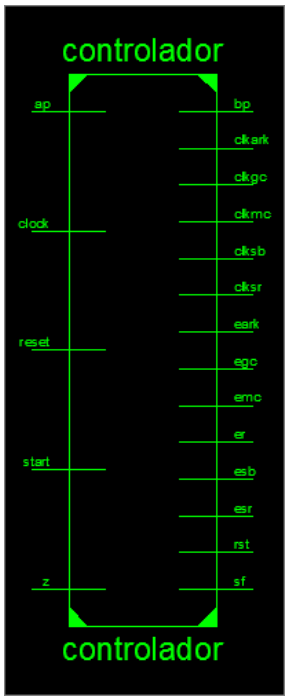


Fig. 22. Diagrama de la máquina de estados

Por otro lado a la máquina de estados le llegan 4 señales externas; ap, start, Reset (lleva la máquina al estado SP) y CLK (reloj externo de todo el sistema) y también le llega la señal z procedente del contador. Las salidas de la máquina de estados son: S_M, E_{SB}, CLK_{SB}, E_{SR}, CLK_{SR}, E_{MC}, CLK_{MC}, E_{GC}, CLK_{GC}, E_{ARK} CLK_{ARK}, bp, er y rst. En la figura 22 se dispone un diagrama con todas las señales de entrada y de salida de la máquina de estados que forma el controlador.

Los estados que forma el controlador guardan un significado que se explica a continuación:

SP: Es el estado de espera del que solo se sale si las señales ap y start están activadas cuando se produce un flanco ascendente de reloj. Además es el estado al que vuelve la máquina tras la activación de la señal de reset.

INI: Es el estado inicial en el que se resetea el contador mediante la señal rst.

FIN: Estado final en el que se activa bp para mostrar que los datos ya han sido encriptados y donde se para de ejecutar el algoritmo.

ERROR: Si se produce alguna transición que no esté contemplada se considera un error y se pasa a este estado donde se para el algoritmo y se activa la señal er para indicar que ha ocurrido un error.

Resto de estados: Se cambian los valores de selección del multiplexor y los habilitadores y relojes del resto de módulos. En la simulación funcional de la figura 23 se puede ver cuales son los estados y como afectan a las señales de salida.

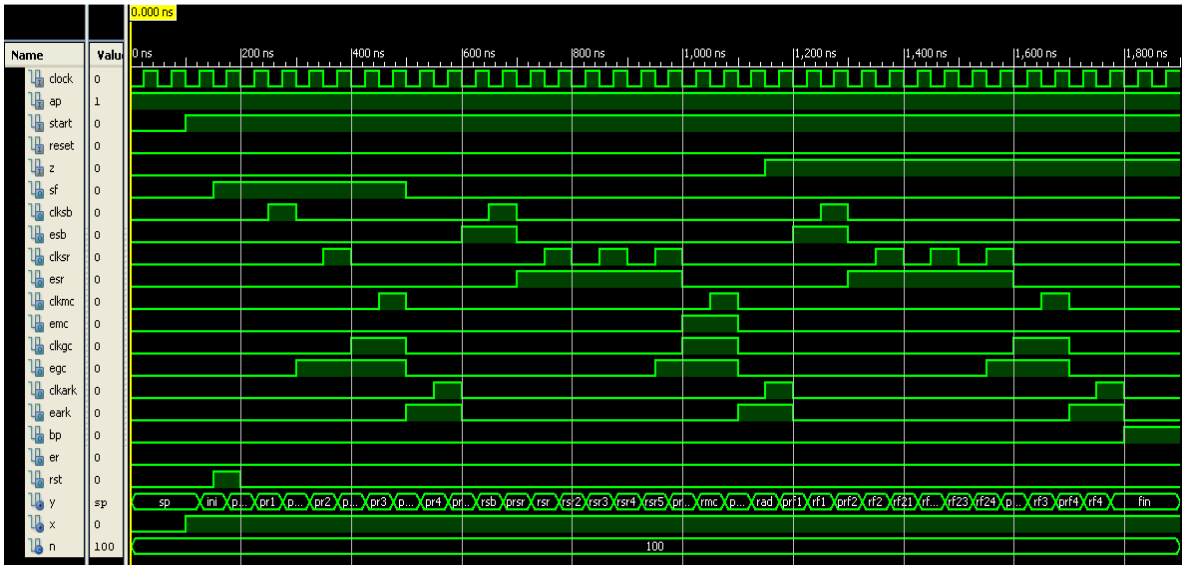


Fig. 23. Simulación funcional del controlador

4.2- Implementación sobre la FPGA

Después de haber comprobado la funcionalidad de cada uno de los componentes por separado, como se ha hecho en el apartado 4.1, se realiza uno nuevo denominado AES. El nuevo elemento se forma mediante instanciaciones del resto de componentes conexionándolas del modo apropiado según el diagrama de bloques propuesto en el apartado 4.1. Una vez sintetizado el diseño se procede a realizar un esquema RTL del componente de mayor jerarquía para ver claramente las señales de entrada y de salida (Figura 24).

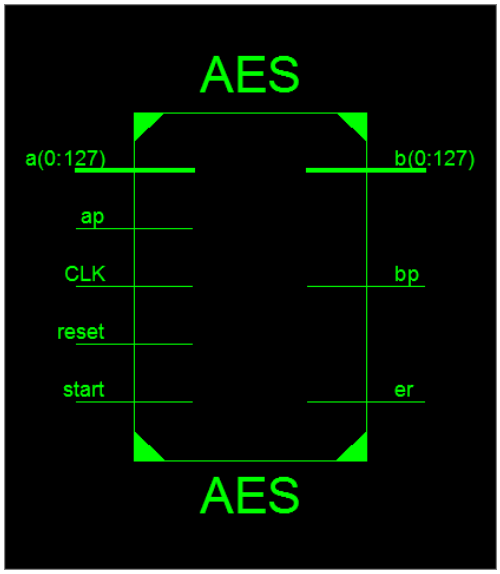


Fig. 24. Esquema RTL del componente de mayor jerarquía AES

Señales de entrada: a (128 bits), ap, CLK (reloj), reset y start.

Señales de salida: b (128 bits), bp y er.

Si a su vez se despliega este componente se puede ver el diagrama de bloques generado por el entorno de diseño. En este diagrama se espera ver si la arquitectura propuesta en el apartado anterior es correcta o si hay formas más eficientes de realizar la implementación. El diagrama de bloques se muestra en la figura 25.

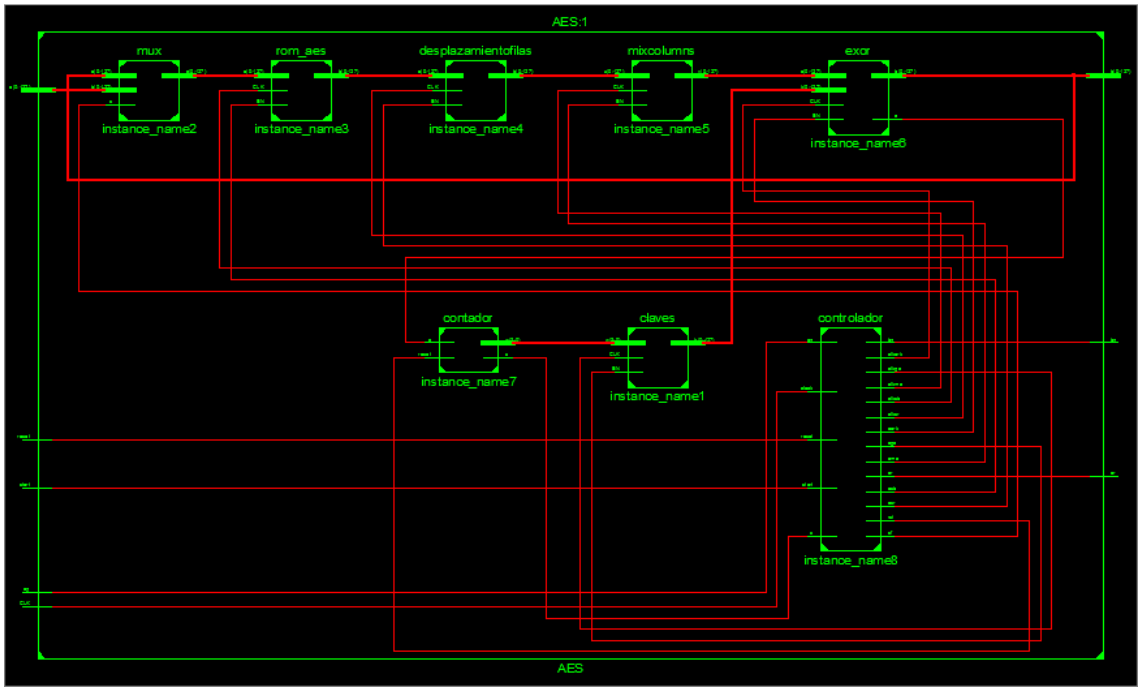


Fig. 25. Diagrama de bloques generado por la herramienta de diseño

Se puede ver que es muy parecida a la arquitectura propuesta ya que solo difiere en que el controlador se divide en dos partes, contador y controlador (máquina de estados). Si se despliegan los componentes se pueden ver qué recursos del dispositivo se están utilizando, siendo los casos de mayor interés los módulos ShiftRows (en la figura aparece como desplazamiento filas), Mixcolumns y controlador.

En el caso del módulo ShiftRows se puede ver como hace uso de registros tipo D conectados de forma análoga a como ocurre en registros de desplazamiento (Figura 26). Algo así se podía esperar dada la funcionalidad del módulo. Además utiliza una serie de multiplexores, que no aparecen todos en la figura, distribuidos de una manera regular y otro registro tipo D para controlar la salida.

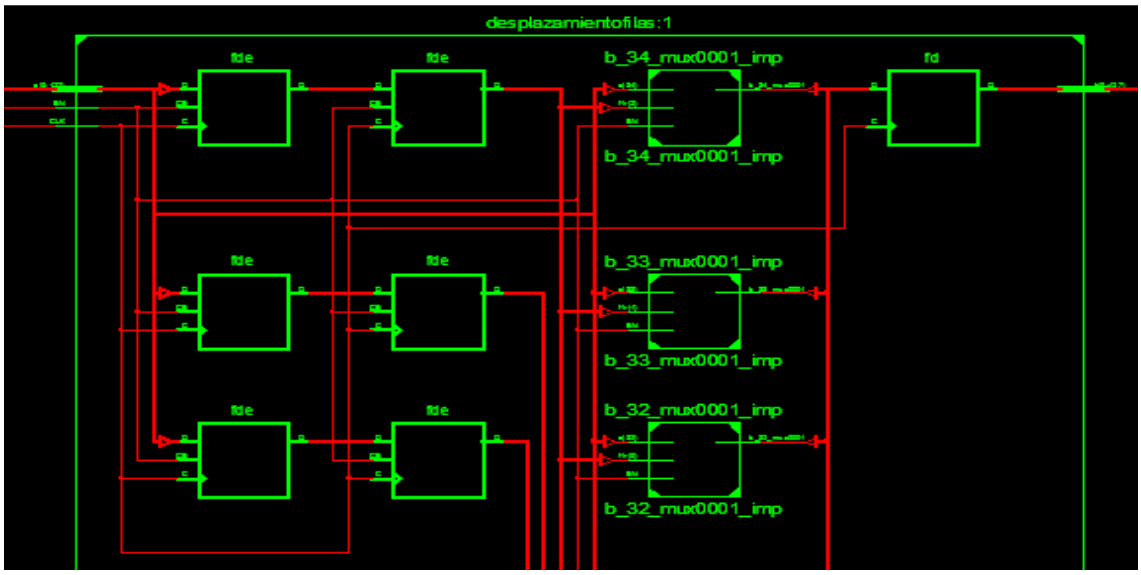


Fig. 26. Elementos utilizados en el módulo ShiftRows

El módulo MixColumns utiliza multiplexores y elementos estándar tipo EXOR ya que en la operación debe realizar esta función es clave como se mencionó en el apartado 2. Este bloque, al igual que el anterior, cuenta con un registro tipo D que controla la salida (Figura 27). Del mismo modo en la figura 27 tampoco aparecen todos los elementos utilizados pero se distribuyen y conectan de una forma muy parecida a como sucede en la figura.

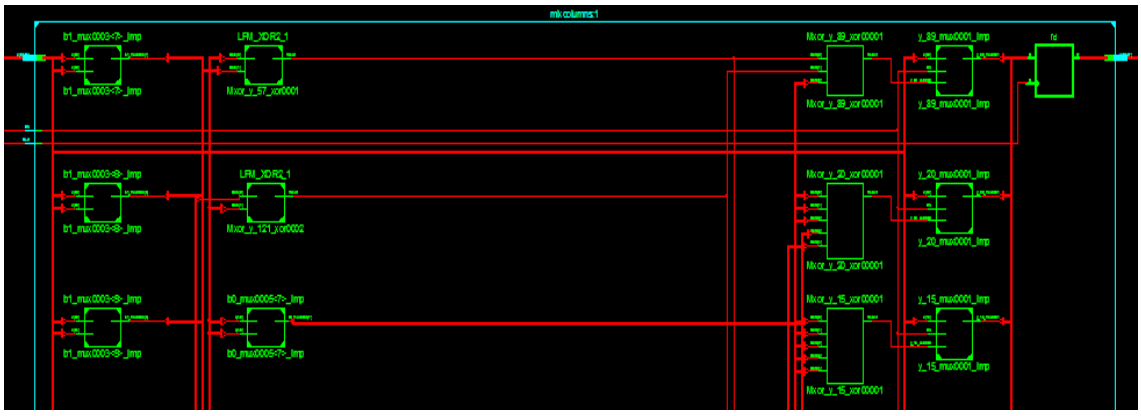


Fig. 27. Componentes del módulo MixColumns

En la figura 28 se puede ver como se implementa el controlador mediante una máquina de estados finitos y una serie de puertas lógicas AND, OR y NOT.

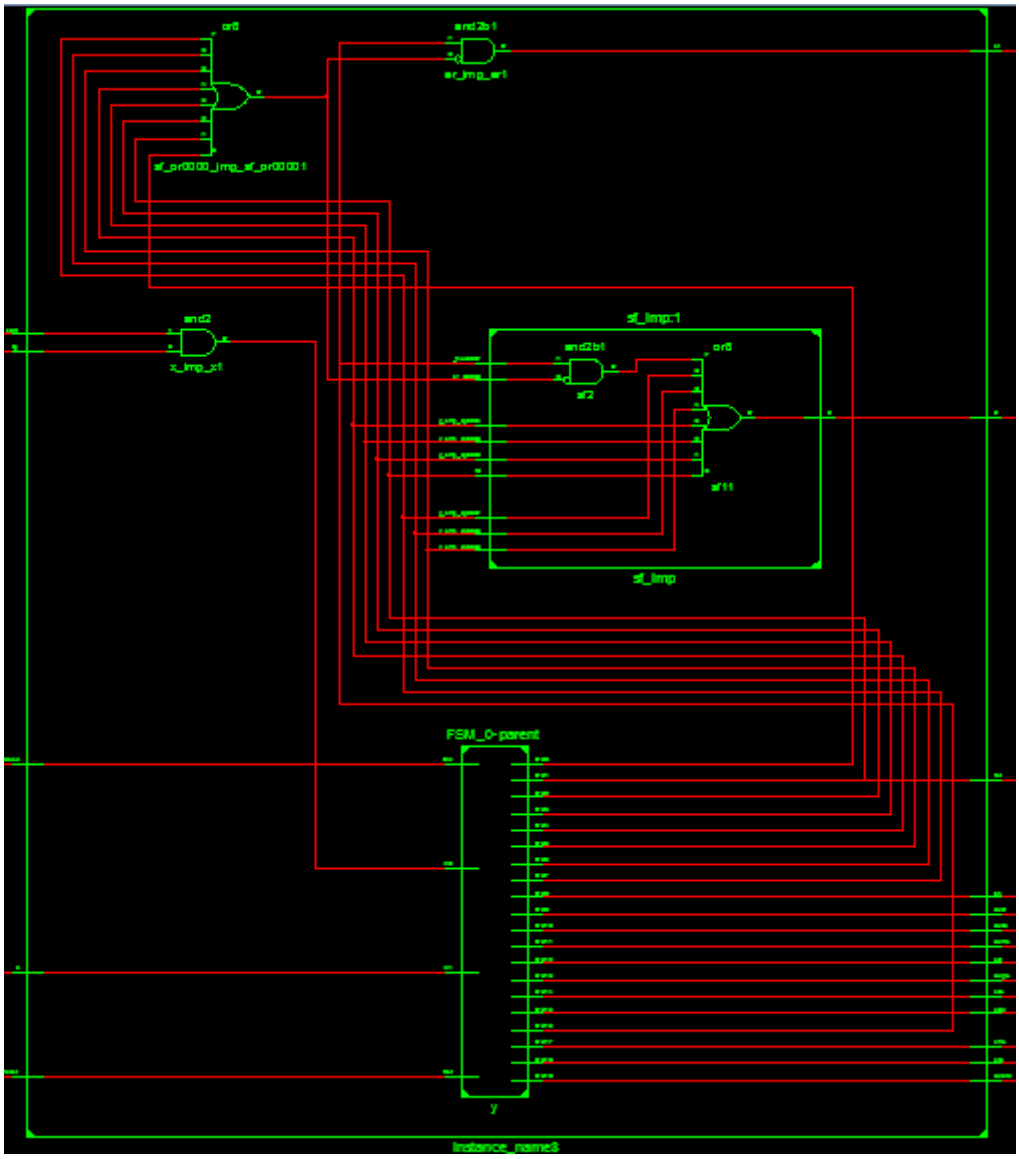


Fig. 28. Implementación del controlador

Una vez realizada la síntesis y la implementación del diseño se puede obtener, de los informes elaborados por el entorno de diseño ISE, los recursos concretos utilizados y la frecuencia de operación máxima. La tabla II recoge los datos de los recursos utilizados de la FPGA.

Tabla II: Recursos utilizados en la implementación

Componente	Utilizados	Disponibles	Porcentaje
Bloques Lógicos	2,035	28,800	7%
Bloques de RAM	4	60	6%
Bloques DSP	0	48	0%
Pines I/O	262	480	54%
Bloque CMT	0	6	0%

Si se desglosan los bloques lógicos se ve que se han usado 648 bloques como registros lo que supone un 2% de los bloques lógicos disponibles mientras que 1,387 de estos bloques se usan como LUTs lo que supone un 5%. De estos 1,387 bloques 96 se han usado como registros de desplazamiento y de este modo solo se pueden usar 7,680 de los bloques. En lo que respecta a la memoria se han utilizado 4 bloques RAM que suponen un total de 144 Kb y es probable que no se hayan usado bloques DSP debido a la naturaleza de las operaciones.

En el informe temporal se establece que el periodo mínimo admitido es de 1.498 ns lo que supone una frecuencia máxima de operación de 667 MHz. Teniendo en cuenta la velocidad a la que pueden funcionar los elementos que forman el dispositivo se obtiene una frecuencia máxima de operación considerable.

Una vez estudiados los resultados de la síntesis y de la implementación hay que comprobar que el diseño realiza correctamente la tarea para la que se ha diseñado. Por ello se procede a realizar una simulación.

4.3- Simulación

4.3.1- Simulación funcional

A continuación se muestran una serie de figuras donde se realiza una simulación funcional con un ejemplo concreto. En este ejemplo el periodo del reloj es 50 ns y los datos a cifrar son en hexadecimal: '328831e0435a3137f6309807a88da234'.

Y tiene que arrojar como resultado: '3902dc1925dc116a8409850b1dfb9732'

De este modo se comienza con los resultados intermedios correspondiente a la ronda inicial y a la primera ronda se muestran en la figura 29 entre 450 ns y 1,650 ns.

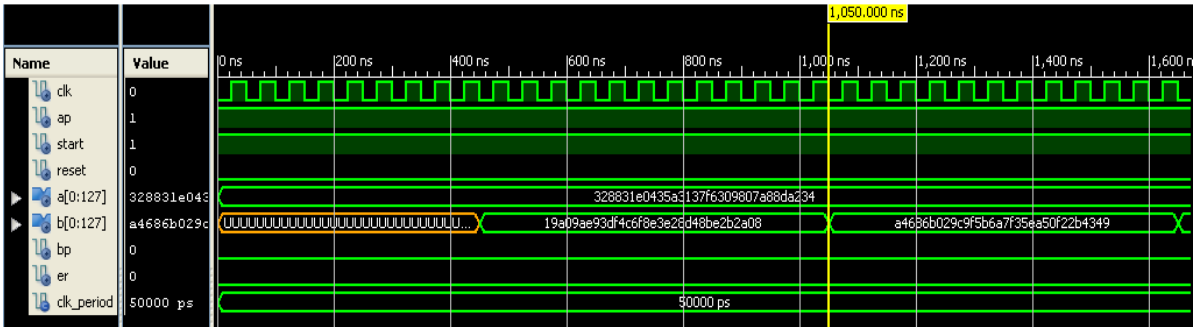


Fig. 29. Resultados intermedios tras la ronda inicial y la primera ronda

En la figura 30 se continúa con la simulación con los resultados intermedios correspondientes a las rondas 2 y 3 comprendidas entre 1,650 ns y 2,850 ns.

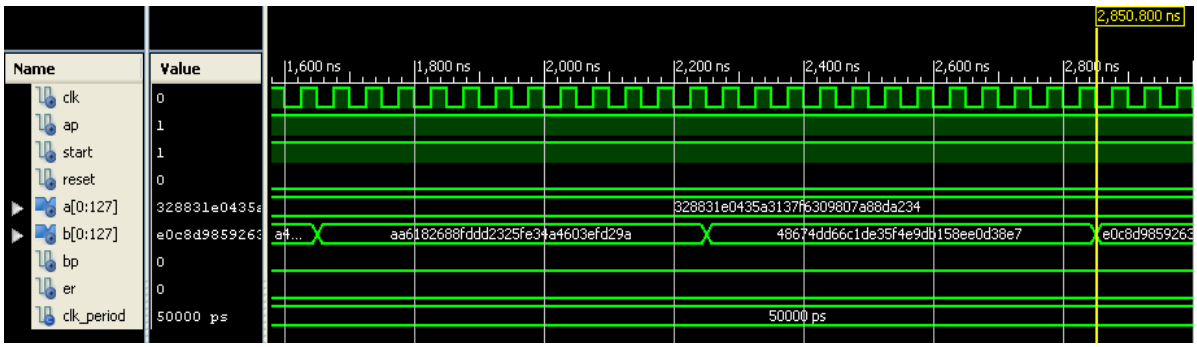


Fig. 30. Resultados intermedios tras las rondas 2 y 3

Se sigue de la misma manera con los resultados intermedios de las rondas 4 y 5 entre 2,850 ns y 4,050 ns en la figura 31.

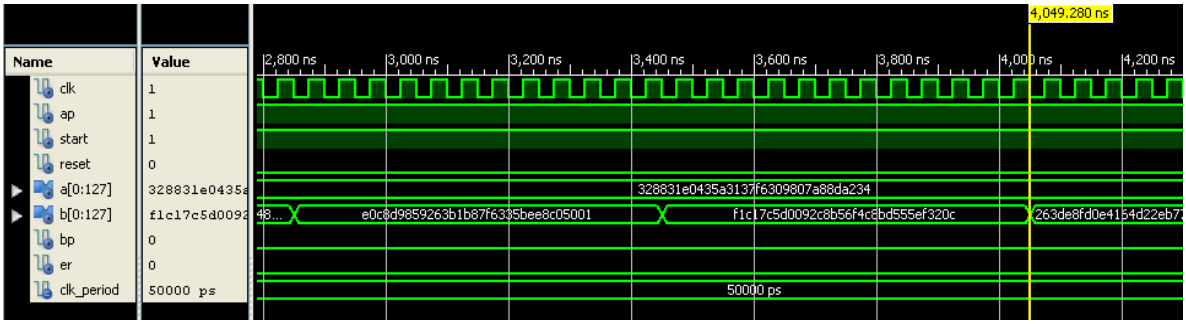


Fig. 31. Resultados intermedios tras las rondas 4 y 5

Continúa con los resultados intermedios de las rondas 6 y 7 entre 4,050 ns y 5,250 ns en la figura 32.

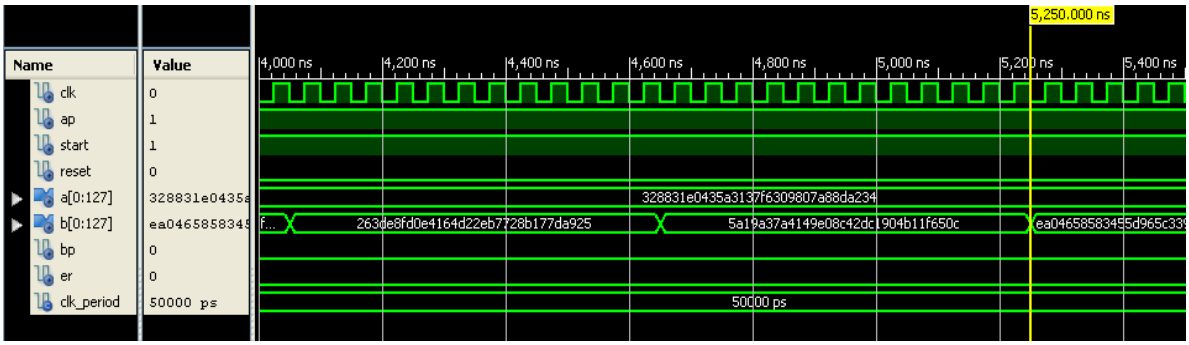


Fig. 32. Resultados intermedios tras las rondas 6 y 7

Finalmente en la figura 33 se enseña el resultado intermedio correspondiente a las rondas 8 y 9 y el resultado final a partir de 5,250 ns.

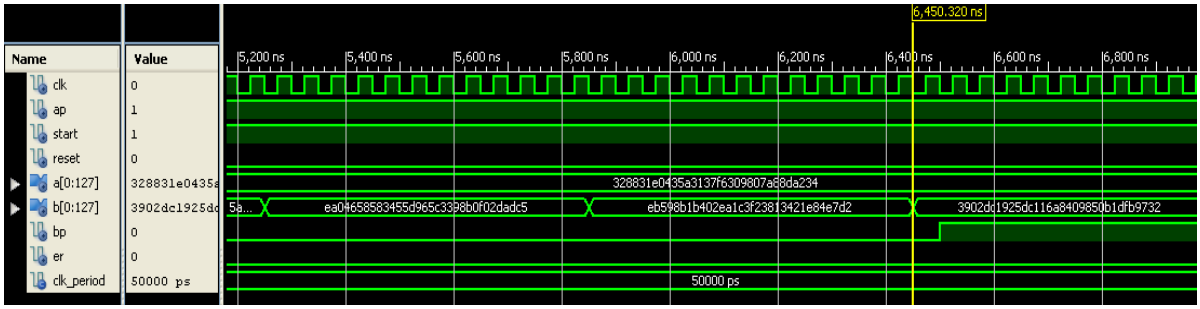


Fig. 33. Resultado intermedio tras las rondas 8 y 9 y el resultado final

Desde el punto de vista funcional trabaja correctamente porque se obtiene el resultado que se había calculado y se necesitan 130 ciclos de reloj para completar el cifrado completamente.

4.3.2- Simulación temporal

Para finalizar con la comprobación del diseño hay que realizar una simulación temporal para poder observar el efecto de los retardos. Una vez realizada la simulación temporal, se muestra el resultado en la figura 34 donde aparecen el resultado intermedio correspondiente a la ronda 9 y el resultado final y se observa que existen retardos pero el algoritmo se ejecuta correctamente con lo que queda completada la fase de comprobación de la implementación.

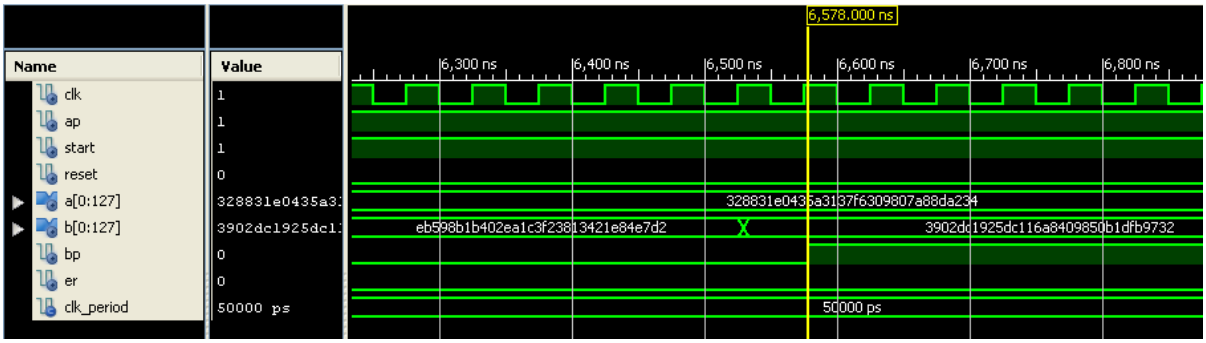


Fig. 34. Resultado final de la simulación temporal

Conclusiones

En primer lugar hay que comentar que se ha conseguido el objetivo principal de este trabajo, implementar el algoritmo de cifrado AES mediante una FPGA de propósito general utilizando. De esta manera también se ha podido comprobar que el algoritmo es implementable en hardware, la eficiencia computacional es elevada ya que los recursos consumidos son reducidos y los recursos de memoria necesarios son escasos (144Kb en este caso) y además con un diseño relativamente simple se consiguen resultados satisfactorios.

Se ha comprobado que el uso de técnicas orientadas a reducir el consumo deben ser tenidas en cuenta desde las primeras fases del diseño y que estas hacen que el mismo sea más complejo. Se han aplicado l as técnicas de reducción de consumo relacionadas con el control del reloj que eran las únicas aplicables debido a la no disponibilidad de biestables de doble flanco y a que la actuación sobre las entradas y la reducción de voltaje no era posible ya que dependen exclusivamente del dispositivo y no del diseño.

La frecuencia máxima de operación del diseño es de 667 MHz, una frecuencia mayor que la frecuencia con la que trabajan los bloques DSP y RAM del dispositivo, lo que vuelve a incidir sobre la eficiencia del diseño. El uso de recursos era uno de los criterios que se usó a la hora de elegir el nuevo algoritmo de cifrado estándar por lo que era de esperar fuera reducido ya que habitualmente en los dispositivos cuando se implementa este algoritmo, suele ir acompañado de un diseño o función principal para el que el desarrollador quiere cifrar la información. De este modo, al algoritmo AES puede ser usado como un coprocesador hardware en un SoPC para acelerar el cálculo de algoritmo.

A la hora de realizar el diseño la clave ha sido proponer una arquitectura en la que se podía trabajar sobre los módulos separadamente, lo que ha provocado que haya una semejanza muy fuerte entre el diagrama de bloques propuesto y la arquitectura resultante de la implementación. El trabajo sobre los módulos también ha permitido que sobre el diagrama RTL de cada uno ellos se pudiera intuir que función del algoritmo realizada dicho módulo.

Por otra parte con el diseño en VHDL ha sido posible crear un código con el que no solo se puede implementar el algoritmo AES si no que, cambiando solamente el módulo correspondiente a las claves, se pueden reproducir el algoritmo de Rijndael para 192 y 256 bits. Esto permite la reutilización del código ya que el diseño realizado es independiente del dispositivo y del fabricante, aunque luego serán distintos los datos relacionados con la síntesis y la implementación, lo que otorga una gran flexibilidad y explica el éxito de los dispositivos reconfigurables.

Finalmente, hay que considerar que debido al aumento de la integración en los circuitos integrados está suponiendo que el consumo estático cobre cada vez más importancia frente al consumo dinámico. Aun así, es importante el uso de este tipo técnicas para reducir el consumo ya que el consumo dinámico también es considerable.

Referencias

[1] Joan Daemen and Vincent Rijmen, "The Design of Rijndael: AES - The Advanced Encryption Standard." Springer-Verlag, 2002.

[2] www.xilinx.com

[3] Peter J. Ashenden, “Digital design: an embedded systems approach using VHDL”, Morgan Kaufmann, 2007.

[4] www.csrc.nist.gov/publications/fips/fips197/fips-197.pdf

[5] Adel S. Sedra, Kenneth C. Smith, “Microelectronics circuits”, Oxford University Press.

[6] www.wikipedia.org/wiki/Circuito_integrado

[7] www.intel.com

[8] V. Nelson, H. Nagle, B. Carrol, J. Irwin, “Análisis y diseño de Circuitos Lógicos Digitales”, Prentice Hall, 1996.

[9] www. wikipedia.org/wiki/FPGA

[10] R. Sass, A. Schmidt, “Embedded Systems Design with Platform FPGAs: Principles and Practices “, Morgan Kaufmann, 2010.

[11] P. Abusaidi, M. Klein, B. Philofsky, “Virtex-5 FPGA System Power Design Considerations”, Xilinx, 2008

[12]Steve Kilts, Advanced FPGA Design: Architecture, Implementation, and Optimization, Wiley-IEEE Press; 1 edition (June 29, 2007), Capítulo 3.

[13] R. Manteena, “A VHDL Implementation of Advanced Encryption Standard-Rijndael Algorithm”, 2004

[14] A. Vera, F.J. Vera, “Introducción al álgebra”, Ellacuria Lab, 1986.

[15] wikipedia.org/wiki/Advanced_Encryption_Standard

[16] S. Brown, Z. Vranesic, “Fundamentals of Digital Logic with VHDL Design”, Tata Mcgraw Hill, 2006.