UNIVERSIDAD CARLOS III DE MADRID
ESCUELA POLITÉCNICA SUPERIOR

DEPARTAMENTO DE TECNOLOGÍA ELECTRÓNICA

INGENIERÍA TÉCNICA INDUSTRIAL
ELECTRÓNICA INDUSTRIAL

PROYECTO FIN DE CARRERA

Implementación del algoritmo de cifrado AES para bajo consumo sobre FPGA

AUTOR: MIGUEL GARCÍA OCÓN
TUTOR: LUIS MENGIBAR POZO

Agradecimientos

En primer lugar, agradecer a mi madre toda la paciencia y el apoyo que me ha ofrecido durante estos años. Sin ella, este proyecto nunca habría llegado a redactarse.

A mi tutor, Luis, por dedicarme tantas horas de densas explicaciones hasta asegurarse de que lo entendía todo y animarme prometiendome que quedaba poco para el final cuando aún quedaban meses de trabajo.

A mis compañeros habituales de prácticas durante la carrera, Ángel y Rafa. Sin duda formábamos un buen equipo y, en el fondo, agradezco todas esas tardes redactando memorias de prácticas en las que nos las ingeniábamos para buscar huecos en los que perder el tiempo de la manera más original posible. Si la carrera ha sido llevadera ha sido gracias a vosotros.

A mi novia, Patricia, por su apoyo y por aguantar múltiples conversaciones y razonamientos acerca de este proyecto, aun sin comprender una palabra de lo que estaba hablando.

Por último, a mis amigos. Todos decidimos en su momento cursar ingenierías o derivados y, poco a poco, vamos finalizando. Gracias por haber estado siempre ahí. Se que siempre estaréis.

Gracias.

Resumen

Este Proyecto de Fin de Carrera consiste en la implementación del algoritmo AES (Advanced Encryption Standard) mediante VHDL en FPGA, haciendo especial hincapié en el consumo de potencia, de manera que sea lo más bajo posible, ya que, hoy en día, es importante debido a la utilización, cada vez mayor, de equipos portátiles.

Para ello, se han utilizado diversas herramientas de diseño digital, como herramientas de síntesis, simuladores y estimadores de consumo.

Se ha escogido el algoritmo AES porque actualmente es el algoritmo estándar utilizado por el gobierno de los Estados Unidos.
Además su nivel de seguridad sigue siendo elevado y goza de una gran popularidad.

Por otra parte, se ha elegido la implementación sobre FPGA debido a su alto rendimiento, fiabilidad, flexibilidad y precio.

Tras los estudios realizados, se ha observado una creciente importancia del consumo estático en tencnologías modernas frente al consumo dinámico. Además, la temperatura se convierte en una variable muy a tener en cuenta, ya que, regulándola, se consigue una reducción importante en el consumo de potencia. En total se ha conseguido una reducción del consumo de potencia de hasta un 23.427 %

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

La finalidad de este proyecto es implementar el algoritmo AES (Advanced Encryption Standard), analizando aspectos del consumo de potencia y estudiando métodos para reducir la misma.

Hoy en día, el tesoro más valioso que podemos poseer es la información, por lo que no es de extrañar que sea de suma importancia protegerla de intrusos que puedan beneficiarse o causar daño con ella.
En un mundo donde existe un uso masivo de comunicaciones digitales, es imprescindible garantizar su seguridad.

Uno de los métodos más antigüos para proteger la información es la criptografía.
Básicamente, la criptografía consiste en disfrazar la información deseada mediante técnicas de cifrado, de manera que quede irreconocible, enviarla a un destinatario que sea capaz de quitarle el disfraz para, finalmente, poder acceder a la información original.

Uno de los motivos por el que se ha elegido el algoritmo AES frente a otros es que es el adoptado como estándar de cifrado por el gobierno de los Estados Unidos, después de un proceso de estandarización que duró 5 años.
Además, aunque hay preocupación por parte de los criptólogos debido a la estructura matemática del AES, los diferentes ataques a este algoritmo son meramente especulativos y no son prácticos en implementaciones del mundo real.
De hecho, no ha habido ningún ataque con éxito hasta el año 2005[1], pero este ataque requiere que el atacante pueda ejecutar programas en el mismo

[1] AES se transformó en un estándar efectivo el 26 de mayo de 2002.

sistema que realiza el cifrado de AES.
A pesar de ello, el AES se ha convertido, desde el 2006, en uno de los algoritmos más populares dentro de la criptografía simétrica, de la cual hablaremos más adelante.

En cuanto al consumo de potencia, en la actualidad es un aspecto esencial en el diseño de sistemas digitales, debido principalmente a dos razones: la alta densidad de integración alcanzada con las tecnologías actuales y la necesidad de reducir el consumo en los equipos portátiles alimentados por baterías.

Además, el motivo de implementar el algoritmo en una FPGA es que éstas poseen un gran rendimiento, son baratas y fiables y, debido a su flexibilidad y capacidad de rápido desarrollo, su tiempo en llegar al mercado es mucho menor que el de otras tecnologías.

1.1. Objetivos

Al comenzar este proyecto, se propuso una serie de pasos a seguir para conseguir implementar el AES y hacer estudios sobre su consumo de potencia para así poder trabajar sobre ella.

1.1.1. Estudio del algoritmo AES

Antes de nada, es imprescindible comprender como funciona el AES, estudiando cuales son las diversas transformaciones que se aplican a los datos originales.

1.1.2. Descripción, simulación funcional y síntesis

Después de haber comprendido el funcionamiento del AES, el siguiente paso es la descripción en VHDL del algoritmo sin tener en cuenta restricciones en el consumo, ya que trabajaremos sobre él más adelante.
Para ello, se ha optado por descomponer en bloques individuales o componentes la estructura global, ya que facilita el trabajo.

Una vez descrito el algoritmo, se comprueba su funcionalidad mediante

simulaciones[2].

Cuando se comprueba que el algoritmo descrito es funcional, se realiza la síntesis, que consiste en adaptar el diseño a una FPGA en concreto. El sintetizador optimiza las expresiones lógicas con objeto de que ocupen menor área, es decir, que requieran menos recursos de la FPGA.

1.1.3. Emplazamiento, rutado y simulación postlayout

Cuando el algoritmo descrito está sintetizado, se realiza el emplazamiento, proceso en el que se sitúan los bloques digitales obtenidos en la síntesis de una manera óptima, y el rutado, que consiste en interconectar adecuadamente dichos bloques entre sí.

La simulación postlayout consiste en una simulación no ideal en la que se tienen en cuenta los retardos de los bloques y sus interconexiones, estos retardos se extraen de un archivo generado por los procesos de emplazamiento y rutado.
Esta simulación es muy precisa y se acerca mucho a la realidad.

Aunque la simulación funcional sea correcta, es posible que esta simulación no de los resultados esperados. Una de las causas puede ser los retardos internos del chip.

En cualquier caso, si la simulación postlayout es errónea, hay que volver a realizar pasos anteriores.

1.1.4. Estimación y reducción del consumo

Una vez realizada una simulación postlayout correcta, se utilizan herramientas y archivos generados por dicha simulación para realizar una estimación de la potencia consumida.

Una vez estimada, la idea es centrarse en las partes que más consumen y aplicar técnicas de reducción de consumo tales como modificaciones de la arquitectura, de manera que el algoritmo descrito no pierda su funcionalidad.

[2]En las simulaciones realizadas se han utilizado vectores del Instituto Nacional de Estándares y Tecnología (NIST) de los Estados Unidos [Vec].

Capítulo 2

Criptología

Según [Lop02], la criptología (del griego *krypto*: "oculto" y *logos*: "discurso") es la disciplina que se dedica al estudio de los mensajes que, procesados de cierta manera, se convierten en difíciles o imposibles de leer para entidades no autorizadas.
Esta ciencia está dividida en cuatro ramas:

- **Criptografía:** Se ocupa del estudio de los algoritmos, protocolos y sistemas que se utilizan para proteger la información.
- **Criptoanálisis:** Se ocupa de descifrar sin autorización la información contenida en criptogramas.
- **Esteganografía:** Se ocupa de ocultar información por un canal inseguro, de manera que no sea siquiera percibida.
- **Estegoanálisis:** Se ocupa de detectar información oculta mediante la esteganografía.

Por lo tanto, podemos apreciar que la criptografía y la esteganografía son las ramas que aplican la idea en la que se basa la criptología, mientras que el criptoanálisis y el estegoanálisis persiguen precisamente lo contrario, es decir, burlar la seguridad de la información.

A continuación estudiaremos cada una de estas ramas, centrándonos sobre todo en la criptografía, ya que es la rama que realmente nos interesa para este proyecto.

2.1. Criptografía

El término "criptografía" viene del griego *krypto*: "oculto" y *graphos*: "escribir". Por lo tanto, por criptografía entendemos "escritura oculta".
Como ya se ha dicho, la finalidad principal de la criptografía es la de proteger la privacidad y confidencialidad de la información, es decir, que la información sea accesible únicamente para el personal debidamente autorizado.

Sin embargo, hoy en día, también persigue otros fines no menos importantes. Dichos fines son la integridad (la información que leemos es idéntica a la que se envió), la autenticación (el emisor de la información es quien dice ser) y el no repudio (ninguna de las entidades implicadas en la comunicación puede negar su participacíon total o parcial).

A continuación se muestra una ilustración (figura 2.1) que explica, de manera sencilla, lo que es, en esencia, la criptografía.

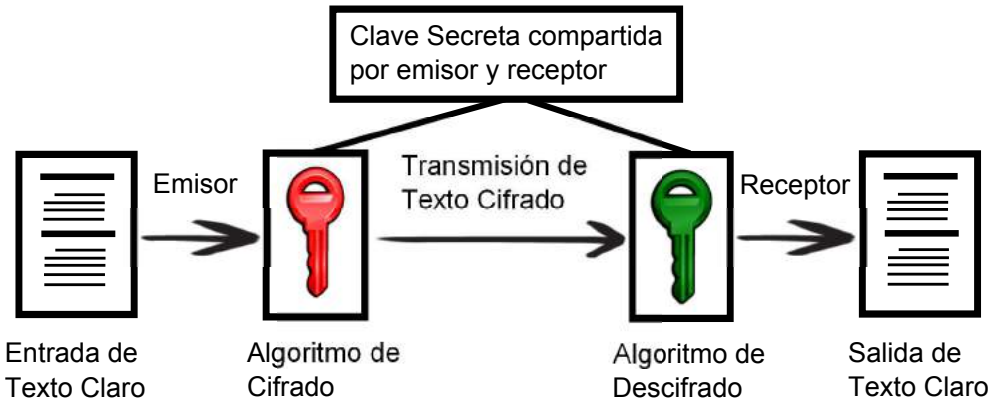


Figura 2.1: Esencia de la criptografía

Como vemos, se trata de cifrar un texto plano mediante ténicas de cifrado (en nuestro caso, el AES). Con ello, obtenemos un texto cifrado, ininteligible, que el emisor envía al receptor. Una vez recibido, el receptor, conocedor de la clave, aplica un algoritmo de descifrado (inversa del algoritmo de cifrado), de manera que es capaz de leer el texto plano.
A esto se le llama criptosistema.

2.1.1. Historia de la criptografía

La necesidad de proteger la información se remonta a hace miles de años. Ya las primeras civilizaciones desarrollaron técnicas criptográficas para enviar mensajes durante las campañas militares, de forma que si el mensajero era interceptado, la información que portaba no caía en manos enemigas.

El primer método criptográfico del que se tiene constancia era conocido como "Escítala"[1]. La escítala consistía en dos varas del mismo grosor que se entregaban a los participantes de la comunicación. El emisor enrollaba una cinta en su vara en forma de espiral y escribía el mensaje longitudinalmente, de forma que en cada vuelta de cinta aparecía una letra.
Luego se desenrollaba y se enviaba al receptor, que únicamente tenía que volver a enrollar la cinta en su vara para leer el mensaje.

Figura 2.2: Escítala espartana [Esc]

El segundo criptosistema del que se tiene constancia fue documentado por Polibilio, un historiador griego, y consistía en un sistema de sustitución basado en la posición de las letras en una tabla.

También los romanos usaban métodos de sustitución, siendo uno de los más conocidos el método César. Recibe ese nombre porque, supuestamente, fue utilizado por Julio César en sus campañas.

Durante los siglos XVII, XVIII y XIX, el interés de los monarcas por la criptografía fue notable.
Las tropas de Felipe II utilizaron una cifra con un alfabeto de más de 500 símbolos que los mátematicos consideraban inexpugnable. Cuando un matemático francés consiguió criptoanalizar aquel sistema para Enrique IV, rey de Francia, el conocimiento mostrado por el monarca francés impulsó una

[1]La escítala data del siglo V a.C.

queja por parte de la corte española ante el papa Pío V, acusando a Enrique IV de utilizar magia negra para vencer a su ejército.

En el siglo XX, la criptografía experimenta grandes avances, debido especialmente a las dos guerras mundiales.
El principal avance es el uso de máquinas de cálculo, siendo la más conocida la máquina alemana Enigma, una máquina de rotores que automatizaba considerablemente el cálculo necesario para cifrar y descifrar mensajes.

Después de la Segunda Guerra Mundial, la criptografía tiene un desarrollo teórico importante, destancando Claude Shannon[2], ingeniero electrónico y matemático estadounidense.

A mediados de los años 70, el Departamento de Normas y Estándares norteamericano publica el primer diseño lógico de un cifrador que estaría llamado a ser el principal sistema criptográfico de finales de siglo: el Estándar de Cifrado de Datos o DES.

El algoritmo que nos ocupa en este proyecto, el AES, es técnicamente el sustituto del DES, ya que se han descrito multitud de ataques que permiten criptoanalizarlo más rápidamente que con un ataque por fuerza bruta (recuperar una clave probando todas las combinaciones posibles).

2.1.2. Criptografía clásica

Cuestiones militares, religiosas y comerciales impulsaron desde tiempos remotos el uso de escrituras secretas con el fin de ocultar información.

Ya los antiguos egipcios usaron métodos criptográficos. Por ejemplo, los sacerdotes egipcios utilizaron la escritura hierática (jeroglífica), que era claramente incomprensible para el resto de la población. Los antiguos babilonios también utilizaron métodos criptográficos en su escritura cuneiforme.

Una de las primeras formas de enmascarar la información fue con la rama de la criptología conocida como esteganografía, de la cual hablaremos en los siguientes apartados.

En la criptografía clásica, los mensajes se transmitían cifrados con clave

[2]Claude Shannon es considerado el padre de la teoría de la información

secreta, de manera que tanto el emisor como el receptor eran conocedores de dicha clave. A ésto se le conoce como ***criptografía simétrica***.
Esto planteaba dos problemas:

- El transporte de las claves debía realizarse a través de correos de confianza.
- Si el correo no llegaba al receptor, éste quedaba incomunicado.

2.1.2.1. Tipos de cifrados clásicos

Los cifrados clásicos suelen dividirse en dos tipos:

- **Cifrado por sustitución:** Letras o grupos de letras son sistemáticamente reemplazadas por otras letras o grupos de letras.
- **Cifrado por transposición:** Se cambia el orden de las letras de acuerdo con un esquema bien definido.

Cabe destacar que la dificultad en el cifrado y descifrado de los cifrados clásicos no es muy compleja, pero sientan las bases de la criptografía moderna.

Tipos de cifrado por sustitución

El cifrado por sustitución es un método por el que unidades de texto son sustituidas por nuevos caracteres de cualquier tipo, es decir, letras, números, símbolos, etc.
Los caracteres permanecen en el mismo orden y el receptor descifra el texto realizando la sustitución inversa.

Hay dos tipos de sustitución:

- **Sustitución monoalfabética:** Cada carácter se sustituye siempre por un determinado carácter del alfabeto del texto cifrado. Ésto significa que si la letra "A" se sustituye por la letra "Z", ésto va a ser así a lo largo de todo el texto. Dentro de este tipo se localiza el ya mencionado cifrado César.
- **Sustitución polialfabética:** Se dice que un cifrado de sustitución es polialfabético cuando un determinado carácter no se sustituye siempre por el mismo carácter. Es decir, hay implicados varios alfabetos de texto cifrado y, dependiendo de las circunstancias, se aplicará uno u otro.

Tipos de cifrado por transposición

A diferencia del cifrado por sustitución, el cifrado de transposición no disfraza los caracteres, simplemente los reordena.
Este tipo de algoritmos son de clave simétrica, por lo que tanto el emisor como el receptor tienen que ser conocedores de la clave.

Por mencionar un ejemplo del que ya se ha hablado, la escítala espartana es de este tipo. En este caso, la clave simétrica es la analogía en el grosor de las varas que se utilizaban para enrollar las cintas que contenían el texto cifrado.

2.1.3. Criptografía moderna

En la criptografía existe una relación entre la complejidad o longitud de la clave y el tiempo necesario para cifrar y descifrar la información.
Debido principalmente a esto, la criptografía clásica carece de una complejidad excesiva.

Sin embargo, en la era moderna esta relación se rompe, debido fundamentalmente a tres factores:

- **Velocidad de cálculo:** La aparición de los ordenadores permitió aumentar drásticamente la potencia de cálculo.
- **Avance de las matemáticas:** Se pudieron encontrar y definir con claridad sistemas criptográficos estables y seguros.
- **Necesidades de seguridad:** Debido también a la aparición de las computadoras, surgieron muchas actividades que precisaban de protección de la información.

En la criptografía moderna, existen dos tipos de cifrado según el tratamiento que se le de al mensaje:

- **Cifrado en bloque:** El cifrado se realiza en grupos de bits de longitud fija, llamados bloques. El algoritmo que nos interesa, el AES, es de este tipo, trabajando con bloques de 128 bits (aunque puede trabajar con otros tamaños). Otros algoritmos de este tipo son el DES y el IDEA.
- **Cifrado en flujo:** Consiste en convertir el texto claro en texto cifrado bit a bit mediante un generador de flujo de clave. Encontramos un ejemplo de cifrado en flujo en el algoritmo RC4.

Además, según el tipo de clave utilizada para el cifrado, encontramos:

- Criptografía simétrica.
- Criptografía asimétrica.

2.1.3.1. Criptografía simétrica

La criptografía simétrica o de clave privada, también conocida como criptografía de una clave, es un método criptografíco en el cual se usa una sola clave, la misma para cifrar y descifrar la información.

Las dos partes que se comunican han de ponerse de acuerdo de antemano acerca de la clave a utilizar. Después, el remitente cifra el mensaje con la clave pactada y le envía al destinatario el mensaje cifrado. Posteriormente, el destinatario utiliza la misma clave para descifrar los datos para finalmente acceder a la información original.
El algoritmo con el que vamos a trabajar, AES, es de clave privada.

Las desventajas asociadas a este sistema son, principalmente:

- Problemas en la seguridad de la comunicación de las claves.
- Necesidad de un gran número de claves cuando se incrementa el número de personas que necesitan comunicarse entre sí.

2.1.3.2. Criptografía asimétrica

La criptografía asimétrica o de clave pública, también conocida como criptografía de dos claves, es un método criptográfico en el cual se usan dos claves para el envío de mensajes.

Ambas claves pertenecen a la misma persona. Una es pública y se puede entregar a cualquier persona, la otra es privada y el propietario debe guardarla a toda costa.

Consiste en que el remitente utiliza la clave pública del destinatario para cifrar el mensaje antes de enviarlo. Después, el destinatario utiliza su clave privada para descifrar el mensaje. Se soluciona, por tanto, el problema de la criptografía simétrica referido a los problemas en la seguridad de la comunicación de las claves. Se asegura también la confidencialidad, ya que únicamente el destinatario tiene acceso a la información.

En el caso de que el propietario del par de claves sea el remitente y utilice su clave privada para cifrar el mensaje, cualquiera podrá descifrarlo con su clave pública. Se asegura, por tanto, la autenticación, es decir, que el emisor es quien dice ser.

La principal desventaja de este sistema es que las claves son vulnerables a ataques por fuerza bruta. Y, mientras que en la clave del sistema simétrico es suficiente una longitud de clave de 128 bits, se recomienda que hoy en día se utilicen, para claves públicas, longitudes de 1024 bits.

La ilustración 2.3 pretende explicar de manera sencilla las diferencias entre criptografía simétrica y criptografía asimétrica.

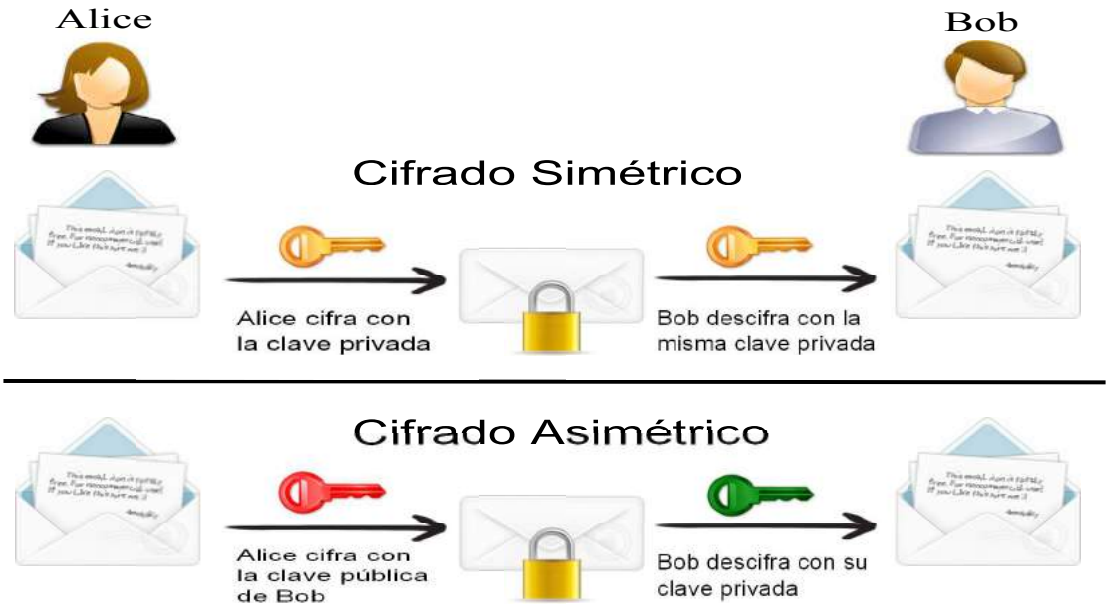


Figura 2.3: Diferencias entre criptografía simétrica y asimétrica

2.2. Criptoanálisis

El criptoanálisis es la rama de la criptología que se dedica al estudio de los sistemas criptográficos con el fin de romper su seguridad y acceder a la información privada.

Estos ataques se pueden clasificar en función de sus características.

- **Actitud del atacante**

  1. **Ataques pasivos:** El atacante no altera la comunicación, sólo la escucha o monitoriza para obtener información.
  2. **Ataques activos:** El atacante modifica el flujo de datos o crea flujos falsos.

- **Conocimiento previo**

  1. **Ataque con texto cifrado conocido:** El atacante sólo tiene acceso a los textos cifrados.
  2. **Ataque con texto plano conocido:** El atacante tiene una serie de textos cifrados de los que conoce sus correspondientes textos planos.
  3. **Ataque con texto plano escogido:** El atacante puede obtener los textos cifrados correspondientes a un conjunto de textos planos de su propia elección.
  4. **Ataque con texto cifrado escogido:** El atacante puede obtener los textos planos correspondientes a un conjunto de textos cifrados de su propia elección.
  5. **Ataque adaptativo con texto plano escogido:** Similar al ataque con texto plano escogido, pero el atacante puede elegir los subsiguientes textos planos basándose en la información de los descifrados con anterioridad. De la misma manera, existe un **ataque adaptativo con texto cifrado escogido**.
  6. **Ataque de clave relacionada:** Similar a un ataque con texto plano escogido, con la particularidad de que el atacante puede obtener el texto cifrado con dos claves diferentes, las cuales son desconocidas, pero se conoce la relación entre ambas.

- **Objetivo**

  1. **Ruptura total:** El atacante averigüa la clave secreta
  2. **Deducción global:** El atacante descubre un algoritmo funcionalmente equivalente para el cifrado y descifrado, pero no obtiene la clave.
  3. **Deducción local:** El atacante descubre textos planos o cifrados, adicionales a los conocidos previamente.
  4. **Deducción de información:** El atacante descubre información que no era conocida previamente.

5. **Distinción del algoritmo:** El atacante puede distinguir la información cifrada de una permutación al azar.

- **Coste**

1. **Tiempo:** Número de operaciones primitivas que deben ser realizadas.
2. **Memoria:** Cantidad de almacenamiento necesario para realizar el ataque.
3. **Datos:** Cantidad de textos planos y cifrados necesaria.

Existen multitud de métodos de ataque criptoanalíticos y todos han ido surgiendo o evolucionando de la mano de los avances criptográficos.
Entre los más conocidos se encuentran el *análisis de frecuencias*, especializado en cifrado clásico, el *criptoanálisis diferencial* y *criptoanálisis lineal*, especializados en criptografía simétrica, y, hablando en ámbitos generales, encontramos otros como el *ataque de fuerza bruta*.

2.2.1. Análisis de frecuencias

Se trata de uno de los métodos más usados en el criptoanálisis a la hora de romper cifrados clásicos.

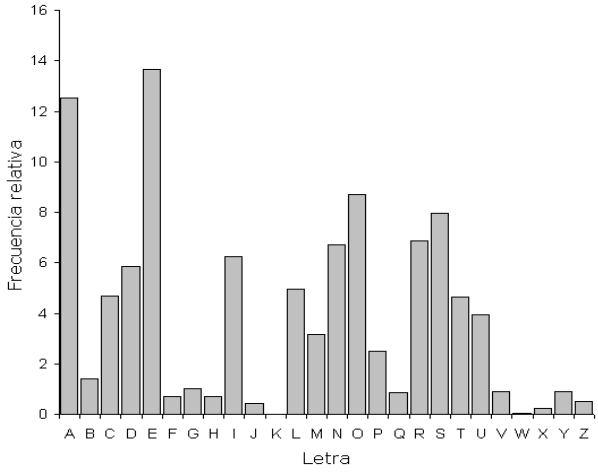


Figura 2.4: Frecuencia de las letras en castellano [Ana]

Si observamos la ilustración 2.4, descubriremos que ciertas letras o grupos de letras se repiten con más frecuencia que otras. Por ejemplo, la letra E en castellano es la más usada (13.68 %), mientras que la letra W apenas tiene presencia (0.02 %), por lo tanto, si nos encontramos ante un texto cifrado y se observa una letra repetida varia veces, hay más probabilidades de que esa letra descifrada sea una E que de que sea una W.

No siempre ocurre así, ya que, por ejemplo, la letra A también tiene mucha presencia en nuestro idioma. Por ello también se trabaja con frecuencias de grupos de letras.

2.2.2. Criptoanálisis diferencial

El criptoanálisis diferencial es un tipo de ataque que puede aplicarse a cifradores de bloque iterativos, como lo son el DES, el IDEA y el AES.

Hablando en términos generales, podemos decir que el critoanálisis diferencial consiste en estudiar varios textos claros escogidos, de manera que se pueda apreciar como las diferencias entre ellos afectan a sus correspondientes textos cifrados bajo la misma clave.
De este modo, se puede reconocer los patrones de comportamiento no aleatorios del cifrador y, eventualmente, descubrir la clave secreta.

Este tipo de ataque es atribuido a Eli Biham y Adi Shamir, a finales de los 80.

2.2.3. Criptoanálisis lineal

Al igual que el critoanálisis diferencial, el criptoanálisis lineal es un tipo de ataque concebido para romper cifradores de bloque.

Su descubrimiento es atribuido a Mitsuru Matsui, quien aplicó el ataque al cifrador FEAL y, posteriormente, al DES.

Consiste en la búsqueda de aproximaciones lineales al comportamiento del cifrador para así poder describirlo.
Dados suficientes textos sin cifrar y sus correspondientes textos cifrados, se puede obtener la clave secreta.

2.2.4. Ataque de fuerza bruta

Se denomina ataque de fuerza bruta a la forma de recuperar una clave probando todas las combinaciones posibles hasta encontrarla.

La cantidad posible de claves es denominada "Espacio de Claves", y viene definida por:

$$\text{Espacio de claves} = 2^n$$

Donde n es la longitud de nuestra clave.

A modo de ejemplo, nuestro algoritmo AES utiliza claves de 128, 192 y 256 bits.
Por lo tanto, el caso más "vulnerable" tendría 2^{128} claves de 128 bits. Sobra decir que la cantidad es tan abrumadora que, con los procesadores que existen hoy día, sería una locura intentar un ataque de fuerza bruta contra este cifrado.

2.3. Esteganografía

La esteganografía es la rama de la criptología que se encarga de estudiar las técnicas que permiten ocultar un mensaje dentro de otro, al que se le conoce como portador, de modo que se establece un canal encubierto de comunicación y el mensaje pasa inadvertido a ojos de los observadores.
En la esteganografía debe haber voluntad de comunicacion tanto por parte del emisor como del receptor.

Su uso está presente desde tiempos antigüos y, hoy en día, existen numerosos métodos esteganográficos para ocultar información, principalmente dentro de archivos multimedia.

Por ejemplo, es muy común esconder información dentro de imágenes a través del método conocido como LSB (*Least Significant Bit* o *Bit Menos Significativo*).
Cambiar el bit menos significativo para ocultar un mensaje altera la imagen de una manera imperceptible para el ojo humano.

Es muy habitual recurrir a este método para insertar marcas de agua con información acerca de derechos de autor, propiedad o licencias, pero también es posible esconder imágenes con este método, como se muestra en la figura 2.5.

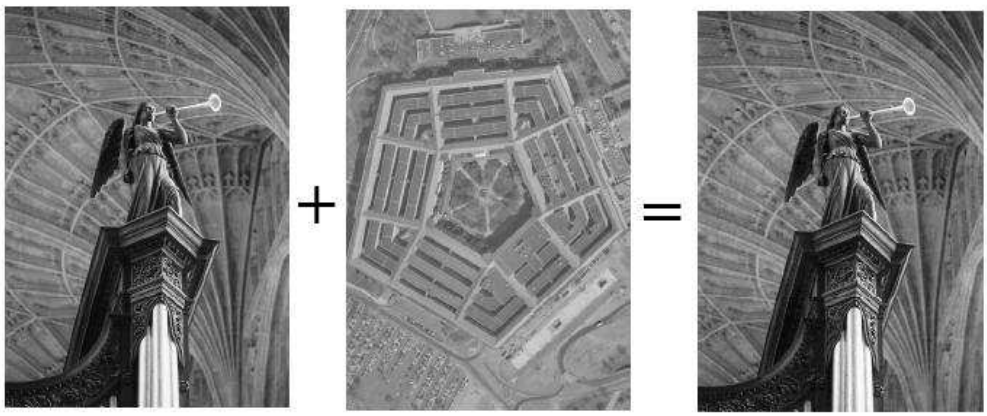


Figura 2.5: Ejemplo del método esteganográfico LSB [Est]

El cambio entre la imagen portadora y la imagen final, que porta el mensaje esteganográfico, o estego-mensaje, en sus bits menos significativos, es inapreciable.

2.4. Estegoanálisis

El estegoanálisis es la disciplina dedicada al estudio de la detección de mensajes ocultos mediante técnicas esteganográficas.

Para considerar un sistema roto, basta con ser capaz de detectar la existencia de un mensaje oculto, a diferencia del criptoanálisis, en el que es necesario descifrar el mensaje.

Generalmente, el estegoanálisis es una tarea muy compleja, debido principalmente al gran número de medios por los que se puede transmitir la información oculta y a la existencia de diversas técnicas esteganográficas.

Capítulo 3

Advanced Encryption Standard

3.1. Introducción

El AES (Advanced Encryption Standard) fue desarrollado por dos criptólogos belgas, Joan Daemen y Vincent Rijmen.

En realidad, el nombre del cifrado es **Rijndael**, el cual fue ganador del concurso público de candidato a AES.

Debido a que el anterior algoritmo estándar, el DES, había consumido su tiempo de vida, en el año 1997, el NIST (Instituto Nacional de Estándares y Tecnología) de Estados Unidos emprendió un proceso abierto para la selección de un nuevo algoritmo de cifrado, el cual sería sometido a la crítica de especialistas e instituciones de seguridad.

Se decidió hacer un concurso público porque ya había habido quejas debido a que partes del anterior estándar, el DES, no habían sido documentadas, lo que daba la impresión de que el gobierno de los Estados Unidos mantenía puertas traseras.

Según [Mun04], los candidatos a AES debían reunir unos requisitos mínimos:

- El algoritmo debía ser público.
- Debía ser un algoritmo en bloque simétrico.
- La longitud de clave debía ser, al menos, de 128 bits.

- Su diseño debía permitir aumentar la longitud de clave según necesidades.
- Debía poder implementarse tanto en hardware como en software.

Si los algoritmos cumplían con esos requisitos mínimos, serían juzgados por diversos factores:

- Seguridad.
- Eficiencia computacional.
- Requisitos de memoria.
- Simplicidad del diseño.
- Flexibilidad.

Además, los algoritmos presentados a concurso debían soportar obligatoriamente una longitud de bloque de al menos 128 bits y una longitud de clave de 128, 192 y 256 bits, al margen de otras longitudes posibles.

El NIST propuso que cualquier organización, institución o persona pudiera participar en el concurso de forma activa, ya fuera presentando algoritmos o enviando informes o pruebas para poner en evidencia a los algoritmos candidatos.

En Octubre del año 2000, tras dos años de concurso y tres rondas de votaciones, el algoritmo de cifrado Rijndael se proclama vencedor, por permitir la mejor combinación de seguridad/velocidad/eficiencia, por su sencillez y por su gran flexibilidad.

3.2. Definición del algoritmo

Como ya se ha dicho, el AES es un cifrador en bloque de criptografía simétrica, es decir, trabaja cifrando y descifrando bloque a bloque, utilizando la misma clave privada para ambos procesos.

Según [NIS01], en el estándar, el algoritmo Rijndael divide los datos de entrada en bloques de 4 palabras de 32 bits, es decir, $4 \times 32 = 128$ bits.
Es necesario decir que el algoritmo Rijndael puede trabajar también con bloques mayores de 192 y 256 bits, pero no vienen contemplados en el estándar.

Al bloque de datos se le conoce como **Estado**.

En cuanto a la longitud de clave, el estándar trabaja con longitudes de Nk palabras de 32 bits, donde $Nk = 4$, 6 ó 8.
Es decir, que el algoritmo trabaja con longitudes de clave de 128 (4×32), 192 (6×32) ó 256 (8×32) bits.

Si observamos la figura 3.1, veremos que, tanto el texto claro como el texto cifrado se dividen en bloques de 128 bits, mientras que la longitud de clave puede variar entre 128, 192 y 256 bits. Más adelante estudiaremos qué implican en el cifrado (o descifrado) las distintas longitudes de clave.

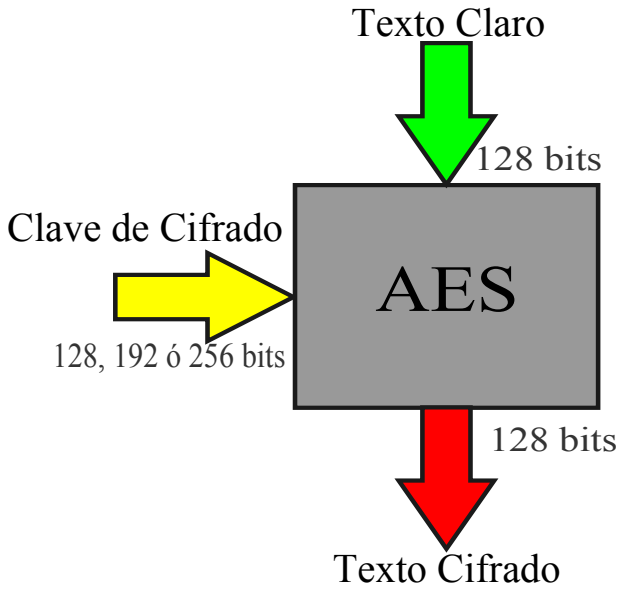


Figura 3.1: Cifrador AES

3.2.1. Conocimientos previos

Antes de entrar en la descripción del funcionamiento del cifrado y descifrado, es necesario tener en cuenta cierta información.

3.2.1.1. Unidad básica

Según [NIS01], la unidad básica de procesamiento del AES es el byte. Un byte es una cadena de 8 bits, representados como $\{b_7, b_6, b_5, b_4, b_3, b_2, b_1, b_0\}$. Además, un byte se puede representar en un campo finito mediante un

polinomio a través de la siguiente fórmula:

$$\sum_{i=0}^{7} b_i x^i$$

Por ejemplo, si partimos del siguiente byte: {0010 1101}, su polinomio será: $x^5 + x^3 + x^2 + 1$

También es útil usar la notación hexadecimal para referirse a los bytes, dividiéndolos en dos grupos de cuatro bits.

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

Tabla 3.1: Notación hexadecimal

De esta manera, el byte {0010 1101} puede ser representado por el número hexadecimal {2D}.

3.2.1.2. El Estado

Como ya se ha dicho, el Estado es un bloque de 128 bits en el estándar. Se trata, por tanto, de una sucesión de 16 bytes.
Para facilitar visualmente la comprensión de las operaciones realizadas en el AES, estos 16 bytes se suelen representar como una matriz de cuatro filas por cuatro columnas de la siguiente manera:

Si a un byte lo denominamos con la letra B, la sucesión de bytes será $\{B_{15}, B_{14}, B_{13}, B_{12}, B_{11}, B_{10}, B_9, B_8, B_7, B_6, B_5, B_4, B_3, B_2, B_1, B_0\}$.

En este proyecto se ha elegido que su representación matricial sea:

$$\begin{pmatrix} B_{15} & B_{11} & B_7 & B_3 \\ B_{14} & B_{10} & B_6 & B_2 \\ B_{13} & B_9 & B_5 & B_1 \\ B_{12} & B_8 & B_4 & B_0 \end{pmatrix}$$

Además, en estas matrices se utilizará, por comodidad, la notación hexadecimal.

3.2.1.3. Operaciones matemáticas

Básicamente hay dos operaciones mátematicas que se realizan entre unidades básicas:

- Suma.
- Multiplicación.

Según [NIS01], la suma (o resta) de dos bytes se realiza a través de la operación OR Exclusiva (XOR) (tabla 3.2).

A	B	$A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

Tabla 3.2: Operación XOR

Vamos a realizar un ejemplo de la suma de {0100 1001} y {1101 0011}

$01001001 + 11010011 = 10011010$ (Notación binaria)

$49 + D3 = 9A$ (Notación hexadecimal)

$(x^6+x^3+1)+(x^7+x^6+x^4+x+1) = x^7+x^4+x^3+x$ (Notación polinómica)

Para explicar la multiplicación es necesario conocer el **polinomio irreductible**[1] **de AES**. Este polinomio se define como:

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

[1]Un polinomio es irreductible si sus únicos divisores son él mismo y la unidad

Para multiplicar dos bytes se utiliza su forma polinómica, recordando siempre que la suma es equivalente a una operación XOR. El polinomio de AES es necesario porque es muy probable que la multiplicación de dos bytes dé un resultado de más de 8 bits. De modo que necesitamos hacer el módulo de ese polinomio con el polinomio de AES, para así volver a tener 1 byte, que es la unidad con la que trabaja el algoritmo.

Por ejemplo, si queremos multiplicar $\{01010111\} \bullet \{10000011\}$

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

y

$$x^{13}+x^{11}+x^9+x^8+x^6+x^5+x^4+x^3+1 \; modulo \; x^8+x^4+x^3+x+1 = x^7+x^6+1$$

3.2.2. Encriptador AES

Para definir el proceso de cifrado del AES, vamos a asumir que la longitud de clave escogida es de 128 bits, ya que la longitud de clave no afecta a las diversas operaciones que realiza el cifrado.

Con el objetivo de facilitar el entendimiento de las operaciones realizadas por el cifrador, se realizarán pequeños ejemplos.

Básicamente, el cifrador aplica al Estado cuatro operaciones durante un número determinado de rondas.
Dicho número de rondas (Nr) viene definido por la longitud de clave utilizada, siendo $Nr = 10$ para una longitud de clave de 128 bits, $Nr = 12$ para 192 bits y $Nr = 14$ para 256 bits.

Las cuatro operaciones realizadas en el cifrado son denominadas:

- SubBytes.
- ShiftRows.
- MixColumns.
- AddRoundKey.

En la figura 3.2 se explica como se distribuyen las operaciones realizadas en el cifrado a lo largo de las 10 rondas necesarias para una clave de 128 bits.

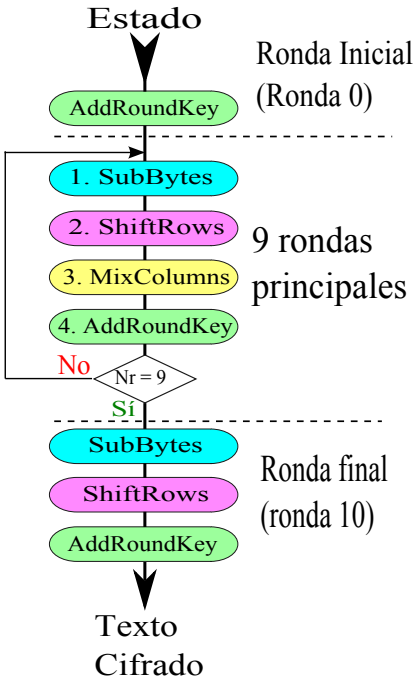


Figura 3.2: Proceso de cifrado

Como vemos, en primer lugar se realiza una ronda inicial en la que únicamente se aplica una operación *AddRoundKey*.
Posteriormente, se realizan las nueve rondas principales, en las que se aplican las cuatro operaciones del cifrado en este orden: *SubBytes*, *ShitfRows*, *MixColumns* y *AddRoundKey*.
Por último, se realiza la ronda final, en la que se aplican las operaciones *SubBytes*, *ShiftRows* y *AddRoundKey*, obteniendo así nuestro texto cifrado.

Cabe destacar que en cada ronda se utilizan diferentes subclaves, derivadas de la clave original. De modo que en la ronda inicial se utiliza la clave original (si la clave es de 128 bits) y en la ronda final se utiliza la subclave número 10.

A continuación se explicarán, una a una, las diversas operaciones que realiza el cifrado.

3.2.2.1. SubBytes

La operación *SubBytes* consiste, según [NIS01], en una sutitución no lineal de bytes. Dicha sustitución se realiza aplicando la fórmula:

$$S'_{i,j} = M \cdot S^{-1}_{i,j} + C$$

Donde $M = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}, C = \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}$

$S^{-1}_{i,j}$ = Multiplicativo inverso del byte a transformar, con el bit menos significativo arriba.

$S'_{i,j}$ = Transformación *SubBytes*, con el bit menos significativo arriba.

Existe una tabla de sustitución fija, llamada **S-box** (tabla 3.3), que aplica estas operaciones y permite realizar la operación *SubBytes* mediante un simple vistazo.

	x0	**x1**	**x2**	**x3**	**x4**	**x5**	**x6**	**x7**	**x8**	**x9**	**xA**	**xB**	**xC**	**xD**	**xE**	**xF**
0x	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
1x	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
2x	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
3x	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
4x	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
5x	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
6x	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
7x	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
8x	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
9x	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
Ax	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
Bx	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
Cx	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
Dx	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
Ex	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
Fx	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

Tabla 3.3: S-Box

Entonces, si necesitamos realizar la operación *SubBytes* al número {19}, no tenemos más que mirar la tabla, fijándonos en la fila **1x** y la columna **x9**, obteniendo así que la transformación de {19} es {D4}.

3.2.2.2. ShiftRows

La operación *ShiftRows* consiste, según [NIS01], en una rotación cíclica hacia la izquierda de las filas de la notación matricial del Estado, de manera que la primera fila permanece igual, la segunda fila se rota hacia la izquierda una posición, la tercera fila se rota hacia la izquierda dos posiciones y, por último, la cuarta fila se rota hacia la izquierda tres posiciones.

En la ilustración 3.3 se observa con claridad la función de esta operación.

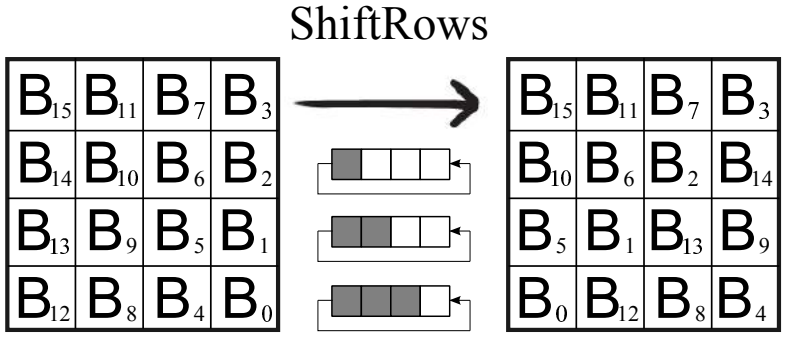


Figura 3.3: Operación ShiftRows

3.2.2.3. MixColumns

En la operación *MixColumns*, los cuatro bytes de cada columna de la notación matricial del Estado se combinan utilizando una transformación lineal inversible, como establece [NIS01].

Cada columna se trata como un polinomio y luego se multiplica el módulo $x^4 + 1$ con un polinomio fijo $a(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}$

Es más sencillo verlo como una multiplicación matricial, donde el Estado siempre se multiplica a la derecha de la misma matriz:

$$\begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix}$$

$$\text{Es decir, } \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \cdot \begin{pmatrix} B_{15} & B_{11} & B_7 & B_3 \\ B_{14} & B_{10} & B_6 & B_2 \\ B_{13} & B_9 & B_5 & B_1 \\ B_{12} & B_8 & B_4 & B_0 \end{pmatrix}$$

Multiplicando de esta manera, siempre teniendo en cuenta los aspectos matemáticos que hemos comentado en el apartado 3.2.1.3, obtendremos el resultado de la operación *MixColumns*.

Dentro del AES, este es, sin duda, uno de los procesos más costosos en cuanto a cantidad de operaciones realizadas.

3.2.2.4. AddRoundKey

La operación *AddRoundKey* consiste, según [NIS01], en la combinación de la subclave de ronda correspondiente con el Estado.
Esta combinación se realiza a través de la operación XOR.

En la ilustración que representa el proceso de cifrado (3.2) se observa que en la ronda inicial (ronda 0) se realiza esta operación. En el caso que estamos estudiando, longitud de clave de 128 bits, la subclave de ronda 0 es la propia clave de cifrado.

Aunque se estudiará más adelante, es importante señalar que, en el caso de tener claves de 192 y 256 bits, su correspondiente subclave de ronda 0 no es, lógicamente, la clave original, sino que será un fragmento de 128 bits de ésta, en concreto sus 128 bits más significativos.

3.2.3. Desencriptador AES

De nuevo, para definir el proceso de descifrado, vamos a suponer que la longitud de clave escogida es de 128 bits, lo cual implica 10 rondas.

Hay que recordar que AES es un cifrador de clave simétrica, lo que quiere decir que la misma clave que se ha usado para cifrar los datos es la que se usará para descifrarlos.

Básicamente, el proceso de descifrado consiste en aplicar, en orden inverso al del cifrado, las operaciones inversas a las descritas en el encriptador. De nuevo, tenemos cuatro operaciones diferentes:

- InvSubBytes.
- InvShiftRows.
- InvMixColumns.
- AddRoundKey.

En la figura 3.4 se explica como se distribuyen las operaciones realizadas en el descifrado.

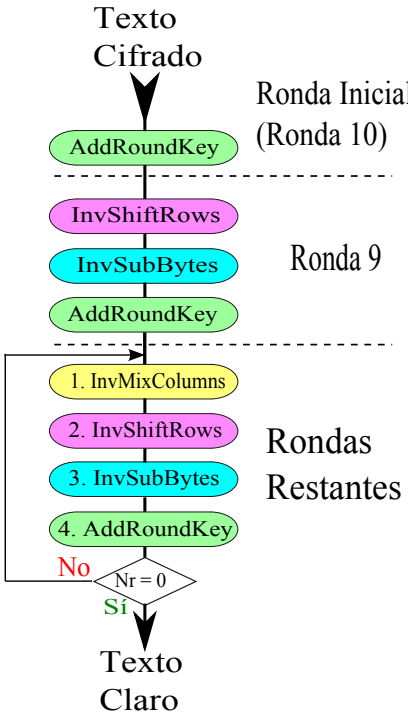


Figura 3.4: Proceso de descifrado

La operación *AddRoundKey* es la misma que en la encriptación, es decir, una operación XOR entre el Estado y la subclave correspondiente, por lo que no se volverá a explicar.

Como vemos en la figura 3.4, el proceso empieza en la última ronda, es decir, que la primera operación que aparece, *AddRoundKey*, utilizará la subclave número 10.
Se puede deducir que si las longitudes de clave son 192 ó 256 bits, la primera ronda que se usará en el descifrado será la 12 o la 14 respectivamente.

Posteriormente se realiza la ronda 9, en la que se aplican al Estado las operaciones *InvShiftRows*, *InvSubBytes* y *AddRoundKey*.

Por último, se realizan, hasta la ronda final (ronda 0), las diversas operaciones de descifrado en este orden: *InvMixColumns*, *InvShiftRows*, *InvSubBytes* y *AddRoundKey*.

De nuevo, la subclave de ronda 0 es la clave original (si la longitud de clave es de 128 bits).

A continuación describiremos las operaciones necesarias para el descifrado.

3.2.3.1. InvSubBytes

Según [NIS01], la operación *InvSubBytes* es, al igual que la operación *SubBytes*, una sutitución no lineal de bytes. Dicha sustitución se realiza aplicando la fórmula:

$$S'_{i,j} = (M^{-1} \cdot (S_{i,j} + C))^{-1}$$

De nuevo, existe una tabla de sustitución fija a la que hemos denominado **InvS-box** (tabla 3.4), la cual permite realizar la operación *InvSubBytes* de manera análoga a la operación *SubBytes*.

	x0	**x1**	**x2**	**x3**	**x4**	**x5**	**x6**	**x7**	**x8**	**x9**	**xA**	**xB**	**xC**	**xD**	**xE**	**xF**
0x	52	09	6A	D5	30	36	A5	38	BF	40	A3	9E	81	F3	D7	FB
1x	7C	E3	39	82	9B	2F	FF	87	34	8E	43	44	C4	DE	E9	CB
2x	54	7B	94	32	A6	C2	23	3D	EE	4C	95	0B	42	FA	C3	4E
3x	08	2E	A1	66	28	D9	24	B2	76	5B	A2	49	6D	8B	D1	25
4x	72	F8	F6	64	86	68	98	16	D4	A4	5C	CC	5D	65	B6	92
5x	6C	70	48	50	FD	ED	B9	DA	5E	15	46	57	A7	8D	9D	84
6x	90	D8	AB	00	8C	BC	D3	0A	F7	E4	58	05	B8	B3	45	06
7x	D0	2C	1E	8F	CA	3F	0F	02	C1	AF	BD	03	01	13	8A	6B
8x	3A	91	11	41	4F	67	DC	EA	97	F2	CF	CE	F0	B4	E6	73
9x	96	AC	74	22	E7	AD	35	85	E2	F9	37	E8	1C	75	DF	6E
Ax	47	F1	1A	71	1D	29	C5	89	6F	B7	62	0E	AA	18	BE	1B
Bx	FC	56	3E	4B	C6	D2	79	20	9A	DB	C0	FE	78	CD	5A	F4
Cx	1F	DD	A8	33	88	07	C7	31	B1	12	10	59	27	80	EC	5F
Dx	60	51	7F	A9	19	B5	4A	0D	2D	E5	7A	9F	93	C9	9C	EF
Ex	A0	E0	3B	4D	AE	2A	F5	B0	C8	EB	BB	3C	83	53	99	61
Fx	17	2B	04	7E	BA	77	D6	26	E1	69	14	63	55	21	0C	7D

Tabla 3.4: InvS-Box

La tabla 3.4 es inversa a la tabla 3.3, que representaba la S-box. Recordemos el ejemplo que se propuso, en el que el byte $\{19\}$, a través de la operación

SubBytes se transformaba en el byte {D4}.
Si ahora queremos aplicar la operación *InvSubBytes* al byte {D4}, miramos en la tabla 3.4 la fila **Dx** y la columna **x4**.
Obtenemos el byte {19}.

3.2.3.2. InvShiftRows

La operación *InvShiftRows* consiste, según [NIS01], en una rotación cíclica hacia la derecha de las filas de la notación matricial del Estado, de manera que la primera fila permanece igual, la segunda fila se rota hacia la derecha una posición, la tercera fila se rota hacia la derecha dos posiciones y, por último, la cuarta fila se rota hacia la derecha tres posiciones.

Consiste, por tanto, en una rotación en dirección opuesta a la que se propuso en la operación *ShiftRows*, como se expone en la figura 3.5.

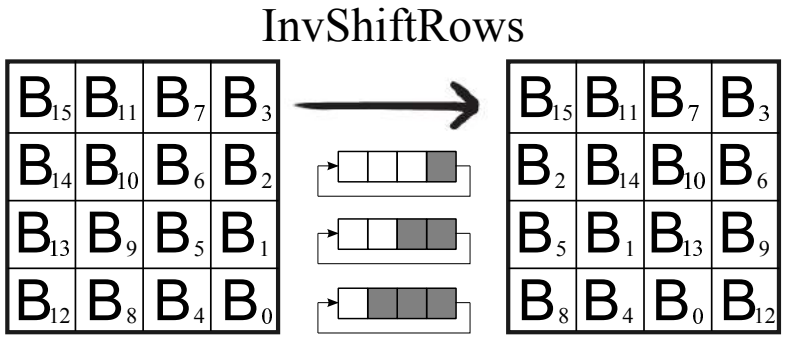


Figura 3.5: Operación InvShiftRows

3.2.3.3. InvMixColumns

Según [NIS01], es la operación inversa a *MixColumns*.
En ella, cada columna se trata como un polinomio y luego se multiplica el módulo x^4+1 con un polinomio fijo $a^{-1}(x) = \{0B\}x^3+\{0D\}x^2+\{09\}x+\{0E\}$

De nuevo, es más intuitivo verlo como una multiplicación del Estado a la derecha de una matriz fija. Dicha matriz es:

$$\begin{pmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{pmatrix}$$

Es decir, $\begin{pmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{pmatrix} \cdot \begin{pmatrix} B_{15} & B_{11} & B_7 & B_3 \\ B_{14} & B_{10} & B_6 & B_2 \\ B_{13} & B_9 & B_5 & B_1 \\ B_{12} & B_8 & B_4 & B_0 \end{pmatrix}$

3.2.4. Cálculo de subclaves

En el AES, concretamente en la operación *AddRoundKey*, se utilizan diferentes subclaves, todas derivadas de la clave original.

La clave expandida, una sucesión de todas las subclaves, puede verse como una matriz de 4 filas por $[4 \times (Nr + 1)]$ columnas.
Es decir, que la longitud de la clave expandida varía dependiendo de Nr, que a su vez varía dependiendo de la longitud de clave.

Por ejemplo, la clave expandida para una longitud de clave de 128 bits se representa como una matriz de 4 filas por (4×11), es decir, 44 columnas.

Análogamente, para una longitud de clave de 192 bits (la cual se representa en la clave expandida como una matriz de 4×6), la clave expandida es una matriz de 4 filas por (4×13), o lo que es lo mismo, 52 columnas.

Todas las subclaves utilizadas, para cualquier longitud de clave, son de 128 bits, o lo que es lo mismo, 4 columnas. Esa es la razón por la que a más longitud de clave, más número de rondas.

Es importante señalar que el cálculo de subclaves es idéntico para longitudes de clave de 128 y 192 bits, mientras que para 256 bits varía en ciertos detalles.

Un dato del AES, necesario para el cálculo de subclaves para claves de cualquier longitud, es la matriz Rcon.
Dicha matriz, según [NIS01] es de la siguiente forma:

$$Rcon = \begin{pmatrix} 01 & 02 & 04 & 08 & 10 & 20 & 40 & 80 & 1B & 36 \\ 00 & 00 & 00 & 00 & 00 & 00 & 00 & 00 & 00 & 00 \\ 00 & 00 & 00 & 00 & 00 & 00 & 00 & 00 & 00 & 00 \\ 00 & 00 & 00 & 00 & 00 & 00 & 00 & 00 & 00 & 00 \end{pmatrix}$$

Es necesario saber que en este proyecto se ha designado a la primera columna de Rcon como Rcon(1), a la segunda como Rcon(2) y así sucesivamente hasta la última columna, Rcon(10).

Las columnas de la matriz de clave ampliada se calculan en diez grupos (uno por cada columna de la matriz Rcon) de Nk columnas cada uno.

3.2.4.1. Cálculo de subclaves a partir de una clave de longitud 128 ó 192 bits

Ya que el procedimiento es idéntico para ambas longitudes, se asumirá que la longitud de la clave a expandir es de 128 bits.
Para facilitar la explicación, vamos a apoyarnos en un ejemplo real.

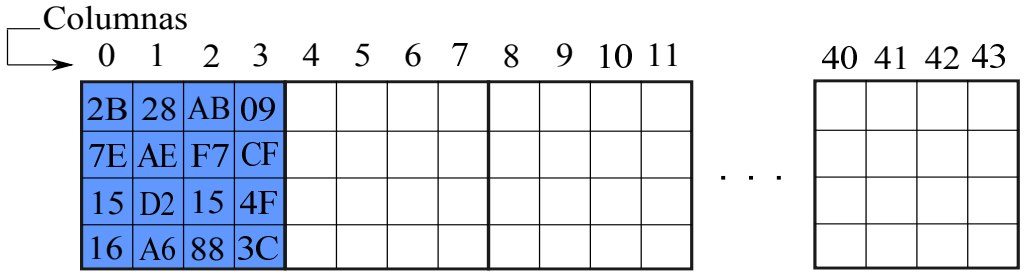


Figura 3.6: Matriz de clave expandida

Como vemos en la figura 3.6, la matriz de clave expandida es de 4 filas por 44 columnas.

1. El primer paso para calcular las subclaves es introducir la clave original en las primeras columnas. En el caso de 128 bits, ocupa 4 columnas, como se puede apreciar en la ilustración.

Ahora hay que calcular las columnas de las diferentes subclaves una a una.
Básicamente, la primera columna del grupo de Nk columnas (en este caso 4) aplica una serie de operaciones que describiremos a continuación, mientras que el resto de columnas del grupo se calculan de manera análoga.

Vamos a denominar como C_i a la columna actual que queremos hallar. En este momento $i = 4$

2. Se aplica sobre C_{i-1} una rotación hacia arriba.

   Es decir, $\begin{pmatrix} 09 \\ CF \\ 4F \\ 3C \end{pmatrix} \Rightarrow \begin{pmatrix} CF \\ 4F \\ 3C \\ 09 \end{pmatrix}$

3. Sobre esa columna rotada, se aplica la operación *SubBytes*.

   Entonces, $\begin{pmatrix} CF \\ 4F \\ 3C \\ 09 \end{pmatrix} \Rightarrow \begin{pmatrix} 8A \\ 84 \\ EB \\ 01 \end{pmatrix}$

4. A la transformación *SubBytes* que acabamos de hallar se le suman C_{i-nk} y $Rcon(i/nk)$, obteniendo así la columna C_i que estabamos buscando.

Hay que recordar que la suma es una operación XOR y que $nk = 4$, 6 ó 8 para longitudes de clave 128, 192 ó 256 bits respectivamente. En este caso $nk = 4$, por lo tanto, la columna de la matriz Rcon que hay que utilizar en este momento ($i = 4$) es Rcon(1), es decir, la primera columna.

Por lo tanto, $\begin{pmatrix} 8A \\ 84 \\ EB \\ 01 \end{pmatrix} \oplus \begin{pmatrix} 2B \\ 7E \\ 15 \\ 16 \end{pmatrix} \oplus \begin{pmatrix} 01 \\ 00 \\ 00 \\ 00 \end{pmatrix} = \begin{pmatrix} A0 \\ FA \\ FE \\ 17 \end{pmatrix} = C_4$

Llegados a este punto, $i = 5$.

5. Para hallar C_i no hay más que realizar la suma de C_{i-1} y C_{i-nk}.

   Es decir, $\begin{pmatrix} A0 \\ FA \\ FE \\ 17 \end{pmatrix} \oplus \begin{pmatrix} 28 \\ AE \\ D2 \\ A6 \end{pmatrix} = \begin{pmatrix} 88 \\ 54 \\ 2C \\ B1 \end{pmatrix} = C_5$

Una vez hecho esto, las dos columnas que quedan por calcular de la primera subclave, C_6 y C_7, se hallan aplicando el paso 5.

Para calcular una nueva subclave, volvemos al paso 2 y comenzamos de nuevo.

3.2.4.2. Cálculo de subclaves a partir de una clave de longitud 256 bits

Como hemos dicho, el cálculo de las subclaves para una longitud de clave de 256 bits difiere en algunos aspectos.

Dentro de la matriz de clave expandida, la clave original ocupa 8 columnas. A partir de éstas, y siguiendo exactamente los mismos pasos descritos en el apartado anterior, obtendríamos columnas en grupos de 8. Sin embargo, hay que introducir un paso adicional.

Este paso consiste en que, para hallar la columna número cinco de cada grupo de nuevas columnas, hay que realizar la operación *SubBytes* de C_{i-1} y, posteriormente, sumarla con C_{i-nk}, donde $nk = 8$.

Si una clave original ocupa los lugares del 0 al 7 en la matriz de clave expandida, la columna que ocupa el quinto puesto en el nuevo grupo de columnas (del 8 al 15) será la número 12. En el siguiente grupo de columnas (del 16 al 23), el quinto puesto es ocupado por la columna número 20. Y así sucesivamente.

Por lo tanto, si queremos hallar C_{12}, necesitaremos aplicar la operación *SubBytes* a la C_{11} y sumar el resultado con C_4.

Por lo demás es equivalente al apartado anterior, para hallar la primera columna del grupo se siguen los pasos 2, 3 y 4. Para el resto de columnas, exceptuando la quinta, se sigue el paso 5 del apartado anterior.

Capítulo 4

Herramientas de trabajo y flujo de diseño

En el presente capítulo se mostrarán las principales herramientas que se han utilizado para la consecución de este proyecto.

Además, se explicará la manera en la que se han usado esas herramientas a lo largo del flujo de diseño, es decir, qué herramienta se ha utilizado en cada paso y qué hemos obtenido con ella.

4.1. Herramientas utilizadas

Las herramientas de trabajo utilizadas son las siguientes:

- Xilinx ISE Design Suite 13.2
- Modelsim 10.1c SE
- Rijndael Inspector 1.1
- VIM
- XPower Analyzer

4.1.1. Xilinx ISE Design Suite 13.2

Xilinx ISE (Integrated Software Enviroment) en una herramienta software, propiedad de Xilinx, que permite, entre otras funciones, describir el diseño y realizar su síntesis, su emplazamiento y su rutado.

Además, con esta herramienta software también se distribuye la herramienta **Xpower Analyzer** (ver apartado 4.1.5).

4.1.2. Modelsim 10.1c SE

Modelsim es una herramienta software, propiedad de Mentor Graphics, que permite, entre otras cosas, realizar simulaciones funcionales y simulaciones temporales, con las que podremos ver el funcionamiento real del circuito teniendo en cuenta especificaciones como los retardos.

4.1.3. Rijndael Inspector 1.1

Rinjndael Inspector es un software diseñado por Enrique Zabala para el proyecto Cryptool.

En él se permite, aplicando el algoritmo AES, realizar el cifrado o descifrado de un bloque de 128 bits, utilizando una clave de una longitud de 128 bits. Se puede introducir manualmente el valor tanto del bloque a cifrar o descifrar como de la clave.

Ha sido especialmente de utilidad porque no sólo encripta o desencripta los datos, sino que muestra todos y cada uno de los valores que adquiere el Estado, además de las diferentes subclaves, es decir, que muestra los resultados después de cada operación realizada (*SubBytes*, *MixColumns*, etc).

4.1.4. VIM

VIM es un editor de texto que comprende más de 200 tipos de sintaxis.

La utilidad más importante para este proyecto es la de la comparación de ficheros. En ella, los dos (o más) ficheros de texto se comparan y sus diferencias se remarcan con diversos colores. Por ejemplo, las eliminaciones aparecen en rojo, mientras que las nuevas inserciones aparecen en violeta.

4.1.5. XPower Analyzer

Como hemos dicho, el *XPower Analyzer* viene distribuida con el *Xilinx ISE*.

Con esta herramienta se puede hacer una estimación detallada del consumo de potencia, teniendo en cuenta valores como la temperatura.

Para ello, precisa de información que se obtendrá con el uso de las herramientas antes descritas. Esto será explicado más adelante.

4.2. Flujo de diseño

La realización de este proyecto sigue unos pasos muy específicos. En cada paso precisaremos de herramientas diferentes.

El objeto de esta sección es explicar uno a uno los pasos que se siguen en el diseño, aclarando qué herramienta se utiliza en cada uno y describiendo qué se obtiene de ellos.

Básicamente, los pasos son:

1. Descripción en lenguaje VHDL.
2. Simulación funcional.
3. Síntesis, emplazamiento y rutado.
4. Simulación postlayout.
5. Estimación del consumo.

Estos pasos hay que ejecutarlos en orden. Si alguno no proporciona los resultados deseados, se vuelve al paso 1.

4.2.1. Descripción en lenguaje VHDL

El primer paso consiste en diseñar los ficheros .vhd que describen el comportamiento de nuestro circuito.
En nuestro caso, el AES se ha dividido en componentes, con lo cual tendremos un archivo .vhd para cada componente.

El requisito necesario para poder avanzar hasta el siguiente paso es realizar una sintaxis correcta, es decir, compilar los archivos .vhd de manera que no haya ningun error.

Si lo hay, será necesario modificar el código de descripción.

4.2.1.1. Herramientas utilizadas para describir el diseño

Cualquier editor de texto puede ser utilizado para describir el diseño, sin embargo, se ha optado por utilizar el editor del *Xilinx ISE Design Suite 13.2*, ya que también nos permite realizar la sintaxis, que es necesaria para verificar que el código descrito no tiene errores.

Que el código sea válido no significa que el funcionamiento sea el deseado. Hay que comprobarlo en el siguiente paso.

4.2.2. Simulación funcional

En este paso podremos comprobar si nuestro circuito funciona tal y como lo hemos descrito.

Si el resultado es el esperado podremos avanzar hasta el siguiente paso. En caso contrario, es necesario volver al primer paso.

4.2.2.1. Herramientas utilizadas para la simulación funcional

En este caso hemos utilizado las herramientas *Modelsim 10.1c SE* y *VIM*.

En la herramienta *Modelsim 10.1c SE* tenemos información visual de los valores o estados tanto de los pines de entrada y salida como de las señales y variables internas descritas en el diseño en función del tiempo. Es decir, que somo capaces de ver como reacciona cada señal o variable interna ante un estímulo conocido de los pines de entrada (el cual podemos realizar a través de esta misma herramienta).

Si la salida no es la deseada, es necesario identificar el problema estudiando los valores que adquieren las señales o variables internas, para así poder determinar donde está el fallo y volver al paso 1.

Si la salida es la deseada podremos avanzar al siguiente paso.

La herramienta *VIM* se ha utilizado porque, en simulaciones largas, en las que podemos cifrar o descifrar un número elevado de bloques de 128 bits, generando un archivo de texto con los resultados del proceso, es útil poder comparar ese archivo de texto con otro que contenga las salidas esperadas.

En el proyecto que nos ocupa, se han realizado simulaciones de hasta 100 vectores oficiales del NIST [Vec].
Hablando en términos de tiempo, es más económico utilizar esta herramienta que comparar los resultados uno a uno.

4.2.3. Síntesis, emplazamiento y rutado

La síntesis es el paso en el que se adapta el diseño decrito a una FPGA en concreto de manera que ocupe el menor área posible.

El emplazamiento es un proceso en el que se sitúan los bloques digitales de una manera óptima, mientras que el rutado se encarga de interconectarlos.

Este paso genera unos ficheros necesarios para los siguientes pasos:

- **.ncd (Native Circuit Description)**: Contiene la descripción del circuito.
- **.pcf (Physical Constraints File)**: Contiene las restricciones de emplazamiento y rutado.
- **.sdf (Standard Delay Format)**: Contiene los retardos internos de la FPGA seleccionada.
- **.vhd**: contiene el código del archivo .vhd original pero a un nivel más bajo, teniendo en cuenta retardos.

Los archivos .ncd y .pcf se utilizarán en la estimación de la potencia consumida, mientras que los archivos .sdf y .vhd se utilizarán en el paso de la simulación postlayout.

4.2.3.1. Herramientas utilizadas para la síntesis, emplazamiento y rutado

La herramienta utilizada para este paso es *Xilinx ISE Design Suite 13.2*.

Antes de poder realizar el emplazamiento y el rutado es necesario obtener una síntesis correcta, es decir, que no haya errores ni warnings (latches, etc).

En nuestro caso, ha sido una de las etapas más costosas ya que, a veces, era muy difícil identificar el problema al que se referían los warnings.

Cuando se realiza una síntesis correcta, el proceso de emplazamiento y rutado no tiene por qué dar problemas, pudiendo avanzar así al siguiente paso.

4.2.4. Simulación postlayout

Consiste en una simulación teniendo en cuenta todos los posibles retardos. Con ella obtendremos unos resultados muy aproximados a la realidad.

Para poder avanzar, es necesario obtener los resultados esperados en la simulación.

Este paso no genera ningún archivo adicional por sí mismo, pero es necesario que generemos uno para obtener el consumo de potencia en la etapa siguiente.

- **.vcd (Value Change Dump)**: Contiene una traza de la evolución de todas las señales del circuito.

Para obtener el archivo .vcd es necesario escribir en un editor de texto un archivo con extensión .do.
Se trata de un script de instrucciones que generará el archivo buscado tras realizar la simulación postlayout. Este script se puede encontrar en el capítulo de Anexos (capítulo 10).

4.2.4.1. Herramientas utilizadas para la simulación postlayout

La herramienta utilizada en ese paso es, de nuevo, el *Modelsim 10.1c SE*. La mecánica es la misma que en la simulación funcional, con la excepción de que, para realizarla, utiliza los archivos .vhd de bajo nivel y .sdf, ambos generados en el paso anterior (síntesis, emplazamiento y rutado).

Tambien se utiliza la herramienta *VIM* por el mismo motivo que en la simulación funcional.

4.2.5. Estimación del consumo

En este último paso, en el que todo el flujo de diseño ha sido correcto, se estima la potencia consumida por el circuito para, posteriormente, poder trabajar en su reducción.

4.2.5.1. Herramientas utilizadas para la estimación del consumo

La herramienta utilizada es el *XPower Analyzer*.

Para poder realizar una estimación lo más próxima posible a la realidad, el *XPower Analyzer* requiere tres archivos:

- **.ncd**: Obtenido en el paso "*Síntesis, emplazamiento y rutado*".
- **.pcf**: Obtenido igualmente en el paso "*Síntesis, emplazamiento y rutado*".
- **.vcd**: Obtenido en el paso "*Simulación postlayout*".

Para ayudar a comprender mejor el flujo de diseño, se adjunta la figura 4.1.

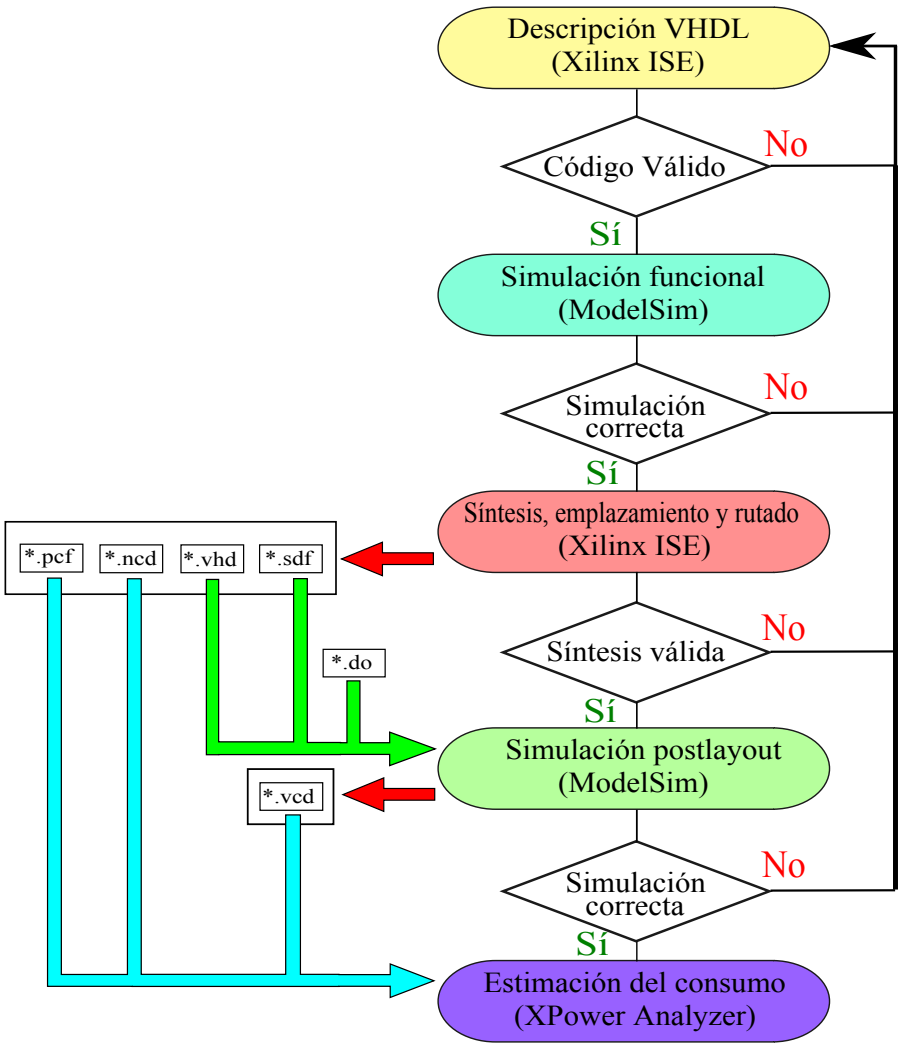


Figura 4.1: Flujo de diseño

Capítulo 5

Implementación del algoritmo AES

En este capítulo se explicará la implementación realizada del algoritmo AES, describiendo cada uno de los bloques y componentes que conforman el diseño. Para hacerlo nos ayudaremos de imágenes y los resultados de las simulaciones.

Es necesario señalar que para realizar las simulaciones del algoritmo se han utilizado vectores del NIST [Vec], mientras que para comprobar el correcto funcionamiento de cada componente por separado se ha utilizado la herramienta *Rinjndael Inspector 1.1.*

El código VHDL del diseño se puede encontrar en los anexos.

5.1. Implementación de AES

Como hemos dicho, el AES puede trabajar con tres longitudes de clave diferentes: 128, 192 y 256 bits.

Esta implementación permite el uso de las tres longitudes, ya que se han aprovechado los recursos que ofrecen la declaración de **genéricos**.
Como siempre, para realizar la explicación de la implementación, asumiremos que la longitud de clave es de 128 bits.
Además, se ha optado por realizar una descripción estructurada, es decir, que se han descrito varios componentes interconectados mediante señales.

A continuación se incluye una lista donde se describen brevemente los

diferentes componentes que conforman nuestro bloque principal AES (tabla 5.1).

Componente	**Descripción**
subbytes	Realiza tanto la operación *SubBytes* como *InvSubBytes*.
shiftrows	Realiza tanto la operación *ShiftRows* como *InvShiftRows*.
mixcolumns	Realiza tanto la operación *MixColumns* como *InvMixcolumns*.
addroundkey	Realiza la operación AddRoundKey.
keygen	Genera todas las subclaves de ronda y devuelve una en función del número de ronda.

Tabla 5.1: Componentes de AES

En la ilustración 5.1 se muestran los bloques que forman nuestro diseño, incluyendo todas las señales que se han descrito. Podemos ver que, además de los componentes antes mencionados, hay otros bloques adicionales.

El elevado número de señales es el motivo por el que en el diagrama no se han interconectado los bloques, sin embargo en todos se detallan las señales y pines de entrada que ejercen influencia sobre ellos y las señales y pines de salida a los que modifican.

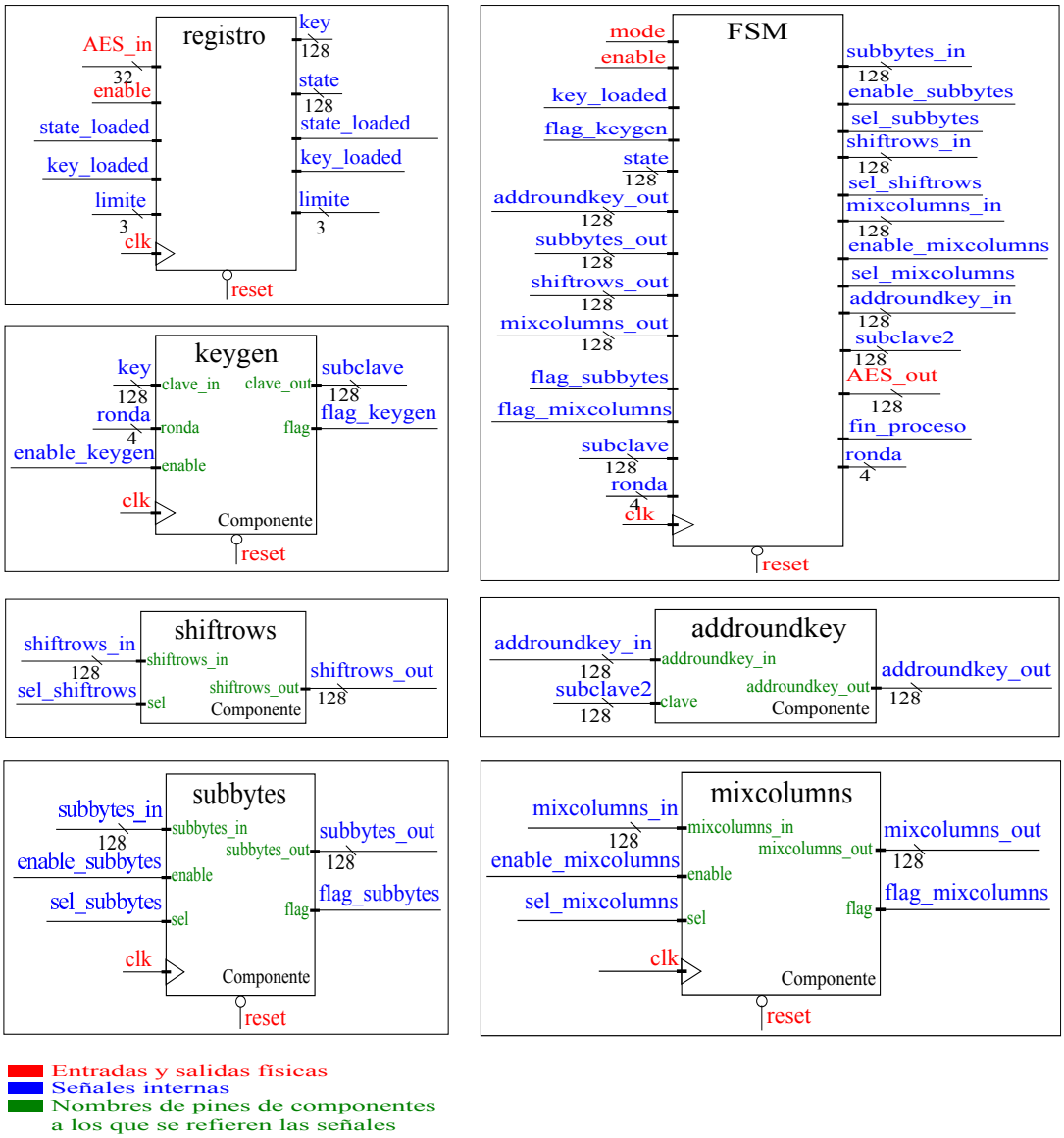


Figura 5.1: Diagrama de bloques del AES

Si observamos el diagrama, vemos que nuestra descripción contempla una entrada de únicamente 32 bits (AES_in), por la que entran los datos tanto de la clave como de la información a cifrar o descifrar.

En principio esto no era así, teniendo 128 bits de entrada para los datos y otros 128, 192 ó 256 para la clave.
Sin embargo, el elevado número de pines resultante nos obligó a adaptarlo

de esta manera, ya que nos permitía implementar el diseño en una FPGA de tamaño adecuado, es decir, con mayor aprovechamiento del área, mejorando también su consumo de potencia.

Debido a esto, fue necesario incluir el bloque al que hemos denominado como **registro**. Su función simplemente es agrupar los datos de entrada y separarlos en clave y en datos a cifrar (o descifrar).
Además, cuando realiza este cometido, habilita a la FSM para comenzar su funcionamiento.

El bloque denominado **FSM** en la ilustración 5.1 es una máquina de estados en la que realiza el procesamiento de los datos de entrada, ofreciendo el resultado buscado.

Cabe destacar que se tuvo en cuenta la opción de comenzar con el cifrado mientras aun se estaban calculando las diferentes subclaves (en el descifrado esto no es posible, ya que la primera subclave que se requiere es la última). Sin embargo no se llevó a cabo porque se comprobó que cuando se requería la primera subclave, ésta no había sido calculada todavía.

5.1.1. Genéricos, pines de entrada y salida y señales internas de AES

A continuación haremos una descripción de todos los puertos genéricos (tabla 5.2), los pines de entrada y salida (tabla 5.3) y las señales internas del circuito (tabla 5.4).

Puertos genéricos

Nombre	Tipo	Descripción
key_length	integer	Longitud en bits de la clave.
nk	integer	Número de longitud de palabras (32 bits) de la clave.
nr	integer	Número de rondas.

Tabla 5.2: Puertos genéricos del AES

Esta implementación se ha diseñado con genéricos para poder cambiar con facilidad la longitud de clave que queremos utilizar. Como se ha comentado con anterioridad, la clave puede tener una longitud de 128, 192 ó 256 bits, *Nk*

puede tener un valor de 4, 6 ó 8 y Nr varía entre 10, 12 ó 14, respectivamente.

Entonces, si precisamos de una longitud de clave de 128 bits no hay más que configurar los genéricos de la siguiente manera:

- $key\_length = 128$
- $nk = 4$
- $nr = 10$

Entradas y salidas

Nombre	**Tipo**	**Dirección**	**Descripción**
AES_in	std_logic_vector(31 downto 0)	E	Entrada tanto de los datos a procesar como de la clave.
mode	std_logic	E	Indica si el proceso es de cifrado(1) o descifrado (0).
clk	std_logic	E	Señal de reloj.
reset	std_logic	E	Señal de reset, activa a nivel alto.
enable	std_logic	E	Señal de enable, activa a nivel alto.
AES_out	std_logic_vector(127 downto 0)	S	Salida de un bloque de datos procesado.
flag	std_logic	S	Indica que *AES_out* ofrece un dato válido.

Tabla 5.3: Entradas y salidas de AES

Como hemos dicho, la entrada *AES_in*, de 32 bits, está diseñada de modo que los datos entren en flujo, es decir, que cada ciclo de reloj entren los 32 bits correspondientes.

Está diseñado de tal manera que primeramente tiene que entrar la clave e inmediatamente después los datos a procesar.

Además el orden de entrada debe ser, en primer lugar, los 32 bits menos significativos.

Esto se entiende mejor con el siguiente ejemplo: Si partimos de la siguiente clave de 128 bits (hexadecimal):

- 00010203 04050607 08090A0B 0C0D0E0F

El primer grupo de 32 bits a introducir en nuestro diseño sería:

- 0C0D0E0F

Esto sólo ocurre con el primer bloque de datos, ya que, para el siguiente bloque, se asume que se utiliza la misma clave.
Es decir, que, tras finalizar el procesado de un bloque, se recogen únicamente cuatro palabras de 32 bits (128 bits), que son el siguiente bloque a procesar con la misma clave del anterior, evitando calcular de nuevo todas las subclaves.

Si quisieramos procesar datos con una clave diferente sería necesario aplicar un reset.

Señales internas

Nombre	**Tipo**	**Descripción**
key	std_logic_vector (key_length-1 downto 0)	Señales de
ronda	std_logic_vector(3 downto 0)	correspondencia
enable_keygen	std_logic	de puertos del
flag_keygen	std_logic	componente
subclave	std_logic_vector(127 downto 0)	**keygen**.
addroundkey_in	std_logic_vector(127 downto 0)	Señales de correspondencia
subclave2	std_logic_vector(127 downto 0)	de puertos del componente
addroundkey_out	std_logic_vector(127 downto 0)	**addroundkey**.
subbytes_in	std_logic_vector(127 downto 0)	Señales de
enable_subbytes	std_logic	correspondencia
sel_subbytes	std_logic	de puertos del
flag_subbytes	std_logic	componente
subbytes_out	std_logic_vector(127 downto 0)	**subbytes**.
shiftrows_in	std_logic_vector(127 downto 0)	Señales de correspondencia
sel_shiftrows	std_logic	de puertos del componente
shiftrows_out	std_logic_vector(127 downto 0)	**shiftrows**.
mixcolumns_in	std_logic_vector(127 downto 0)	Señales de
enable_mixcolumns	std_logic	correspondencia
sel_mixcolumns	std_logic	de puertos del
flag_mixcolumns	std_logic	componente
mixcolumns_out	std_logic_vector(127 downto 0)	**mixcolumns**.
state	std_logic_vector(127 downto 0)	Contiene los datos a procesar.
state_loaded	std_logic	Indica que la señal *state* está preparada.
key_loaded	std_logic	Indica que la señal *key* está preparada.
limite	std_logic_vector(2 downto 0)	Marca los límites dentro de *key* y *state* donde se almacena *AES_in*.
fin_proceso	integer	Indica que el proceso ha terminado.

Tabla 5.4: Señales internas del AES

Las señales de correspondencia de puertos de los diversos componentes se explicarán en los apartados de dichos componentes, pero son muy similares entre sí.

Básicamente las señales denominadas "enable" (habilitación) permiten al componente comenzar su función, las señales denominadas "flag" indican que el componente ha realizado su función y las señales denominadas "sel" indican si el proceso es de cifrado o descifrado.
El resto son entradas y salidas del Estado al componente, con excepción del componente **keygen**, al que le llega la clave y devuelve las diferentes subclaves dependiendo de la señal *ronda*, y del componente **addroundkey**, al que, aparte del Estado, le llega la correspondiente subclave por medio de la señal *subclave2*.

5.1.2. Máquinas de estado (FSM) de AES

Como hemos dicho y mostrado en el diagrama de bloques (figura 5.1), en la implementación del AES se ha descrito una máquina de estados cuya función es realizar el procesado de los datos de entrada, proporcionando un resultado válido mostrado en la salida *AES_out*.

Además, esta FSM cambia sus estados en función de la entrada *mode*, es decir, que cada estado realiza una función diferente dependiendo de si el proceso es de cifrado o de descifrado.

5.1.2.1. FSM

Lo primero que se realiza en nuestro diseño una vez que tenemos separados los 128 bits de los datos y los 128 bits de la clave, es el cálculo de subclaves.

Se ha optado por no comenzar el procesado de datos hasta que todas las subclaves hayan sido calculadas, ya que si los dos procesos comienzan al mismo tiempo, en el cifrado se requieren las subclaves cuando no han sido calculadas todavía.
Es decir, que el tiempo requerido para realizar una ronda completa es inferior al requerido para el cálculo de una subclave.

Además, debido a que en el proceso de descifrado sí se requiere tener todas las subclaves calculadas (la primera que utiliza es la última que se calcula), se ha elegido esperar también en el cifrado. De este modo tampoco se

observarán diferencias en el consumo de potencia entre un proceso u otro.

La función de **FSM** es calcular las subclaves y realizar el procesado de datos.

Consta de 9 estados:

- **inicio:** Es el estado de reposo. Cuando la señal *key_loaded* se activa, se realiza la transición de estados.
- **calculo_subclaves:** En él se desarrolla todo el proceso de cálculo de subclaves (sólo si es el primer bloque). Cuando la señal *flag_keygen* y *state_loaded* están activadas, se pasa al siguiente estado.
  Es necesario apuntar que *flag_keygen* no se desactiva en ningún momento una vez se han calculado las subclaves. De esta forma conseguimos que la transicion de estado para los siguientes bloques a procesar se produzca inmediatamente cuando el bloque a procesar ha sido cargado.
- **ronda_inicial:** Se realiza la operación que se requiere en la ronda inicial del cifrado o descifrado.
- **Osubbytes:** Realiza la operación *SubBytes* o la *InvSubBytes*. Cuando la señal *flag_subbytes* se activa, cambia de estado.
- **Oshiftrows:** Realiza la operación *ShiftRows* o la *InvShiftRows*.
- **Omixcolumns:** Realiza la operación *MixColums* o la *InvMixColumns*. Cuando la señal *flag_mixcolumns* se activa, se realiza la transición de estados.
- **Oaddroundkey:** Se realiza la operación *AddRoundKey*.
- **auxiliar:** La última ronda del cifrado y la ronda 1 inversa del descifrado son ligeramente diferentes del resto (sin contar la ronda inicial o ronda 0). Este estado ayuda a realizar esta ronda.
- **salida:** En este estado se establece la salida *AES_out* y se activa la señal *fin_proceso*.

Además, para la transición de estados, todos requieren que la entrada *enable* esté activa.
Si se desactivase el *enable*, el procesado quedaría pausado hasta su reactivación, a no ser que el proceso haya finalizado, en cuyo caso se volvería al

estado inicial para procesar el siguiente bloque, una vez el *enable* haya sido reactivado.

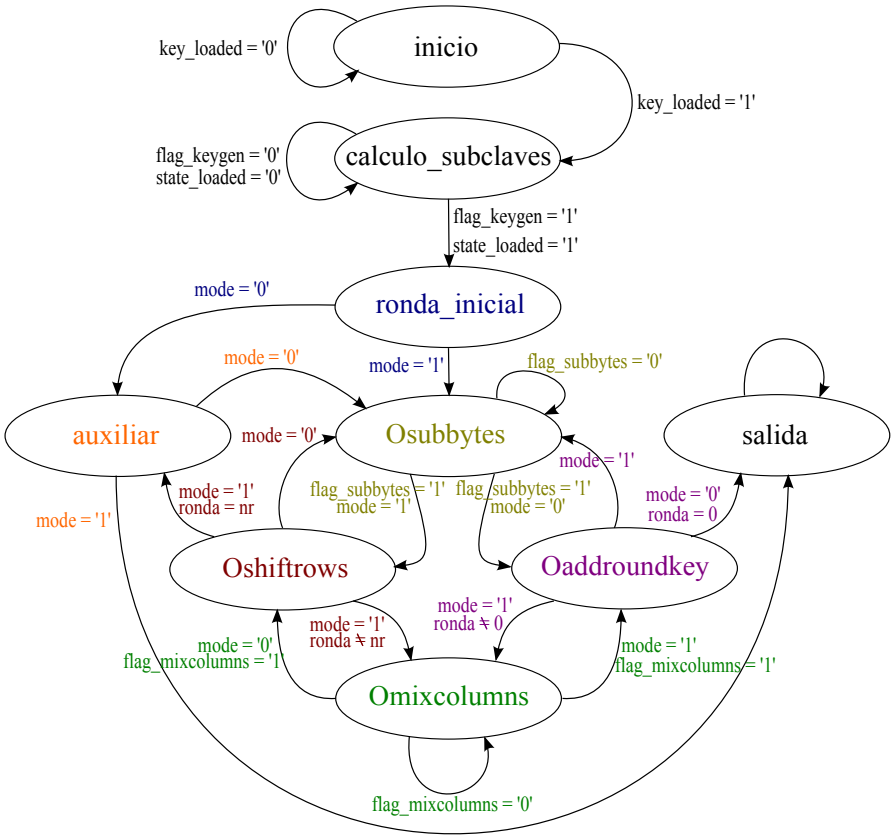


Figura 5.2: Máquina de estados del AES

En la figura 5.2 se observa como se realiza la transición entre estados. Recordemos que la entrada *mode* establece si el proceso es de cifrado (1) o descifrado (0).

5.1.3. Simulaciones de AES

En esta sección se presentarán simulaciones que nos ayudarán a demostrar el perfecto funcionamiento del algoritmo descrito.

5.1.3.1. Reset

Algunas de las señales que se utilizan en esta descripción necesitan inicializarse a un valor conocido. La señal de *Reset* es la encargada de realizar esta función.

Como ya hemos dicho, nuestra señal de reset es activa a nivel alto, es decir, que las señales se inicializarán cuando $Reset = 1$. La inicialización de señales puede apreciarse en la figura 5.3.

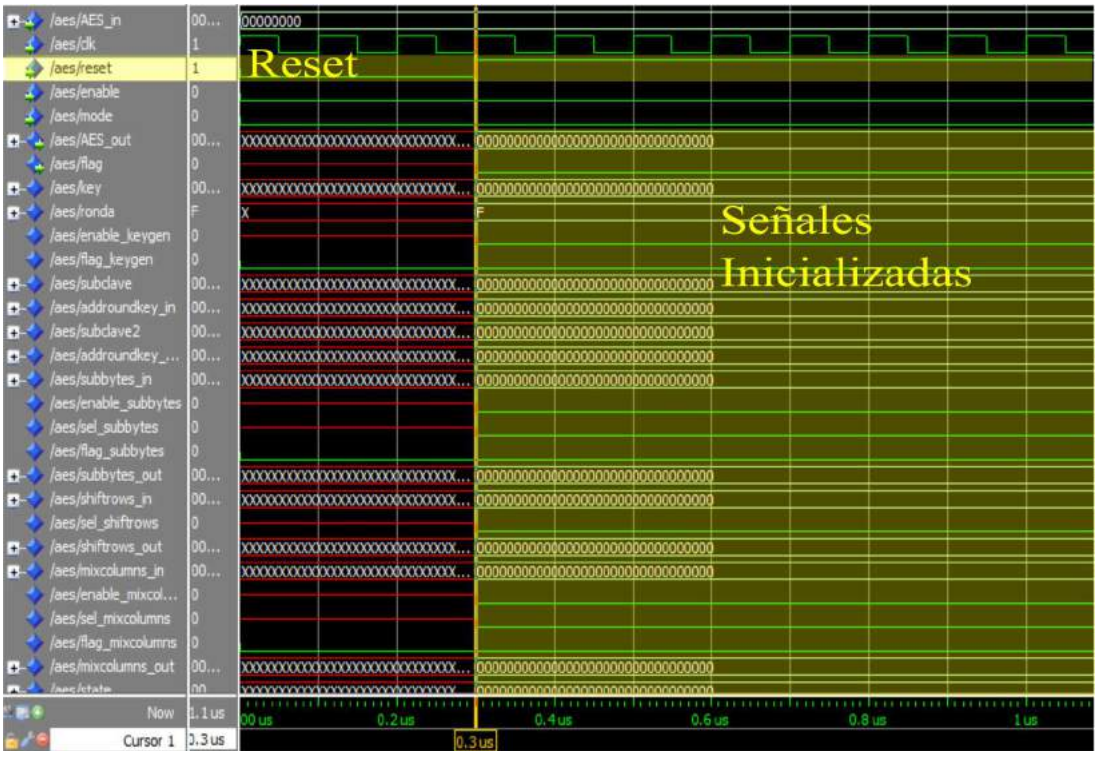


Figura 5.3: Simulación Reset (AES)

5.1.3.2. Enable

La señal *enable* habilita el proceso. En nuestro caso, debe activarse cuando los primeros 32 bits de entrada estén preparados.

Pero su función también es pausar el proceso. Como vemos en la ilutración 5.4, cuando se interrumpe la señal *enable*, la operación en la que interfiere se ejecuta hasta el final (aparición de su respectiva señal de *flag*), pero no ejecuta la siguiente hasta que la señal *enable* vuelve a activarse, es decir, el flujo normal de operación del circuito se detiene.

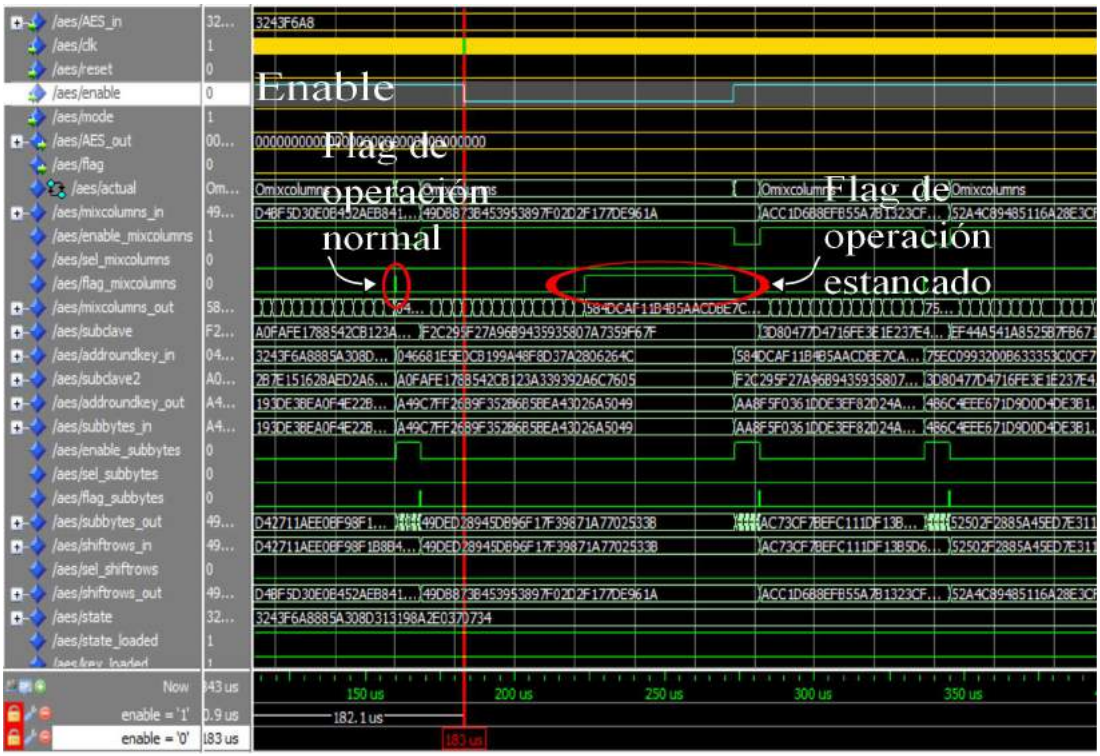


Figura 5.4: Simulación Enable (AES)

5.1.3.3. Toma de datos

Cuando se activa por primera vez la señal *enable* se procede, como se ha explicado, a la recogida de la clave y de los datos a cifrar para, posteriormente, comenzar con el cálculo de subclaves.

Como se puede apreciar en la ilustración 5.5, los cuatro primeros bloques de 32 bits se asignan a la señal *key*, activando la señal *key_loaded*, que da permiso a la máquina de estados para comenzar con el cálculo de subclaves mientras que los cuatro últimos bloques de 32 bits se asignan a la señal *state*, activando la señal *state_loaded*.

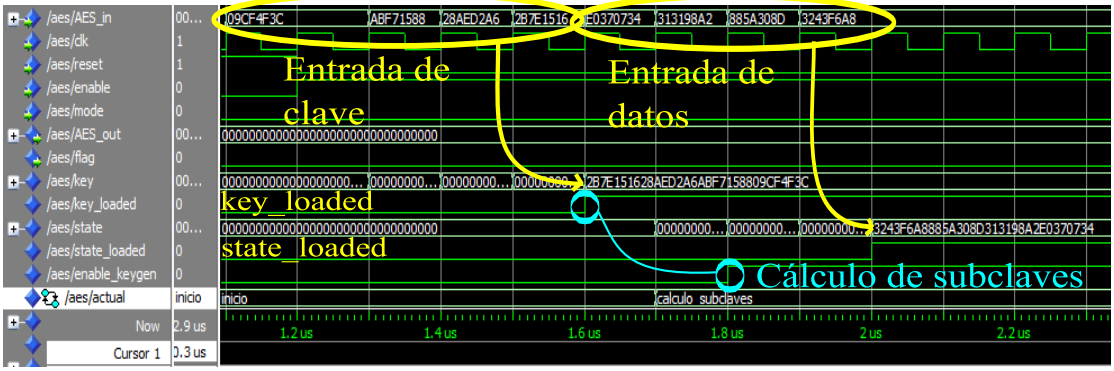


Figura 5.5: Toma de datos (AES)

5.1.3.4. Salida

Cuando se alcanza la última ronda (10 en nuestro caso, "A" hexadecimal) se realizan las operaciones necesarias y la máquina de estados pasa al estado *salida* , donde se establece el resultado buscado en la salida *AES_out*, activando también la salida *flag* (la cual copia el valor de la señal *fin_proceso*), indicando así que el proceso ha terminado y *AES_out* tiene un valor correcto (figura 5.6).

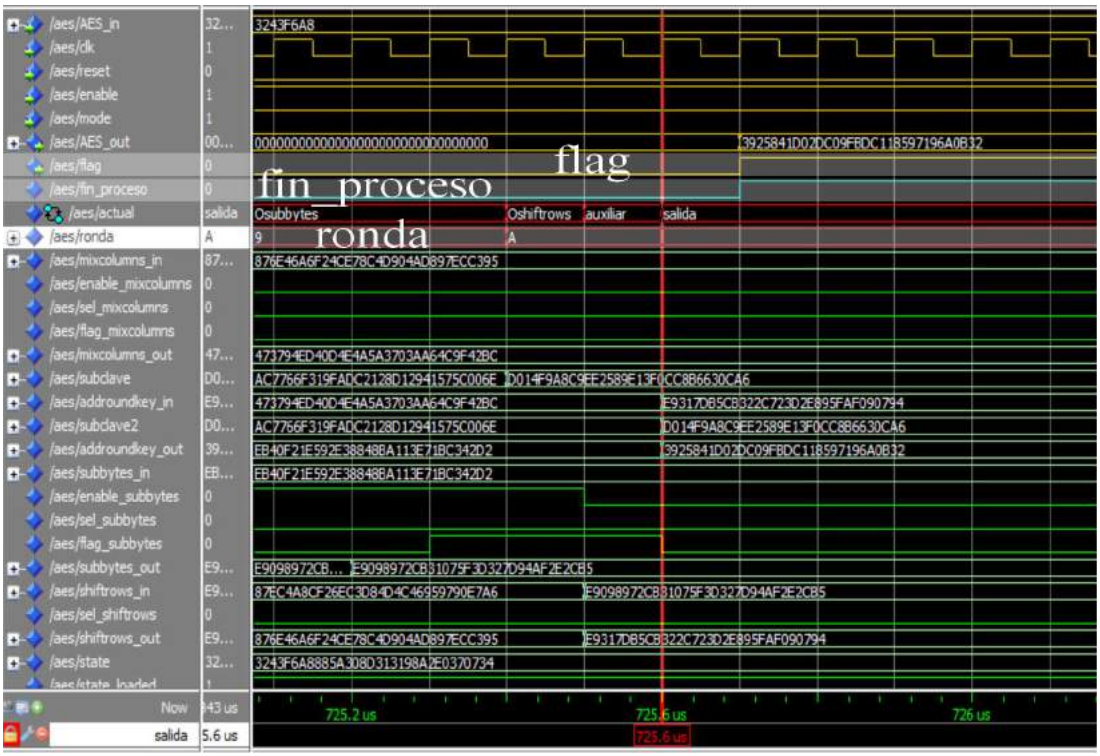


Figura 5.6: Salida (AES)

5.2. Implementación del componente SubBytes

Como ya se ha explicado, la operación *SubBytes* consiste en una sutitución no lineal de bytes, que al final se podía simplificar con la existencia de una tabla fija.

El componente *SubBytes* diseñado realiza tanto la operación *SubBytes* como *InvSubBytes*.

Además también contiene los subcomponentes mostrados en la tabla 5.5.:

Componente	Descripción
sbox	Memoria RAM de sólo lectura que contiene la tabla S-Box.
invsbox	Memoria RAM de sólo lectura que contiene la tabla InvS-Box.

Tabla 5.5: Componentes de SubBytes

En la ilustración 5.7 se muestran los bloques que forman el componente *SubBytes*, además de las señales descritas.

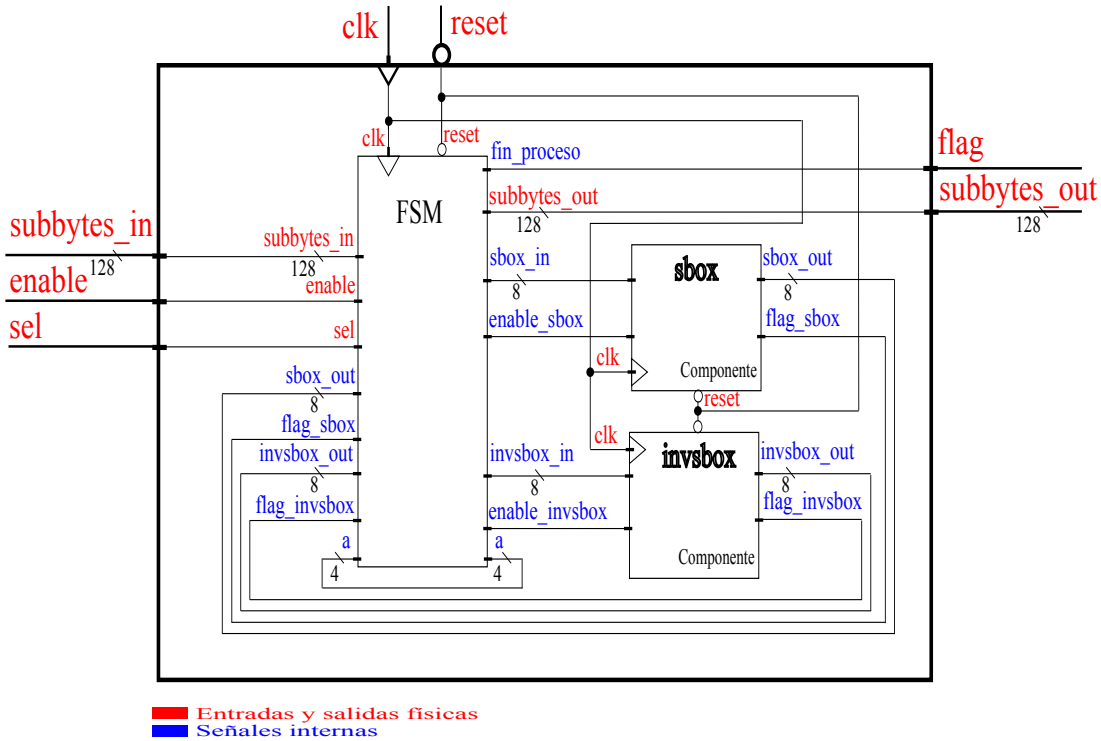


Figura 5.7: Diagrama de bloques del componente SubBytes

La máquina de estados utiliza los componentes descritos para transformar, de byte en byte, la entrada *subbytes_in*, que se va almacenando en la salida *subbytes_out* byte a byte a modo de registro de desplazamiento, marcando los límites con ayuda de la señal llamada *a*.

5.2.1. Pines de entrada y salida y señales internas del componente SubBytes

A continuación haremos una descripción de todas los pines de entrada y salida (tabla 5.6) y las señales internas (tabla 5.7) del componente *SubBytes*.

Entradas y salidas

Nombre	**Tipo**	**E/S**	**Descripción**
subbytes_in	std_logic_vector(127 downto 0)	E	Entrada del Estado al que aplicaremos la operación *SubBytes*/*InvSubbytes*.
sel	std_logic	E	Indica si la operación es *SubBytes*(0) o *InvSubBytes* (1).
clk	std_logic	E	Señal de reloj.
reset	std_logic	E	Señal de reset, activa a nivel alto.
enable	std_logic	E	Señal de enable, activa a nivel alto.
subbytes_out	std_logic_vector(127 downto 0)	S	Salida del Estado tras la operación.
flag	std_logic	S	Indica que *subbytes_out* ofrece un dato válido.

Tabla 5.6: Entradas y salidas del componente SubBytes

Señales internas

Nombre	**Tipo**	**Descripción**
sbox_in	std_logic_vector(7 downto 0)	Byte de entrada a la memoria S-Box.
enable_sbox	std_logic_vector	Permite leer la memoria S-Box.
sbox_out	std_logic_vector(7 downto 0)	Byte de salida de la memoria S-Box.
flag_sbox	std_logic_vector	Indica que la memoria S-Box ha sido leída.
invsbox_in	std_logic_vector(7 downto 0)	Byte de entrada a la memoria InvS-Box.
enable_invsbox	std_logic_vector	Permite leer la memoria InvS-Box.
invsbox_out	std_logic_vector(7 downto 0)	Byte de salida de la memoria InvS-Box.
flag_invsbox	std_logic_vector	Indica que la memoria InvS-Box ha sido leída.
finproceso	std_logic_vector	Indica que la operación ha finalizado.
a	std_logic_vector(3 downto 0)	Nos marca los límites dentro de *subbytes_in*.

Tabla 5.7: Señales internas del componente SubBytes

La salida *flag* copia los valores de la señal *finproceso* mediante un bloque de lógica adicional.

5.2.2. Máquinas de estado (FSM) del componente SubBytes

El componente *SubBytes* descrito incluye una máquina de estados cuya función es realizar la operación *SubBytes*/*InvSubBytes* byte a byte.

Para ello se gestionan los componentes que contienen las memorias RAM y se utiliza la señal *a* para emular un registro de desplazamiento, es decir, que la salida se irá actualizando cada ciclo de estados, activando la salida *flag* cuando la salida sea la correcta.

Los estados de nuestra máquina son los siguientes:

- **inicio:** Es el estado de reposo. No necesita ningún estímulo para cambiar al siguiente, salvo el que necesitan todos, es decir, *enable* = '1'.
- **mem_in:** En este estado se establece el byte a transformar de la entrada *subbytes_in* en la dirección de la memoria RAM que necesitemos, dando permiso a la misma para leer.
- **espera:** Antes de poder actualizar la salida, necesitamos un estado de transición, ya que necesitamos un pulso de reloj para que la memoria nos proporcione el valor buscado. De nuevo, no necesita estímulos para la transición.
- **mem_out:** Establece el byte transformado en la posición adecuada de la salida *subbytes_out*.
- **final:** Cuando la señal *a* nos dice que todos los bytes han sido transformados, se llega a este estado, en el que se activa la señal *finproceso*, copiada por la salida *flag*, indicandonos así que el dato está listo para ser leído.

La transición entre todos y cada uno de los estados requiere que la entrada *enable* esté activa.

En este caso, y, en general, para todos los componentes de nuestra descripción completa, cuando se desactiva, el *enable* ejerce una función similar al *reset*, ya que a lo largo del procesado de datos se utiliza más de una vez y se necesita devolver ciertas señales a valores conocidos.

Entonces, si se desactiva el *enable*, el proceso no se pausará, sino que la FSM volverá al estado de reposo.

En la figura 5.8 se observa como se realiza la transición entre estados, sin olvidar que para todas ellas es necesario que el *enable* esté activo, volviendo al estado *inicio* en caso contrario.

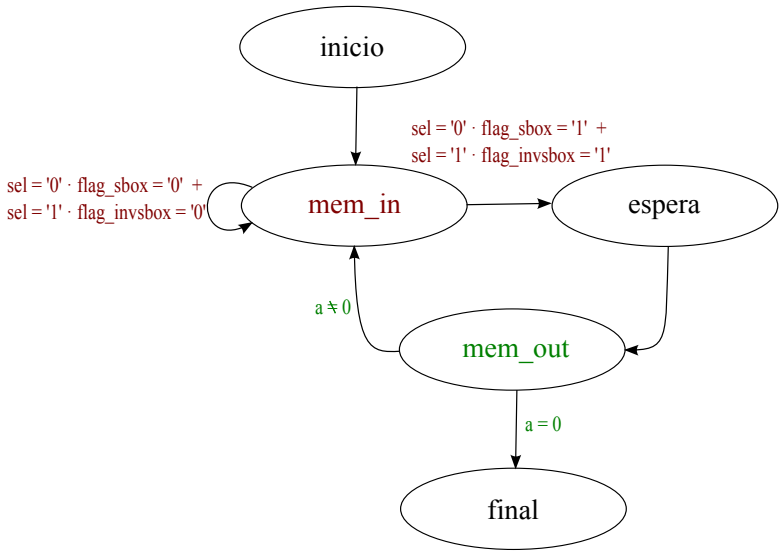


Figura 5.8: Máquina de estados del componente SubBytes

En el estado *mem_in*, aunque la transición es siempre al estado *espera*, es necesario diferenciar entre la operación *SubBytes* (*sel* = '0'), ya que entonces esperaremos el cambio en la señal *flag_sbox*, y la operación *InvSubBytes* (*sel* = '1'), en cuyo caso esperaremos el cambio en la señal *flag_invsbox*.

En el estado *mem_out* se produce la actualización del valor de la señal *a*, inicializada a un valor conocido. Cuando llega a 0 es cuando se produce la transición al estado *final*, en el que permanece hasta que se desactiva el *enable* y se vuelve al estado de reposo.

5.2.3. Simulaciones del componente SubBytes

5.2.3.1. Lectura y escritura (SubBytes)

Como se ve en la figura 5.9, cuando se activa el *enable* se coge el byte más significativo y se realiza su sustitución a través de la memoria RAM de la operación respectiva, en este caso *SubBytes*.

Posteriormente, la salida de la memoria RAM se establece en la salida *subbytes_out* en el mismo lugar que ocupaba el byte original en la entrada *subbytes_in*.

Finalmente, se actualiza el valor de la señal *a* y se repite el proceso, cogiendo el siguiente byte más significativo.

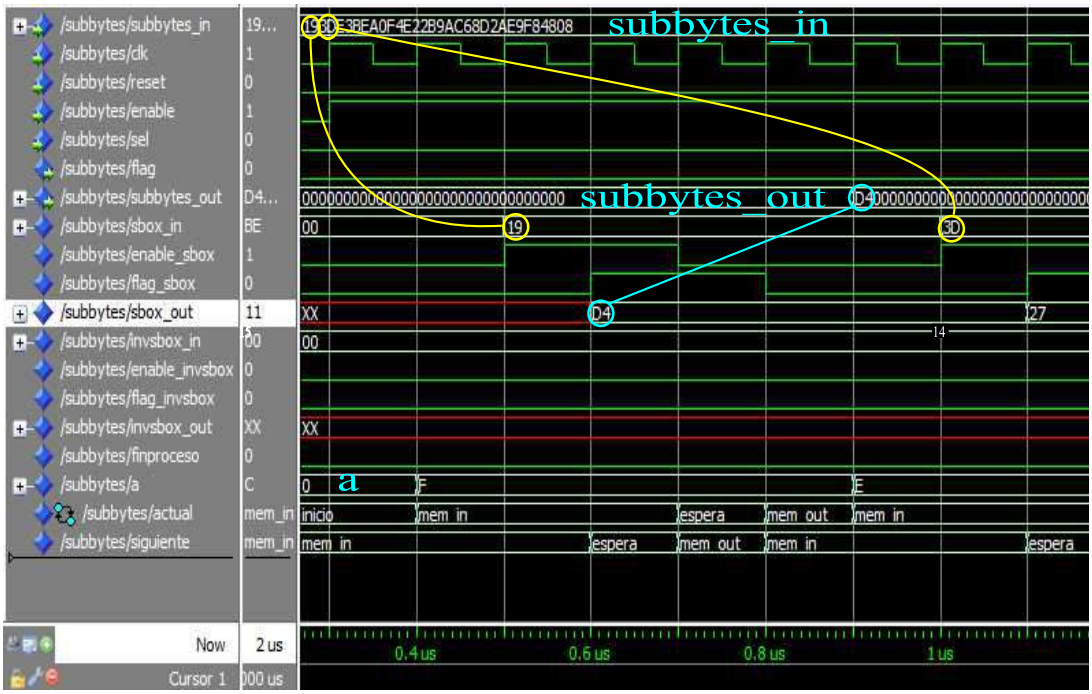


Figura 5.9: Lectura y escritura del componente SubBytes

5.2.3.2. Fin del proceso (SubBytes)

Cuando el proceso finaliza se activa la salida *flag* y permanece en el estado *salida* hasta que se desactiva la entrada *enable* y, como consecuencia, se vuelve al estado de reposo.

En la figura 5.10 podemos comprobar que, efectivamente, ocurre así.

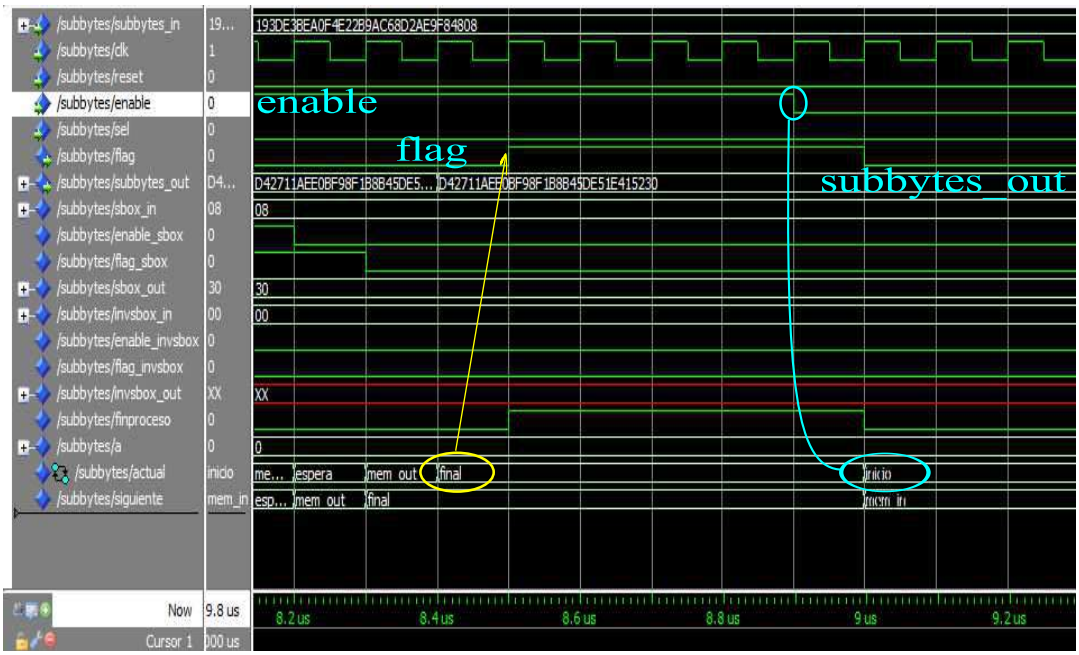


Figura 5.10: Fin del proceso del componente SubBytes

5.3. Implementación del componente ShiftRows

La operación ShiftRows/InvShiftRows consiste en una rotación cíclica hacia la izquiera/derecha de las filas de la notación matricial del Estado.

Este componente realiza tanto la operación *ShiftRows* como la operación *InvShiftRows*.

El diseño es muy sencillo ya que se trata de un bloque de lógica combinacional, es decir, de sentencias que no se rigen por el reloj.

5.3.1. Pines de entrada y salida del componente ShiftRows

En está sección se estudiarán los pines de entrada y salida (tabla 5.8) del componente *ShiftRows*.

En este componente no hay descritas señales internas.

Entradas y salidas

Nombre	**Tipo**	**E/S**	**Descripción**
shiftrows_in	std_logic_vector (127 downto 0)	E	Entrada del Estado al que aplicaremos la operación *ShiftRows*/*InvShiftRows*.
sel	std_logic	E	Indica si la operación es *ShiftRows*(0) o *InvShiftRows* (1).
shiftrows_out	std_logic_vector (127 downto 0)	S	Salida del Estado tras aplicar la operación *ShiftRows*/*InvShiftRows*.

Tabla 5.8: Entradas y salidas del componente ShiftRows

La operación descrita es, básicamente, la siguiente:

Si partimos de una cadena de bytes en la entrada *shiftrows_in* de esta forma $\{B_{15}, B_{14}, B_{13}, B_{12}, B_{11}, B_{10}, B_9, B_8, B_7, B_6, B_5, B_4, B_3, B_2, B_1, B_0\}$, dependiendo de la entrada *sel* se copiarán en la salida *shiftrows_out* de la siguiente manera:

- **sel = '0'(ShiftRows):**
  $\{B_{15}, B_{10}, B_5, B_0, B_{11}, B_6, B_1, B_{12}, B_7, B_2, B_{13}, B_8, B_3, B_{14}, B_9, B_4\}$.
- **sel = '1'(InvShiftRows):**
  $\{B_{15}, B_2, B_5, B_8, B_{11}, B_{14}, B_1, B_4, B_7, B_{10}, B_{13}, B_0, B_3, B_6, B_9, B_{12}\}$.

Para ayudar a entender el cambio de posiciones de los bytes se adjunta la figura 5.11.

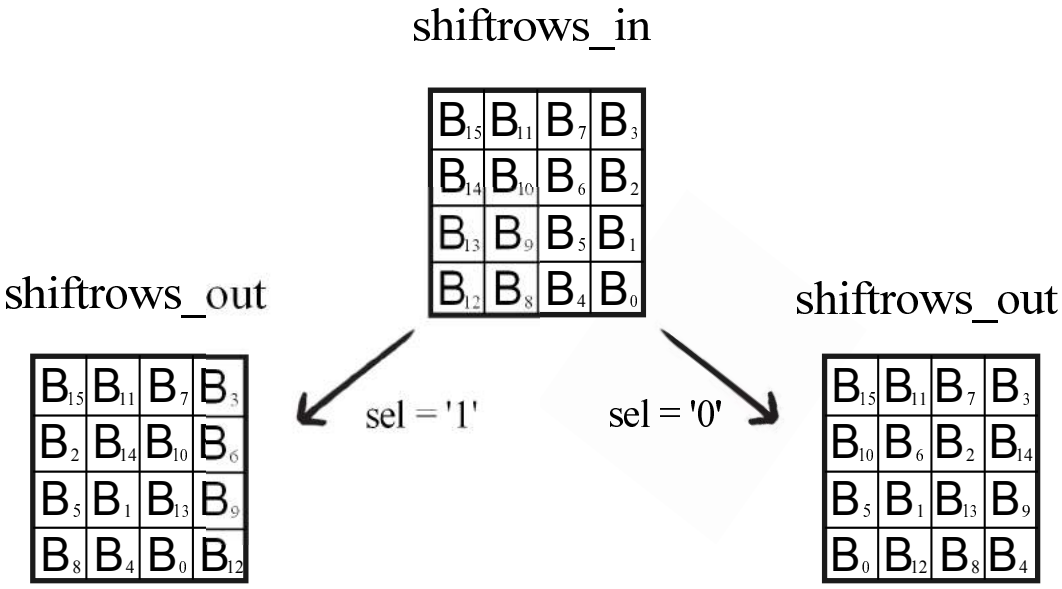


Figura 5.11: Componente ShiftRows

5.3.2. Simulaciones del componente ShiftRows

Como ya hemos explicado, el orden de los bytes resultante tiene que ser de una determinada manera dependiendo de la entrada *sel*, por lo que vamos a añadir una simulación en la que veremos este cambio.

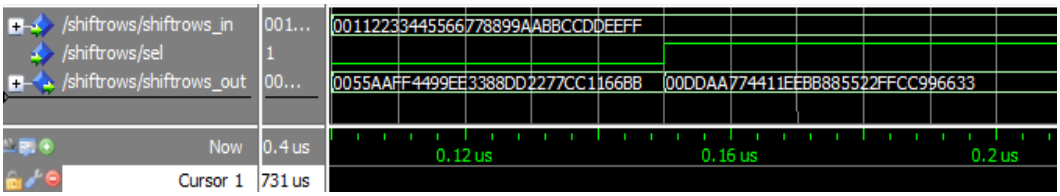


Figura 5.12: Simulación (ShiftRows)

Ante una entrada:

{00 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF}

Vemos que para *sel* = '0' la salida es:

{00 55 AA FF 44 99 EE 33 88 DD 22 77 CC 11 66 BB}

Mientras que para *sel* = '1' la salida es:

{00 DD AA 77 44 11 EE BB 88 55 22 FF CC 99 66 33}.

5.4. Implementación del componente MixColumns

Si recordamos, la operación *MixColumns*/*InvMixColumns* se podía realizar como una multiplicación de una determinada matriz a la izquierda de la notación matricial del estado.

Estas matrices son:

- **Para la operación *MixColumns*:**

$$\begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix}$$

- **Para la operación *InvMixColumns*:**

$$\begin{pmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{pmatrix}$$

Es decir, para la operación *MixColumns* tendremos una operación de la siguiente forma:

$$\begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \times \begin{pmatrix} E_{1,1} & E_{1,2} & E_{1,3} & E_{1,4} \\ E_{2,1} & E_{2,2} & E_{2,3} & E_{2,4} \\ E_{3,1} & E_{3,2} & E_{3,3} & E_{3,4} \\ E_{4,1} & E_{4,2} & E_{4,3} & E_{4,4} \end{pmatrix} = \\ = \begin{pmatrix} S_{1,1} & S_{1,2} & S_{1,3} & S_{1,4} \\ S_{2,1} & S_{2,2} & S_{2,3} & S_{2,4} \\ S_{3,1} & S_{3,2} & S_{3,3} & S_{3,4} \\ S_{4,1} & S_{4,2} & S_{4,3} & S_{4,4} \end{pmatrix}$$

Si recordamos la operación de multiplicación matricial:

$$S_{1,1} = (02 \times E_{1,1}) + (03 \times E_{2,1}) + (01 \times E_{3,1}) + (01 \times E_{4,1})$$

$$S_{2,1} = (01 \times E_{1,1}) + (02 \times E_{2,1}) + (03 \times E_{3,1}) + (01 \times E_{4,1})$$

$$S_{3,1} = (01 \times E_{1,1}) + (01 \times E_{2,1}) + (02 \times E_{3,1}) + (03 \times E_{4,1})$$

$$S_{4,1} = (03 \times E_{1,1}) + (01 \times E_{2,1}) + (01 \times E_{3,1}) + (02 \times E_{4,1})$$

$$S_{1,2} = (02 \times E_{1,2}) + (03 \times E_{2,2}) + (01 \times E_{3,2}) + (01 \times E_{4,2})$$

$$S_{2,2} = (01 \times E_{1,2}) + (02 \times E_{2,2}) + (03 \times E_{3,2}) + (01 \times E_{4,2})$$

$$S_{3,2} = (01 \times E_{1,2}) + (01 \times E_{2,2}) + (02 \times E_{3,2}) + (03 \times E_{4,2})$$

$$S_{4,2} = (03 \times E_{1,2}) + (01 \times E_{2,2}) + (01 \times E_{3,2}) + (02 \times E_{4,2})$$

$$S_{1,3} = (02 \times E_{1,3}) + (03 \times E_{2,3}) + (01 \times E_{3,3}) + (01 \times E_{4,3})$$

$$S_{2,3} = (01 \times E_{1,3}) + (02 \times E_{2,3}) + (03 \times E_{3,3}) + (01 \times E_{4,3})$$

$$S_{3,3} = (01 \times E_{1,3}) + (01 \times E_{2,3}) + (02 \times E_{3,3}) + (03 \times E_{4,3})$$

$$S_{4,3} = (03 \times E_{1,3}) + (01 \times E_{2,3}) + (01 \times E_{3,3}) + (02 \times E_{4,3})$$

$$S_{1,4} = (02 \times E_{1,4}) + (03 \times E_{2,4}) + (01 \times E_{3,4}) + (01 \times E_{4,4})$$

$$S_{2,4} = (01 \times E_{1,4}) + (02 \times E_{2,4}) + (03 \times E_{3,4}) + (01 \times E_{4,4})$$

$$S_{3,4} = (01 \times E_{1,4}) + (01 \times E_{2,4}) + (02 \times E_{3,4}) + (03 \times E_{4,4})$$

$$S_{4,4} = (03 \times E_{1,4}) + (01 \times E_{2,4}) + (01 \times E_{3,4}) + (02 \times E_{4,4})$$

Debemos recordar los tipos de operaciones matemáticas que se explicaron en el apartado 3.2.1.3, como que la suma se realiza a través de una operación XOR.
Se puede observar que, para hallar cada byte transformado, se requieren cuatro multiplicaciones y una suma (XOR) entre sus respectivos resultados. Este dato es importante para explicar el diseño propuesto.

Como se aprecia, esta operación realiza un gran numero de cálculos, por lo que es una de las más complejas y que requieren más tiempo de ejecución.

El componente descrito realiza tanto la operación *MixColumns* como la *InvMixcolumns*.

Además también contiene un subcomponente, el cual se muestra en la tabla 5.9.

Componente	**Descripción**
multiplicacion	Multiplica ocho vectores dos a dos, devolviendo cuatro vectores de 8 bits.

Tabla 5.9: Componente de MixColumns

Este componente realiza las multiplicaciones descritas, realizando además la operación módulo, ya que la multiplicación da como resultado, en la mayoría de los casos, un vector de más de 8 bits.
Además tiene capacidad para realizar cuatro multiplicaciones por cada utilización.

A continuación, en la figura 5.13, se muestra el diagrama de bloques del componente *MixColumns*.

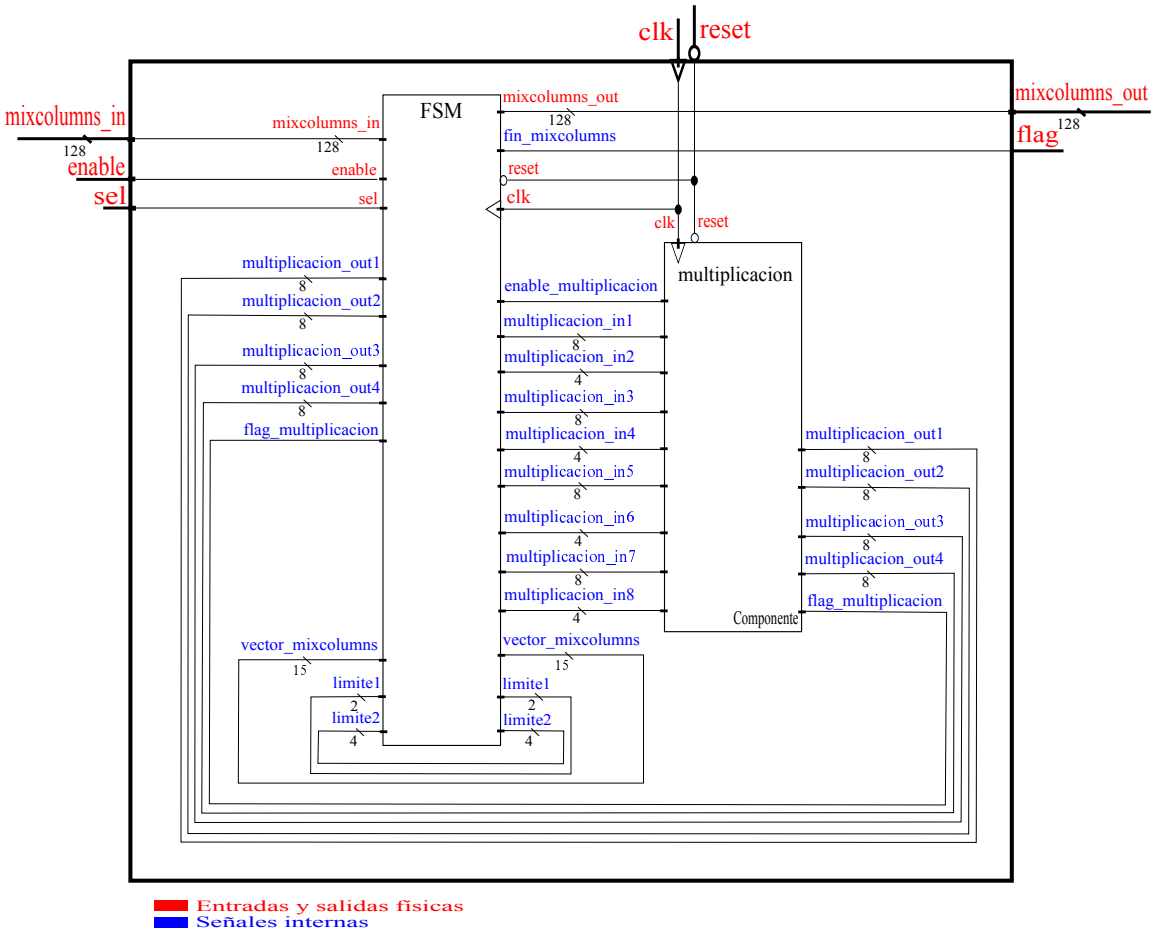


Figura 5.13: Diagrama de bloques del componente MixColumns

La función de la FSM es transformar, byte a byte, la entrada *mixcolumns_in* y depositarla, byte a byte, en la salida *mixcolumns_out*. La transformación se hace a modo de registro de desplazamiento con ayuda de las señales *limite1* y *limite2*.

Es necesario señalar que los elementos de la matriz fija por la que hay que multiplicar el Estado, ya sea la matriz de la operación *MixColumns* o *InvMixColumns*, son de esta forma: "0X" (notación hexadecimal).
Es decir, que sus cuatro bits más significativos valen cero siempre, por lo tanto es equivalente a multiplicar por un vector de 4 bits.

5.4.1. Pines de entrada y salida y señales internas del componente MixColumns

En está sección haremos una descripción los pines de entrada y salida del componente *MixColumns* (tabla 5.10), así como de sus señales internas descritas (tabla 5.11).

Entradas y salidas

Nombre	**Tipo**	**E/S**	**Descripción**
mixcolumns_in	std_logic_vector (127 downto 0)	E	Entrada del Estado al que aplicaremos la operación *MixColumns/InvMixColumns.*
sel	std_logic	E	Indica si la operación es *MixColumns*(0) o *InvMixColumns* (1).
clk	std_logic	E	Señal de reloj.
reset	std_logic	E	Señal de reset, activa a nivel alto.
enable	std_logic	E	Señal de enable, activa a nivel alto.
mixcolumns_out	std_logic_vector (127 downto 0)	S	Salida del Estado tras aplicar la operación *MixColumns/InvMixColumns.*
flag	std_logic	S	Indica que *mixcolumns_out* ofrece un dato válido.

Tabla 5.10: Entradas y salidas del componente MixColumns

Señales internas

Nombre	**Tipo**	**Descripción**
multiplicacion_in1	std_logic_vector(7 downto 0)	Multiplicación 1, factor 1.
multiplicacion_in2	std_logic_vector(3 downto 0)	Multiplicación 1, factor 2.
multiplicacion_in3	std_logic_vector(7 downto 0)	Multiplicación 2, factor 1.
multiplicacion_in4	std_logic_vector(3 downto 0)	Multiplicación 2, factor 2.
multiplicacion_in5	std_logic_vector(7 downto 0)	Multiplicación 3, factor 1.
multiplicacion_in6	std_logic_vector(3 downto 0)	Multiplicación 3, factor 2.
multiplicacion_in7	std_logic_vector(7 downto 0)	Multiplicación 4, factor 1.
multiplicacion_in8	std_logic_vector(3 downto 0)	Multiplicación 4, factor 2.
enable_multiplicacion	std_logic_vector	Permite multiplicar los factores.
flag_multiplicacion	std_logic_vector	Indica que los factores han sido multiplicados.
multiplicacion_out1	std_logic_vector(7 downto 0)	Resultado de la multiplicación 1.
multiplicacion_out2	std_logic_vector(7 downto 0)	Resultado de la multiplicación 2.
multiplicacion_out3	std_logic_vector(7 downto 0)	Resultado de la multiplicación 3.
multiplicacion_out4	std_logic_vector(7 downto 0)	Resultado de la multiplicación 4.
fin_mixcolumns	std_logic_vector	Indica que la operación ha finalizado.
limite1	std_logic_vector(1 downto 0)	Nos marca los límites dentro de *mixcolumns_in*.
limite2	std_logic_vector(3 downto 0)	Nos marca los límites dentro de *mixcolumns_out*.
vector_mixcolumns	std_logic_vector(15 downto 0)	Se fija el valor del vector de multiplicación.

Tabla 5.11: Señales internas del componente MixColumns

Básicamente, en las señales *multiplicacion_in* impares se establecen las entradas del Estado (*mixcolumns_in*) correspondientes con ayuda de la señal *limite1*, mientras que en las señales *multiplicacion_in* pares se establecen los correspondientes elementos de la matriz de multiplicación fija (de 4 bits), contenidos en la señal *vector_mixcolumns* (de 4×4 bits).

Para entender la señal *vector_mixcolumns* es necesario fijarse de nuevo en la matriz fija de multiplicación, recordando que se sitúa a la izquierda en la operación:

$$\begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix}$$

La señal *vector_mixcolumns* está inicializada en {2311}(Notación hexadecimal) en el caso de que la operación sea *MixColumns*, es decir, la primera fila de la matriz.
Si nos fijamos, cada fila es idéntica a la anterior pero añadiendo una rotación hacia la derecha.
En la máquina de estados se irá rotando la señal *vector_mixcolumns* hacia la derecha cuando corresponda y la forma de saber que una columna de la salida ha sido calculada, o lo que es lo mismo, que las cuatro multiplicaciones necesarias se han realizado, es comparando el valor de la señal *vector_mixcolumns* con el valor que le habíamos dado en un principio.

El resultado de las multiplicaciones se suma (XOR) en la máquina de estados y se establece en su lugar correspondiente dentro de *mixcolumns_out* con ayuda de la señal *limite2*.

La salida *flag* copia el valor de la señal *fin_mixcolumns*.

5.4.2. Máquinas de estado (FSM) del componente MixColumns

La función de la FSM descrita en el componente *MixColumns* es realizar la operación *MixColums* o *InvMixColumns* byte a byte, a modo de registro de desplazamiento, avisándonos de cuando ha concluído el proceso y gestionando el funcionamiento del componente *multiplicacion*.

La máquina de estados decrita está formada por cinco estados:

- **inicio:** Es el estado de reposo. En él se inicializan las señales *limite1* y *limite2* al valor necesario para realizar todo el proceso.

  También es inicializada la señal *vector_mixcolumns* en función de la entrada *sel*, es decir, {2311} para *sel* = '0'(*MixColumns*) ó {EBD9} para *sel* = '1'(*InvMixColumns*).

  No necesita estímulos para la transición de estados, salvo el que necesitan todos, es decir, *enable* = '1'.

- **Emultiplicacion:** En este estado se asignan a las señales *multiplicacion_in* impares los bytes de *mixcolumns_in* correspondientes con ayuda de la señal *limite1*.

  También se asignan a las señales *multiplicacion_in* pares los valores de la señal *vector_mixcolumns*.

  Por último, se permite al componente multiplicacion realizar su función, permaneciendo en este estado hasta que no haya concluído.

- **resultado:** En este estado se hace la suma (XOR) de los cuatro resultados de las multiplicaciones y se establece en su lugar correspondiente de la salida *mixcolumns_out* con ayuda de la señal *limite2*.

  Además también se realiza la rotación hacia la derecha de la señal *vector_mixcolumns*, dejándolo preparado para la siguiente multiplicación.

  De nuevo, no necesita estímulos para su transición.

- **limites:** En este estado se produce la actualización de las señales *limite1* y *limite2*.

  limite2 se actualiza siempre que se llegue a este estado mientras su valor sea superior a 0, mientras que *limite1* también necesita la condición de que la señal *vector_mixcolumns* haya dado una vuelta completa, es decir, que tenga el valor que se le asignó en el estado de reposo.

  Cuando *limite2* vale 0 quiere decir que la ultima posición de *mixcolumns_out* en la que se ha establecido el resultado calculado es la del

byte 0, la última. Por lo que mientras *limite2* no valga 0 volverá al estado *Emultiplicacion*.
En caso contrario irá al siguiente estado.

- **final:** En este estado se activa la señal *fin_mixcolumns*, la cual es copiada por la salida *flag*, indicándonos que el proceso ha terminado. Permanecerá en este estado indefinidamente hasta que *enable* = '0'.

Como es habitual, la transición entre todos y cada uno de los estados requiere que la entrada *enable* esté activa.

Además, como ocurría con el componente *SubBytes*, si la entrada *enable* se desactiva, el proceso no se pausa, sino que se reinicia.

Esto es así por la misma razón, la operación *MixColumns* es realizada varias veces a lo largo de todo el procesado de datos, por lo que necesitamos devolver ciertas señales a valores conocidos.
Es misión del módulo que gestiona todos los componentes (AES) el no desactivar su correspondiente *enable* hasta que el componente haya realizado su función.

En la figura 5.14 se observa de manera gráfica las transiciones entre estados descritas.

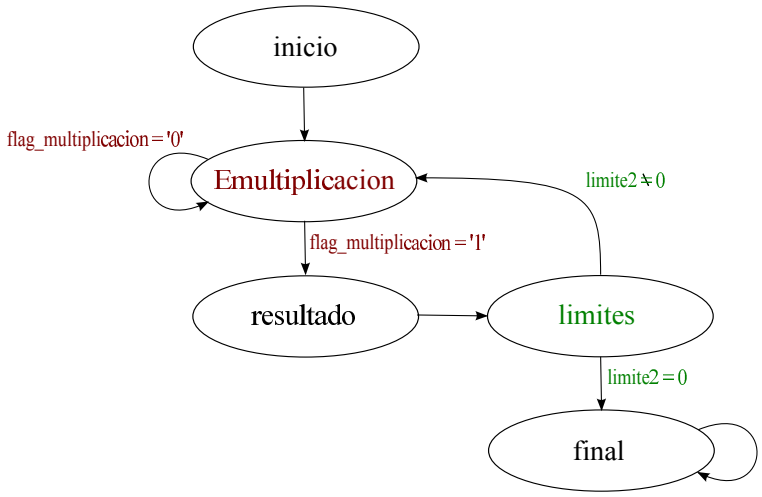


Figura 5.14: Máquina de estados del componente MixColumns

5.4.3. Simulaciones del componente MixColumns

5.4.3.1. Lectura y escritura (MixColumns)

Como se ve en la figura 5.15, la columna de bytes de *mixcolumns_in* que se copia en las señales de *multiplicacion_in* impares viene definida por el valor de la señal *limite1*.

Se pueden apreciar las cuatro multiplicaciones que se realizan fijándonos en la señal *vector_mixcolumns*, la cual va rotando a la derecha por cada multiplicación realizada.
Cuando *vector_mixcolumns* hace la rotación completa, es decir, que su valor vuelve a ser el de inicio, la señal *limite1* actualiza su valor, recogiendo la siguiente columna de bytes de la entrada *mixcolumns_in*.

En cuanto a la escritura en la salida *mixcolumns_out*, se aprecia que la posición donde se escribe el byte calculado es establecido por la señal *limite2*, la cual actualiza su valor cada vez que se realiza una multiplicación.
El valor que se almacena es el resultado de una operación XOR entre las cuatro señales de *multiplicacion_out*.

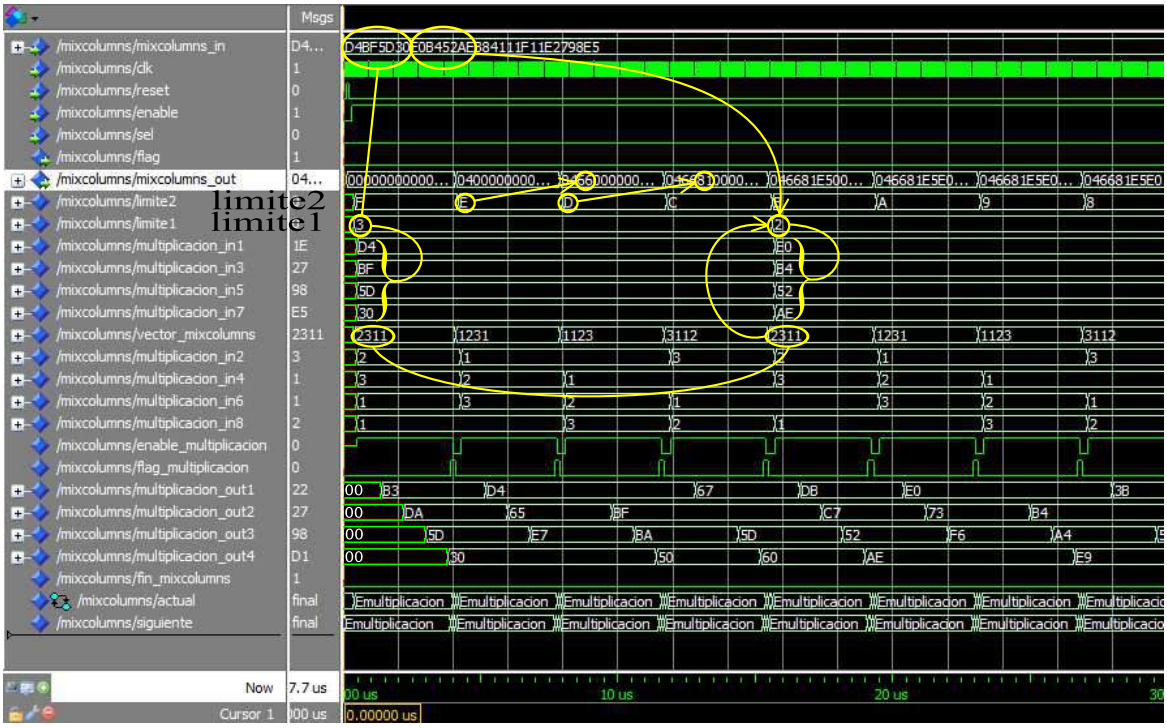


Figura 5.15: Lectura y escritura del componente MixColumns

5.4.3.2. Fin del proceso (MixColumns)

Cuando la señal *limite2* llega a valor 0 quiere decir que el último resultado que se calcula se copia en la última posición por ocupar de la salida *mixcolumns_out*, por lo que, después de copiarlo, se activa la señal *fin_mixcolumns*, cuyo valor es copiado en todo momento por la salida *flag*, indicándonos que el valor contenido en *mixcolumns_out* es válido.

Esto es apreciado en la figura 5.16. Además se permanecerá en el estado *final* hasta que el *enable* sea desactivado.

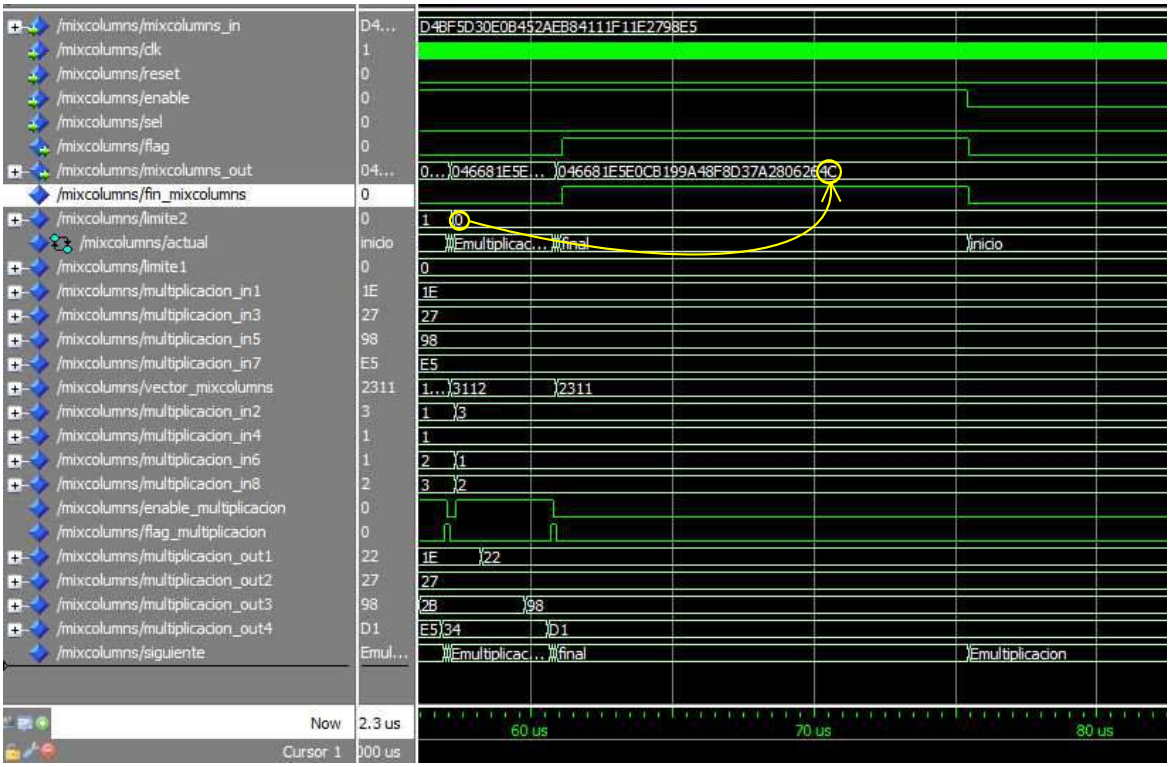


Figura 5.16: Fin del proceso del componente MixColumns

5.5. Implementación del componente Multiplicación (MixColumns)

El componente *Multiplicación* incluido en *MixColumns* tiene la función de realizar cuatro multiplicaciones entre dos vectores, uno de 8 bits y otro

de 4, devolviendo sus cuatro resultados, de 8 bits.

La multiplicación se realiza a través de un proceso combinacional. Sin embargo este proceso no nos proporcionará, en la gran mayoría de los casos, el resultado buscado, ya que éste será de más de 8 bits.

El peor caso se dará cuando los bits más significativos de los vectores a multiplicar tengan valor '1', sin importar el valor de los demás bits.

Por ejemplo, el peor caso se dará cuando:

- $factor\ 1 = \{1000\ 0000\} = x^7$
- $factor\ 2 = \{1000\} = x^3$

Multiplicando ambos factores:

$$x^7 \times x^3 = x^{10} = 100\ 0000\ 0000$$

Es decir, un vector de 11 bits.
Es por ello que *Multiplicación* consta de un subcomponente, el cual aparece en la tabla 5.12.

Componente	**Descripción**
modulo	Recoge un vector de 11 bits y le aplica la operación módulo, devolviendo uno de 8 bits.

Tabla 5.12: Componente de Multiplicación

El proceso consiste básicamente en pasar por el componente *Módulo* los resultados de las multiplicaciones uno a uno para obtener su módulo. Como siempre, este componente está gestionado por una máquina de estados.

A continuación, en la figura 5.17, se muestra el diagrama de bloques del componente *Multiplicación.*

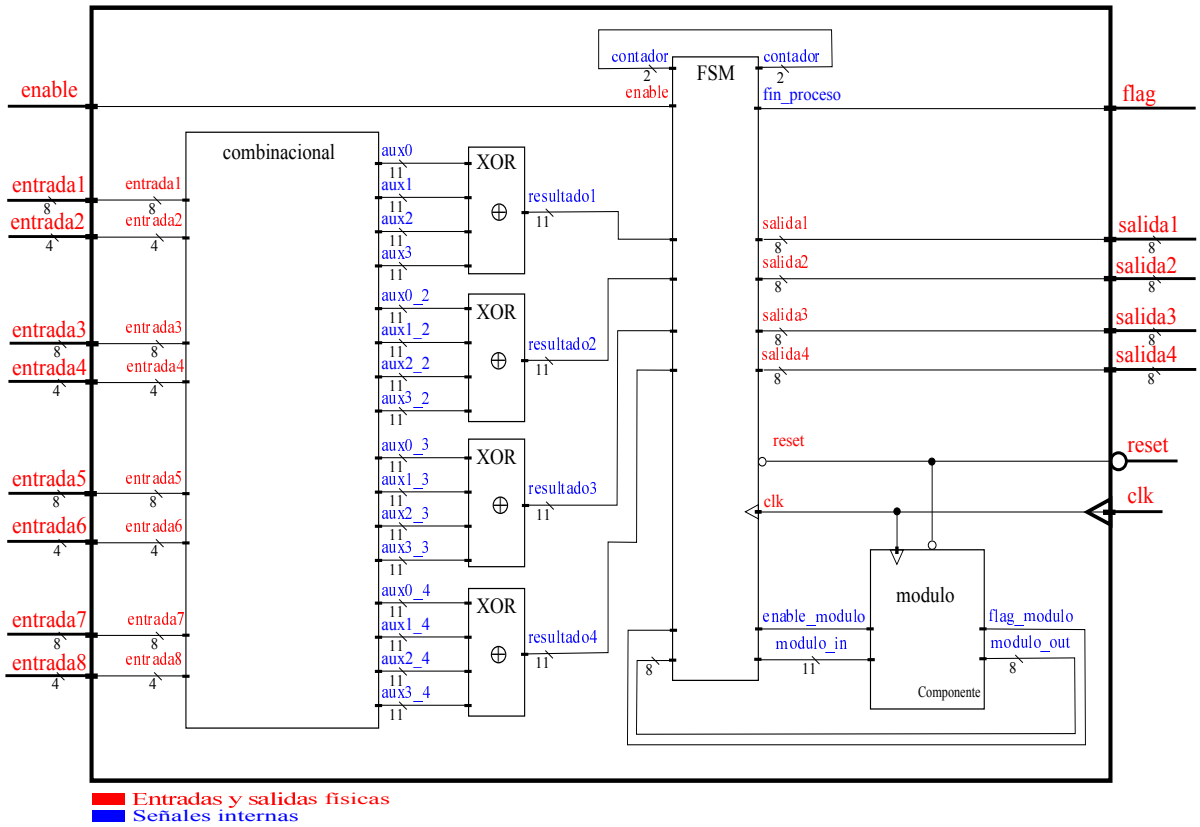
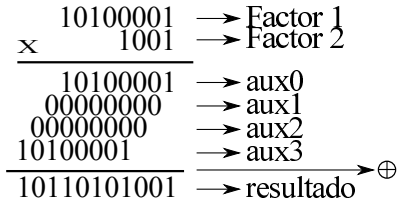


Figura 5.17: Diagrama de bloques del componente Multiplicación

No existe ningún operador directo en VHDL que nos proporcione los resultados de la multiplicación contenida en el bloque combinacional, ya que la suma de bits es con una operación XOR.
Para entender la función que realiza el bloque combinacional vamos a proponer un ejemplo:

Partiendo de:

- $factor\ 1 = \{1010\ 0001\}$
- $factor\ 2 = \{1001\}$



Como vemos, la multiplicación se realiza bit a bit. Si el bit del factor 2 vale uno, se copia el factor 1 en una variable auxiliar, mientras que si vale cero, la variable auxiliar toma el valor cero.
Todas las variables auxiliares necesarias para la multiplicación son de 11 bits y, dependiendo del bit del factor 2 que estemos multiplicando, el factor 1 se copiará en una posición específica dentro del auxiliar, rellenando el resto de ceros (ya que los ceros no afectan a la operación XOR).

Finalmente, a través de bloques de lógica adicional, se realiza la operación XOR entre todas las variables auxiliares correspondientes, obteniendo un vector de 11 bits que mandaremos a la FSM.

5.5.1. Pines de entrada y salida y señales internas del componente Multiplicación

En está sección haremos una descripción los pines de entrada y salida del componente *Multiplicación* (tabla 5.13), así como de sus señales internas descritas (tabla 5.14).

Entradas y salidas

Nombre	**Tipo**	**E/S**	**Descripción**
entrada1	std_logic_vector (7 downto 0)	E	Multiplicación 1, factor 1.
entrada2	std_logic_vector (3 downto 0)	E	Multiplicación 1, factor 2.
entrada3	std_logic_vector (7 downto 0)	E	Multiplicación 2, factor 1.
entrada4	std_logic_vector (3 downto 0)	E	Multiplicación 2, factor 2.
entrada5	std_logic_vector (7 downto 0)	E	Multiplicación 3, factor 1.
entrada6	std_logic_vector (3 downto 0)	E	Multiplicación 3, factor 2.
entrada7	std_logic_vector (7 downto 0)	E	Multiplicación 4, factor 1.
entrada8	std_logic_vector (3 downto 0)	E	Multiplicación 4, factor 2.
clk	std_logic	E	Señal de reloj.
reset	std_logic	E	Señal de reset, activa a nivel alto.
enable	std_logic	E	Señal de enable, activa a nivel alto.
salida1	std_logic_vector (7 downto 0)	S	Resultado de la multiplicación 1 después de módulo.
salida2	std_logic_vector (7 downto 0)	S	Resultado de la multiplicación 2 después de módulo.
salida3	std_logic_vector (7 downto 0)	S	Resultado de la multiplicación 3 después de módulo.
salida4	std_logic_vector (7 downto 0)	S	Resultado de la multiplicación 4 después de módulo.
flag	std_logic_vector	S	Indica que las salidas son correctas.

Tabla 5.13: Entradas y salidas del componente Multiplicación

Señales internas

Nombre	**Tipo**	**Descripción**
modulo_in	std_logic_vector(10 downto 0)	Entrada al componente Módulo
enable_modulo	std_logic_vector	Permite realizar el módulo a la entrada.
flag_modulo	std_logic_vector	El módulo ha sido realizado.
modulo_out	std_logic_vector(7 downto 0)	Resultado de la operación módulo.
aux0	std_logic_vector(10 downto 0)	Señales
aux1	std_logic_vector(10 downto 0)	auxiliares
aux2	std_logic_vector(10 downto 0)	para la
aux3	std_logic_vector(10 downto 0)	multiplicación 1.
aux0_2	std_logic_vector(10 downto 0)	Señales
aux1_2	std_logic_vector(10 downto 0)	auxiliares
aux2_2	std_logic_vector(10 downto 0)	para la
aux3_2	std_logic_vector(10 downto 0)	multiplicación 2.
aux0_3	std_logic_vector(10 downto 0)	Señales
aux1_3	std_logic_vector(10 downto 0)	auxiliares
aux2_3	std_logic_vector(10 downto 0)	para la
aux3_3	std_logic_vector(10 downto 0)	multiplicación 3.
aux0_4	std_logic_vector(10 downto 0)	Señales
aux1_4	std_logic_vector(10 downto 0)	auxiliares
aux2_4	std_logic_vector(10 downto 0)	para la
aux3_4	std_logic_vector(10 downto 0)	multiplicación 4.
resultado1	std_logic_vector(10 downto 0)	Producto de la multiplicación 1.
resultado2	std_logic_vector(10 downto 0)	Producto de la multiplicación 2.
resultado3	std_logic_vector(10 downto 0)	Producto de la multiplicación 3.
resultado4	std_logic_vector(10 downto 0)	Producto de la multiplicación 4.
fin_proceso	std_logic_vector	Indica que la operación ha finalizado.
contador	std_logic_vector(1 downto 0)	Nos indica a que resultado hay que aplicar la operación módulo.

Tabla 5.14: Señales internas del componente Multiplicación

5.5.2. Máquinas de estado (FSM) del componente Multiplicación

La función de la FSM descrita es gestionar el uso del componente *Módulo* para realizar la operación módulo a los resultados de las multiplicaciones uno a uno (señales *resultado1*, *resultado2*, *resultado3* y *resultado4*).

La máquina de estados decrita está formada por siete estados:

- **inicio:** Es el estado de reposo. En el se inicializa la señal *contador* a valor cero. No necesita estímulos para la transición de estados, únicamente que la entrada *enable* = '1'.
- **modulo1:** El resultado de la multiplicación 1 es copiado en la entrada del componente *Módulo*, permaneciendo en este estado hasta que no haya completado su función (flag_modulo = '1').
- **modulo2:** El resultado de la multiplicación 2 es copiado en la entrada del componente *Módulo*, permaneciendo en este estado hasta que no haya completado su función (flag_modulo = '1').
- **modulo3:** El resultado de la multiplicación 3 es copiado en la entrada del componente *Módulo*, permaneciendo en este estado hasta que no haya completado su función (flag_modulo = '1').
- **modulo4:** El resultado de la multiplicación 4 es copiado en la entrada del componente *Módulo*, permaneciendo en este estado hasta que no haya completado su función (flag_modulo = '1').
- **escritura:** En este estado se escribe la salida del componente Módulo (*modulo_out*) en la correspondiente salida de nuestro circuito (*salida1*, *salida2*, *salida3* o *salida4*) con ayuda de la señal *contador*.

  También se produce la actualización de la señal *contador*.
- **final:** A este estado se llega cuando todas las salidas del circuito tienen los valores que deseamos. Su única función, por tanto, es activar la señal *fin_proceso*, que es copiada por la salida *flag* mediante un bloque de lógica adicional, indicándonos que el proceso ha concluido.

Como es habitual, la transición entre todos los estados requiere que la entrada *enable* esté activa, volviendo al estado *inicio* en caso contrario.

Para explicar de una forma más precisa la transición entre estados, se adjunta la figura 5.18

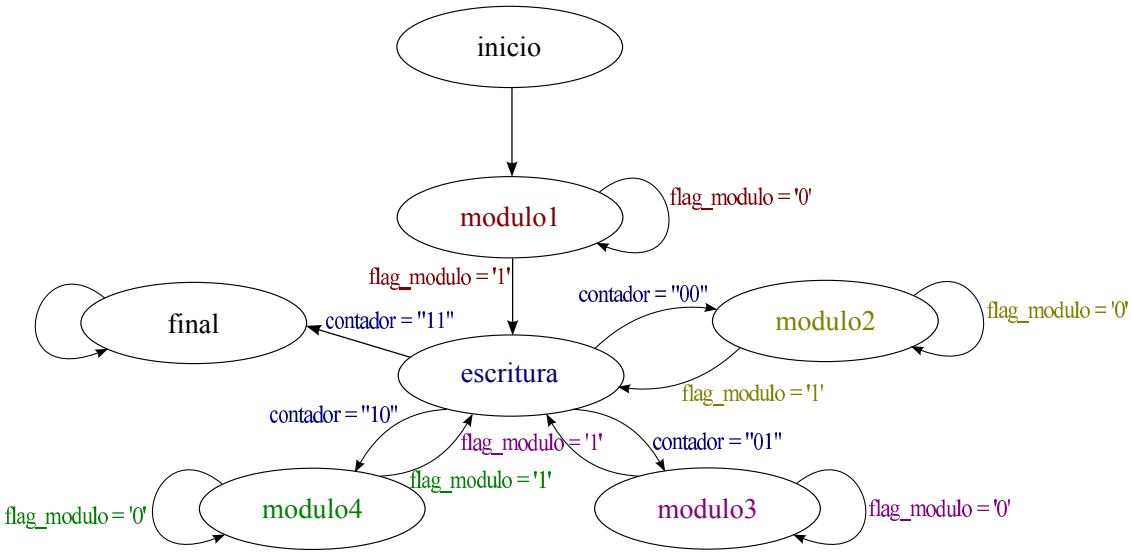


Figura 5.18: Máquina de estados del componente Multiplicación

5.5.3. Simulaciones del componente Multiplicación

5.5.3.1. Lectura y escritura (Multiplicación)

Como se aprecia en la figura 5.19, las multiplicaciones se realizan sin necesidad de esperar al reloj, ya que se trata de un proceso combinacional.

Cuando se activa la señal *enable*, el resultado da la primera multiplicación, *resultado1*, es copiado en la entrada del componente *Módulo*, *modulo_in*, y, cuando éste termina su función, copia su salida, *modulo_out*, a la salida *salida1* para, posteriormente, comenzar el procesado del resultado de la segunda multiplicación, *resultado2*.

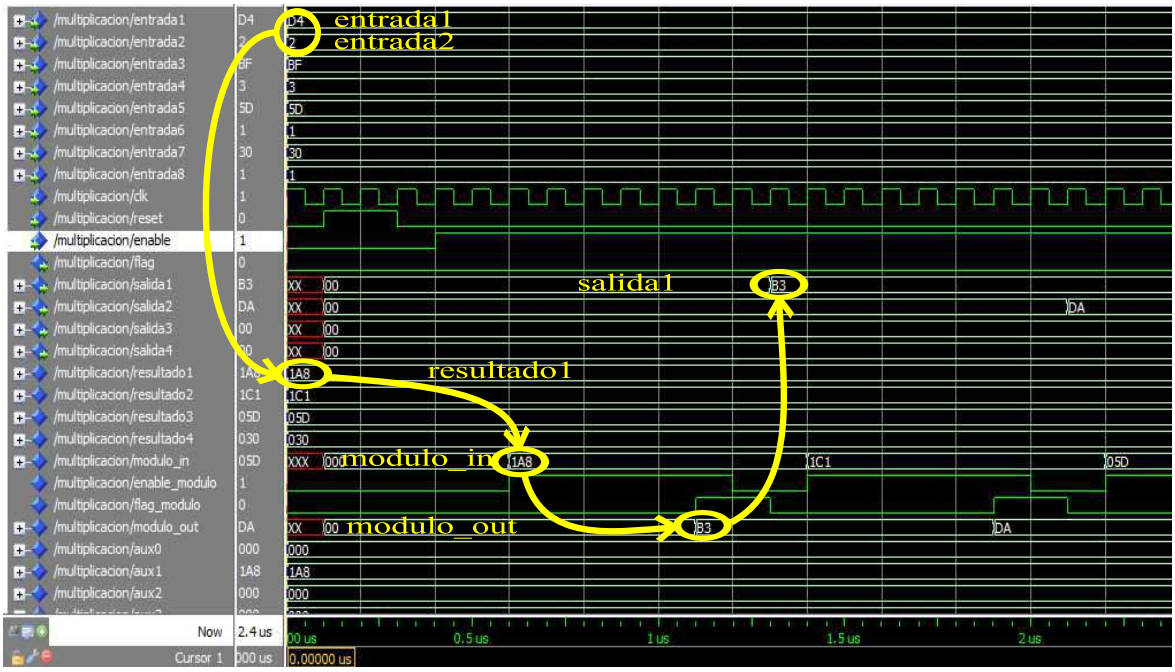


Figura 5.19: Lectura y escritura del componente Multiplicación

5.5.3.2. Fin del proceso (Multiplicación)

Cuando las cuatro salidas (*salida1*, *salida2*, *salida3* y *salida4*) han sido establecidas con sus correspondientes valores se produce la transición al estado *final*, donde se activa la señal *fin_proceso*, copiada por la salida *flag*, indicándonos que el proceso ha finalizado.

Esto se aprecia en la figura 5.20. Además se permanecerá en el estado *final* hasta que el *enable* sea desactivado.

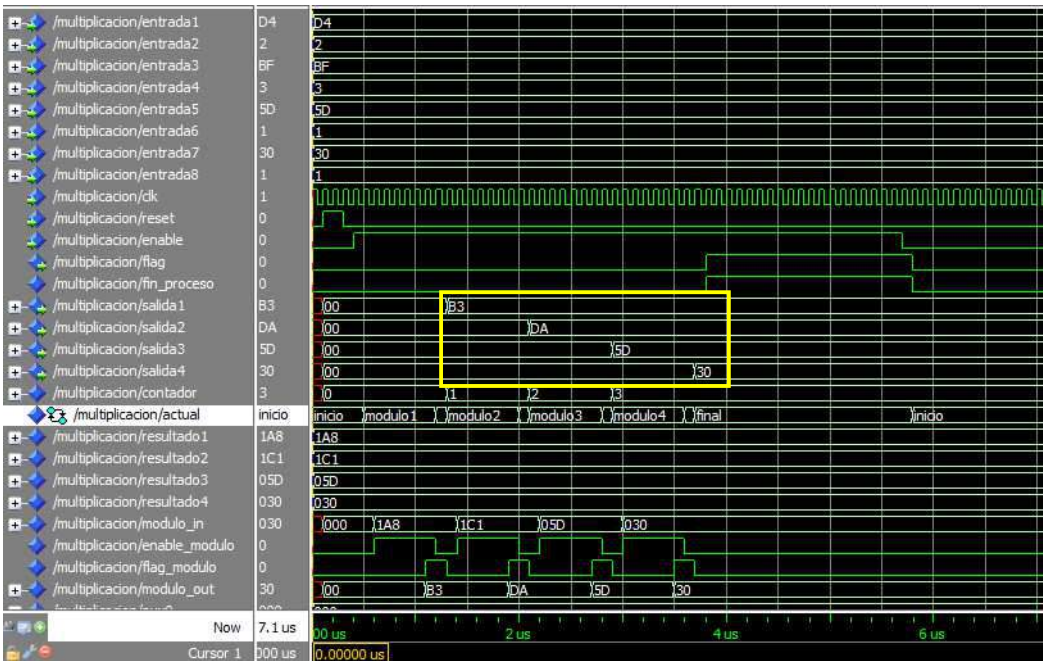


Figura 5.20: Fin del proceso del componente Multiplicación

5.6. Implementación del componente Módulo (Multiplicación)

La unidad básica de trabajo del algoritmo AES es el byte.

Sin embargo, como hemos visto, tras una multiplicación entre un vector de 8 bits y uno de 4 bits, se puede obtener un vector de hasta 11 bits.

Es por ello que es necesario realizar la operación módulo sobre ese vector, reduciéndolo de nuevo a uno de longitud 8 bits.

El componente *Módulo* incluido en *Multiplicación* tiene la función de realizar la operación módulo del vector de entrada con el polinomio irreductible de AES.

Este polinomio era de la siguiente forma:

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

Como podemos ver, se trata de un vector de 9 bits.

Para explicar la operación módulo de una manera sencilla, básicamente consiste en realizar una operación XOR entre los 9 bits más significativos de nuestro vector (siempre y cuando el más significativo sea 1) y el polinomio irreductible de AES tantas veces como sean necesarias para que la longitud final de nuestro vector sea 8 bits.

Por ejemplo, si partimos de este vector de 11 bits:

- {101 1010 0001}

Su módulo con el polinomio de AES será:

$$\overbrace{1\,0110\,1000}^{9\,bits\,mas\,significativos} \oplus \overbrace{1\,0001\,1011}^{Polinomio\,de\,AES} = 0\,0111\,0011 = 111\,0011$$

A este resultado se le concatenan los bits de nuestro vector que no se han incluido en la operación XOR, es decir:

$$111\,0011\,\&\,01 = 1\,1100\,1101$$

Como vemos, hemos obtenido un vector de 9 bits, por lo que hay que realizar de nuevo el mismo proceso.

$$\overbrace{1\,1100\,1101}^{9\,bits\,mas\,significativos} \oplus \overbrace{1\,0001\,1011}^{Polinomio\,de\,AES} = 0\,1101\,0110 = 1101\,0110$$

Ya hemos obtenido un vector de 8 bits, por lo tanto podemos establecer que:

$$101\,1010\,0001\ modulo\ 1\,0001\,1011 = 1101\,0110$$

El diagrama de bloques mostrado en la figura 5.21 muestra los bloques de los que se compone el circuito.

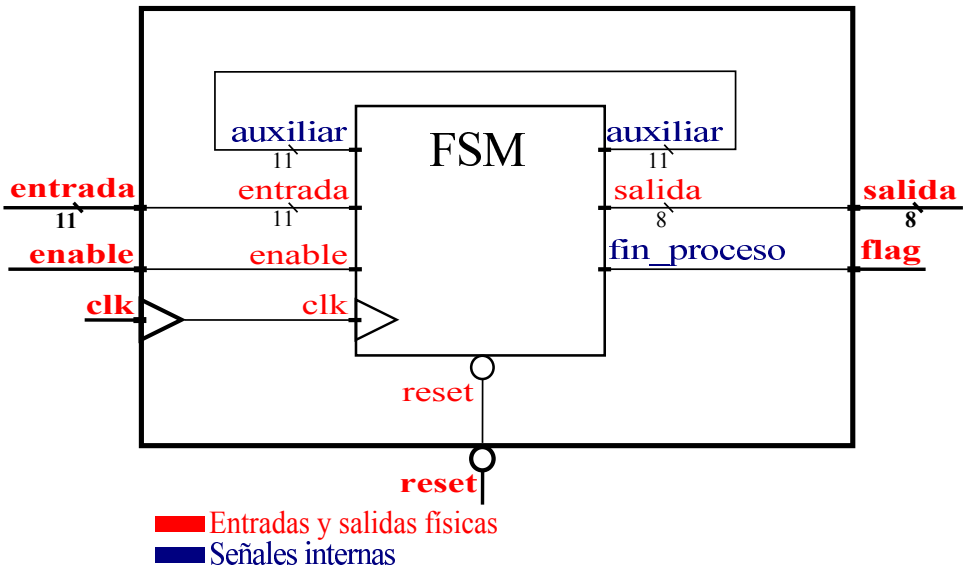


Figura 5.21: Diagrama de bloques del componente Módulo

Como vemos, se trata de un diseño muy simple que únicamente consta de una máquina de estados, aparte de un bloque de lógica adicional en el que la salida *flag* copia el valor de la señal *fin_proceso*.

5.6.1. Pines de entrada y salida y señales internas del componente Módulo

En esta sección se describen los pines de entrada y de salida del componente *Módulo* (tabla 5.15) y sus respectivas señales internas (tabla 5.16).

Entradas y salidas

Nombre	Tipo	E/S	Descripción
entrada	std_logic_vector (10 downto 0)	E	Entrada de 11 bits a la que realizaremos el módulo.
clk	std_logic	E	Señal de reloj.
reset	std_logic	E	Señal de reset, activa a nivel alto.
enable	std_logic	E	Señal de enable, activa a nivel alto.
salida	std_logic_vector (7 downto 0)	S	Salida del byte tras la operación módulo.
flag	std_logic_vector	S	Indica que la salida es correcta.

Tabla 5.15: Entradas y salidas del componente Módulo

Señales internas

Nombre	Tipo	Descripción
auxiliar	std_logic_vector(10 downto 0)	Se almacenarán los cambios en el vector.
fin_proceso	std_logic_vector	Indica que el proceso ha finalizado.

Tabla 5.16: Señales internas del componente Módulo

5.6.2. Máquinas de estado (FSM) del componente Módulo

La función de la máquina de estados descrita es realizar la operación módulo de la entrada *entrada* con el polinomio de AES, obteniendo un vector de 8 bits que será almacenado en la salida *salida*.

Se compone de cinco estados:

- **inicio:** Es el estado de reposo. En él se copia el valor de la entrada *entrada* a la señal *auxiliar*.

No necesita estímulos para la transición de estados.

- **modbit10:** En este estado se comprueba si el bit 11 de *auxiliar* vale uno o cero. Si vale uno se realiza la operación XOR con el polinomio de AES (añadiendo dos ceros al final, ya que el polinomio de AES es de 9 bits), sobreescribiendo su nuevo valor en la señal *auxiliar*. Si vale cero no se realiza ninguna operación.

  No necesita estímulos para la transición de estados.

- **modbit9:** En este estado se comprueba si el bit 10 de *auxiliar* vale uno o cero. Si vale uno se realiza la operación XOR con el polinomio de AES (añadiendo un cero al principio y un cero al final), sobreescribiendo su nuevo valor en la señal *auxiliar*. Si vale cero no se realiza ninguna operación.

  No necesita estímulos para la transición de estados.

- **modbit9:** En este estado se comprueba si el bit 9 de *auxiliar* vale uno o cero. Si vale uno se realiza la operación XOR con el polinomio de AES (añadiendo dos ceros al principio), sobreescribiendo su nuevo valor en la señal *auxiliar*. Si vale cero no se realiza ninguna operación.

  Llegados a este punto ya habremos obtenido un vector de 8 bits.

  No necesita estímulos para la transición de estados.

- **final:** En este estado los 8 bits menos significativos de la señal *auxiliar* (el resto valen cero) son copiados en la salida *salida*.

  Además se activa la señal *fin_proceso*, indicando que el proceso ha terminado.

Todos los estados necesitan que la señal *enable* esté activa para realizar la transición entre ellos. Si no ocurriera así se volvería al estado *inicio*.

Como vemos en la figura 5.22 son transiciones directas que únicamente requieren que la entra *enable* esté activa.

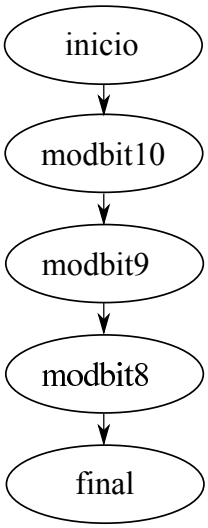


Figura 5.22: Máquina de estados del componente Módulo

5.6.3. Simulaciones del componente Módulo

Para comprobar la funcionalidad de este componente vamos a realizar la simulación del mismo ejemplo que hemos visto en esta sección:

$$101\ 1010\ 0001\ \textit{modulo}\ 1\ 0001\ 1011 = 1101\ 0110$$

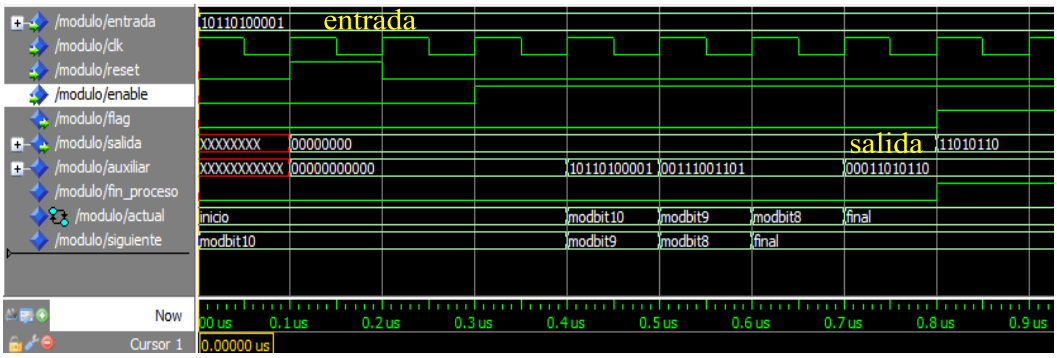


Figura 5.23: Simulación del componente Módulo

Como vemos en la figura 5.23, da el resultado esperado.

Si nos fijamos, en el estado *modbit10*, al tener el bit 11 de la señal *auxiliar* valor uno, se realiza la operación XOR, sin embargo, en el estado *modbit9*, al tener el bit 10 de la señal *auxiliar* valor cero, *auxiliar* conserva su valor pasando al siguiente estado, *modbit8*, donde se vuelve a realizar la operación XOR por tener el bit 9 valor uno.

Cuando la salida tiene el valor deseado (8 bits menos significativos de la señal *auxiliar*) se activa el pin de salida*flag*.

5.7. Implementación del componente AddRoundKey

La operación AddRoundkey consiste en la combinación entre el Estado y la correspondiente subclave de ronda a través de la operación XOR.

Se trata, por tanto, de uno de los bloques más sencillos (figura 5.24).

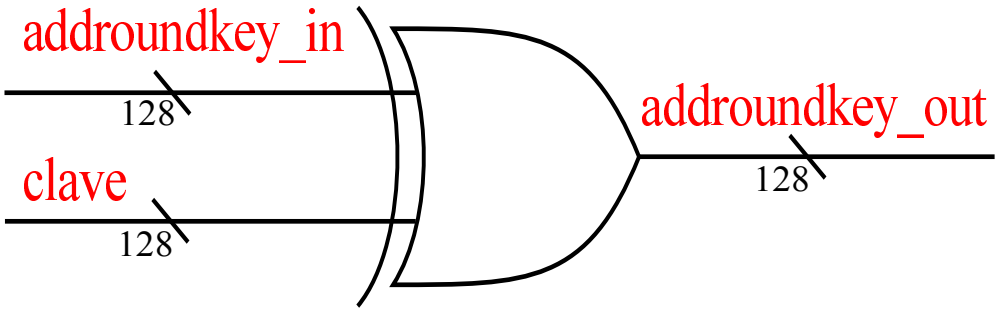


Figura 5.24: Diagrama de bloques del componente AddRoundKey

Como vemos, se trata de un simple bloque de lógica combinacional.

5.7.1. Pines de entrada y salida del componente AddRoundKey

En esta sección, en la tabla 5.17, se detallan los pines de entrada y de salida del componente *AddRoundKey*.

Nombre	**Tipo**	**E/S**	**Descripción**
addroundkey_in	std_logic_vector (127 downto 0)	E	Entrada del Estado.
clave	std_logic_vector (127 downto 0)	E	Entrada de la subclave de ronda.
addroundkey_out	std_logic_vector (127 downto 0)	S	Salida del Estado tras la operación.

Tabla 5.17: Entradas y salidas del componente AddRoundKey

5.8. Implementación del componente Keygen

El componente keygen es el encargado de calcular todas las subclaves de ronda y proporcionarnos las mismas en función del número de ronda en el que nos encontremos.

Este bloque es de los más complejos junto al bloque *MixColumns*.
Como hemos visto, el cálculo de subclaves requiere la operación *SubBytes* en ciertos momentos, por lo que nuestro componente *keygen* tiene como subcomponente el que describe la operación *SubBytes*.

El diagrama de bloques del componente *Keygen* se muestra en la figura 5.25.

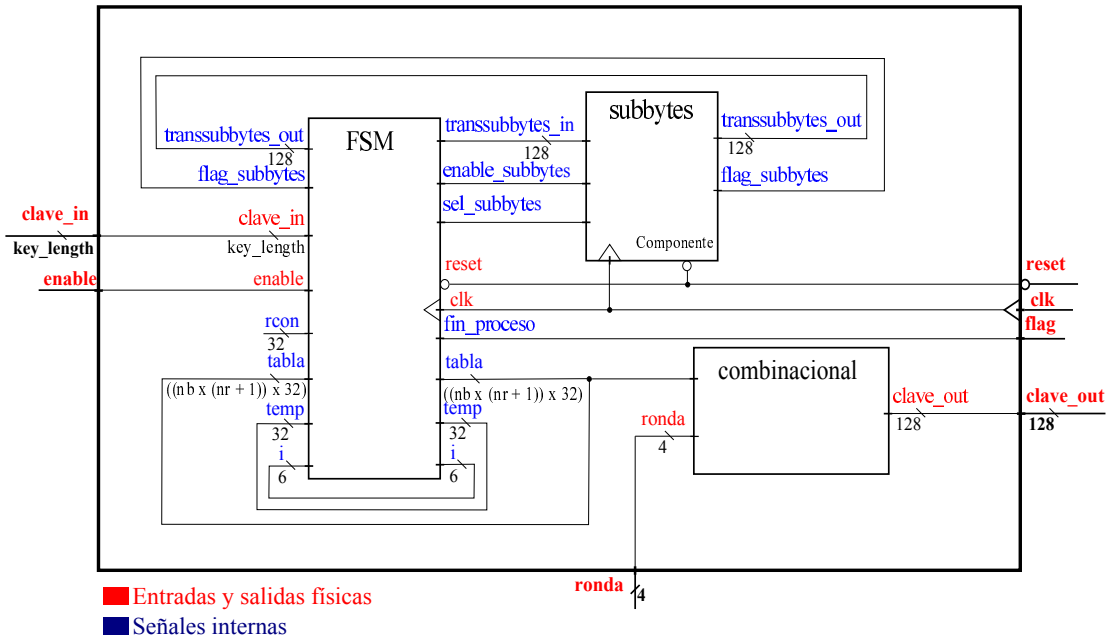


Figura 5.25: Diagrama de bloques del componente Keygen

La máquina de estados (FSM) se encarga de calcular la matriz de clave expandida (señal *tabla*), gestionando todas las señales y el componente *SubBytes*, activando la salida *flag* cuando este proceso ha concluído.

A partir de ese momento es cuando el bloque combinacional podrá realizar de una manera correcta su función, que es escribir en la salida *clave_out* la correspondiente subclave de ronda. Es tarea del circuito de AES no pedir las subclaves de ronda hasta que este componente activa su flag.

5.8.1. Genéricos, pines de entrada y salida y señales internas del componente Keygen

A continuación haremos una descripción de todos los puertos genéricos (tabla 5.18), los pines de entrada y salida (tabla 5.19) y las señales internas del circuito (tabla 5.20).

Puertos genéricos

Nombre	Tipo	Descripción
key_length	integer	Longitud en bits de la clave.
nb	integer	Número de longitud de palabras (32 bits) del estado.
nk	integer	Número de longitud de palabras (32 bits) de la clave.
nr	integer	Número de rondas.

Tabla 5.18: Puertos genéricos del componente Keygen

De nuevo, es imprescindible el uso de genéricos para poder implementar el algoritmo AES de forma que se puedan utilizar claves de 128, 192 ó 256 bits (key_length).

En el estándar el valor del genérico Nb es siempre 4.

Como ya se ha explicado, Nk varía entre 4, 6 y 8.

Finalmente, Nr varía entre 10, 12 y 14.

Entonces, si queremos configurar este circuito para que genere las subclaves de una clave de 128 bits, quedaría de la siguiente manera:

$$key\_length = 128$$

$$Nb = 4$$

$$Nk = 4$$

$$Nr = 10$$

.

Si nos fijamos en el diagrama de bloques (figura 5.25) La señal *tabla* tiene una longitud de $[(Nb \times (Nr + 1) \times 32]$ bits. Ya se ha comentado que la señal *tabla* es la matriz de clave expandida.

Recordamos que, para una longitud de clave de 128 bits, esta matriz tenía dimensiones de cuatro filas por 44 columnas (1 columna = 32 bits).

En este caso $Nr = 10$, por lo que:

$$[(Nb \times (Nr + 1) \times 32] = [(4 \times 11) \times 32] = 44 \times 32$$

Es decir, 44 columnas de 32 bits cada una.

Entradas y salidas

Nombre	Tipo	E/S	Descripción
clave_in	std_logic_vector (key_length-1 downto 0)	E	Clave original.
ronda	std_logic_vector (3 downto 0)	E	Número de ronda de la que necesitamos la subclave.
clk	std_logic	E	Señal de reloj.
reset	std_logic	E	Señal de reset, activa a nivel alto.
enable	std_logic	E	Señal de enable, activa a nivel alto.
clave_out	std_logic_vector (127 downto 0)	S	Subclave de ronda.
flag	std_logic_vector	S	Indica que todas las subclaves han sido calculadas.

Tabla 5.19: Entradas y salidas del componente Keygen

Señales internas

Nombre	Tipo	Descripción
transsubbytes_in	std_logic_vector (127 downto 0)	Señales de
enable_subbytes	std_logic	correspondencia
sel_subbytes	std_logic	de puertos del
flag_subbytes	std_logic	componente
subbytes_out	std_logic_vector (127 downto 0)	**subbytes**.
tabla	std_logic_vector $(((Nb \times (Nr+1) \times 32)) - 1$ downto 0)	Se almacenará la matriz de clave expandida.
i	std_logic_vector (5 downto 0)	Marcador de columna de clave expandida.
temp	std_logic_vector (31 downto 0)	Se almacenarán cambios en columnas.
rcon	tRcon (1 to 10)	Constante que contiene la matriz Rcon.
fin_proceso	std_logic_vector	Indica que el proceso ha finalizado.

Tabla 5.20: Señales internas del componente Keygen

El tipo *tRcon* es una constante en la que se encuentran descritos 10 vectores de 32 bits, que son la matriz Rcon.

Si recordamos, la matriz Rcon era la siguiente:

$$Rcon = \begin{pmatrix} 01 & 02 & 04 & 08 & 10 & 20 & 40 & 80 & 1B & 36 \\ 00 & 00 & 00 & 00 & 00 & 00 & 00 & 00 & 00 & 00 \\ 00 & 00 & 00 & 00 & 00 & 00 & 00 & 00 & 00 & 00 \\ 00 & 00 & 00 & 00 & 00 & 00 & 00 & 00 & 00 & 00 \end{pmatrix}$$

Por lo tanto, si seleccionamos $Rcon(9)$, por ejemplo, obtendremos el vector {1B000000}.

5.8.2. Máquinas de estado (FSM) del componente Keygen

Como ya se ha dicho, la función de la máquina de estados es recoger la clave original y calcular su matriz de clave expandida.

La máquina de estados decrita está formada por diez estados:

- **inicio:** Es el estado de reposo. Su función es esperar a la señal de "enable" y no necesita estímulos para cambiar de estado, aparte del que necesitan todos (*enable* = '1').
- **estado1:** En este estado se copia la clave original en las Nk columnas menos significativas, ayudados por la señal i, que nos indica en que columna nos encontramos. Cuando $i = Nk - 1$ se produce la transición al siguiente estado.
- **estado2:** En este estado se copia a la señal *temp* la última columna calculada, es decir, la columna $i - 1$.

  Aquí hay varias opciones de transición:

  1. Si el resto de $i/Nk = 0$, quiere decir que estamos calculando la primera columna del nuevo grupo de Nk columnas, por lo que pasaremos al estado correspondiente.
  2. Si el resto de $i/Nk = 4$ (para claves de longitud 256 bits), quiere decir que estamos calculando la quinta columna del nuevo grupo de Nk columnas, por lo que se pasará al estado que le corresponda.

3. Si el resto de $i/Nk \neq 0$ en general y además $i/Nk \neq 4$ para claves de 256 bits, quiere decir que estamos calculando el resto de columnas del nuevo grupo de Nk columnas, por lo que pasaremos al estado correspondiente.

- **estado3:** En este estado se produce la rotación hacia arriba de la columna contenida en la señal *temp*.

  No necesita condiciones para la transición.

- **estado4:** A la columna rotada contenida en *temp* se le aplica la operación *SubBytes*.

  Permanece en este estado hasta que la operación ha finalizado, lo cual es indicado por la señal *flag_subbytes*.

- **estado5:** Se hace la operación XOR entre el resultado de la operación SubBytes y Rcon(i/nk) (recordemos que en este punto esa división es exacta), almacenando el resultado en la señal *temp*.

  No precisa de condiciones para la transición de estados.

- **estado6:** Se realiza la operación XOR entre la señal *temp* y la columna $i - Nk$, almacenando el resultado en la señal *tabla*, en la posición que nos indica i.

  En este estado se comprueba el valor de la señal i, si ha llegado al máximo (44 para longitud de clave 128 bits) quiere decir que todas las columnas han sido calculadas, por lo que se pasará al estado *final*. En caso contrario se pasará al estado correspondiente.

- **estado7:** Este estado sólo se alcanza si la longitud de clave (*key_length*) es de 256 bits. En él se realiza la operación *SubBytes* de la señal *temp*. Se trata de la operación extra que requerían las claves de 256 bits para calcular la quinta columna del nuevo grupo de 8 (Nk) columnas.

  Para cambiar de estado se necesita que la operación *SubBytes* haya finalizado.

- **estado8:** Este estado se alcanza únicamente si la longitud de clave es de 256 bits como consecuencia de lo explicado en el *estado7*.

En él, el resultado de la operación SubBytes es almacenado en la señal *temp*.

No necesita condiciones para la transición de estados.

- **final:** En este estado la señal *fin_proceso*, que es copiada por la salida *flag*, se activa.

  Se permanecerá en este estado indefinidamente.

Todos los estados requieren que la entrada *enable* esté activa, volviendo al estado *inicio* en caso contrario.

A continuación se adjunta la figura 5.26, donde se ve de una forma más clara las transiciones entre estados.

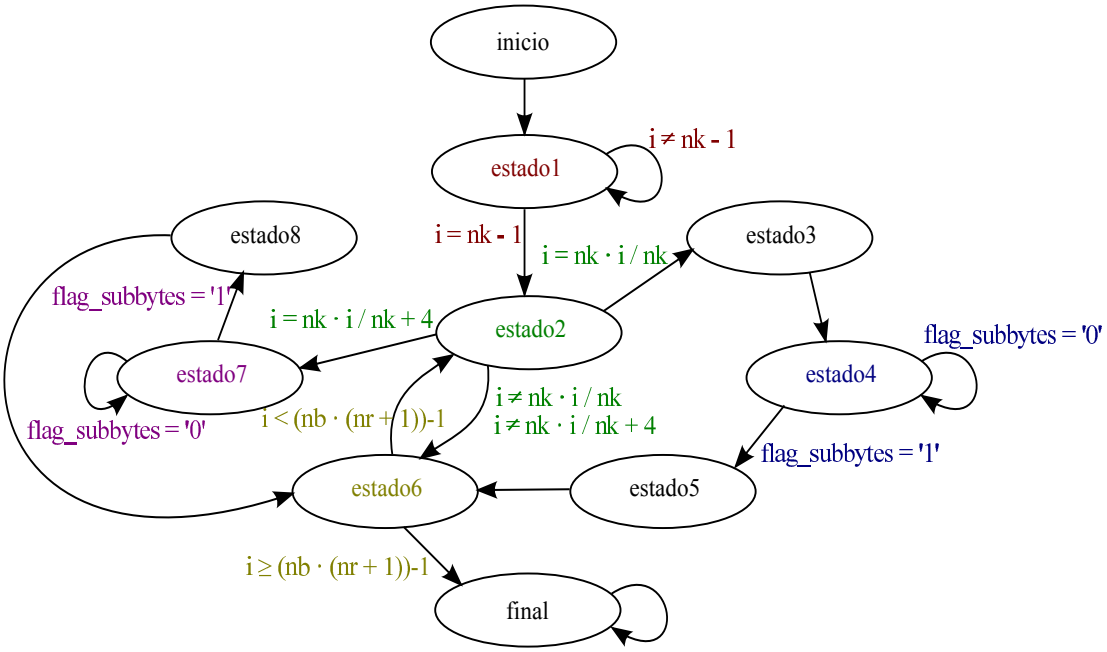


Figura 5.26: Máquina de estados del componente Keygen

Es necesario explicar las transiciones del *estado2* (color verde). Éstas se producían calculando restos.

Existe un operador en VHDL que calcula los restos entre dos números (MOD), sin embargo sólo lo hace para potencias de 2.

En el caso de tener longitudes de clave de 128 y 256 bits, sus correspondientes Nk(4 y 8) sí son potencias de 2 y podría describirse la condición de transición con el operador MOD.

En el caso de tener longitud de clave 192 bits, su Nk (6) no es potencia de 2, por lo que hay que buscar un método alternativo.

Tan simple como aplicar la prueba de la división:

$$Dividendo = Cociente \times Divisor + Resto$$

donde:

$$Dividendo = i$$

$$Divisor = Nk$$

Por lo tanto, si necesitamos la condición de que el resto sea cero, la descripción sería:

$$i = i/Nk \times Nk + 0$$

Mientras que si necesitamos que la condición es que el resto sea cuatro:

$$i = i/Nk \times Nk + 4$$

5.8.3. Simulaciones del componente Keygen

En esta sección vamos a realizar una simulación en la que comprobaremos el correcto funcionamiento del bloque combinacional, ya que si éste proporciona los resultados esperados, el hecho de que la máquina de estados funcione correctamente viene implícito en ello.

Dicha simulación se adjunta en la figura 5.27.

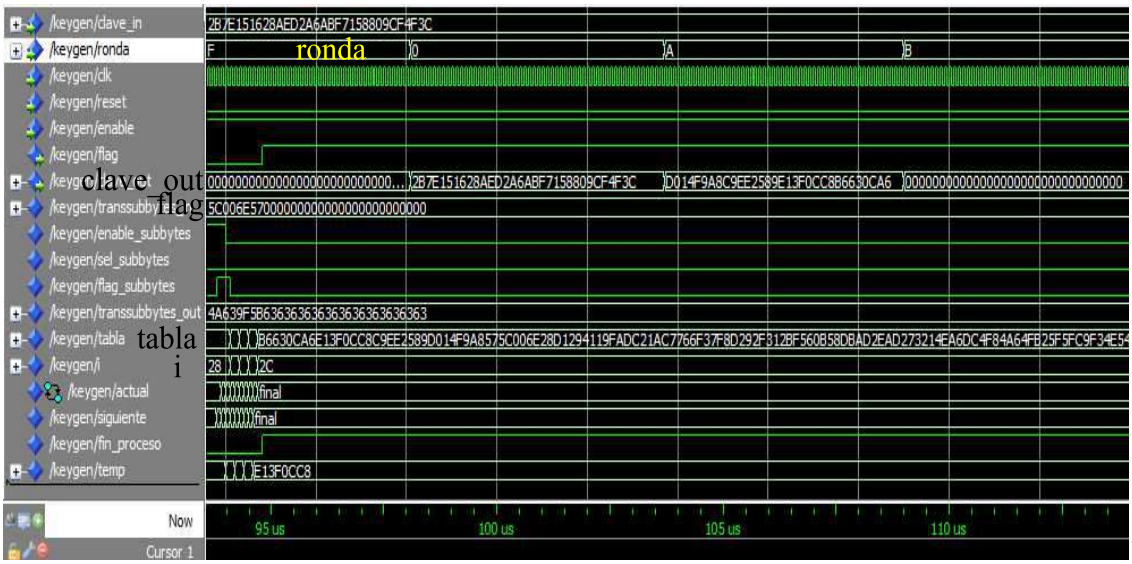


Figura 5.27: Escritura de datos del componente Keygen

Como vemos, cuando todos los bits de la señal *tabla* han sido rellenados con sus respectivos valores (en el caso de longitud de clave 128 bits, $i = 44$), se activa la salida *flag*.

En ese momento se proporcionarán en *clave_out* valores válidos para todos los valores posibles de la entrada *ronda* (un ciclo de reloj antes se proporcionarían valores válidos para todo valor de *ronda* excepto el último, 10 en el caso de longitud de clave 128).

Como vemos, ante un valor de *ronda* 0 se copia en la salida la clave original, mientras que para un valor de *ronda* 10 se copia la subclave correspondiente.

Además, si se establece en la entrada *ronda* un valor que no entra dentro de lo establecido en el estándar (por ejemplo, ronda 11 para longitudes de clave de 128 bits), *clave_out* tomará el valor cero.

Capítulo 6

Resultados

En este capítulo nos centraremos en presentar los resultados que se han ido obteniendo a lo largo del flujo de diseño (capítulo 4).

En este proyecto, nuestro diseño se ha realizado para dos FPGAs de familias diferentes que han sido elegidas con el criterio de ajustar el área lo máximo posible, realizando simulaciones temporales idénticas para ambas.

Será, por tanto, interesante comprobar las diferencias en aspectos de síntesis y consumo de potencia que pueden existir entre dos familias de FPGAs.

Cabe destacar que, primeramente, se realizó una implementación del algoritmo AES en el que se habían descrito muchos más pines de entrada y de salida, ya que existía una entrada para los 128 bits del estado y otra de rango 128-256 bits para la clave.

Esto produce un consumo estático elevado y se desaprovechan muchos recursos de la FPGA.
Sin embargo, esta implementación se cambió mucho antes de comenzar con las simulaciones postlayout, es decir, que se redujo una fuente importante de consumo de potencia antes incluso de iniciar el estudio de reducción de potencia.

En nuestra implementación, los pines dedicados al estado y la clave suman 32 bits en total para cualquier longitud de clave.

Los resultados que se van a mostrar son para implementaciones de longitudes de clave de 128 bits, pero se ha realizado también la síntesis para longitudes de 256 bits, ya que se trata del caso en el que más recursos de la

FPGA necesitaremos, es decir, nos hemos asegurado de que la misma FPGA tiene capacidad para implementar las tres longitudes de clave.

Las familias de FPGAs escogidas son:

- Spartan-6
- Virtex-4

Las Virtex-4 son anteriores a las Spartan-6. Se ha elegido así porque las Virtex son más potentes que las Spartan y, de haber elegido la familia Virtex-6, habría sobrado mucho área.
También será interesante estudiar como afecta la antigüedad al consumo de potencia, viendo como los nuevos diseños de FPGAs son más eficientes.

Como hemos dicho, el principal criterio de elección de FPGAs dentro de esas familias ha sido el de ajustar el área lo máximo posible.
Siendo esto así, las dos FPGAs escogidas son:

- **Spartan-6:** XC6SLX16FTG256-3
- **Virtex-4:** XC4VFX12SF363-12

6.1. Resultados de la síntesis

En esta sección se expondrán los resultados obtenidos tras la síntesis en ambas FPGAs para claves de 128 bits de longitud.

6.1.1. Spartan-6 (Síntesis)

Advanced HDL Synthesis Report

```
Advanced HDL Synthesis Report

Macro Statistics
# RAMs                                                : 5
 16x32-bit single-port distributed Read Only RAM      : 1
 256x8-bit single-port block Read Only RAM            : 4
# Adders/Subtractors                                  : 16
 12-bit adder                                         : 6
 4-bit addsub                                         : 1
 7-bit subtractor                                     : 1
 8-bit adder                                          : 7
 9-bit adder                                          : 1
# Counters                                            : 7
 2-bit down counter                                   : 1
 2-bit up counter                                     : 1
 3-bit up counter                                     : 1
 4-bit down counter                                   : 3
 6-bit up counter                                     : 1
# Registers                                           : 3065
 Flip-Flops                                           : 3065
# Comparators                                         : 4
 2-bit comparator lessequal                           : 1
 4-bit comparator greater                             : 1
 6-bit comparator equal                               : 1
 6-bit comparator greater                             : 1
# Multiplexers                                        : 5154
 1-bit 2-to-1 multiplexer                             : 5122
 11-bit 2-to-1 multiplexer                            : 6
 11-bit 4-to-1 multiplexer                            : 1
 128-bit 2-to-1 multiplexer                           : 9
 16-bit 2-to-1 multiplexer                            : 1
 32-bit 2-to-1 multiplexer                            : 3
 4-bit 2-to-1 multiplexer                             : 4
 8-bit 2-to-1 multiplexer                             : 8
# FSMs                                                : 11
# Xors                                                : 50
 11-bit xor2                                          : 3
 11-bit xor4                                          : 4
 128-bit xor2                                         : 1
 32-bit xor2                                          : 41
 8-bit xor4
```

Tabla 6.1: Advanced HDL Synthesis Report (Spartan-6)

Device Utilization Summary

```
Device utilization summary:
---------------------------

Selected Device : 6slx16ftg256-3


Slice Logic Utilization:
 Number of Slice Registers:            3027  out of  18224    16%
 Number of Slice LUTs:                 4576  out of   9112    50%
    Number used as Logic:              4576  out of   9112    50%

Slice Logic Distribution:
 Number of LUT Flip Flop pairs used:   4677
   Number with an unused Flip Flop:    1650  out of   4677    35%
   Number with an unused LUT:           101  out of   4677     2%
   Number of fully used LUT-FF pairs:  2926  out of   4677    62%
   Number of unique control sets:        35

IO Utilization:
 Number of IOs:                         165
 Number of bonded IOBs:                 165  out of    186    88%

Specific Feature Utilization:
 Number of Block RAM/FIFO:                2  out of     32     6%
    Number using Block RAM only:          2
 Number of BUFG/BUFGCTRLs:                1  out of     16     6%
```

Tabla 6.2: Device Utilization Summary (Spartan-6)

Timing Summary

```
Timing Summary:
---------------
Speed Grade: -3

   Minimum period: 6.035ns (Maximum Frequency: 165.687MHz)
   Minimum input arrival time before clock: 6.271ns
   Maximum output required time after clock: 3.791ns
   Maximum combinational path delay: No path found

Timing Details:
---------------
All values displayed in nanoseconds (ns)
```

Tabla 6.3: Timing Summary (Spartan-6)

6.1.2. Virtex-4 (Síntesis)

Advanced HDL Synthesis Report

```
Advanced HDL Synthesis Report

Macro Statistics
# RAMs                                          : 4
 256x8-bit single-port block RAM                : 4
# Adders/Subtractors                            : 11
 2-bit adder                                    : 1
 2-bit subtractor                               : 1
 3-bit adder                                    : 1
 4-bit addsub                                   : 1
 4-bit subtractor                               : 3
 6-bit adder                                    : 1
 6-bit subtractor                               : 1
 7-bit subtractor                               : 1
 8-bit adder                                    : 1
# Registers                                     : 3139
 Flip-Flops                                     : 3139
# Comparators                                   : 5
 2-bit comparator greater                       : 1
 2-bit comparator less                          : 1
 4-bit comparator greater                       : 2
 6-bit comparator less                          : 1
# Multiplexers                                  : 7
 32-bit 44-to-1 multiplexer                     : 1
 8-bit 16-to-1 multiplexer                      : 2
 8-bit 4-to-1 multiplexer                       : 4
# Xors                                          : 39
 1-bit xor2                                     : 32
 11-bit xor4                                    : 4
 128-bit xor2                                   : 1
 32-bit xor2                                    : 1
 8-bit xor4                                     : 1
```

Tabla 6.4: Advanced HDL Synthesis Report (Virtex-4)

Device Utilization Summary

```
Device utilization summary:
---------------------------

Selected Device : 4vfx12sf363-12

 Number of Slices:                     3548  out of   5472    64%
 Number of Slice Flip Flops:           3033  out of  10944    27%
 Number of 4 input LUTs:               6259  out of  10944    57%
 Number of IOs:                         165
 Number of bonded IOBs:                 165  out of    240    68%
 Number of FIFO16/RAMB16s:                3  out of     36     8%
    Number used as RAMB16s:               3
 Number of GCLKs:                         1  out of     32     3%
```

Tabla 6.5: Device Utilization Summary (Virtex-4)

Timing Summary

```
Timing Summary:
---------------
Speed Grade: -12

   Minimum period: 6.504ns (Maximum Frequency: 153.740MHz)
   Minimum input arrival time before clock: 6.822ns
   Maximum output required time after clock: 3.793ns
   Maximum combinational path delay: No path found

Timing Detail:
--------------
All values displayed in nanoseconds (ns)
```

Tabla 6.6: Timing Summary (Virtex-4)

6.2. Consumo de potencia

Hoy en día se está realizando un gran esfuerzo en relación a las mejoras tecnológicas.

Dichas mejoras implican el aumento del número de transistores utilizados para una única implementación, con el consecuente aumento de la potencia consumida. Sin embargo, estos transistores han ido disminuyendo de tamaño, de manera que el consumo total ha sido reducido.

El consumo de potencia se puede dividir en [MP96][Men03]:

- **Consumo estático:** Se produce por la existencia de caminos entre alimentación y masa por los que circula la corriente cuando el circuito se encuentra en estado de reposo y no hay involucrada ninguna señal de reloj. Está relacionado directamente con la tecnología empleada.

  Viene dado por la ecuación:

  $$P_{estatica} = I_{estatica} \cdot V_{DD}$$

- **Consumo dinámico:** Se produce por las conmutaciones en los nodos circuitales, ya que para que puedan variar su valor lógico deben producirse movimientos de cargas por un medio disipativo, lo que origina un consumo de energía..

  Viene dado por la ecuación:

  $$P = \sum_{i=1}^{n} \alpha_i \cdot C_i \cdot Vdd^2 \cdot f_{clk}$$

  Donde α = Probabilidad de que el nodo de entrada conmute de "0" a "1".
  C = Capacidad del condensador.
  Vdd = Tensión de alimentación.
  f_{clk} = Frecuencia del reloj.

Nuestras dos FPGAs utilizadas son de las familias Spartan-6 y Virtex-4.

Según [Xil], las Virtex-4 utilizan tecnología de 90 nm [Vir], mientras que las Spartan-6 usan tecnología de 45 nm [Spa].
Esto quiere decir que, en el mismo área, en una Spartan-6 se encuentran el doble de transistores que en una Virtex-4.

El hecho de que el número de transistores sea mayor en la Spartan-6 se va a ver traducido en un incremento de la potencia estática con respecto a la dinámica. Pero el hecho de que estos transistores sean de menor tamaño va a implicar una reducción del consumo en general.

En este apartado se van a estudiar las diferencias que existen en el consumo de las dos FPGAs escogidas para claves de 128 bits.
Posteriormente se expondrá la estimación del consumo de las FPGAs si la longitud de clave es de 256 bits.
Finalmente se compararán todos y cada uno de los consumos.
Para ello se utilizarán simulaciones postlayout en las que se procesará un bloque de datos (128 bits), ya que se ha observado que un mayor número de bloques procesados, con la consecuente mayor duración de la simulación postlayout, no implica un aumento significativo del consumo (se ha comprobado que el consumo producido por el procesado de 60 bloques de datos es aproximadamente igual que el consumo producido por el procesado de un único bloque de datos).

Las FPGAs escogidas son:

- **Spartan-6:** XC6SLX16FTG256-3
- **Virtex-4:** XC4VFX12SF363-12

Para realizar la estimación del consumo, como se ha indicado en el capítulo 4, se utilizará la herramienta XPower.

6.2.1. Consumo de la FPGA Spartan-6

A continuación, en las tablas 6.7, 6.8 y 6.9, se exponen los resultados de las estimaciones del consumo de potencia para longitudes de clave de 128 bits para la FPGA de la familia Spartan-6.

En la tabla 6.7, en la primera columna se observan las tensiones de alimentación de la FPGA. En la segunda columna aparecen las corrientes, tanto estática como dinámica, que dicha alimentación produce. En la tercera columna se ven las potencias consumidas en función de cada corriente y su respectiva tensión de alimentación.

	Tensión (V)	**Corriente (mA)**	**Potencia (mW)**	**%**
Vccint	1.20			
Dinámico		2.58	3.096	13.438 %
Estático		6.14	7.368	31.981 %
Vccaux	2.50			
Dinámico		0.00	0.00	0.00 %
Estático		3.03	7.575	32.879 %
Vcco25	2.50			
Dinámico		0.00	0.00	0.00 %
Estático		2.00	5.00	21.702 %
Potencia Total			23.039	100 %

Tabla 6.7: Consumo Spartan-6 I

En la tabla 6.8 se divide el consumo en estático y dinámico, estudiando en detalle la parte del consumo dinámico, producido por el reloj, la lógica, las señales, las memorias y las entradas y salidas.

	Potencia (mW)	**%**
Reloj	2.95	12.798 %
Lógica	0.05	0.217 %
Señales	0.08	0.347 %
BRAMs	0.01	0.043 %
E/S	0.02	0.087 %
Estático	19.94	86.508 %
TOTAL	23.05	100 %

Tabla 6.8: Consumo Spartan-6 II

Como vemos, la inmensa mayoría del consumo dinámico va a parar al reloj, siendo un 12.798 % del consumo total, que representa casi la totalidad del consumo dinámico.

En la tabla 6.9 aparece el consumo total, dividido en estático y dinámico.

	Potencia (mW)	**%**
Dinámico	3.10	13.449 %
Estático	19.94	86.508 %
TOTAL	23.05	100 %

Tabla 6.9: Consumo Spartan-6 III

En la tabla 6.10 vemos el consumo total, dividido en estático y dinámico producido por el circuito cuando trabaja con longitudes de clave de 256 bits.

	Potencia (mW)	**%**
Dinámico	3.82	16.07 %
Estático	19.95	83.93 %
TOTAL	23.77	100 %

Tabla 6.10: Consumo Spartan-6 IV

Como vemos, el consumo estático para ambas longitudes de clave es prácticamente igual, mientras que el consumo dinámico se ve incrementado para claves de 256 bits de longitud.

6.2.2. Consumo de la FPGA Virtex-4

De nuevo, en las tablas 6.11, 6.12 y 6.13, se exponen los resultados de las estimaciones del consumo de potencia para longitudes de clave de 128 bits para la FPGA de la familia Virtex-4.

En la tabla 6.11, en la primera columna se observan las tensiones de alimentación de la FPGA. En la segunda columna aparecen las corrientes, tanto estática como dinámica, que dicha alimentación produce. En la tercera columna se ven las potencias consumidas en función de cada corriente y su respectiva tensión de alimentación.

	Tensión (V)	**Corriente (mA)**	**Potencia (mW)**	**%**
Vccint	1.20			
Dinámico		30.13	36.156	20.608 %
Estático		48.89	58.668	33.439 %
Vccaux	2.50			
Dinámico		0.00	0.00	0.00 %
Estático		31.00	77.500	44.172 %
Vcco25	2.50			
Dinámico		0.00	0.00	0.00 %
Estático		1.25	3.125	1.781 %
Potencia Total			175.449	100 %

Tabla 6.11: Consumo Virtex-4 I

En la tabla 6.12 se divide el consumo en estático y dinámico, estudiando en detalle la parte del consumo dinámico, producido por el reloj, la lógica, las señales, las memorias y las entradas y salidas.

	Potencia (mW)	**%**
Reloj	35.81	20.409 %
Lógica	0.10	0.057 %
Señales	0.22	0.125 %
BRAMs	0.01	0.006 %
E/S	0.03	0.017 %
Estático	139.29	79.386 %
TOTAL	175.46	100 %

Tabla 6.12: Consumo Virtex-4 II

De nuevo, dentro del consumo dinámico, el reloj es el que mas potencia consume, siendo un 20.409 % del total, que representa casi la totalidad del consumo dinámico.

En la tabla 6.13 aparece el consumo total, dividido en estático y dinámico.

	Potencia (mW)	**%**
Dinámico	36.17	20.614 %
Estático	139.29	79.386 %
TOTAL	175.46	100 %

Tabla 6.13: Consumo Virtex-4 III

En la tabla 6.14 vemos el consumo total, dividido en estático y dinámico producido por el circuito cuando trabaja con longitudes de clave de 256 bits.

	Potencia (mW)	**%**
Dinámico	37.12	18.175 %
Estático	167.12	81.825 %
TOTAL	204.24	100 %

Tabla 6.14: Consumo Virtex-4 IV

Como vemos, en este caso, aunque el consumo dinámico también incrementa, tiene mas importancia la mayor utilización de área que requiere este circuito, ya que el consumo estático aumenta en aproximádamente 28 mW.

6.2.3. Comparación del consumo

Según los datos presentados en las tablas anteriores se puede observar el impresionante avance en cuanto a consumo de potencia que se ha conseguido con la fabricación de nuevas familias de FPGAs.

Para longitudes de clave de 128 bits, de los 175,46 mW que son consumidos por la Virtex-4 hemos pasado a los 23,05 mW que requieren la Spartan-6. Eso implica una reducción del 86.86 % del consumo de potencia.

Sin embargo y como esperábamos, debido al mayor número de transistores, el consumo de potencia estática ha pasado de un 79.39 % para la familia Virtex-4 a un 86.51 % para la familia Spartan-6.

Además, aunque la utilización de una longitud de clave de 256 bits nos proporciona un mayor consumo dinámico, la principal componente del consumo en ambos casos sigue siendo el consumo estático, siendo hasta casi siete veces superior al dinámico.

Es por ello que nos vamos a centrar en la reducción de esta componente estática, ya que, antes de llegar a este apartado, se realizó una reducción del número de pines de entrada (lo que significa que ya se hizo una mejora para reducir el consumo debido a que nos ajustamos más al área de la FPGA, pero el resultado sigue siendo un elevado porcentaje de consumo estático).

Desafortunadamente, no existen mucha posibilidades a la hora de realizar estas mejoras, ya que el consumo se debe a la tecnología existente, es decir, no podemos evitar que haya el número de transistores que hay. Por lo que técnicas como las mejoras de arquitectura y la codificación de las máquinas de estados (dedicadas a la reducción del consumo dinámico) no nos van a proporcionar mejoras significativas en el consumo total.

6.3. Reducción del consumo

Llegados a este punto se hace necesario preguntarse: ¿Qué opciones tenemos?

Como ya se ha dicho, las posibilidades de reducción del consumo estático son muy reducidas.

Según [Nev09], la más significativa se basa en la reducción del voltaje de alimentación del núcleo (vccint), del voltaje auxiliar (vccaux) y vcco25.

Esta técnica se basa en reducir estas tensiones hasta encontrar el voltaje mínimo de alimentación con el cual nuestro circuito sigue funcionando correctamente.

Es necesario tener en cuenta que la reducción del voltaje, si no está debidamente controlada, puede ocasionar fallos debido al incumplimiento de tiempos en los caminos críticos del circuito. Debido a ello, los fabricantes establecen un rango de voltajes del que no es recomendable salir.

Si bien es cierto que las FPGAs pueden funcionar fuera de esos rangos, en nuestras estimaciones vamos a trabajar siempre dentro del rango establecido por el fabricante.

Otro factor a considerar con el que está íntimamente ligado el consumo de potencia es la temperatura.
En estos casos se puede estudiar el encapsulamiento y el uso de sumideros o disipadores de calor.

En esta sección se estudiará la aplicación de estas técnicas de reducción de consumo sobre las estimaciones anteriores para claves de longitud 128 bits.

6.3.1. Reducción del consumo para Spartan-6

En las siguientes tablas, entre paréntesis aparecen los valores originales de la sección anterior con el objeto de facilitar la comparación entre ambos casos.

En las tablas 6.15, 6.16 y 6.17 se estudia la reducción del consumo aplicando la técnica de reducción de la tensión de alimentación al mínimo recomendado por el fabricante.

	Tensión (V)	**Corriente (mA)**	**Potencia (mW)**	**% Reducción**
Vccint	1.140 (**1.20**)			
Dinámico		2.53 (**2.58**)	2.884 (**3.096**)	6.848 %
Estático		5.10 (**6.14**)	5.814 (**7.368**)	21.091 %
Vccaux	2.375 (**2.50**)			
Dinámico		0.00	0.00	0.00 %
Estático		2.98 (**3.03**)	7.078 (**7.575**)	6.561 %
Vcco25	2.25 (**2.50**)			
Dinámico		0.00	0.00	0.00 %
Estático		2.00 (**2.00**)	4.50 (**5.00**)	10.00 %
Potencia Total			20.276 (**23.039**)	11.993 %

Tabla 6.15: Reducción del connsumo Spartan-6 I

	Potencia (mW)	**% Reducción**
Reloj	2.76 (**2.95**)	6.44 %
Lógica	0.05 (**0.05**)	0.00 %
Señales	0.07 (**0.08**)	12.50 %
BRAMs	0.01 (**0.01**)	0.00 %
E/S	0.02(**0.02**)	0.00 %
Estático	17.89 (**19.94**)	10.281 %
TOTAL	20.78 (**23.05**)	9.848 %

Tabla 6.16: Reducción del consumo Spartan-6 II

	Potencia (mW)	**% Reducción**
Dinámico	2.90 (**3.10**)	6.452 %
Estático	17.39 (**19.94**)	12.788 %
TOTAL	20.28 (**23.05**)	12.017 %

Tabla 6.17: Reducción del consumo Spartan-6 III

Como vemos, con el cambio en el voltaje de alimentación, conseguimos reducir en un 12.788 % el consumo estático, además de una reducción del 6.452 % en el consumo dinámico, que se traduce en un 12.017 % de reducción

del consumo total.

Todas las estimaciones hasta ahora realizadas son para una temperatura ambiente de 25 °C.

Para evaluar el efecto de la temperatura, se han realizado unos nuevos experimentos manteniendo todas las condiciones iniciales (es decir, sin aplicar la reducción del voltaje de alimentación), a excepción de la temperatura ambiente, que se ha fijado en 15 °C (tabla 6.18).

	Potencia (mW)	**% Reducción**
Dinámico	3.10 (**3.10**)	0.00 %
Estático	16.80 (**19.94**)	15.747 %
TOTAL	19.91 (**23.05**)	13.623 %

Tabla 6.18: Reducción del consumo Spartan-6 IV

Como vemos, no conviene subestimar el factor temperatura, ya que una diferencia de 10 °C en la temperatura ambiente es capaz de reducir considerablemente el consumo de potencia estática.

En la tabla 6.19 se estudia como afectan al consumo la aplicación de ambas técnicas, reducción de la tensión de alimentación y reducción de la temperatura ambiente.

	Potencia (mW)	**% Reducción**
Dinámico	2.90 (**3.10**)	6.452 %
Estático	14.76 (**19.94**)	25.978 %
TOTAL	17.65 (**23.05**)	23.427 %

Tabla 6.19: Reducción del consumo Spartan-6 V

Se observa que se puede conseguir una reducción del consumo total de hasta un 23.427 %.

6.3.2. Reducción del consumo para Virtex-4

En las siguientes tablas, entre paréntesis aparecen los valores originales de la sección anterior con el objeto de facilitar la comparación entre ambos casos.

En las tablas 6.20, 6.21 y 6.22 se estudia la reducción del consumo aplicando la técnica de reducción de la tensión de alimentación al mínimo recomendado por el fabricante.

	Tensión (V)	**Corriente (mA)**	**Potencia (mW)**	**% Reducción**
Vccint	1.140 (**1.20**)			
Dinámico		29.99 (**30.13**)	34.189 (**36.156**)	5.440 %
Estático		43.90 (**48.89**)	50.046 (**58.668**)	16.696 %
Vccaux	2.125 (**2.50**)			
Dinámico		0.00	0.00	0.00 %
Estático		31.00 (**31.00**)	65.875 (**77.500**)	15.00 %
Vcco25	2.380 (**2.50**)			
Dinámico		0.00	0.00	0.00 %
Estático		1.25 (**1.25**)	2.975 (**3.125**)	4.80 %
Potencia Total			153.085 (**175.449**)	12.747 %

Tabla 6.20: Reducción del connsumo Virtex-4 I

	Potencia (mW)	**% Reducción**
Reloj	33.87 (**35.81**)	5.417 %
Lógica	0.09 (**0.10**)	10.00 %
Señales	0.20 (**0.22**)	9.091 %
BRAMs	0.01 (**0.01**)	0.00 %
E/S	0.02 (**0.03**)	33.333 %
Estático	119.05 (**139.29**)	14.531 %
TOTAL	153.24 (**175.46**)	12.664 %

Tabla 6.21: Reducción del consumo Virtex-4 II

	Potencia (mW)	**% Reducción**
Dinámico	34.20 (**36.17**)	5.447 %
Estático	118.90 (**139.29**)	14.639 %
TOTAL	153.09 (**175.46**)	12.749 %

Tabla 6.22: Reducción del consumo Virtex-4 III

Como vemos, con el cambio en el voltaje de alimentación, conseguimos reducir en un 14.639 % el consumo estático, además de un 5.447 % del consumo dinámico, que se traduce en un 12.749 % de reducción del consumo total.

De nuevo, vamos a estudiar de qué manera afecta una bajada de 10 °C en la temperatura ambiente y volviendo a alimentar la FPGA a tensiones nominales (tabla 6.23).

	Potencia (mW)	**% Reducción**
Dinámico	36.17 (**36.17**)	0.00 %
Estático	130.88 (**139.29**)	6.038 %
TOTAL	167.05 (**175.46**)	4.793 %

Tabla 6.23: Reducción del consumo Virtex-4 IV

Si además de la bajada de 10 °C realizamos la bajada de la tensión de alimentación (tabla 6.24), obtenemos:

	Potencia (mW)	**% Reducción**
Dinámico	34.20 (**36.17**)	5.447 %
Estático	111.74 (**139.29**)	19.779 %
TOTAL	145.93 (**175.46**)	16.830 %

Tabla 6.24: Reducción del consumo Virtex-4 V

Se observa una reducción de hasta el 16.83 %.

6.3.3. Conclusiones de la reducción del consumo

En el apartado 6.3 hemos estudiado el consumo de nuestro circuito para dos FPGAs diferentes y las posibles maneras de reducirlo.

Estas posibilidades eran:

- Reducción del voltaje de alimentación.
- Reducción de la temperatura ambiente.
- Reducción del voltaje de alimentación y de la temperatura ambiente.

En la tabla 6.25 se comparan las reducciones en el consumo totales debido a estas tres técnicas propuestas entre las dos FPGAs seleccionadas.

	Bajada de tensión de alimentación	**Bajada de temperatura**	**Bajada de tensión de alimentación + temperatura**
Spartan-6	12.017 %	13.623 %	23.427 %
Virtex-4	12.749 %	4.793 %	16.830 %

Tabla 6.25: Comparación de la reducción del consumo

Como vemos las reducciones producidas por la bajada de tensión son prácticamente iguales.

Sin embargo, se observa una gran diferencia entre la bajada del consumo por cambio de la temperatura ambiente. Esto es debido, probablemente, a que la Virtex-4 utiliza tecnología más antigüa, es decir, un menor número de transistores, con el consecuente menor consumo de potencia estática.

Se puede observar que la temperatura se convierte en una variable muy a tener en cuenta en las tecnologías actuales. Es por ello que el diseño para bajo consumo es crucial.

Capítulo 7

Conclusiones y trabajos futuros

En el presente capítulo se presentarán las conclusiones que se han obtenido a lo largo del estudio realizado y se propondrán futuros trabajos.

7.1. Conclusiones

Para concluir que el trabajo realizado ha sido satisfactorio es necesario recordar los objetivos que se propusieron al principio de este proyecto:

- Estudio del algoritmo AES.
- Descripción en VHDL, simulación funcional y síntesis.
- Emplazamiento, rutado y simulación postlayout.
- Estimación y reducción del consumo.

Tras los estudios realizados, en los que se han realizado simulaciones satisfactorias con varios paquetes de hasta 100 vectores oficiales del NIST cada uno [Vec] (los cuales se adjuntan en el capítulo de anexos), podemos concluir que nuestra descripción funciona correctamente.

A pesar de la drástica reducción de pines de entrada y salida que se realizó (se pasaron de un máximo de 517 a 165) el estudio del consumo ha revelado que la mayor parte (86.508 % en Spartan-6) es consumo estático.

La mayor importancia del consumo estático frente al dinámico es algo que limita mucho al diseñador a la hora de la reducción del consumo, ya que la mayor parte de las técnicas existentes están vinculadas a la reducción del consumo dinámico, que es realmente el que tiene posibilidades de ser

reducido. Recordemos que la potencia estática consumida está directamente relacionada con la tecnología.

A pesar de ello y agotando las pocas posibilidades de las que disponíamos, hemos concluido que, en una Spartan-6, podemos reducir el consumo total hasta en un 12.017 % bajando la tensión de alimentación hasta el mínimo recomendado por el fabricante.

Además, hemos observado que, en tecnologías modernas, la temperatura se convierte en un factor de creciente importancia a tener en cuenta, ya que, con una reducción de 10 °C en la temperatura ambiente, añadido a la mejora por bajada de tensión de alimentación, obtenemos una reducción en el consumo total de hasta un **23.427 %**.

Por lo tanto sería de especial interés estudiar el uso de disipadores de calor.

7.2. Trabajos futuros

En esta sección se propondrán una serie de trabajos futuros de interés con respecto al nivel de seguridad y reducción del consumo.

7.2.1. Modo de operación

Existen varios modos de operación dentro del cifrado por bloques [Dwo01].

Nuestro diseño comprende el modo de operación **ECB** (Electronic Code Book), en el que cada bloque se cifra de manera separada.

Sin embargo, se ha observado que este modo de operación puede no ser completamente seguro, como se muestra en la figura 7.1.

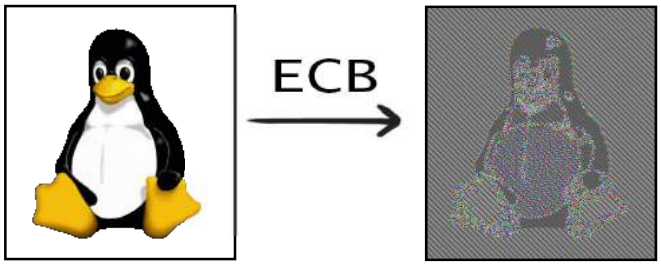


Figura 7.1: Codificación ECB [Mod]

Como vemos, la figura del pingüino sigue siendo reconocible.

En nuestro proyecto hemos diseñado el modo de operación ECB porque nuestro objetivo era conseguir implementar un algoritmo AES completamente funcional. Sin embargo, está diseñado como un núcleo sobre el que sería relativamente sencillo aplicar cualquier mejora con la seguridad de que el diseño no perderá funcionalidad.

Por lo tanto sería interesante estudiar otros modos de operación más seguros para aplicarlos sobre este algoritmo.

Por ejemplo, otro modo de operación es el denominado CBC (Cipher Block Chaining).
En este modo, cada bloque recién procesado se combina con el siguiente a procesar a través de una operación XOR, de esta forma se consigue que los bloques no se procesen de forma independiente, sino que cada uno dependa de todos los anteriores.

Otros modos de operación son el CFB (Cipher Feedback), OFB (Output Feedback) y CTR (Counter)

7.2.2. Reducción del consumo dinámico

En nuestro proyecto nos hemos centrado en la reducción del consumo estático porque se trataba del que mayor proporción tenía. Sin embargo, sería interesante estudiar reducciones del consumo dinámico.

Como se ha visto, casi la totalidad del consumo dinámico es producido por el reloj, por lo que la técnica más interesante a aplicar es la denominada "gated clock" [JK11].

Se trata de una técnica conocida que consiste en desactivar partes del circuito que no se necesitan en momentos determinados, evitando así que los biestables contenidos en dichas partes cambien de estado, evitando la disipación de potencia dinámica.
Aunque se sabe que con esta técnica se obtendrían buenos resultados, en este proyecto se ha intentado reducir el consumo lo máximo con el menor esfuerzo posible, y no siempre es sencillo modificar la estructura del circuito, y más aun en una FPGA con una arquitectura rígida.

Por otra parte, en nuestro diseño hay definidas numerosas máquinas de estado, por lo que sería de especial interés estudiar la codificación de las mismas.

Capítulo 8

Presupuestos

En este capítulo se expondrá el coste total que ha supuesto realizar este proyecto.

8.1. Costes en hardware

PRESUPUESTO

Implementación del algoritmo de cifrado AES para bajo consumo sobre FPGA

CÓDIGO	DESCRIPCIÓN	UNIDAD	MEDICIÓN	PRECIO	IMPORTE
	CAPÍTULO C1 HARDWARE				
C101	**HP Pavilion dv6 Notebook PC** Ordenador portátil con procesador Intel® Core™ i3 2,53 GHz y memoria RAM de 4,00 GB (3,80 GB utilizables).	Ud	1	699.99	699.99

TOTAL CAPÍTULO C1 HARDWARE: 699.99

8.2. Costes en software

PRESUPUESTO

Implementación del algoritmo de cifrado AES para bajo consumo sobre FPGA

CÓDIGO	DESCRIPCIÓN	UNIDAD	MEDICIÓN	PRECIO	IMPORTE
	CAPÍTULO C2 SOFTWARE				
C201	**Sistema Operativo** Sistema Operativo Windows 7 Home Premium, Service Pack 1. Licencia incluida con el PC.	Ud	1	0	0.00
C202	**Xilinx ISE Design Suite 13.2** Software de síntesis y análisis de diseños HDL. Capaz de realizar simulaciones del comportamiento y temporales.	Ud	1	0	0.00
C203	**Modelsim 10.1c SE** Entorno de simulación de diseños HDL, capaz de realizar simulaciones del comportamiento y temporales. Licencia de un año de duración.	Ud	1	1,148.57	1,148.57
C204	**VIM** Editor de texto que comprende múltiples lenguajes. Útil para la comparación entre diversos archivos de texto.	Ud	1	0	0.00
C205	**TeXnicCenter 2.0** Editor de LaTeX que integra las herramientas necesarias para la composición de diversos tipos de textos.	Ud	1	0	0.00

TOTAL CAPÍTULO C2 SOFTWARE: 1,148.57

8.3. Costes en personal

PRESUPUESTO

Implementación del algoritmo de cifrado AES para bajo consumo sobre FPGA

CÓDIGO	DESCRIPCIÓN	UNIDAD	MEDICIÓN	PRECIO	IMPORTE
	CAPÍTULO C3 PERSONAL				
C301	**Miguel García Ocón** Ingeniero encargado del estudio del algoritmo AES, descripción en VHDL, optimización de la misma, simulaciones de comportamiento y temporales, estudio de los resultados, estudio del consumo de potencia y redacción del presente documento.	Hora	660	40	26,400.00
C302	**Dr. Luis Mengibar Pozo** Ingeniero encargado de la dirección del proyecto	Hora	40	40	1,600.00

TOTAL CAPÍTULO C3 PERSONAL: 28,000.00

8.4. Otros costes

PRESUPUESTO

Implementación del algoritmo de cifrado AES para bajo consumo sobre FPGA

CÓDIGO	DESCRIPCIÓN	UNIDAD	MEDICIÓN	PRECIO	IMPORTE
	CAPÍTULO C4 OTROS COSTES				
C401	**Consumo eléctrico** Consumo eléctrico estimado en modalidad sin discriminación horaria.	KwH	597	0,15256	91.08

TOTAL CAPÍTULO C4 OTROS COSTES: 91.08

8.5. Presupuesto general

PRESUPUESTO - RESUMEN

Implementación del algoritmo de cifrado AES para bajo consumo sobre FPGA

CAPÍTULO	DESCRIPCIÓN	IMPORTE (€)	%
C1	HARDWARE	699,99	2,33800406
C2	SOFTWARE	1.148,57	3,83628527
C3	PERSONAL	28.000	93,5214986
C4	OTROS COSTES	91,08	0,30421207

	TOTAL EJECUCIÓN:		29.939,64
	13,00% Gastos Generales:	3892,1532	
	Sumados G.G.:		33.831,79
	16.00% de IVA:	5413,08691	
	TOTAL:		**39.244,87**

Asciende el presupuesto general a la expresada cantidad de TREINTA Y NUEVE MIL DOSCIENTOS CUARENTA Y CUATRO EUROS CON OCHENTA Y SIETE CÉNTIMOS

Pinto, a 12 de Abril del 2013

Capítulo 9

Referencias

[Ana] `http://www.mkit.com.ar/blog/tag/codigo-cesar/`. Accedido 09-Junio-2013.

[Dwo01] Morris Dworkin. Computer Security. *Recommendation for Block Cipher Modes of Operation*, pages 9–15, 2001.

[Esc] `http://www.mathe.tu-freiberg.de/~hebisch/cafe/kryptographie/skytale.html`. Accedido 09-Junio-2013.

[Est] `http://csr.lanl.gov/covert/Lsb-example.JPG`. Accedido 09-Junio-2013.

[JK11] M. Ayoubkhan y Arti Noor Jagrit Kathuria. MIT International Journal of Electronics and Communication Engineering. *MIT Publications*, pages 106–114, 2011.

[Lop02] Manuel Jose Lucena Lopez. Criptografia y Seguridad en Computadores. *Tercera Edicion*, pages 29–34, 2002.

[Men03] Luis Mengibar. Contribucion al Diseno para bajo consumo en FPGAs. *Universidad Carlos III de Madrid*, pages 17–21, 2003.

[Mod] `http://www.isc.tamu.edu/~lewing/linux/`. Accedido 09-Junio-2013.

[MP96] J.M. Rabaey M. Pedram. Low Power Design Methodologies. *Kluwer Academic Publishers*, 1996.

[Mun04] Alfonso Munoz Munoz. Seguridad europea para EEUU. *Algoritmo Criptográfico Rijndael*, pages 6–10, 2004.

[Nev09] Santiago Nevado. Implementación de técnicas de bajo consumo en FPGAs. *Universidad de Alcalá de Henares*, pages 6–7, 2009.

[NIS01] NIST. Announcing the Advanced Encryption Standard (AES). *Federal Information Processing Standards Publication 197*, pages 7–24, 2001.

[Spa] `http://www.xilinx.com/support/documentation/data_sheets/ds160.pdf`. Accedido 09-Junio-2013.

[Vec] `http://csrc.nist.gov/archive/aes/index.html`. Accedido 09-Junio-2013.

[Vir] `http://www.xilinx.com/support/documentation/data_sheets/ds112.pdf`. Accedido 09-Junio-2013.

[Xil] `http://www.xilinx.com/`. Accedido 09-Junio-2013.

Capítulo 10

Anexos

10.1. Anexo I: Código VHDL

10.1.1. Algoritmo AES

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_unsigned.all;

ENTITY aes IS
  GENERIC(
    key_length : integer := 128;
    nk         : integer := 4;
    nr         : integer := 10
  );
  PORT(
    AES_in  : in std_logic_vector (31 downto 0);   --Dato a encriptar/desencriptar.
    clk     : in std_logic;                        --Reloj.
    reset   : in std_logic;                        --Reset.
    enable  : in std_logic;                        --A nivel alto permite el inicio del proceso.
    mode    : in std_logic;                        --Encriptación(1)/Desencriptación(0).

    AES_out : out std_logic_vector (127 downto 0); --Dato encriptado/desencriptado.
    flag    : out std_logic                        --Indica que AES_out es válido.
  );
END aes;

ARCHITECTURE aes_beh OF aes IS

COMPONENT keygen
  GENERIC(
    key_length : integer; --Longitud de la clave: 128, 192 y 256.
    nb         : integer; --Número de columnas del estado: siempre es 4 en el estándar.
    nk         : integer; --Número de columnas de la clave: 4 para 128, 6 para 192 y 8 para 256.
    nr         : integer  --Número de rondas: 10 para 128, 12 para 192 y 14 para 256.
  );
  PORT(
    clave_in  : in std_logic_vector (key_length-1 downto 0); --Clave original.
    ronda     : in std_logic_vector (3 downto 0);            --Ronda del proceso.
    clk       : in std_logic;                                --Reloj.
    reset     : in std_logic;                                --Reset.
    enable    : in std_logic;                                --El proceso comienza cuando
                                                               --está a nivel alto.

    flag      : out std_logic;                               --Indica si ha terminado el proceso.
    clave_out : out std_logic_vector (127 downto 0)          --Nos dará las subclaves dependiendo
                                                               --de la entrada 'ronda'.
  );
END COMPONENT;
-- PORT MAP SIGNALS
signal key           : std_logic_vector (key_length-1 downto 0); --'clave_in'
signal ronda         : std_logic_vector (3 downto 0);            --'ronda'
signal enable_keygen : std_logic;                                --'enable'
signal flag_keygen   : std_logic;                                --'flag'
signal subclave      : std_logic_vector (127 downto 0);          --'clave_out'
```

```
COMPONENT addroundkey
  port(
    addroundkey_in  : in std_logic_vector (127 downto 0); --Entrada del ESTADO.
    clave           : in std_logic_vector (127 downto 0); --Subclave de ronda.

    addroundkey_out : out std_logic_vector (127 downto 0) --ESTADO después de AddRoundKey.
  );
END COMPONENT;
-- PORT MAP SIGNALS
signal addroundkey_in  : std_logic_vector (127 downto 0); --'addroundkey_in'.
signal subclave2       : std_logic_vector (127 downto 0); --'clave'. En esta señal se
                                                            -- copiará el valor de 'subclave'
signal addroundkey_out : std_logic_vector (127 downto 0); --'addroundkey_out'.


COMPONENT subbytes
  port(
    subbytes_in  : in std_logic_vector (127 downto 0); --Entrada del ESTADO.
    clk          : in std_logic;                       --Reloj.
    reset        : in std_logic;                       --Reset.
    enable       : in std_logic;                       --El proceso comienza a nivel alto.
    sel          : in std_logic;                       --Operación SubBytes(0)/InvSubBytes(1).

    flag         : out std_logic;                      --Indica si el proceso ha terminado.
    subbytes_out : out std_logic_vector (127 downto 0) --ESTADO después de SubBytes/InvSubBytes.
  );
END COMPONENT;
-- PORT MAP SIGNALS
signal subbytes_in     : std_logic_vector (127 downto 0); --'subbytes_in'.
signal enable_subbytes : std_logic;                         --'enable'.
signal sel_subbytes    : std_logic;                         --'sel'.
signal flag_subbytes   : std_logic;                         --'flag'.
signal subbytes_out    : std_logic_vector (127 downto 0); --'subbytes_out'.


COMPONENT shiftrows
  port(
    shiftrows_in  : in std_logic_vector (127 downto 0); --Entrada del ESTADO.
    sel           : in std_logic;                        --Operación
                                                           --ShiftRows(0)/InvShiftRows(1).

    shiftrows_out : out std_logic_vector (127 downto 0) --ESTADO después de
                                                           --ShiftRows/InvShiftRows.
   );
END COMPONENT;
-- PORT MAP SIGNALS
signal shiftrows_in  : std_logic_vector (127 downto 0); --'shiftrows_in'.
signal sel_shiftrows : std_logic;                       --'sel'.
signal shiftrows_out : std_logic_vector (127 downto 0); --'shiftrows_out'.
```

```
COMPONENT mixcolumns
  port(
    mixcolumns_in  : in std_logic_vector (127 downto 0); --Entrada del ESTADO.
    clk            : in std_logic;                       --Reloj.
    reset          : in std_logic;                       --Reset.
    enable         : in std_logic;                       --El proceso comienza a nivel alto.
    sel            : in std_logic;                       --Operación MixColumns(0)/
                                                           --InvMixColumns(1).

    flag           : out std_logic;                      --Indica si el proceso ha terminado.
    mixcolumns_out : out std_logic_vector (127 downto 0) --ESTADO después de
                                                           --MixColumns/InvMixColumns.
  );
END COMPONENT;
--PORT MAP SIGNALS
signal mixcolumns_in     : std_logic_vector (127 downto 0); --'mixcolumns_in'.
signal enable_mixcolumns : std_logic;                        --'enable'.
signal sel_mixcolumns    : std_logic;                        --'sel'.
signal flag_mixcolumns   : std_logic;                        --'flag'.
signal mixcolumns_out    : std_logic_vector (127 downto 0); --'mixcolumns_out'.

--OTRAS SEÑALES

signal state        : std_logic_vector (127 downto 0);  --Dato a encriptar/desencriptar.
signal state_loaded : std_logic :='0';                  --Indica si la señal state está preparada.
signal key_loaded   : std_logic :='0';                  --Indica si la señal key está preparada
signal limite       : std_logic_vector(2 downto 0);     --Señal que marca los límites dentro de
                                                          --las señales 'state'(128) y
                                                          --'key'(128, 192 ó 256) donde copiamos
                                                          --la entrada 'AES_in'(32)
signal fin_proceso  : std_logic;                        --Indica si el procesado ha concluído

type tEstados is (inicio, calculo_subclaves, ronda_inicial, Osubbytes, Oshiftrows,
                  Omixcolumns, Oaddroundkey, auxiliar, salida);
signal actual, siguiente: tEstados;

BEGIN
  PMKEYGEN      : keygen GENERIC MAP(key_length=>key_length, nb=>4, nk=>nk,nr=>nr)
                         PORT MAP(key, ronda, clk, reset, enable_keygen, flag_keygen, subclave);

  PMADDROUNDKEY : addroundkey PORT MAP(addroundkey_in, subclave2, addroundkey_out);

  PMSUBBYTES    : subbytes    PORT MAP(subbytes_in, clk, reset, enable_subbytes,sel_subbytes,
                                       flag_subbytes, subbytes_out);

  PMSHIFTROWS   : shiftrows   PORT MAP(shiftrows_in, sel_shiftrows, shiftrows_out);

  PMMIXCOLUMNS  : mixcolumns  PORT MAP(mixcolumns_in, clk, reset, enable_mixcolumns,
                                       sel_mixcolumns, flag_mixcolumns, mixcolumns_out);

  --lógica adicional
  flag <= fin_proceso;
```

```
REGISTRO: process (clk, reset)
BEGIN

  IF reset='1' THEN
    key_loaded   <= '0';
    state_loaded <= '0';
    limite       <= "000";
    state        <= (others =>'0');
    key          <= (others =>'0');

  ELSIF clk'event AND clk = '1' THEN
    IF enable = '1' THEN
      IF key_loaded = '0' THEN
        key (conv_integer(limite)*32+31 downto conv_integer(limite)*32) <= AES_in;
        -- la primera vez => key(31 downto 0)
        limite                                                          <= limite+"001";
        IF conv_integer(limite)*32+31 = key_length-1 THEN
          key_loaded <= '1';
          limite     <= "000";
        END IF;

      ELSE --key_loaded = '1'
        IF state_loaded = '0' THEN
          state(conv_integer(limite)*32+31 downto conv_integer(limite)*32) <= AES_in;
          limite                                                           <= limite+"001";
          IF conv_integer(limite)*32+31 = 127 THEN
            state_loaded <= '1';
            limite       <= "000";
          END IF;

        END IF;

      END IF;

    ELSE --enable ='0'
      IF fin_proceso = '1' THEN
        state_loaded <= '0';
      END IF;
    END IF;

  END IF;

END process REGISTRO;
```

```
FSM_SEQ:    process (clk, reset)
BEGIN
  IF reset = '1' THEN
    actual <= inicio;

  ELSIF clk'event AND clk = '1' THEN
    IF enable = '1' THEN
      actual <= siguiente;
    ELSE
      IF fin_proceso = '1' THEN
        actual <= inicio;
      END IF;
    END IF;

  END IF;

END process FSM_SEQ;



FSM_transiciones: process(actual, key_loaded, flag_keygen, mode, flag_subbytes,
                          ronda, flag_mixcolumns)
BEGIN

  CASE actual IS

    WHEN inicio =>

      IF key_loaded = '1' THEN
        siguiente <= calculo_subclaves;
      ELSE
        siguiente <= inicio;
      END IF;

    WHEN calculo_subclaves =>

      IF flag_keygen = '1' AND state_loaded = '1' THEN
        siguiente <= ronda_inicial;
      ELSE
        siguiente <= calculo_subclaves;
      END IF;
```

```
WHEN ronda_inicial =>

  IF mode = '1' THEN
    siguiente <= Osubbytes;
  ELSE
    siguiente <= auxiliar;
  END IF;

WHEN Osubbytes =>

  IF flag_subbytes = '1' THEN
    IF mode = '1' THEN
      siguiente <= Oshiftrows;
    ELSE
      siguiente <= Oaddroundkey;
    END IF;
  ELSE
    siguiente <= Osubbytes;
  END IF;

WHEN Oshiftrows =>

  IF mode = '1' THEN
    IF conv_integer(ronda) = nr THEN --¿Se han hecho todas las rondas?
      siguiente <= auxiliar;
    ELSE
      siguiente <= Omixcolumns;
    END IF;
  ELSE
    siguiente <= Osubbytes;
  END IF;

WHEN Omixcolumns =>

  IF flag_mixcolumns = '1' THEN
    IF mode = '1' THEN
      siguiente <= Oaddroundkey;
    ELSE
      siguiente <= Oshiftrows;
    END IF;
  ELSE
    siguiente <= Omixcolumns;
  END IF;
```

```
    WHEN Oaddroundkey =>

      IF mode = '1' THEN
        siguiente <= Osubbytes;
      ELSE
        IF ronda = "0000" THEN
          siguiente <= salida;
        ELSE
          siguiente <= Omixcolumns;
        END IF;
      END IF;

    WHEN auxiliar =>

      IF mode = '1' THEN
        siguiente <= salida;
      ELSE
        siguiente <= Osubbytes;
      END IF;

    WHEN salida =>

      siguiente <= salida;

  END CASE;

END process FSM_transiciones;


PROCESADO: process (clk, reset)
BEGIN
  IF reset = '1' THEN
    ronda             <= "1111"; --Inicializo a ese valor porque
                                 -- no se alcanza en ningún caso.
    enable_keygen     <= '0';
    AES_out           <= (others => '0');
    fin_proceso       <= '0';
    addroundkey_in    <= (others => '0');
    subclave2         <= (others => '0');
    subbytes_in       <= (others => '0');
    enable_subbytes   <= '0';
    sel_subbytes      <= '0';
    shiftrows_in      <= (others => '0');
    sel_shiftrows     <= '0';
    mixcolumns_in     <= (others => '0');
    enable_mixcolumns <= '0';
    sel_mixcolumns    <= '0';


  ELSIF clk'event AND clk = '1' THEN
    IF enable = '1' THEN

      IF actual = calculo_subclaves THEN
        enable_keygen <= '1';
        IF flag_keygen = '1' THEN
          IF mode = '1' THEN
            ronda <="0000";
          ELSE
            IF key_length = 128 THEN
              ronda <= "1010"; --ronda 10.
            ELSIF key_length = 192 THEN
              ronda <= "1100"; --ronda 12.
            ELSIF key_length = 256 THEN
              ronda <= "1110"; --ronda 14.
            END IF;

          END IF;

        END IF;
```

```
ELSIF actual = ronda_inicial THEN
  addroundkey_in <= state;
  subclave2      <= subclave;

ELSIF actual = Osubbytes THEN
  enable_subbytes <= '1';
  IF mode = '1' THEN
    subbytes_in  <= addroundkey_out;
    sel_subbytes <= '0'; --Operación SubBytes.
    IF flag_subbytes = '1' THEN
      ronda <= ronda+"0001";
    END IF;
  ELSE
    subbytes_in  <= shiftrows_out;
    sel_subbytes <= '1'; --Operación InvSubBytes.
  END IF;

ELSIF actual = Oshiftrows THEN
  IF mode = '1' THEN
    enable_subbytes <= '0';
    shiftrows_in    <= subbytes_out;
    sel_shiftrows   <= '0'; --Operación ShiftRows.
  ELSE
    enable_mixcolumns <= '0';
    shiftrows_in      <= mixcolumns_out;
    sel_shiftrows     <= '1'; --operación InvShiftRows.
  END IF;

ELSIF actual = Omixcolumns THEN
  enable_mixcolumns <= '1';
  IF mode = '1' THEN
    mixcolumns_in  <= shiftrows_out;
    sel_mixcolumns <= '0'; --operación MixColumns.
  ELSE
    mixcolumns_in  <= addroundkey_out;
    sel_mixcolumns <= '1'; --operación InvMixColumns.
    IF flag_mixcolumns = '1' THEN
      ronda <= ronda-"0001";
    END IF;
  END IF;

ELSIF actual = Oaddroundkey THEN
  subclave2 <= subclave;
  IF mode = '1' THEN
    enable_mixcolumns <= '0';
    addroundkey_in    <= mixcolumns_out;
  ELSE
    enable_subbytes <= '0';
    addroundkey_in  <= subbytes_out;
  END IF;
```

```
        ELSIF actual = auxiliar THEN
          IF mode = '1' THEN
            addroundkey_in <= shiftrows_out;
            subclave2      <= subclave;
          ELSE
            ronda         <= ronda-"0001";
            shiftrows_in  <= addroundkey_out;
            sel_shiftrows <= '1'; --Operación InvShiftRows.
          END IF;

        ELSIF actual = salida THEN
          AES_out     <= addroundkey_out;
          fin_proceso <='1';
        END IF;

      ELSE --enable = '0'
        fin_proceso <= '0';
      END IF;
    END IF;
  END process PROCESADO;
END aes_beh;
```

10.1.2. Componente SubBytes (AES)

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_unsigned.all;

-- componente SubBytes
ENTITY subbytes IS
  PORT(
    subbytes_in  : in std_logic_vector (127 downto 0); --Entrada del ESTADO.
    clk          : in std_logic;                       --Reloj.
    reset        : in std_logic;                       --Reset.
    enable       : in std_logic;                       --El proceso comienza a nivel alto.
    sel          : in std_logic;                       --Operación SubBytes(0)/InvSubBytes(1).

    flag         : out std_logic;                      --Indica si el proceso ha terminado.
    subbytes_out : out std_logic_vector (127 downto 0) --ESTADO después de SubBytes/InvSubBytes.
  );
END subbytes;

ARCHITECTURE subbytes_beh OF subbytes IS

COMPONENT sbox
  PORT(
    sbox_in      : in std_logic_vector (7 downto 0); --Entrada de 8 bits del ESTADO
    clk          : in std_logic;                     --Reloj.
    reset        : in std_logic;                     --Reset.
    enable       : in std_logic;                     --El proceso comienza a nivel alto.

    flag         : out std_logic;                    --El proceso termina a nivel alto.
    sbox_out     : out std_logic_vector (7 downto 0) --Salida de sbox.
  );
END COMPONENT;
--PORT MAP SIGNALS
signal sbox_in     : std_logic_vector(7 downto 0); --'sbox_in'
signal enable_sbox : std_logic;                    --'enable'
signal flag_sbox   : std_logic;                    --'flag'
signal sbox_out    : std_logic_vector(7 downto 0); --'sbox_out'

COMPONENT invsbox
  PORT(
    invsbox_in  : in std_logic_vector (7 downto 0);   --Entrada de 8 bits del ESTADO.
    clk         : in std_logic;                       --Reloj.
    reset       : in std_logic;                       --Reset.
    enable      : in std_logic;                       --El proceso comienza a nivel alto.

    flag        : out std_logic;                      --El proceso termina a nivel alto.
    invsbox_out : out std_logic_vector (7 downto 0)   --Salida de invsbox.
  );
END COMPONENT;
--PORT MAP SIGNALS
signal invsbox_in     : std_logic_vector(7 downto 0); --'invsbox_in'
signal enable_invsbox : std_logic;                    --'enable'
signal flag_invsbox   : std_logic;                    --'flag'
signal invsbox_out    : std_logic_vector(7 downto 0); --'invsbox_out'

--OTRAS SEÑALES.
signal finproceso : std_logic;                    --Indica si el proceso ha terminado.
signal a          : std_logic_vector(3 downto 0); --Nos guiará a lo largo de las posiciones
                                                  --del ESTADO.
```

```
--FSM para memoria RAM.
type tEstados IS (inicio, mem_in, espera, mem_out, final);
signal actual, siguiente: tEstados;

BEGIN
  PMSBOX    : sbox    PORT MAP(sbox_in, clk, reset, enable_sbox, flag_sbox, sbox_out);
  PMINVSBOX : invsbox PORT MAP(invsbox_in, clk, reset, enable_invsbox, flag_invsbox,
                               invsbox_out);

  --lógica adicional.
  flag <= finproceso;

  FSM_SEQ: process (clk, reset)
  BEGIN
    IF reset = '1' THEN
      actual <= inicio;

    ELSIF clk'event AND clk = '1' THEN
      IF enable = '1' THEN
        actual <= siguiente;
      ELSE
        actual <= inicio;
      END IF;

    END IF;

  END process FSM_SEQ;


  FSM_transiciones: process(actual, flag_sbox, flag_invsbox, a, sel)
  BEGIN

    CASE actual IS

      WHEN inicio =>

        siguiente <= mem_in;

      WHEN mem_in =>

        IF sel='0' THEN -- SubBytes
          IF flag_sbox='1' THEN
            siguiente <= espera;
          ELSE
            siguiente <= mem_in;
          END IF;
        ELSE --InvSubBytes
          IF flag_invsbox='1' THEN
            siguiente <= espera;
          ELSE
            siguiente <= mem_in;
          END IF;
        END IF;

      WHEN espera => --Es un estado de transición, ya que necesito un pulso de reloj
                     --para usar la memoria.
        siguiente <= mem_out;
```

```
    WHEN mem_out =>

      IF conv_integer(a)=0 THEN
        siguiente <= final;
      ELSE
        siguiente <= mem_in;
      END IF;

    WHEN final =>

      siguiente <= final;

  END CASE;

END process FSM_transiciones;


SECUENCIAL: process(clk,reset)
begin
  if reset = '1' then
    finproceso     <= '0';
    subbytes_out   <= (others => '0');
    sbox_in        <= (others => '0');
    invsbox_in     <= (others => '0');
    enable_sbox    <= '0';
    enable_invsbox <= '0';
    a              <= (others => '0');

  elsif clk'event and clk='1' then

    if enable='1' then

      IF actual = inicio THEN

        a <= "1111";

      ELSIF actual = mem_in THEN
        IF sel='0' THEN
          sbox_in <= subbytes_in(conv_integer(a)*8+7 downto conv_integer(a)*8);
          --conv_integer(a) está inicializado en 15.
          --La primera vez: sbox_in <= subbytes_in(127 downto 120);
          enable_sbox <= '1';
          IF flag_sbox='1' THEN
            enable_sbox <= '0';
          END IF;
        ELSE
          invsbox_in     <= subbytes_in(conv_integer(a)*8+7 downto conv_integer(a)*8);
          enable_invsbox <= '1';
          IF flag_invsbox='1' THEN
            enable_invsbox <= '0';
          END IF;
        END IF;

      ELSIF actual = mem_out THEN
        IF sel='0' THEN
          subbytes_out(conv_integer(a)*8+7 downto conv_integer(a)*8) <= sbox_out;
          --Lo almacenado en la memoria va la primera vez a subbytes_out(127 downto 120);
        ELSE
          subbytes_out(conv_integer(a)*8+7 downto conv_integer(a)*8) <= invsbox_out;
        END IF;

        IF conv_integer(a)/=0 THEN
          a      <= a-"0001";
        END IF;
```

```
            ELSIF actual = final THEN
              finproceso <= '1';
            END IF;

          ELSE --enable='0'
            finproceso   <= '0';

          END IF;
        end if;
      end process SECUENCIAL;

    END subbytes_beh;
```

10.1.3. Componente SBox (SubBytes)

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_unsigned.all;

-- componente sbox de subbytes
--este componente recoge 8 bits y saca su transformada.
ENTITY sbox IS
  PORT(
    sbox_in  : in std_logic_vector (7 downto 0); --Entrada de 8 bits del ESTADO.
    clk      : in std_logic;                     --Reloj.
    reset    : in std_logic;                     --Reset
    enable   : in std_logic;                     --El proceso comienza a nivel alto.

    flag     : out std_logic;                    --El proceso termina a nivel alto.
    sbox_out : out std_logic_vector (7 downto 0) --Salida.
  );
END sbox;

ARCHITECTURE sbox_beh OF sbox IS

--Memoria ram (only read)
type ram is array(natural range<>) of std_logic_vector(7 downto 0);
constant sbox: ram(255 downto 0) :=
(
X"16",X"bb",X"54",X"b0",X"0f",X"2d",X"99",X"41",X"68",X"42",X"e6",X"bf",X"0d",X"89",X"a1",X"8c",
X"df",X"28",X"55",X"ce",X"e9",X"87",X"1e",X"9b",X"94",X"8e",X"d9",X"69",X"11",X"98",X"f8",X"e1",
X"9e",X"1d",X"c1",X"86",X"b9",X"57",X"35",X"61",X"0e",X"f6",X"03",X"48",X"66",X"b5",X"3e",X"70",
X"8a",X"8b",X"bd",X"4b",X"1f",X"74",X"dd",X"e8",X"c6",X"b4",X"a6",X"1c",X"2e",X"25",X"78",X"ba",
X"08",X"ae",X"7a",X"65",X"ea",X"f4",X"56",X"6c",X"a9",X"4e",X"d5",X"8d",X"6d",X"37",X"c8",X"e7",
X"79",X"e4",X"95",X"91",X"62",X"ac",X"d3",X"c2",X"5c",X"24",X"06",X"49",X"0a",X"3a",X"32",X"e0",
X"db",X"0b",X"5e",X"de",X"14",X"b8",X"ee",X"46",X"88",X"90",X"2a",X"22",X"dc",X"4f",X"81",X"60",
X"73",X"19",X"5d",X"64",X"3d",X"7e",X"a7",X"c4",X"17",X"44",X"97",X"5f",X"ec",X"13",X"0c",X"cd",
X"d2",X"f3",X"ff",X"10",X"21",X"da",X"b6",X"bc",X"f5",X"38",X"9d",X"92",X"8f",X"40",X"a3",X"51",
X"a8",X"9f",X"3c",X"50",X"7f",X"02",X"f9",X"45",X"85",X"33",X"4d",X"43",X"fb",X"aa",X"ef",X"d0",
X"cf",X"58",X"4c",X"4a",X"39",X"be",X"cb",X"6a",X"5b",X"b1",X"fc",X"20",X"ed",X"00",X"d1",X"53",
X"84",X"2f",X"e3",X"29",X"b3",X"d6",X"3b",X"52",X"a0",X"5a",X"6e",X"1b",X"1a",X"2c",X"83",X"09",
X"75",X"b2",X"27",X"eb",X"e2",X"80",X"12",X"07",X"9a",X"05",X"96",X"18",X"c3",X"23",X"c7",X"04",
X"15",X"31",X"d8",X"71",X"f1",X"e5",X"a5",X"34",X"cc",X"f7",X"3f",X"36",X"26",X"93",X"fd",X"b7",
X"c0",X"72",X"a4",X"9c",X"af",X"a2",X"d4",X"ad",X"f0",X"47",X"59",X"fa",X"7d",X"c9",X"82",X"ca",
X"76",X"ab",X"d7",X"fe",X"2b",X"67",X"01",X"30",X"c5",X"6f",X"6b",X"f2",X"7b",X"77",X"7c",X"63"
);

signal aux         : std_logic_vector (1 downto 0);
signal fin_proceso : std_logic;
```

```
BEGIN
  --lógica adicional
  fin_proceso <= '1' when aux="01" else '0';
  flag        <= fin_proceso;

  PROCESS (clk,reset)
  BEGIN
    IF reset='1' THEN
      aux <= "00";
    ELSIF clk'event AND clk='1' THEN
      IF enable='1' THEN
        IF aux="00" THEN
          sbox_out <= sbox(conv_integer(sbox_in));
          aux      <= "01";
        ELSIF aux="01" THEN
          aux <= "01";
        END IF;
      ELSE
        aux <= "00";
      END IF;
    END IF;
  END PROCESS;

END sbox_beh;
```

10.1.4. Componente InvSBox (SubBytes)

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_unsigned.all;

-- componente invsbox de subbytes
ENTITY invsbox IS
  PORT(
    invsbox_in  : in std_logic_vector (7 downto 0); --Entrada del ESTADO.
    clk         : in std_logic;                     --Reloj.
    reset       : in std_logic;                     --Reset.
    enable      : in std_logic;                     --El proceso comienza a nivel alto.

    flag        : out std_logic;                    --El proceso termina a nivel alto.
    invsbox_out : out std_logic_vector (7 downto 0) --Salida
  );
END invsbox;

ARCHITECTURE invsbox_beh OF invsbox IS

--Memoria RAM (only read)
type ram is array(natural range<>) of std_logic_vector(7 downto 0);
constant invsbox: ram (0 to 255) :=
(
x"52",x"09",x"6a",x"d5",x"30",x"36",x"a5",x"38",x"bf",x"40",x"a3",x"9e",x"81",x"f3",x"d7",x"fb",
x"7c",x"e3",x"39",x"82",x"9b",x"2f",x"ff",x"87",x"34",x"8e",x"43",x"44",x"c4",x"de",x"e9",x"cb",
x"54",x"7b",x"94",x"32",x"a6",x"c2",x"23",x"3d",x"ee",x"4c",x"95",x"0b",x"42",x"fa",x"c3",x"4e",
x"08",x"2e",x"a1",x"66",x"28",x"d9",x"24",x"b2",x"76",x"5b",x"a2",x"49",x"6d",x"8b",x"d1",x"25",
x"72",x"f8",x"f6",x"64",x"86",x"68",x"98",x"16",x"d4",x"a4",x"5c",x"cc",x"5d",x"65",x"b6",x"92",
x"6c",x"70",x"48",x"50",x"fd",x"ed",x"b9",x"da",x"5e",x"15",x"46",x"57",x"a7",x"8d",x"9d",x"84",
x"90",x"d8",x"ab",x"00",x"8c",x"bc",x"d3",x"0a",x"f7",x"e4",x"58",x"05",x"b8",x"b3",x"45",x"06",
x"d0",x"2c",x"1e",x"8f",x"ca",x"3f",x"0f",x"02",x"c1",x"af",x"bd",x"03",x"01",x"13",x"8a",x"6b",
x"3a",x"91",x"11",x"41",x"4f",x"67",x"dc",x"ea",x"97",x"f2",x"cf",x"ce",x"f0",x"b4",x"e6",x"73",
x"96",x"ac",x"74",x"22",x"e7",x"ad",x"35",x"85",x"e2",x"f9",x"37",x"e8",x"1c",x"75",x"df",x"6e",
x"47",x"f1",x"1a",x"71",x"1d",x"29",x"c5",x"89",x"6f",x"b7",x"62",x"0e",x"aa",x"18",x"be",x"1b",
x"fc",x"56",x"3e",x"4b",x"c6",x"d2",x"79",x"20",x"9a",x"db",x"c0",x"fe",x"78",x"cd",x"5a",x"f4",
x"1f",x"dd",x"a8",x"33",x"88",x"07",x"c7",x"31",x"b1",x"12",x"10",x"59",x"27",x"80",x"ec",x"5f",
x"60",x"51",x"7f",x"a9",x"19",x"b5",x"4a",x"0d",x"2d",x"e5",x"7a",x"9f",x"93",x"c9",x"9c",x"ef",
x"a0",x"e0",x"3b",x"4d",x"ae",x"2a",x"f5",x"b0",x"c8",x"eb",x"bb",x"3c",x"83",x"53",x"99",x"61",
x"17",x"2b",x"04",x"7e",x"ba",x"77",x"d6",x"26",x"e1",x"69",x"14",x"63",x"55",x"21",x"0c",x"7d"
);

--lógica adicional
signal aux         : std_logic_vector (1 downto 0);
signal fin_proceso : std_logic;
```

```
BEGIN
  fin_proceso <= '1' when aux="01" else '0';
  flag        <= fin_proceso;

  PROCESS (clk,reset)
    BEGIN
    IF reset='1' THEN
      aux <= "00";
    ELSIF clk'event AND clk='1' THEN
      IF enable='1' THEN
        IF aux="00" THEN
          invsbox_out <= invsbox(conv_integer(invsbox_in));
          aux         <= "01";
        ELSIF aux="01" THEN
          aux <= "01";
        END IF;
      ELSE
        aux <= "00";
      END IF;
    END IF;
  END PROCESS;

END invsbox_beh;
```

10.1.5. Componente ShiftRows (AES)

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_unsigned.all;
--Componente ShiftRows
-- Rotación de filas a la izquierda (ShiftRows) o a la derecha (InvShiftRows)

ENTITY shiftrows IS
  PORT(
    shiftrows_in  : in std_logic_vector (127 downto 0); --Entrada del ESTADO.
    sel           : in std_logic;                       --Operación
                                                          --ShiftRows(0)/InvShiftRows(1).
    shiftrows_out : out std_logic_vector (127 downto 0) --ESTADO después de
                                                          --ShiftRows/InvShiftRows.
  );
END shiftrows;

ARCHITECTURE shiftrows_beh OF shiftrows IS

BEGIN
  shiftrows_out(127 downto 120) <= shiftrows_in(127 downto 120);
  shiftrows_out(23 downto 16)   <= shiftrows_in(119 downto 112) when sel='0'
                              else shiftrows_in(55 downto 48);
  shiftrows_out(47 downto 40)   <= shiftrows_in(111 downto 104);
  shiftrows_out(71 downto 64)   <= shiftrows_in(103 downto 96) when sel='0'
                              else shiftrows_in(39 downto 32);
  shiftrows_out(95 downto 88)   <= shiftrows_in(95 downto 88);
  shiftrows_out(119 downto 112) <= shiftrows_in(87 downto 80) when sel='0'
                              else shiftrows_in(23 downto 16);
  shiftrows_out(15 downto 8)    <= shiftrows_in(79 downto 72);
  shiftrows_out(39 downto 32)   <= shiftrows_in(71 downto 64) when sel='0'
                              else shiftrows_in(7 downto 0);
  shiftrows_out(63 downto 56)   <= shiftrows_in(63 downto 56);
  shiftrows_out(87 downto 80)   <= shiftrows_in(55 downto 48) when sel='0'
                              else shiftrows_in(119 downto 112);
  shiftrows_out(111 downto 104) <= shiftrows_in(47 downto 40);
  shiftrows_out(7 downto 0)     <= shiftrows_in(39 downto 32) when sel='0'
                              else shiftrows_in(103 downto 96);
  shiftrows_out(31 downto 24)   <= shiftrows_in(31 downto 24);
  shiftrows_out(55 downto 48)   <= shiftrows_in(23 downto 16) when sel='0'
                              else shiftrows_in(87 downto 80);
  shiftrows_out(79 downto 72)   <= shiftrows_in(15 downto 8);
  shiftrows_out(103 downto 96)  <= shiftrows_in(7 downto 0) when sel='0'
                              else shiftrows_in(71 downto 64);


END shiftrows_beh;
```

10.1.6. Componente MixColumns (AES)

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_unsigned.all;
-- componente Mixcolumns
ENTITY mixcolumns IS
  PORT(
    mixcolumns_in  : in std_logic_vector (127 downto 0); --Entrada del ESTADO.
    clk            : in std_logic;                       --Reloj.
    reset          : in std_logic;                       --Reset.
    enable         : in std_logic;                       --El proceso comienza a nivel alto.
    sel            : in std_logic;                       --Operación
                                                           --MixColumns(0)/InvMixColumns(1).
    flag           : out std_logic;                      --Indica si el proceso ha terminado.
    mixcolumns_out : out std_logic_vector (127 downto 0) --ESTADO después de
                                                          --MixColumns/InvMixColumns.
  );
END mixcolumns;

ARCHITECTURE mixcolumns_beh OF mixcolumns IS

COMPONENT multiplicacion
 port(
   entrada1 : in std_logic_vector (7 downto 0);  --Multiplicación 1: Factor 1.
   entrada2 : in std_logic_vector (3 downto 0);  --Multiplicación 1: Factor 2.
   entrada3 : in std_logic_vector (7 downto 0);  --Multiplicación 2: Factor 1.
   entrada4 : in std_logic_vector (3 downto 0);  --Multiplicación 2: Factor 2.
   entrada5 : in std_logic_vector (7 downto 0);  --Multiplicación 3: Factor 1.
   entrada6 : in std_logic_vector (3 downto 0);  --Multiplicación 3: Factor 2.
   entrada7 : in std_logic_vector (7 downto 0);  --Multiplicación 4: Factor 1.
   entrada8 : in std_logic_vector (3 downto 0);  --Multiplicación 4: Factor 2.
   clk      : in std_logic;                      --Reloj.
   reset    : in std_logic;                      --Reset.
   enable   : in std_logic;                      --El proceso comienza a nivel alto.

   flag     : out std_logic;                     --El proceso ha finalizado.
   salida1  : out std_logic_vector (7 downto 0); --Multiplicación 1: Producto.
   salida2  : out std_logic_vector (7 downto 0); --Multiplicación 2: Producto.
   salida3  : out std_logic_vector (7 downto 0); --Multiplicación 3: Producto.
   salida4  : out std_logic_vector (7 downto 0)  --Multiplicación 4: Producto.
  );
END COMPONENT;
--PORT MAP SIGNALS
signal multiplicacion_in1    : std_logic_vector(7 downto 0);  --'entrada1'.
signal multiplicacion_in2    : std_logic_vector (3 downto 0); --'entrada2'.
signal multiplicacion_in3    : std_logic_vector(7 downto 0);  --'entrada3'.
signal multiplicacion_in4    : std_logic_vector (3 downto 0); --'entrada4'.
signal multiplicacion_in5    : std_logic_vector(7 downto 0);  --'entrada5'.
signal multiplicacion_in6    : std_logic_vector (3 downto 0); --'entrada6'.
signal multiplicacion_in7    : std_logic_vector(7 downto 0);  --'entrada7'.
signal multiplicacion_in8    : std_logic_vector (3 downto 0); --'entrada8'.
signal enable_multiplicacion : std_logic;                     --'enable'.
signal flag_multiplicacion   : std_logic;                     --'flag'.
signal multiplicacion_out1   : std_logic_vector (7 downto 0); --'salida1'.
signal multiplicacion_out2   : std_logic_vector (7 downto 0); --'salida2'.
signal multiplicacion_out3   : std_logic_vector (7 downto 0); --'salida3'.
signal multiplicacion_out4   : std_logic_vector (7 downto 0); --'salida4'.
```

```
--OTRAS SEÑALES
signal fin_mixcolumns   : std_logic;         --Nos indicará que el proceso ha terminado.
signal limite1          : std_logic_vector(1 downto 0);
signal limite2          : std_logic_vector(3 downto 0);
signal vector_mixcolumns: std_logic_vector(15 downto 0);

type tEstados is (inicio, Emultiplicacion, resultado, limites, final);
signal actual, siguiente: tEstados;

BEGIN

  PMmultiplicacion: multiplicacion PORT MAP(multiplicacion_in1, multiplicacion_in2,
                                            multiplicacion_in3, multiplicacion_in4,
                                            multiplicacion_in5, multiplicacion_in6,
                                            multiplicacion_in7, multiplicacion_in8,
                                            clk, reset, enable_multiplicacion,
                                            flag_multiplicacion,
                                            multiplicacion_out1, multiplicacion_out2,
                                            multiplicacion_out3, multiplicacion_out4);
  --logica adicional
  flag<=fin_mixcolumns;

  FSM_SEQ: process (clk, reset)
  BEGIN
    IF reset = '1' THEN
      actual <= inicio;

    ELSIF clk'event AND clk = '1' THEN
      IF enable = '1' THEN
        actual <= siguiente;
      ELSE
        actual <= inicio;
      END IF;

    END IF;
  END process FSM_SEQ;


  FSM_transiciones: process(actual, flag_multiplicacion, limite2)
  BEGIN

    CASE actual IS

      WHEN inicio =>

        siguiente <= Emultiplicacion;

      WHEN Emultiplicacion =>

        IF flag_multiplicacion = '1' THEN
          siguiente <= resultado;
        ELSE
          siguiente <= Emultiplicacion;
        END IF;

      WHEN resultado =>

        siguiente <= limites;
```

```
    WHEN limites =>

      IF limite2 /= "0000" THEN
        siguiente <= Emultiplicacion;
      ELSE
        siguiente <= final;
      END IF;

    WHEN final =>

      siguiente <= final;

  END CASE;

END process FSM_transiciones;


SECUENCIAL: process (clk, reset)
BEGIN
  IF reset='1' THEN
    fin_mixcolumns        <='0';
    mixcolumns_out        <= (others => '0');
    multiplicacion_in1    <= (others => '0');
    multiplicacion_in2    <= (others => '0');
    multiplicacion_in3    <= (others => '0');
    multiplicacion_in4    <= (others => '0');
    multiplicacion_in5    <= (others => '0');
    multiplicacion_in6    <= (others => '0');
    multiplicacion_in7    <= (others => '0');
    multiplicacion_in8    <= (others => '0');
    enable_multiplicacion <= '0';
    limite1               <= (others => '0');
    limite2               <= (others => '0');
    vector_mixcolumns     <= (others => '0');
  ELSIF clk'event and clk='1' THEN
    IF enable='1' THEN
      IF actual = inicio THEN

        limite1 <= "11";
        limite2 <= "1111";
        IF sel = '0' THEN
          vector_mixcolumns<=x"2311";
        ELSE
          vector_mixcolumns<=x"ebd9";
        END IF;

      ELSIF actual = Emultiplicacion THEN

        multiplicacion_in1 <= mixcolumns_in(conv_integer(limite1)*32+31 downto
                                            conv_integer(limite1)*32+24);
        multiplicacion_in3 <= mixcolumns_in(conv_integer(limite1)*32+23 downto
                                            conv_integer(limite1)*32+16);
        multiplicacion_in5 <= mixcolumns_in(conv_integer(limite1)*32+15 downto
                                            conv_integer(limite1)*32+8);
        multiplicacion_in7 <= mixcolumns_in(conv_integer(limite1)*32+7 downto
                                            conv_integer(limite1)*32);
        -- se asigna el primer factor de cada multiplicación

        multiplicacion_in2 <= vector_mixcolumns(15 downto 12);
        multiplicacion_in4 <= vector_mixcolumns(11 downto 8);
        multiplicacion_in6 <= vector_mixcolumns(7 downto 4);
        multiplicacion_in8 <= vector_mixcolumns(3 downto 0);
        --se asigna el segundo factor.

        enable_multiplicacion <= '1';
        -- permitimos que se inicie la multiplicación.
        IF flag_multiplicacion='1' THEN
          enable_multiplicacion <= '0';
        END IF;
```

```
        ELSIF actual = resultado THEN

          mixcolumns_out(conv_integer(limite2)*8+7 downto conv_integer(limite2)*8) <=
                                        multiplicacion_out1 xor multiplicacion_out2 xor
                                        multiplicacion_out3 xor multiplicacion_out4;

          --rotacion del vector_mixcolumns, preparándolo para la siguiente multiplicacion
          vector_mixcolumns(15 downto 12) <= vector_mixcolumns(3 downto 0);
          vector_mixcolumns(11 downto 8) <= vector_mixcolumns(15 downto 12);
          vector_mixcolumns(7 downto 4) <= vector_mixcolumns(11 downto 8);
          vector_mixcolumns(3 downto 0) <= vector_mixcolumns(7 downto 4);


        ELSIF actual = limites THEN

          IF limite2 > "0000" THEN
            limite2 <= limite2-"0001";
          END IF;

          IF vector_mixcolumns = x"2311" or vector_mixcolumns = x"ebd9" THEN
         --el vector ha dado una vuelta entera,
         --por lo tanto se empieza a multiplicar otra columna.
            IF limite1 > "00" THEN
              limite1 <= limite1-"01";
            END IF;
          END IF;

        ELSIF actual = final THEN

          fin_mixcolumns<='1';

        END IF;
      ELSE --enable='0'
        fin_mixcolumns<='0';
      END IF;
    END IF;
  END process SECUENCIAL;

END mixcolumns_beh;
```

10.1.7. Componente Multiplicación (MixColumns)

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_unsigned.all;

-- componente multiplicacion de mixcolumns
-- realiza 4 multiplicaciones entre dos vectores de 8 y 4 bits.
ENTITY multiplicacion IS
  port(
    entrada1 : in std_logic_vector (7 downto 0);  --Multiplicación 1: Factor 1.
    entrada2 : in std_logic_vector (3 downto 0);  --Multiplicación 1: Factor 2.
    entrada3 : in std_logic_vector (7 downto 0);  --Multiplicación 2: Factor 1.
    entrada4 : in std_logic_vector (3 downto 0);  --Multiplicación 2: Factor 2.
    entrada5 : in std_logic_vector (7 downto 0);  --Multiplicación 3: Factor 1.
    entrada6 : in std_logic_vector (3 downto 0);  --Multiplicación 3: Factor 2.
    entrada7 : in std_logic_vector (7 downto 0);  --Multiplicación 4: Factor 1.
    entrada8 : in std_logic_vector (3 downto 0);  --Multiplicación 4: Factor 2.
    clk      : in std_logic;                      --Reloj.
    reset    : in std_logic;                      --Reset.
    enable   : in std_logic;                      --El proceso comienza a nivel alto.

    flag     : out std_logic;                     --El proceso ha finalizado.
    salida1  : out std_logic_vector (7 downto 0); --Multiplicación 1: Producto.
    salida2  : out std_logic_vector (7 downto 0); --Multiplicación 2: Producto.
    salida3  : out std_logic_vector (7 downto 0); --Multiplicación 3: Producto.
    salida4  : out std_logic_vector (7 downto 0)  --Multiplicación 4: Producto.
  );
END multiplicacion;

ARCHITECTURE multiplicacion_beh OF multiplicacion IS

COMPONENT modulo
  port(
    entrada : in std_logic_vector (10 downto 0); --La multiplicación entre vectores de
                                                  --4 y 8 bits dará un máximo de 11 bits.
    clk     : in std_logic;                      --Reloj.
    reset   : in std_logic;                      --Reset.
    enable  : in std_logic;                      --El proceso comienza a nivel alto.

    flag    : out std_logic;                     --El proceso ha finalizado.
    salida  : out std_logic_vector (7 downto 0)  --Salida de 8 bits que buscamos.
  );
END COMPONENT;
--PORT MAP SIGNALS
signal modulo_in     : std_logic_vector (10 downto 0); --'entrada'.
signal enable_modulo : std_logic;                      --'enable'.
signal flag_modulo   : std_logic;                      --'flag'.
signal modulo_out    : std_logic_vector (7 downto 0);  --'salida'.

--OTRAS SEÑALES.
signal aux0        : std_logic_vector (10 downto 0);
signal aux1        : std_logic_vector (10 downto 0);
signal aux2        : std_logic_vector (10 downto 0);
signal aux3        : std_logic_vector (10 downto 0);
--señales auxiliares necesarias para la primera multiplicación.
signal aux0_2      : std_logic_vector (10 downto 0);
signal aux1_2      : std_logic_vector (10 downto 0);
signal aux2_2      : std_logic_vector (10 downto 0);
signal aux3_2      : std_logic_vector (10 downto 0);
--señales auxiliares necesarias para la segunda multiplicación.
```

```
signal aux0_3      : std_logic_vector (10 downto 0);
signal aux1_3      : std_logic_vector (10 downto 0);
signal aux2_3      : std_logic_vector (10 downto 0);
signal aux3_3      : std_logic_vector (10 downto 0);
--señales auxiliares necesarias para la tercera multiplicación.
signal aux0_4      : std_logic_vector (10 downto 0);
signal aux1_4      : std_logic_vector (10 downto 0);
signal aux2_4      : std_logic_vector (10 downto 0);
signal aux3_4      : std_logic_vector (10 downto 0);
--señales auxiliares necesarias para la cuarta multiplicación.
signal resultado1  : std_logic_vector (10 downto 0);
signal resultado2  : std_logic_vector (10 downto 0);
signal resultado3  : std_logic_vector (10 downto 0);
signal resultado4  : std_logic_vector (10 downto 0);
--resultado de las multiplicaciones antes de pasar por el componente MODULO
signal fin_proceso : std_logic;
--Nos indica que en las salidas se leen los resultados de cada una de las multiplicaciones.
signal contador    : std_logic_vector(1 downto 0); --nos ayudará en la transición de la FSM

type tEstados is (inicio, modulo1, modulo2, modulo3, modulo4, escritura, final);
signal actual, siguiente: tEstados;

BEGIN
  PMmodulo: modulo PORT MAP(modulo_in, clk, reset, enable_modulo, flag_modulo, modulo_out);

  --logica adicional
  flag        <= fin_proceso;

  FSM_SEQ: process (clk, reset)
  BEGIN
    IF reset = '1' THEN
      actual <= inicio;

    ELSIF clk'event AND clk = '1' THEN
      IF enable = '1' THEN
        actual <= siguiente;
      ELSE
        actual <= inicio;
      END IF;

    END IF;
  END process FSM_SEQ;


  FSM_transiciones: process(actual, flag_modulo, contador)
  BEGIN

    CASE actual IS

      WHEN inicio =>

        siguiente <= modulo1;

      WHEN modulo1 =>

        IF flag_modulo='1' THEN
          siguiente <= escritura;
        ELSE
          siguiente <= modulo1;
        END IF;
```

```
      WHEN modulo2 =>

        IF flag_modulo='1' THEN
          siguiente <= escritura;
        ELSE
          siguiente <= modulo2;
        END IF;

      WHEN modulo3 =>

        IF flag_modulo='1' THEN
          siguiente <= escritura;
        ELSE
          siguiente <= modulo3;
        END IF;

      WHEN modulo4 =>

        IF flag_modulo='1' THEN
          siguiente <= escritura;
        ELSE
          siguiente <= modulo4;
        END IF;

      WHEN escritura =>
        IF contador="00" THEN
          siguiente <= modulo2;
        ELSIF contador="01" THEN
          siguiente <= modulo3;
        ELSIF contador="10" THEN
          siguiente <= modulo4;
        ELSIF contador="11" THEN
          siguiente <= final;
        ELSE
          siguiente <=inicio;
        END IF;

      WHEN final =>

        siguiente <= final;

    END CASE;

  END process FSM_transiciones;
```

```
SECUENCIAL: process (clk, reset)
begin
  if reset='1' then
    fin_proceso   <= '0';
    enable_modulo <= '0';
    modulo_in     <= (others=>'0');
    salida1       <= (others=>'0');
    salida2       <= (others=>'0');
    salida3       <= (others=>'0');
    salida4       <= (others=>'0');
    contador      <= (others=>'0');

  elsif clk'event and clk='1' then
    IF enable ='1' THEN

      IF actual = inicio THEN
        contador<="00";

      ELSIF actual = modulo1 THEN

        modulo_in     <= resultado1; --'resultado1' es calculado en el proceso combinacional.
        enable_modulo <= '1';
        IF flag_modulo='1' THEN
          enable_modulo <= '0';
        END IF;

      ELSIF actual = modulo2 THEN

        modulo_in     <= resultado2; --'resultado2' es calculado en el proceso combinacional.
        enable_modulo <= '1';
        IF flag_modulo='1' THEN
          enable_modulo <= '0';
        END IF;

      ELSIF actual = modulo3 THEN

        modulo_in     <= resultado3; --'resultado3' es calculado en el proceso combinacional.
        enable_modulo <= '1';
        IF flag_modulo='1' THEN
          enable_modulo <= '0';
        END IF;

      ELSIF actual = modulo4 THEN

        modulo_in     <= resultado4; --'resultado4' es calculado en el proceso combinacional.
        enable_modulo <= '1';
        IF flag_modulo='1' THEN
          enable_modulo <= '0';
        END IF;

      ELSIF actual = escritura THEN
        IF contador < "11" THEN
          contador<=contador+"01";
        END IF;
        IF contador="00" THEN
          salida1 <= modulo_out;
        ELSIF contador="01" THEN
          salida2 <= modulo_out;
        ELSIF contador="10" THEN
          salida3 <= modulo_out;
        ELSIF contador="11" THEN
          salida4 <= modulo_out;
        END IF;

      ELSIF actual = final THEN

        fin_proceso <= '1';

      END IF;
    ELSE --enable='0'
      fin_proceso <= '0';
      enable_modulo <= '0';
    END IF;
  end if;
end process SECUENCIAL;
```

```
COMBINACIONAL: process (entrada1, entrada2, entrada3, entrada4, entrada5, entrada6,
                        entrada7, entrada8)
begin
-- Es un proceso que realiza la multiplicación de dos vectores de 8 y 4 bits
-- La idea es básica, se empieza cogiendo el bit menos significativo del segundo
--FACTOR y se multiplica por el primer FACTOR, como el resultado es un numero de 8 bits,
--los restantes (10 downto 8) los rellenamos con '0' para que no afecten.
--Posteriormente se coge el segundo bit menos significativo del segundo FACTOR y se
--multiplica por el primer FACTOR, estando el resultado "rotado" una posición a la izquierda.
--Asi sucesivamente, el PRODUCTO es la suma (XOR) de todosesos valores auxiliares.

--EJEMPLO 3bits    101   FACTOR 1 (3 bits)
--                x 011   FACTOR 2 (3 bits)
--                 ______ MULTIPLICACION
--                  101   AUX1 (5 bits) => 00101
--                 101    AUX2 (5 bits) => 01010
--                000     AUX3 (5 bits) => 00000
--               _______ XOR, los huecos restantes de cada AUX se rellenan con '0'.
--               01111   RESULTADO
  IF entrada2(0)='1' then
    aux0(7 downto 0)  <= entrada1;
    aux0(10 downto 8) <= "000";
  ELSIF entrada2(0)='0' then
    aux0(10 downto 0) <= "00000000000";
  END IF;

  IF entrada2(1)='1' then
    aux1(0)           <= '0';
    aux1(8 downto 1)  <= entrada1;
    aux1(10 downto 9) <= "00";
  ELSIF entrada2(1)='0' then
    aux1(10 downto 0) <= "00000000000";
  END IF;

  IF entrada2(2)='1' then
    aux2(1 downto 0) <= "00";
    aux2(9 downto 2) <= entrada1;
    aux2(10)         <='0';
  ELSIF entrada2(2)='0' then
    aux2(10 downto 0) <= "00000000000";
  END IF;

  IF entrada2(3)='1' then
    aux3(2 downto 0)  <= "000";
    aux3(10 downto 3) <= entrada1;
  ELSIF entrada2(3)='0' then
    aux3(10 downto 0)<="00000000000";
  END IF;
-------------------------------------------------------------------FIN PRIMERA MULTIPLICACION
```

```
  IF entrada4(0)='1' then
    aux0_2(7 downto 0)  <=entrada3;
    aux0_2(10 downto 8) <="000";
  ELSIF entrada4(0)='0' then
    aux0_2(10 downto 0) <= "00000000000";
  END IF;

  IF entrada4(1)='1' then
    aux1_2(0)           <= '0';
    aux1_2(8 downto 1)  <= entrada3;
    aux1_2(10 downto 9) <= "00";
  ELSIF entrada4(1)='0' then
    aux1_2(10 downto 0) <= "00000000000";
  END IF;

  IF entrada4(2)='1' then
    aux2_2(1 downto 0) <= "00";
    aux2_2(9 downto 2) <= entrada3;
    aux2_2(10)         <= '0';
  ELSIF entrada4(2)='0' then
    aux2_2(10 downto 0) <= "00000000000";
  END IF;

  IF entrada4(3)='1' then
    aux3_2(2 downto 0)  <= "000";
    aux3_2(10 downto 3) <= entrada3;
  ELSIF entrada4(3)='0' then
    aux3_2(10 downto 0) <= "00000000000";
  END IF;
 ----------------------------------------------------------------FIN SEGUNDA MULTIPLICACION
  IF entrada6(0)='1' then
    aux0_3(7 downto 0)  <= entrada5;
    aux0_3(10 downto 8) <="000";
  ELSIF entrada6(0)='0' then
    aux0_3(10 downto 0) <= "00000000000";
  END IF;

  IF entrada6(1)='1' then
    aux1_3(0)           <= '0';
    aux1_3(8 downto 1)  <= entrada5;
    aux1_3(10 downto 9) <= "00";
  ELSIF entrada6(1)='0' then
    aux1_3(10 downto 0) <= "00000000000";
  END IF;

  IF entrada6(2)='1' then
    aux2_3(1 downto 0) <= "00";
    aux2_3(9 downto 2) <= entrada5;
    aux2_3(10)         <='0';
  ELSIF entrada6(2)='0' then
    aux2_3(10 downto 0) <= "00000000000";
  END IF;

  IF entrada6(3)='1' then
    aux3_3(2 downto 0)  <= "000";
    aux3_3(10 downto 3) <= entrada5;
  ELSIF entrada6(3)='0' then
    aux3_3(10 downto 0) <= "00000000000";
  END IF;
 ----------------------------------------------------------------FIN TERCERA MULTIPLICACION
```

```
  IF entrada8(0)='1' then
    aux0_4(7 downto 0)  <= entrada7;
    aux0_4(10 downto 8) <= "000";
  ELSIF entrada8(0)='0' then
    aux0_4(10 downto 0) <= "00000000000";
  END IF;

  IF entrada8(1)='1' then
    aux1_4(0)           <= '0';
    aux1_4(8 downto 1)  <= entrada7;
    aux1_4(10 downto 9) <= "00";
  ELSIF entrada8(1)='0' then
    aux1_4(10 downto 0) <= "00000000000";
  END IF;

  IF entrada8(2)='1' then
    aux2_4(1 downto 0) <= "00";
    aux2_4(9 downto 2) <= entrada7;
    aux2_4(10)         <= '0';
  ELSIF entrada8(2)='0' then
    aux2_4(10 downto 0) <= "00000000000";
  END IF;

  IF entrada8(3)='1' then
    aux3_4(2 downto 0)  <= "000";
    aux3_4(10 downto 3) <= entrada7;
  ELSIF entrada8(3)='0' then
    aux3_4(10 downto 0) <= "00000000000";
  END IF;
-----------------------------------------------------------------------FIN CUARTA MULTIPLICACION
END process COMBINACIONAL;

 --lógica adicional
 resultado1 <= aux0   xor aux1   xor   aux2 xor aux3;
 resultado2 <= aux0_2 xor aux1_2 xor aux2_2 xor aux3_2;
 resultado3 <= aux0_3 xor aux1_3 xor aux2_3 xor aux3_3;
 resultado4 <= aux0_4 xor aux1_4 xor aux2_4 xor aux3_4;

END multiplicacion_beh;
```

10.1.8. Componente Módulo (Multiplicación)

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_unsigned.all;
-- componente modulo de multiplicacion
--Este componente realiza el modulo con el polinomio irreducible de AES, que es "100011011"
--Esto se hace porque la multiplicación entre dos vectores de 8 y 4 bits suele dar un
--vector mayor de 8 bits, hasta un máximo de 11, con el cual no podemos trabajar.
entity modulo is
  port(
    entrada : in std_logic_vector (10 downto 0); --La multiplicación entre vectores de 4 y
                                                   --8 bits dará un máximo de 11 bits.
    clk     : in std_logic;                      --Reloj.
    reset   : in std_logic;                      --Reset.
    enable  : in std_logic;                      --El proceso comienza a nivel alto.

    flag    : out std_logic;                     --El proceso ha finalizado.
    salida  : out std_logic_vector (7 downto 0)  --Salida de 8 bits que buscamos.
  );
end modulo;

ARCHITECTURE modulo_beh OF modulo IS

signal auxiliar  : std_logic_vector (10 downto 0);
--señal en la que se irá almacenando las operaciones realizadas en el proceso.
signal fin_proceso : std_logic;          --Indica que el proceso ha terminado.

type tEstados is (inicio, modbit10, modbit9, modbit8, final);
signal actual, siguiente: tEstados;

BEGIN
  --lógica adicional
  flag       <= fin_proceso;

  FSM_SEQ: process (clk, reset)
  BEGIN
    IF reset = '1' THEN
      actual <= inicio;

    ELSIF clk'event AND clk = '1' THEN
      IF enable = '1' THEN
        actual <= siguiente;
      ELSE
        actual <= inicio;
      END IF;

    END IF;
  END process FSM_SEQ;
```

```
FSM_transiciones: process(actual)
  BEGIN

    CASE actual IS

      WHEN inicio =>

        siguiente <= modbit10;

      WHEN modbit10 =>

        siguiente <= modbit9;

      WHEN modbit9 =>

        siguiente <= modbit8;

      WHEN modbit8 =>

        siguiente <= final;

      WHEN final =>

        siguiente <= final;

    END CASE;

  END process FSM_transiciones;


  SECUENCIAL: process (clk, reset)
  begin
    if reset='1' then
      fin_proceso <= '0';
      salida      <= (others=>'0');
      auxiliar    <= (others=>'0');
    elsif clk'event and clk='1' then
      IF enable ='1' THEN

        IF actual = inicio THEN

          auxiliar <= entrada;

        ELSIF actual = modbit10 THEN

          IF auxiliar(10)='1' THEN
            auxiliar <= auxiliar xor "10001101100";
            -- Para simplificar se añade "0" a las posiciones del polinomio AES que
              --no tienen que realizar operación.
          END IF;

        ELSIF actual = modbit9 THEN

          IF  auxiliar(9)='1' THEN
            auxiliar <= auxiliar xor "01000110110";
          END IF;
```

```
        ELSIF actual = modbit8 THEN

          IF  auxiliar(8)='1' THEN
            auxiliar <= auxiliar xor "00100011011";
          END IF;

        ELSIF actual = final THEN

          salida <= auxiliar(7 downto 0);
          fin_proceso <= '1';

        END IF;
      ELSE --enable='0'
        fin_proceso <= '0';
      END IF;
    end if;
  end process SECUENCIAL;

END modulo_beh;
```

10.1.9. Componente AddRoundKey (AES)

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_unsigned.all;
--Componente AddRoundKey

ENTITY addroundkey IS
 PORT(
  addroundkey_in  : in std_logic_vector (127 downto 0); --Entrada del ESTADO.
  clave           : in std_logic_vector (127 downto 0); --Subclave de ronda.

  addroundkey_out : out std_logic_vector (127 downto 0) --ESTADO después de AddRoundKey.
 );
END addroundkey;

ARCHITECTURE addroundkey_beh OF addroundkey IS
BEGIN
 addroundkey_out <= addroundkey_in XOR clave;
END addroundkey_beh;
```

10.1.10. Componente Keygen (AES)

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_unsigned.all;

ENTITY keygen IS
  GENERIC(
    key_length : integer := 128; --Longitud de la clave: 128, 192 y 256.
    nb         : integer := 4;   --Número de columnas del estado:
                                   --siempre es 4 en el estándar.
    nk         : integer := 4;   --Número de columnas de la clave:
                                   --4 para 128, 6 para 192 y 8 para 256.
    nr         : integer := 10   --Número de rondas:
                                   --10 para 128, 12 para 192 y 14 para 256.
  );
  PORT(
    clave_in  : in std_logic_vector (key_length-1 downto 0); --Clave original.
    ronda     : in std_logic_vector (3 downto 0);            --Ronda del proceso.
    clk       : in std_logic;                                --Reloj.
    reset     : in std_logic;                                --Reset.
    enable    : in std_logic;                                --El proceso comienza cuando
                                                               --está a nivel alto.
    flag      : out std_logic;                          --Indica si ha terminado el proceso.
    clave_out : out std_logic_vector (127 downto 0)   --Nos dará las subclaves dependiendo
                                                        --de la entrada 'ronda'.
  );

END keygen;

ARCHITECTURE keygen_beh OF keygen IS

COMPONENT subbytes IS
  PORT(
    subbytes_in  : in std_logic_vector (127 downto 0); --Entrada del ESTADO.
    clk          : in std_logic;                       --Reloj.
    reset        : in std_logic;                       --Reset.
    enable       : in std_logic;                       --El proceso comienza a nivel alto.
    sel          : in std_logic;                       --Operación
                                                          --SubBytes(0)/InvSubBytes(1).
    flag         : out std_logic;                      --Indica si el proceso ha terminado.
    subbytes_out : out std_logic_vector (127 downto 0) --ESTADO después de operación.
   );
END COMPONENT;
-- PORT MAP SIGNALS
signal transsubbytes_in  : std_logic_vector(127 downto 0); --'subbytes_in'.
signal enable_subbytes   : std_logic;                      --'enable'.
signal sel_subbytes      : std_logic;                      --'sel'.
signal flag_subbytes     : std_logic;                      --'flag'.
signal transsubbytes_out : std_logic_vector(127 downto 0); --'subbytes_out'.

--Otras señales.
signal tabla : std_logic_vector(((nb*(nr+1))*32)-1 downto 0);
-- Se almacenarán todas las subclaves (matriz de clave expandida)
-- key_length=128 => (((4*11)*32)-1) downto 0 => 1407 downto 0 bits necesarios.
-- key_length=192 => (((4*13)*32)-1) downto 0 => 1663 downto 0 bits necesarios.
-- key_length=256 => (((4*15)*32)-1) downto 0 => 1919 downto 0 bits necesarios.
signal i     : std_logic_vector(5 downto 0); --Señal de control de la FSM.
```

```
type tRcon is array(natural range<>) of std_logic_vector(31 downto 0);
constant rcon: tRcon (1 to 10) :=
(x"01000000",x"02000000",x"04000000",x"08000000",x"10000000",
x"20000000",x"40000000",x"80000000",x"1B000000",x"36000000");
-- Constante Rcon.

type tEstados IS(inicio, estado1, estado2, estado3, estado4,
                 estado5, estado6, estado7, estado8, final);
signal actual, siguiente: tEstados;

signal fin_proceso : std_logic; --Indica cuando se han calculado todas las subclaves.
signal temp        : std_logic_vector(31 downto 0); --Se almacenan los cambios en columnas.

BEGIN
  PMSUBBYTES: subbytes PORT MAP (transsubbytes_in, clk, reset, enable_subbytes, sel_subbytes,
                                 flag_subbytes, transsubbytes_out);
  --logica adicional
  flag <= fin_proceso;

  FSM_SEQ: process (clk, reset)
  BEGIN
    IF reset = '1' THEN
      actual <= inicio;

    ELSIF clk'event AND clk = '1' THEN
      IF enable = '1' THEN
        actual <= siguiente;
      ELSE
        actual <= inicio;
      END IF;

    END IF;

  END process FSM_SEQ;


  FSM_transiciones: process(actual, i, flag_subbytes)
  BEGIN

    CASE actual IS

      WHEN inicio =>

        siguiente <= estado1;

      WHEN estado1 =>

        IF i = nk-1 THEN
          siguiente <= estado2;
        ELSE
          siguiente <= estado1;
        END IF;
```

```
      WHEN estado2 =>

        IF conv_integer(i) = nk * (conv_integer(i)/nk) THEN --Si el resto es 0
        -- *Recordatorio: Dividendo = Cociente * Divisor + Resto
          siguiente <= estado3;
        ELSIF conv_integer(i)= nk * (conv_integer(i)/nk) + 4 and key_length=256 THEN
        --Si el resto es 4 y key_length=256.
          siguiente <= estado7;
        ELSE
          siguiente <= estado6;
        END IF;

      WHEN estado3 =>

        siguiente <= estado4;

      WHEN estado4 =>

        IF flag_subbytes = '1' THEN
          siguiente <= estado5;
        ELSE
          siguiente <= estado4;
        END IF;

      WHEN estado5 =>

        siguiente <= estado6;

      WHEN estado6 =>

        IF conv_integer(i)<(nb*(nr+1)-1) THEN --Si aun no se han rellenado todos los valores
                                              --de "tabla"
          siguiente <= estado2;
        ELSE
          siguiente <= final;
        END IF;

      WHEN estado7 => --se alcanza únicamente si key_length=256

        IF flag_subbytes='1' THEN
          siguiente <= estado8;
        ELSE
          siguiente <= estado7;
        END IF;

      WHEN estado8 =>

         siguiente <= estado6;

      WHEN final =>

        siguiente <= final;

    END CASE;

  END process FSM_transiciones;
```

```
SECUENCIAL:     process (clk, reset)
begin
  if reset='1' then
    i                <= (others => '0');
    fin_proceso      <= '0';
    temp             <= (others => '0');
    transsubbytes_in <= (others => '0');
    enable_subbytes  <= '0';
    sel_subbytes     <= '1';
    tabla            <= (others => '0');

  elsif clk'event and clk='1' then
    IF enable='1' THEN

      IF actual = estado1 THEN

        --En este estado se copia la clave original en las primeras posiciones de "tabla"
        tabla((conv_integer(i)*32)+31 downto (conv_integer(i)*32)) <=
        clave_in((key_length-1)-(conv_integer(i)*32) downto
                                                    (key_length-1)-((conv_integer(i)*32)+31));
        -- Almacena, en los 32 bits menos significativos de "tabla" sin ocupar, los 32 bits más
        -- significativos de la clave original que están aun sin copiar.
        -- Ejemplo i=0, key_length=128 =>
        -- tabla((0*32)+31 downto 0*32 <= clave_in((128-1)-(0*32) downto (128-1)-((0*32)+31) =>
        -- tabla(31 downto 0) <= clave_in(127 downto 96);
        i <= i+"000001";

      ELSIF actual = estado2 THEN

        temp <= tabla(((conv_integer(i)-1)*32)+31 downto (conv_integer(i)-1)*32);
        --A "temp" se copia la última columna calculada.

      ELSIF actual = estado3 THEN --Rotación de la columna almacenada.

        temp(31 downto 24) <= temp(23 downto 16);
        temp(23 downto 16) <= temp(15 downto 8);
        temp(15 downto 8)  <= temp(7 downto 0);
        temp(7 downto 0)   <= temp(31 downto 24);

      ELSIF actual = estado4 THEN --Transformación SubBytes de la columna almacenada.
        transsubbytes_in(127 downto 96) <= temp;
        sel_subbytes                    <= '0'; --Operación SubBytes
        enable_subbytes                 <= '1';

      ELSIF actual = estado5 THEN

        temp            <= transsubbytes_out(127 downto 96) XOR rcon(conv_integer(i)/nk);
        enable_subbytes <= '0';

      ELSIF actual = estado6 THEN

        --En la nueva columna de "tabla" se almacena la columna "temp" XOR
          --la columna "nk" posiciones antes.
        tabla((conv_integer(i)*32)+31 downto conv_integer(i)*32) <= temp XOR
                   tabla(((conv_integer(i)-nk)*32)+31 downto (conv_integer(i)-nk)*32);
        i <= i+"000001";

      ELSIF actual = final THEN

        fin_proceso <= '1';
```

```
        ELSIF actual = estado7 THEN --Solo se alcanza si key_length es 256.

          transsubbytes_in(127 downto 96) <= temp;
          sel_subbytes                    <= '0';
          enable_subbytes                 <= '1';

        ELSIF actual = estado8 THEN

          enable_subbytes <= '0';
          temp            <= transsubbytes_out(127 downto 96);

        END IF;

      ELSE --enable='0'
        i      <= "000000";
        fin_proceso <= '0';
      END IF;

    end if;
  end process SECUENCIAL;


  COMBINACIONAL: process (ronda, tabla)
  begin
  -- proceso que establece una salida dependiendo de la entrada 'ronda'
    IF key_length=128 AND ronda>x"a" THEN --Se dan 10 subclaves,
                                          --si preguntamos por una superior, subclave=0
      clave_out <= (others => '0');
    ELSIF key_length=192 AND ronda>x"c" THEN --Se dan 12 subclaves.
      clave_out <= (others => '0');
    ELSIF key_length=256 AND ronda>x"e" THEN --Se dan 14 subclaves.
      clave_out <= (others => '0');
    ELSE
      clave_out(127 downto 96) <=
                   tabla((conv_integer(ronda)*128)+31 downto (conv_integer(ronda)*128)+0);
      clave_out(95 downto 64)  <=
                   tabla((conv_integer(ronda)*128)+63 downto (conv_integer(ronda)*128)+32);
      clave_out(63 downto 32)  <=
                   tabla((conv_integer(ronda)*128)+95 downto (conv_integer(ronda)*128)+64);
      clave_out(31 downto 0)   <=
                   tabla((conv_integer(ronda)*128)+127 downto (conv_integer(ronda)*128)+96);
      --Ejemplo ronda="0001";
      --clave_out(127 downto 96) <= tabla(159 downto 128);
      --clave_out(95 downto 64)  <= tabla(191 downto 160);
      --clave_out(63 downto 32)  <= tabla(223 downto 192);
      --clave_out(31 downto 0)   <= tabla(255 downto 224);
    END IF;
  END process COMBINACIONAL;

END keygen_beh;
```

10.2. Anexo II: Ejemplo de encriptación

En este anexo se expondrá un ejemplo real de encriptación de un documento de texto con clave de 128 bits utilizando el algoritmo descrito en este proyecto.

La clave utilizada es: PFC-algoritmoAES

Texto a encriptar:

```
No toques. No beses. Aprende navegando. Fórmate online. Lee en e-books.
Escolariza a tu bebé a los cero años (en inglés, please) y esterilízalo, Einstein-ízalo, Mozart-ízalo.
Interna a tus ancianos en residencias.
No beses. Esterilízate. No toques. Denuncia a quien toque a tu hijo/a.
No juegues con esto ni con aquello. No te la juegues. ¡Que no juegues!
Sé creativo, que yo te diré cómo hay que hacerlo.
No te diferencies. No decidas. No pienses, limítate a querer consumir lo que te imponen, a temer lo que te acecha,
a pagar lo que te exigen, a conversar sobre lo que apuntan los medios,
a leer por sagas los bestseller de actualidad.
Ve al gimnasio. No fumes. No bebas.
No rías, mejor acude por un módico precio a sesiones de risoterapia (de abrazoterapia no, que te he dicho que no toques).
No salgas sin tu móvil, tu portátil, tu pen drive, tu MP4. Sé el primero en obtener tu Plastic Logic eReader.
Corre, paga, corre, paga, corre al dictado de la todopoderosa banca, de los glamourosos negocios,
de los intereses políticos y financieros internacionales…
No beses. No des la mano. Di hola.
¡Hola!
¿Hay algún ser humano por ahí?
```

Texto encriptado:

```
>©E æ´:ÐÌª Ä  _#ÔùÆ  ²¼Qj pÕ§6+'æ{ ÁÍ9ñ Ð e¿  æ«Ó.,LpàÉF ¾È1Å ]w¸ö_$Ø bØ·
e#£¼²kX  3"ÍKm^ 9 ÛR`XÓ l  ¤e" Zdqp ¥oÍ  DsÀ .è<
êè ¦ÌKÜÞ«¯´¦ (´ -ÊL³{ É«VÜä¶.} >ÊHë+´Þó*m5sjÖø ³%=ß]c3
{¢dfÓõx ö¤ä[ç` ü+mÕ±úKkq ¤m ÑÝ_Ð«f,Bn¦ J`Ý) ÊÜSÐ[ã)æ$@:5l¯@äÜUB (®v¸¦f® ¶å3©%\G G¼°i ¾w Ó,K§+QDjvZ¨¸
 s¶ ïÜü§ z ÜÒ&n?&[Ý C 8»FÐ´y@»çz9'@Có ÈY :%`x)¥×6Äÿ(Ñ A .dÜ¢  ¤¯Öë_oD¯Öà±©6¨&PË I%l¨ßT#$Q4Ü¼ÙÄÞt4\ Ô°·
Ìn81wµÍ4 ê õó 4Â'ÎØþ +Þ~(^.H]
  =ßðù2ò  s jüjx©:ºîµªüU.¡þ"¶Âxg?Ä È ô ed F¤ ì{D$ ÕÁÖ[ µÃ  ¼  )õÆ :$ =ô¡¼¦åYîÞ+)y h LË
 lÉ<p÷Do&ÒyÿÚ ãýõÃ{c IÇF@ìÑMÆÑÑïu írðA
  tt  ¿î¥Ó 6Ñ@@^6 ¬áWp'é®? × >&ÀN ÃKî s  ÄætS  Õ8Aÿ#6 Ü¼o=d×´  Ø"Y `õ0C
(À ¡ª¿ ¶dQo/ªnú`8µØ:íq ju6Ä½j´¢ LD-G ´êó È=b×ââÚû ¡ ²úLÄ. ßvM:W 'Q(CÚ ¶ù ´1v f©â¡Kkók2
ó,©ÍIx· \ ©Ùw ¿ncäÏeI"ÇOo]ï\É 8»dé«Ö'â'û²     AF´ ÒNÁ +°ÁNqÉúrT1v :*©Sa*Bs*ri±ýÜ    Àf¯*ÜtÀ+wÒÿaês«
¿éCÎ æTªX³ø ·&u² Ë¼× ÝýúKN'÷E ÈÕãä:óaîô ýï3ÎªoSyÂ® ³±ð Tÿ¿W.-á~ # X²-J DE   `òîsZ<ÔKæ:E'P lå%  -Ì¦wÍ9¦«GLØþ3
éV   »`Îb *b|» Ú ë¦Ú¦y»Ñ³¿ï¾á(HPK³ /tZé/t1íúÇ$E:][-ÕMY  RZöÖ
´Uîí¡ Iõ®2¥#¶ :æ)fïÀùÃùVaÞ ëêæÀ¶ di`lÞ+Ônn%ï © c9
```

Como vemos, el texto se vuelve completamente ininteligible.

10.3. Anexo III: Testbench

En este anexo se incluye el banco de pruebas que se ha utilizado para comprobar el correcto funcionamiento del algoritmo descrito utilizando los vectores del NIST para cualquier longitud de clave incluidos en el anexo V.

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;
USE std.textio.all;
USE ieee.std_logic_textio.all;

ENTITY test_ficheros IS
GENERIC(
  key_length : integer :=256
);
END test_ficheros;

ARCHITECTURE behavior OF test_ficheros IS

COMPONENT aes
  PORT(
    AES_in : IN  std_logic_vector(31 downto 0);
    clk : IN  std_logic;
    reset : IN  std_logic;
    enable : IN  std_logic;
    mode : IN  std_logic;
    AES_out : OUT  std_logic_vector(127 downto 0);
    flag : OUT std_logic
  );
 END COMPONENT;

   --Inputs
   signal AES_in : std_logic_vector(31 downto 0) := (others => '0');
   signal clk : std_logic := '0';
   signal reset : std_logic := '0';
   signal enable : std_logic := '0';
   signal mode : std_logic := '0';

  --Outputs
   signal AES_out : std_logic_vector(127 downto 0);
   signal flag : std_logic;

   -- Clock period definitions
   constant clk_period : time := 100 ns;


BEGIN

 -- Instantiate the Unit Under Test (UUT)
   uut: aes PORT MAP (
          AES_in => AES_in,
          clk => clk,
          reset => reset,
          enable => enable,
          mode => mode,
          AES_out => AES_out,
          flag => flag
        );

   -- Clock process definitions
   clk_process :process
   begin
     clk <= '0';
     wait for clk_period/2;
     clk <= '1';
     wait for clk_period/2;
   end process;
```

```
 -- Stimulus process
   stim_proc: process
   file input_datos: TEXT open read_mode is "Ruta_del_archivo_de_entrada";
   file output: TEXT open write_mode is "Ruta_del_archivo_de_salida";
   variable buffer_in, buffer_out : line;
   variable entrada: std_logic_vector (key_length+127 downto 0); --key_length de clave y 128 de datos
   variable auxiliar: std_logic_vector (key_length-1 downto 0);
   variable auxiliar: std_logic_vector (127 downto 0);
   begin

     reset <= '0';

     wait for clk_period*2;

     while not endfile(input_datos) loop
       reset  <= '1';
       enable <= '0';
       mode   <= '1';

      wait for clk_period*15;

      readline (input_datos,buffer_in); --la entrada de clave es copiada en buffer_in
      hread (buffer_in, auxiliar); --buffer_in pasa su valor a la variable auxiliar
      wait for clk_period;
      entrada(key_length-1 downto 0):=auxiliar; --auxiliar se copia en "entrada"
      wait for clk_period;
      readline (input_datos,buffer_in); --la entrada de datos es copiada en buffer_in
      hread (buffer_in, auxiliar2); --buffer_in pasa su valor a la variable auxiliar2
      wait for clk_period;
      entrada(key_length+127 downto key_length):=auxiliar2; --auxiliar se copia en "entrada"
      wait for clk_period;

      reset <='0';
      enable<='1';

      for J in 0 to (key_length/32 + 3) LOOP -- se introducen los datos de 32 en 32
        AES_in <= entrada(J*32+31 downto J*32);
        wait for clk_period;
      END LOOP;

      wait until flag='1';
      wait for clk_period*15;

      hwrite (buffer_out, AES_out); --La señal "AES_out" pasa su valor a "buffer_out"
      writeline (output, buffer_out); --"buffer_out" es escrito en el fichero "output"

      wait for clk_period;

    end loop; --fin del fichero

    file_close(input_datos);
    file_close(output);

    wait;
  end process;

END;
```

10.4. Anexo IV: script .do

Para que la simulación postlayout nos proporcione el archivo .vcd necesario para la estimación del consumo era necesario generar un script.
Dicho script se incluye en este anexo.

```
vlib work
vcom -explicit -93 “netgen/par/aes_timesim.vhd”
vcom -explicit -93 “test_ficheros.vhd”
vsim -voptargs=“+acc” -t 1ps -sdfmax “/uut=netgen/par/aes_timesim.sdf”
-lib work work.test_ficheros
do {test_ficheros_wave.tdo}
view wave
view structure
view signals
do {test_ficheros.udo}
vcd file AES_vcd.vcd
vcd add /uut/*
add list /uut/*
run 750 us
vcd checkpoint
```

Tabla 10.1: Script .do

10.5. Anexo V: Vectores de prueba utilizados

En este anexo se incluyen los ficheros de texto que contienen los vectores de prueba del NIST que se han utilizado en este proyecto a través del banco de pruebas del anexo III para verificar la correcta funcionalidad del circuito descrito. También se incluyen las salidas esperadas de cada fichero.

10.5.1. Fichero 1: Entradas Encriptación con clave de 128 bits

00010203050607080A0B0C0D0F101112 CLAVE 1

506812A45F08C889B97F5980038B8359 DATOS 1

14151617191A1B1C1E1F202123242526 CLAVE 2

5C6D71CA30DE8B8B00549984D2EC7D4B DATOS 2

28292A2B2D2E2F30323334353738393A CLAVE 3

53F3F4C64F8616E4E7C56199F48F21F6 DATOS 3

3C3D3E3F41424344464748494B4C4D4E CLAVE 4

A1EB65A3487165FB0F1C27FF9959F703 DATOS 4

50515253555657585A5B5C5D5F606162 CLAVE 5

3553ECF0B1739558B08E350A98A39BFA DATOS 5

64656667696A6B6C6E6F707173747576 CLAVE 6

67429969490B9711AE2B01DC497AFDE8 DATOS 6

78797A7B7D7E7F80828384858788898A CLAVE 7

93385C1F2AEC8BED192F5A8E161DD508 DATOS 7

8C8D8E8F91929394969798999B9C9D9E CLAVE 8

B5BF946BE19BEB8DB3983B5F4C6E8DDB DATOS 8

A0A1A2A3A5A6A7A8AAABACADAFB0B1B2 CLAVE 9

41321EE10E21BD907227C4450FF42324 DATOS 9

B4B5B6B7B9BABBBCBEBFC0C1C3C4C5C6 CLAVE 10

00A82F59C91C8486D12C0A80124F6089 DATOS 10

C8C9CACBCDCECFD0D2D3D4D5D7D8D9DA CLAVE 11

7CE0FD076754691B4BBD9FAF8A1372FE DATOS 11

DCDDDEDFE1E2E3E4E6E7E8E9EBECEDEE CLAVE 12

23605A8243D07764541BC5AD355B3129 DATOS 12

F0F1F2F3F5F6F7F8FAFBFCFDFE010002 CLAVE 13

12A8CFA23EA764FD876232B4E842BC44 DATOS 13

04050607090A0B0C0E0F101113141516 CLAVE 14

BCAF32415E8308B3723E5FDD853CCC80 DATOS 14

2C2D2E2F31323334363738393B3C3D3E CLAVE 15

89AFAE685D801AD747ACE91FC49ADDE0 DATOS 15

40414243454647484A4B4C4D4F505152 CLAVE 16

F521D07B484357C4A69E76124A634216 DATOS 16

54555657595A5B5C5E5F606163646566 CLAVE 17

3E23B3BC065BCC152407E23896D77783 DATOS 17

68696A6B6D6E6F70727374757778797A CLAVE 18

79F0FBA002BE1744670E7E99290D8F52 DATOS 18

7C7D7E7F81828384868788898B8C8D8E CLAVE 19

DA23FE9D5BD63E1D72E3DAFBE21A6C2A DATOS 19

A4A5A6A7A9AAABACAEAFB0B1B3B4B5B6 CLAVE 20

E3F5698BA90B6A022EFD7DB2C7E6C823 DATOS 20

E0E1E2E3E5E6E7E8EAEBECEDEFF0F1F2 CLAVE 21

BDC2691D4F1B73D2700679C3BCBF9C6E DATOS 21

08090A0B0D0E0F101213141517181 91A CLAVE 22

BA74E02093217EE1BA1B42BD5624349A DATOS 22

6C6D6E6F71727374767778797B7C7D7E CLAVE 23

B5C593B5851C57FBF8B3F57715E8F680 DATOS 23

80818283858687888A8B8C8D8F909192 CLAVE 24

3DA9BD9CEC072381788F9387C3BBF4EE DATOS 24

94959697999A9B9C9E9FA0A1A3A4A5A6 CLAVE 25

4197F3051121702AB65D316B3C637374 DATOS 25

A8A9AAABADAEAFB0B2B3B4B5B7B8B9BA CLAVE 26

9F46C62EC4F6EE3F6E8C62554BC48AB7 DATOS 26

BCBDBEBFC1C2C3C4C6C7C8C9CBCCCDCE CLAVE 27

0220673FE9E699A4EBC8E0DBEB6979C8 DATOS 27

D0D1D2D3D5D6D7D8DADBDCDDDFE0E1E2 CLAVE 28

B2B99171337DED9BC8C2C23FF6F18867 DATOS 28

E4E5E6E7E9EAEBECEEEFF0F1F3F4F5F6 CLAVE 29

A7FACF4E301E984E5EFEEFD645B23505 DATOS 29

F8F9FAFBFDFEFE00020304050708090A CLAVE 30

F7C762E4A9819160FD7ACFB6C4EEDCDD DATOS 30

0C0D0E0F11121314161718191B1C1D1E CLAVE 31

9B64FC21EA08709F4915436FAA70F1BE DATOS 31

20212223252627282A2B2C2D2F303132 CLAVE 32

52AF2C3DE07EE6777F55A4ABFC100B3F DATOS 32

34353637393A3B3C3E3F404143444546 CLAVE 33

2FCA001224386C57AA3F968CBE2C816F DATOS 33

48494A4B4D4E4F50525354555758595A CLAVE 34

4149C73658A4A9C564342755EE2C132F DATOS 34

5C5D5E5F61626364666768696B6C6D6E CLAVE 35

AF60005A00A1772F7C07A48A923C23D2 DATOS 35

70717273757677787A7B7C7D7F808182 CLAVE 36

6FCCBC28363759914B6F0280AFAF20C6 DATOS 36

84858687898A8B8C8E8F909193949596 CLAVE 37

7D82A43DDF4FEFA2FC5947499884D386 DATOS 37

98999A9B9D9E9FA0A2A3A4A5A7A8A9AA CLAVE 38

5D5A990EAAB9093AFE4CE254DFA49EF9 DATOS 38

ACADAEAFB1B2B3B4B6B7B8B9BBBCBDBE CLAVE 39

4CD1E2FD3F4434B553AAE453F0ED1A02 DATOS 39

C0C1C2C3C5C6C7C8CACBCCCDCFD0D1D2 CLAVE 40

5A2C9A9641D4299125FA1B9363104B5E DATOS 40

D4D5D6D7D9DADBDCDEDFE0E1E3E4E5E6 CLAVE 41

B517FE34C0FA217D341740BFD4FE8DD4 DATOS 41

E8E9EAEBEDEEEFF0F2F3F4F5F7F8F9FA CLAVE 42

014BAF2278A69D331D5180103643E99A DATOS 42

FCFDFEFF01020304060708090B0C0D0E CLAVE 43

B529BD8164F20D0AA443D4932116841C DATOS 43

10111213151617181A1B1C1D1F202122 CLAVE 44

2E596DCBB2F33D4216A1176D5BD1E456 DATOS 44

24252627292A2B2C2E2F303133343536 CLAVE 45

7274A1EA2B7EE2424E9A0E4673689143 DATOS 45

38393A3B3D3E3F40424344454748494A CLAVE 46

AE20020BD4F13E9D90140BEE3B5D26AF DATOS 46

4C4D4E4F51525354565758595B5C5D5E CLAVE 47

BAAC065DA7AC26E855E79C8849D75A02 DATOS 47

60616263656667686A6B6C6D6F707172 CLAVE 48

7C917D8D1D45FAB9E2540E28832540CC DATOS 48

74757677797A7B7C7E7F808183848586 CLAVE 49

BDE6F89E16DAADB0E847A2A614566A91 DATOS 49

88898A8B8D8E8F90929394959798999A CLAVE 50

C9DE163725F1F5BE44EBB1DB51D07FBC DATOS 50

9C9D9E9FA1A2A3A4A6A7A8A9ABACADAE CLAVE 51

3AF57A58F0C07DFFA669572B521E2B92 DATOS 51

B0B1B2B3B5B6B7B8BABBBCBDBFC0C1C2 CLAVE 52

3D5EBAC306DDE4604F1B4FBBBFCDAE55 DATOS 52

C4C5C6C7C9CACBCCCECFD0D1D3D4D5D6 CLAVE 53

C2DFA91BCEB76A1183C995020AC0B556 DATOS 53

D8D9DADBDDDEDFE0E2E3E4E5E7E8E9EA CLAVE 54

C70F54305885E9A0746D01EC56C8596B DATOS 54

ECEDEEEFF1F2F3F4F6F7F8F9FBFCFDFE CLAVE 55

C4F81B610E98012CE000182050C0C2B2 DATOS 55

00010203050607080A0B0C0D0F101112 CLAVE 56

EAAB86B1D02A95D7404EFF67489F97D4 DATOS 56

14151617191A1B1C1E1F202123242526 CLAVE 57

7C55BDB40B88870B52BEC3738DE82886 DATOS 57

28292A2B2D2E2F30323334353738393A CLAVE 58

BA6EAA88371FF0A3BD875E3F2A975CE0 DATOS 58

3C3D3E3F41424344464748494B4C4D4E CLAVE 59

08059130C4C24BD30CF0575E4E0373DC DATOS 59

50515253555657585A5B5C5D5F606162 CLAVE 60

9A8EAB004EF53093DFCF96F57E7EDA82 DATOS 60

64656667696A6B6C6E6F707173747576 CLAVE 61

0745B589E2400C25F117B1D796C28129 DATOS 61

78797A7B7D7E7F80828384858788898A CLAVE 62

2F1777781216CEC3F044F134B1B92BBE DATOS 62

8C8D8E8F91929394969798999B9C9D9E CLAVE 63

353A779FFC541B3A3805D90CE17580FC DATOS 63

A0A1A2A3A5A6A7A8AAABACADAFB0B1B2 CLAVE 64

1A1EAE4415CEFCF08C4AC1C8F68BEA8F DATOS 64

B4B5B6B7B9BABBBCBEBFC0C1C3C4C5C6 CLAVE 65

E6E7E4E5B0B3B2B5D4D5AAAB16111013 DATOS 65

C8C9CACBCDCECFD0D2D3D4D5D7D8D9DA CLAVE 66

F8F9FAFBFBF8F9E677767170EFE0E1E2 DATOS 66

DCDDDEDFE1E2E3E4E6E7E8E9EBECEDEE CLAVE 67

63626160A1A2A3A445444B4A75727370 DATOS 67

F0F1F2F3F5F6F7F8FAFBFCFDFE010002 CLAVE 68

717073720605040B2D2C2B2A05FAFBF9 DATOS 68

04050607090A0B0C0E0F101113141516 CLAVE 69

78797A7BEAE9E8EF3736292891969794 DATOS 69

18191A1B1D1E1F20222324252728292A CLAVE 70

838281803231300FDDDCDBDAA0AFAEAD DATOS 70

2C2D2E2F31323334363738393B3C3D3E CLAVE 71

18191A1BBFBCBDBA75747B7A7F78797A DATOS 71

40414243454647484A4B4C4D4F505152 CLAVE 72

848586879B989996A3A2A5A4849B9A99 DATOS 72

54555657595A5B5C5E5F606163646566 CLAVE 73

00010203222122027CACBF4F551565754 DATOS 73

68696A6B6D6E6F70727374757778797A CLAVE 74

CECFCCCDAFACADB2515057564A454447 DATOS 74

7C7D7E7F81828384868788898B8C8D8E CLAVE 75

92939091CDCECFC813121D1C80878685 DATOS 75

90919293959697989A9B9C9D9FA0A1A2 CLAVE 76

D2D3D0D16F6C6D6259585F5ED1EEEFEC DATOS 76

A4A5A6A7A9AAABACAEAFB0B1B3B4B5B6 CLAVE 77

ACADAEAF878485820F0E1110D5D2D3D0 DATOS 77

B8B9BABBBDBEBFC0C2C3C4C5C7C8C9CA CLAVE 78

9091929364676619E6E7E0E1757A7B78 DATOS 78

CCCDCECFD1D2D3D4D6D7D8D9DBDCDDDE CLAVE 79

BABBB8B98A89888F74757A7B92959497 DATOS 79

E0E1E2E3E5E6E7E8EAEBECEDEFF0F1F2 CLAVE 80

8D8C8F8E6E6D6C633B3A3D3CCAD5D4D7 DATOS 80

F4F5F6F7F9FAFBFCFEFE010103040506 CLAVE 81

8687848501020304080 8F7F767606162 DATOS 81

08090A0B0D0E0F10121314151718191A CLAVE 82

8E8F8C8D656667788A8B8C8D010E0F0C DATOS 82

1C1D1E1F21222324262728292B2C2D2E CLAVE 83

C8C9CACB858687807A7B7475E7E0E1E2 DATOS 83

30313233353637383A3B3C3D3F404142 CLAVE 84

6D6C6F6E5053525D8C8D8A8BADD2D3D0 DATOS 84

44454647494A4B4C4E4F505153545556 CLAVE 85

28292A2B393A3B3C0607181903040506 DATOS 85

58595A5B5D5E5F60626364656768696A CLAVE 86

A5A4A7A6B0B3B28DDBDADDDCBDB2B3B0 DATOS 86

6C6D6E6F71727374767778797B7C7D7E CLAVE 87

323330316467666130313E3F2C2B2A29 DATOS 87

80818283858687888A8B8C8D8F909192 CLAVE 88

27262524080B0A05171611100B141516 DATOS 88

94959697999A9B9C9E9FA0A1A3A4A5A6 CLAVE 89

040506074142434435340B0AA3A4A5A6 DATOS 89

A8A9AAABADAEAFB0B2B3B4B5B7B8B9BA CLAVE 90

242526271112130C61606766BDB2B3B0 DATOS 90

BCBDBEBFC1C2C3C4C6C7C8C9CBCCCDCE CLAVE 91

4B4A4948252627209E9F9091CEC9C8CB DATOS 91

D0D1D2D3D5D6D7D8DADBDCDDDFE0E1E2 CLAVE 92

68696A6B6665646B9F9E9998D9E6E7E4 DATOS 92

E4E5E6E7E9EAEBECEEEFF0F1F3F4F5F6 CLAVE 93

34353637C5C6C7C0F0F1EEEF7C7B7A79 DATOS 93

F8F9FAFBFDFEFE00020304050708090A CLAVE 94

32333031C2C1C13F0D0C0B0A050A0B08 DATOS 94

0C0D0E0F11121314161718191B1C1D1E CLAVE 95

CDCCCFCEBEBDBCBBABAAA5A4181F1E1D DATOS 95

20212223252627282A2B2C2D2F303132 CLAVE 96

212023223635343BA0A1A6A7445B5A59 DATOS 96

34353637393A3B3C3E3F404143444546 CLAVE 97

0E0F0C0DA8ABAAAD2F2E515002050407 DATOS 97

48494A4B4D4E4F50525354555758595A CLAVE 98

070605042A2928378E8F8889BDB2B3B0 DATOS 98

5C5D5E5F61626364666768696B6C6D6E CLAVE 99

CBCAC9C893909196A9A8A7A6A5A2A3A0 DATOS 99

70717273757677787A7B7C7D7F808182 CLAVE 100

80818283C1C2C3CC9C9D9A9B0CF3F2F1 DATOS 100

10.5.2. Fichero 1: Salidas Encriptación con clave de 128 bits

D8F532538289EF7D06B506A4FD5BE9C9 SALIDA 1
59AB30F4D4EE6E4FF9907EF65B1FB68C SALIDA 2
BF1ED2FCB2AF3FD41443B56D85025CB1 SALIDA 3
7316632D5C32233EDCB0780560EAE8B2 SALIDA 4
408C073E3E2538072B72625E68B8364B SALIDA 5
E1F94DFA776597BEACA262F2F6366FEA SALIDA 6
F29E986C6A1C27D7B29FFD7EE92B75F1 SALIDA 7
131C886A57F8C2E713ABA6955E2B55B5 SALIDA 8
D2AB7662DF9B8C740210E5EEB61C199D SALIDA 9
14C10554B2859C484CAB5869BBE7C470 SALIDA 10
DB4D498F0A49CF55445D502C1F9AB3B5 SALIDA 11
6D96FEF7D66590A77A77BB2056667F7F SALIDA 12

316FB68EDBA736C53E78477BF913725C SALIDA 13
6936F2B93AF8397FD3A771FC011C8C37 SALIDA 14
F3F92F7A9C59179C1FCC2C2BA0B082CD SALIDA 15
6A95EA659EE3889158E7A9152FF04EBC SALIDA 16
1959338344E945670678A5D432C90B93 SALIDA 17
E49BDDD2369B83EE66E6C75A1161B394 SALIDA 18
D3388F19057FF704B70784164A74867D SALIDA 19
23AA03E2D5E4CD24F3217E596480D1E1 SALIDA 20
C84113D68B666AB2A50A8BDB222E91B9 SALIDA 21
AC02403981CD4340B507963DB65CB7B6 SALIDA 22
8D129923622335947401 1F6BF5088414 SALIDA 23
5A1D6AB8605505F7977E55B9A54D9B90 SALIDA 24
72E9C2D519CF555E4208805AABE3B258 SALIDA 25
A8F3E81C4A23A39EF4D745DFFE026E80 SALIDA 26
546F646449D31458F9EB4EF5483AEE6C SALIDA 27
4DBE4BC84AC797C0EE4EFB7F1A07401C SALIDA 28
25E10BFB411BBD4D625AC8795C8CA3B3 SALIDA 29
315637405054EC803614E43DEF177579 SALIDA 30
60C5BC8A1410247295C6386C59E572A8 SALIDA 31
01366FC8CA52DFE055D6A00A76471BA6 SALIDA 32
ECC46595516EC612449C3F581E7D42FF SALIDA 33
6B7FFE4C602A154B06EE9C7DAB5331C9 SALIDA 34
7DA234C14039A240DD02DD0FBF84EB67 SALIDA 35
C7DC217D9E3604FFE7E91F080ECD5A3A SALIDA 36
37785901863F5C81260EA41E7580CDA5 SALIDA 37
A07B9338E92ED105E6AD720FCCCE9FE4 SALIDA 38
AE0FB9722418CC21A7DA816BBC61322C SALIDA 39
C826A193080FF91FFB21F71D3373C877 SALIDA 40
1181B11B0E494E8D8B0AA6B1D5AC2C48 SALIDA 41
6743C3D1519AB4F2CD9A78AB09A511BD SALIDA 42
DC55C076D52BACDF2EEFD952946A439D SALIDA 43
711B17B590FFC72B5C8E342B601E8003 SALIDA 44
19983BB0950783A537E1339F4AA21C75 SALIDA 45
3BA7762E15554169C0F4FA39164C410C SALIDA 46
A0564C41245AFCA7AF8AA2E0E588EA89 SALIDA 47
5E36A42A2E099F54AE85ECD92E2381ED SALIDA 48
770036F878CD0F6CA2268172F106F2FE SALIDA 49
7E4E03908B716116443CCF7C94E7C259 SALIDA 50
482735A48C30613A242DD494C7F9185D SALIDA 51
B4C0F6C9D4D7079ADDF9369FC081061D SALIDA 52
D5810FE0509AC53EDCD74F89962E6270 SALIDA 53
03F17A16B3F91848269ECDD38EBB2165 SALIDA 54
DA1248C3180348BAD4A93B4D9856C9DF SALIDA 55
3D10D7B63F3452C06CDF6CCE18BE0C2C SALIDA 56
4AB823E7477DFDDC0E6789018FCB6258 SALIDA 57
E6478BA56A77E70CFDAA5C843ABDE30E SALIDA 58
1673064895FBEAF7F09C5429FF75772D SALIDA 59
4488033AE9F2EFD0CA9383BFCA1A94E9 SALIDA 60
978F3B8C8F9D6F46626CAC3C0BCB9217 SALIDA 61
E08C8A7E582E15E5527F1D9E2EECB236 SALIDA 62
CEC155B76AC5FFDA4CF4F9CA91E49A7A SALIDA 63
D5AC7165763225DD2A38CDC6862C29AD SALIDA 64
03680FE19F7CE7275452020BE70E8204 SALIDA 65
461DF740C9781C388E94BB861CEB54F6 SALIDA 66
451BD60367F96483042742219786A074 SALIDA 67
E4DFA42671A02E57EF173B85C0EA9F2B SALIDA 68
ED11B89E76274282227D854700A78B9E SALIDA 69
433946EAA51EA47AF33895F2B90B3B75 SALIDA 70
6BC6D616A5D7D0284A5910AB35022528 SALIDA 71
D2A920ECFE919D354B5F49EAE9719C98 SALIDA 72
3A061B17F6A92885EFBD0676985B373D SALIDA 73
FADEEC16E33EA2F4688499D157E20D8F SALIDA 74
5CDEFEDE59601AA3C3CDA36FA6B1FA13 SALIDA 75
9574B00039844D92EBBA7EE8719265F8 SALIDA 76
9A9CF33758671787E5006928188643FA SALIDA 77
2CDDD634C846BA66BB46CBFEA4A674F9 SALIDA 78
D28BAE029393C3E7E26E9FAFBBB4B98F SALIDA 79
EC27529B1BEE0A9AB6A0D73EBC82E9B7 SALIDA 80
3CB25C09472AFF6EE7E2B47CCD7CCB17 SALIDA 81
DEE33103A7283370D725E44CA38F8FE5 SALIDA 82
27F9BCD1AAC64BFFC11E7815702C1A69 SALIDA 83
5DF534FFAD4ED0749A9988E9849D0021 SALIDA 84
A48BEE75DB04FB60CA2B80F752A8421B SALIDA 85
024C8CF70BC86EE5CE03678CB7AF45F9 SALIDA 86
3C19AC0F8A3A3862CE577831301E166B SALIDA 87
C5E355B796A57421D59CA6BE82E73BCA SALIDA 88
D94033276417ABFB05A69D15B6E386E2 SALIDA 89
24B36559EA3A9B9B958FE6DA3E5B8D85 SALIDA 90
20FD4FEAA0E8BF0CCE7861D74EF4CB72 SALIDA 91
350E20D5174277B9EC314C501570A11D SALIDA 92
87A29D61B7C604D238FE73045A7EFD57 SALIDA 93
2C3164C1CC7D0064816BDC0FAA362C52 SALIDA 94
195FE5E8A05A2ED594F6E4400EEE10B3 SALIDA 95

E4663DF19B9A21A5A284C2BD7F905025 SALIDA 96
21B88714CFB4E2A933BD281A2C4743FD SALIDA 97
CBFC3980D704FD0FC54378AB84E17870 SALIDA 98
BC5144BAA48BDEB8B63E22E03DA418EF SALIDA 99
5A1DBAEF1EE2984B8395DA3BDFFA3CCC SALIDA 100

10.5.3. Fichero 2: Entradas Encriptación con clave de 192 bits

00010203050607080A0B0C0D0F10111214151617191A1B1C CLAVE 1
2D33EEF2C0430A8A9EBF45E809C40BB6 DATOS 1
1E1F20212324252628292A2B2D2E2F30323334353738393A CLAVE 2
6AA375D1FA155A61FB72353E0A5A8756 DATOS 2
3C3D3E3F41424344464748494B4C4D4E5051525355565758 CLAVE 3
BC3736518B9490DCB8ED60EB26758ED4 DATOS 3
5A5B5C5D5F60616264656667696A6B6C6E6F707173747576 CLAVE 4
AA214402B46CFFB9F761EC11263A311E DATOS 4
78797A7B7D7E7F808283848587888 98A8C8D8E8F91929394 CLAVE 5
02AEA86E572EEAB66B2C3AF5E9A46FD6 DATOS 5
969798999B9C9D9EA0A1A2A3A5A6A7A8AAABACADAFB0B1B2 CLAVE 6
E2AEF6ACC33B965C4FA1F91C75FF6F36 DATOS 6
B4B5B6B7B9BABBBCBEBFC0C1C3C4C5C6C8C9CACBCDCECFD0 CLAVE 7
0659DF46427162B9434865DD9499F91D DATOS 7
D2D3D4D5D7D8D9DADCDDDEDFE1E2E3E4E6E7E8E9EBECEDEE CLAVE 8
49A44239C748FEB456F59C276A5658DF DATOS 8
F0F1F2F3F5F6F7F8FAFBFCFDFE0100020405060709 0A0B0C CLAVE 9
66208F6E9D04525BDEDB2733B6A6BE37 DATOS 9
0E0F10111314151618191A1B1D1E1F20222324252728292A CLAVE 10
3393F8DFC729C97F5480B950BC9666B0 DATOS 10
2C2D2E2F31323334363738393B3C3D3E4041424345464748 CLAVE 11
606834C8CE063F3234CF1145325DBD71 DATOS 11
4A4B4C4D4F50515254555657595A5B5C5E5F606163646566 CLAVE 12
FEC1C04F529BBD17D8CECFCC4718B17F DATOS 12
68696A6B6D6E6F707273747577778797A7C7D7E7F81828384 CLAVE 13
32DF99B431ED5DC5ACF8CAF6DC6CE475 DATOS 13
868788898B8C8D8E90919293959697989A9B9C9D9FA0A1A2 CLAVE 14
7FDC2B746F3F665296943B83710D1F82 DATOS 14
A4A5A6A7A9AAABACAEAFB0B1B3B4B5B6B8B9BABBBDBEBFC0 CLAVE 15
8FBA1510A3C5B87E2EAA3F7A91455CA2 DATOS 15
C2C3C4C5C7C8C9CACCCDCECFD1D2D3D4D6D7D8D9DBDCDDDE CLAVE 16
2C9B468B1C2EED92578D41B0716B223B DATOS 16
E0E1E2E3E5E6E7E8EAEBECEDEFF0F1F2F4F5F6F7F9FAFBFC CLAVE 17
0A2BBF0EFC6BC0034F8A03433FCA1B1A DATOS 17
FEFE01010304050608090A0B0D0E0F101213141517181 91A CLAVE 18
25260E1F31F4104D387222E70632504B DATOS 18
1C1D1E1F21222324262728292B2C2D2E3031323335363738 CLAVE 19
C527D25A49F08A5228D338642AE65137 DATOS 19
3A3B3C3D3F40414244454647494A4B4C4E4F505153545556 CLAVE 20
3B49FC081432F5890D0E3D87E884A69E DATOS 20
58595A5B5D5E5F60626364656768696A6C6D6E6F71727374 CLAVE 21
D173F9ED1E57597E166931DF2754A083 DATOS 21
767778797B7C7D7E808182838586878 88A8B8C8D8F909192 CLAVE 22
8C2B7CAFA5AFE7F13562DAEAE1ADEDE0 DATOS 22
94959697999A9B9C9E9FA0A1A3A4A5A6A8A9AAABADAEAFB0 CLAVE 23
AAF4EC8C1A815AEB826CAB741339532C DATOS 23
D0D1D2D3D5D6D7D8DADBDCDDDFE0E1E2E4E5E6E7E9EAEBEC CLAVE 24
40BE8C5D9108E663F38F1A2395279ECF DATOS 24
2A2B2C2D2F30313234353637393A3B3C3E3F404143444546 CLAVE 25
0C8AD9BC32D43E04716753AA4CFBE351 DATOS 25
48494A4B4D4E4F5052535455575858595A5C5D5E5F61626364 CLAVE 26
1407B1D5F87D63357C8DC7EBBAEBBFEE DATOS 26
84858687898A8B8C8E8F90919394959698999A9B9D9E9FA0 CLAVE 27
E62734D1AE3378C4549E939E6F123416 DATOS 27
A2A3A4A5A7A8A9AAACADAEAFB1B2B3B4B6B7B8B9BBBCBDBE CLAVE 28
5A752CFF2A176DB1A1DE77F2D2CDEE41 DATOS 28
C0C1C2C3C5C6C7C8CACBCCCDCFD0D1D2D4D5D6D7D9DADBDC CLAVE 29
A9C8C3A4EABEDC80C64730DDD018CD88 DATOS 29
1A1B1C1D1F20212224252627292A2B2C2E2F303133343536 CLAVE 30
EE9B3DBBDB86180072130834D305999A DATOS 30
38393A3B3D3E3F404243444547484 94A4C4D4E4F51525354 CLAVE 31
A7FA8C3586B8EBDE7568EAD6F634A879 DATOS 31
929394959798999A9C9D9E9FA1A2A3A4A6A7A8A9ABACADAE CLAVE 32
37E0F4A87127D45AC936FE7AD88C10A DATOS 32
464748494B4C4D4E50515253555657585A5B5C5D5F606162 CLAVE 33
3F77D8B5D92BAC148E4E46F697A535C5 DATOS 33
828384858788898A8C8D8E8F91929394969798999B9C9D9E CLAVE 34
D25EBB686C40F7E2C4DA1014936571CA DATOS 34
A0A1A2A3A5A6A7A8AAABACADAFB0B1B2B4B5B6B7B9BABBBC CLAVE 35

4F1C769D1E5B0552C7ECA84DEA26A549 DATOS 35
BEBFC0C1C3C4C5C6C8C9CACBCDCECFD0D2D3D4D5D7D8D9DA CLAVE 36
8548E2F882D7584D0FAFC54372B6633A DATOS 36
DCDDDEDFE1E2E3E4E6E7E8E9EBECEDEEF0F1F2F3F5F6F7F8 CLAVE 37
87D7A336CB476F177CD2A51AF2A62CDF DATOS 37
FAFBFCFDFE0100020405060709 0A0B0C0E0F101113141516 CLAVE 38
03B1FEAC668C4E485C1065DFC22B44EE DATOS 38
18191A1B1D1E1F20222324252728292A2C2D2E2F31323334 CLAVE 39
BDA15E66819FA72D653A6866AA287962 DATOS 39
363738393B3C3D3E40414243454647484A4B4C4D4F505152 CLAVE 40
4D0C7A0D2505B80BF8B62CEB12467F0A DATOS 40
54555657595A5B5C5E5F60616364656668696A6B6D6E6F70 CLAVE 41
626D34C9429B37211330986466B94E5F DATOS 41
727374757778797A7C7D7E7F818283848687888 98B8C8D8E CLAVE 42
333C3E6BF00656B088A17E5FF0E7F60A DATOS 42
909192939596979899A9B9C9D9FA0A1A2A4A5A6A7A9AAABAC CLAVE 43
687ED0CDC0D2A2BC8C466D05EF9D2891 DATOS 43
AEAFB0B1B3B4B5B6B8B9BABBBDBEBFC0C2C3C4C5C7C8C9CA CLAVE 44
487830E78CC56C1693E64B2A6660C7B6 DATOS 44
CCCDCECFD1D2D3D4D6D7D8D9DBDCDDDEE0E1E2E3E5E6E7E8 CLAVE 45
7A48D6B7B52B29392AA2072A32B66160 DATOS 45
EAEBECEDEFF0F1F2F4F5F6F7F9FAFBFCFEFE010103040506 CLAVE 46
907320E64C8C5314D10F8D7A11C8618D DATOS 46
08090A0B0D0E0F10121314151718191A1C1D1E1F21222324 CLAVE 47
B561F2CA2D6E65A4A98341F3ED9FF533 DATOS 47
262728292B2C2D2E303132333536373 83A3B3C3D3F404142 CLAVE 48
DF769380D212792D026F049E2E3E48EF DATOS 48
44454647494A4B4C4E4F50515354555658595A5B5D5E5F60 CLAVE 49
79F374BC445BDABF8FCCB8843D6054C6 DATOS 49
626364656768696A6C6D6E6F71727374767778797B7C7D7E CLAVE 50
4E02F1242FA56B05C68DBAE8FE44C9D6 DATOS 50
808182838586878 88A8B8C8D8F909192949596979 99A9B9C CLAVE 51
CF73C93CBFF57AC635A6F4AD2A4A1545 DATOS 51
9E9FA0A1A3A4A5A6A8A9AAABADAEAFB0B2B3B4B5B7B8B9BA CLAVE 52
9923548E2875750725B886566784C625 DATOS 52
BCBDBEBFC1C2C3C4C6C7C8C9CBCCCDCED0D1D2D3D5D6D7D8 CLAVE 53
4888336B723A022C9545320F836A4207 DATOS 53
DADBDCDDDFE0E1E2E4E5E6E7E9EAEBECEEEFF0F1F3F4F5F6 CLAVE 54
F84D9A5561B0608B1160DEE000C41BA8 DATOS 54
F8F9FAFBFDFEFE00020304050708090A0C0D0E0F11121314 CLAVE 55
C23192A0418E30A19B45AE3E3625BF22 DATOS 55
161718191B1C1D1E20212223252627282A2B2C2D2F303132 CLAVE 56
B84E0690B28B0025381AD82A15E501A7 DATOS 56
34353637393A3B3C3E3F40414344454648494A4B4D4E4F50 CLAVE 57
ACEF5E5C108876C4F06269F865B8F0B0 DATOS 57
525354555758595A5C5D5E5F61626364666768696B6C6D6E CLAVE 58
0F1B3603E0F5DDEA4548246153A5E064 DATOS 58
70717273757677787A7B7C7D7F808182848586878 98A8B8C CLAVE 59
FBB63893450D42B58C6D88CD3C1809E3 DATOS 59
8E8F909193949596989 99A9B9D9E9FA0A2A3A4A5A7A8A9AA CLAVE 60
4BEF736DF150259DAE0C91354E8A5F92 DATOS 60
ACADAEAFB1B2B3B4B6B7B8B9BBBCBDBEC0C1C2C3C5C6C7C8 CLAVE 61
7D2D46242056EF13D3C3FC93C128F4C7 DATOS 61
CACBCCCDCFD0D1D2D4D5D6D7D9DADBDCDEDFE0E1E3E4E5E6 CLAVE 62
E9C1BA2DF415657A256EDB33934680FD DATOS 62
E8E9EAEBEDEEEFF0F2F3F4F5F7F8F9FAFCFDFEFF01020304 CLAVE 63
E23EE277B0AA0A1DFB81F7527C3514F1 DATOS 63
060708090B0C0D0E10111213151617181A1B1C1D1F202122 CLAVE 64
3E7445B0B63CAAF75E4A911E12106B4C DATOS 64
24252627292A2B2C2E2F303133343536383 93A3B3D3E3F40 CLAVE 65
767774752023222544455A5BE6E1E0E3 DATOS 65
424344454748494A4C4D4E4F515253545 65758595B5C5D5E CLAVE 66
72737475717E7F7CE9E8EBEA696A6B6C DATOS 66
606162636566676 86A6B6C6D6F707172747577677797A7B7C CLAVE 67
DFDEDDDC25262728C9C8CFCEF1EEEFEC DATOS 67
7E7F808183848568 8898A8B8D8E8F90929394959798999A CLAVE 68
FFFE0100707776755F5E5D5C7675746B DATOS 68
9C9D9E9FA1A2A3A4A6A7A8A9ABACADAEB0B1B2B3B5B6B7B8 CLAVE 69
E0E1E2E3424140479F9E9190292E2F2C DATOS 69
BABBBCBDBFC0C1C2C4C5C6C7C9CACBCCCECFD0D1D3D4D5D6 CLAVE 70
2120272690EFEEED3B3A39384E4D4C4B DATOS 70
D8D9DADBDDDEDFE0E2E3E4E5E7E8E9EAECEDEEEFF1F2F3F4 CLAVE 71
ECEDEEEF5350516EA1A0A7A6A3ACADAE DATOS 71
F6F7F8F9FBFCFDFE000102030507080A0B0C0D0F101112 CLAVE 72
32333C3D25222320E9E8EBEACECDCCC3 DATOS 72
14151617191A1B1C1E1F20212324252628292A2B2D2E2F30 CLAVE 73
404142436261606 78A8BB4B511161714 DATOS 73
323334353738393A3C3D3E3F41424344464748494B4C4D4E CLAVE 74
94959293F5FAFBF81F1E1D1C7C7F7E79 DATOS 74
505152535556575 85A5B5C5D5F606162646566676 96A6B6C CLAVE 75
BEBFBCBD191A1B14CFCEC9C8546B6A69 DATOS 75
6E6F707173747576787 97A7B7D7E7F80828384858788898A CLAVE 76
2C2D3233898E8F8CBBBAB9B8333031CE DATOS 76

8C8D8E8F91929394969798999B9C9D9EA0A1A2A3A5A6A7A8 CLAVE 77
84858687BFBCBDBA37363938FDFAFBF8 DATOS 77
AAABACADAFB0B1B2B4B5B6B7B9BABBBCBEBFC0C1C3C4C5C6 CLAVE 78
828384857669686B909192930B08090E DATOS 78
C8C9CACBCDCECFD0D2D3D4D5D7D8D9DADCDDDEDFE1E2E3E4 CLAVE 79
BEBFBCBD9695948B707176779E919093 DATOS 79
E6E7E8E9EBECEDEEF0F1F2F3F5F6F7F8FAFBFCFDFE010002 CLAVE 80
8B8A858460676665212023220D0D3D2DD DATOS 80
04050607090A0B0C0E0F10111314151618191A1B1D1E1F20 CLAVE 81
76777475F1F2F3F4F8F9E6E777707172 DATOS 81
222324252728292A2C2D2E2F31323334363738393B3C3D3E CLAVE 82
A4A5A2A34F404142B4B5B6B727242522 DATOS 82
404142434546474 84A4B4C4D4F5051525455657595A5B5C CLAVE 83
94959697E1E2E3EC16171011839C9D9E DATOS 83
5E5F60616364656668696A6B6D6E6F70727374757778797A CLAVE 84
03023D3C06010003DEDFDCDDFFFCFDE2 DATOS 84
7C7D7E7F818283848687888 98B8C8D8E9091929395969798 CLAVE 85
10111213F1F2F3F4CECFC0C1DBDCDDDE DATOS 85
9A9B9C9D9FA0A1A2A4A5A6A7A9AAABACAEAFB0B1B3B4B5B6 CLAVE 86
67666160724D4C4F1D1C1F1E73707176 DATOS 86
B8B9BABBBDBEBFC0C2C3C4C5C7C8C9CACCCDCECFD1D2D3D4 CLAVE 87
E6E7E4E5A8ABAAD584858283909F9E9D DATOS 87
D6D7D8D9DBDCDDDEE0E1E2E3E5E6E7E8EAEBECEDEFF0F1F2 CLAVE 88
71707F7E565150537D7C7F7E6162636C DATOS 88
F4F5F6F7F9FAFBFCFEFE01010304050608090A0B0D0E0F10 CLAVE 89
64656667212223245555AAAA03040506 DATOS 89
121314151718191A1C1D1E1F21222324262728292B2C2D2E CLAVE 90
9E9F9899ABA4A5A6CFCECDCC2B28292E DATOS 90
30313233353637383A3B3C3D3F40414244454647494A4B4C CLAVE 91
C7C6C5C4D1D2D3DC626364653A454447 DATOS 91
4E4F50515354555658595A5B5D5E5F60626364656768696A CLAVE 92
F6F7E8E9E0E7E6E51D1C1F1E5B585966 DATOS 92
6C6D6E6F71727374767778797B7C7D7E8081828385868788 CLAVE 93
BCBDBEBF5D5E5F5868696667F4F3F2F1 DATOS 93
8A8B8C8D8F909192949596979 99A9B9C9E9FA0A1A3A4A5A6 CLAVE 94
40414647B0AFAEAD9B9A99989B98999E DATOS 94
A8A9AAABADAEAFB0B2B3B4B5B7B8B9BABCBDBEBFC1C2C3C4 CLAVE 95
69686B6A0201001F0F0E0908B4BBBAB9 DATOS 95
C6C7C8C9CBCCCDCED0D1D2D3D5D6D7D8DADBDCDDDFE0E1E2 CLAVE 96
C7C6C9C8D8DFDEDD5A5B5859BEBDBCB3 DATOS 96
E4E5E6E7E9EAEBECEEEFF0F1F3F4F5F6F8F9FAFBFDFEFE00 CLAVE 97
DEDFDCDD787B7A7DFFFEE1E0B2B5B4B7 DATOS 97
020304050708090A0C0D0E0F11121314161718191B1C1D1E CLAVE 98
4D4C4B4A606F6E6DD0D1D2D3FBF8F9FE DATOS 98
20212223252627282A2B2C2D2F30313234353637393A3B3C CLAVE 99
B7B6B5B4D7D4D5DAE5E4E3E2E1FEFFFC DATOS 99
3E3F40414344454648494A4B4D4E4F50525354555758595A CLAVE 100
CECFB0B1F7F0F1F2AEAFACAD3E3D3C23 DATOS 100

10.5.4. Fichero 2: Salidas Encriptación con clave de 192 bits

DFF4945E0336DF4C1C56BC700EFF837F SALIDA 1
B6FDDEF4752765E347D5D2DC196D1252 SALIDA 2
D23684E3D963B3AFCF1A114ACA90CBD6 SALIDA 3
3A7AC027753E2A18C2CEAB9E17C11FD0 SALIDA 4
8F6786BD007528BA26603C1601CDD0D8 SALIDA 5
D17D073B01E71502E28B47AB551168B3 SALIDA 6
A469DA517119FAB95876F41D06D40FFA SALIDA 7
6091AA3B695C11F5C0B6AD26D3D862FF SALIDA 8
70F9E67F9F8DF1294131662DC6E69364 SALIDA 9
D154DCAFAD8B207FA5CBC95E9996B559 SALIDA 10
4934D541E8B46FA339C805A7AEB9E5DA SALIDA 11
62564C738F3EFE186E1A127A0C4D3C61 SALIDA 12
07805AA043986EB23693E23BEF8F3438 SALIDA 13
DF0B4931038BADE848DEE3B4B85AA44B SALIDA 14
592D5FDED76582E4143C65099309477C SALIDA 15
C9B8D6545580D3DFBCDD09B954ED4E92 SALIDA 16
5DCCD5D6EB7C1B42ACB008201DF707A0 SALIDA 17
A2A91682FFEB6ED1D34340946829E6F9 SALIDA 18
E45D185B797000348D9267960A68435D SALIDA 19
45E060DAE5901CDA8089E10D4F4C246B SALIDA 20
F6951AFACC0079A369C71FDCFF45DF50 SALIDA 21
9E95E00F351D5B3AC3D0E22E626DDAD6 SALIDA 22
9CB566FF26D92DAD083B51FDC18C173C SALIDA 23
C9C82766176A9B228EB9A974A010B4FB SALIDA 24
D8E26AA02945881D5137F1C1E1386E88 SALIDA 25
C0E024CCD68FF5FFA4D139C355A77C55 SALIDA 26

0B18B3D16F491619DA338640DF391D43 SALIDA 27
DBE09AC8F66027BF20CB6E434F252EFC SALIDA 28
6D04E5E43C5B9CBE05FEB9606B6480FE SALIDA 29
DD1D6553B96BE526D9FEE0FBD7176866 SALIDA 30
0260CA7E3F979FD015B0DD4690E16D2A SALIDA 31
9893734DE10EDCC8A67C3B110B8B8CC6 SALIDA 32
93B30B750516B2D18808D710C2EE84EF SALIDA 33
16F65FA47BE3CB5E6DFE7C6C37016C0E SALIDA 34
F3847210D5391E2360608E5ACB560581 SALIDA 35
8754462CD223366D0753913E6AF2643D SALIDA 36
1EA20617468D1B806A1FD58145462017 SALIDA 37
3B155D927355D737C6BE9DDA60136E2E SALIDA 38
26144F7B66DAA91B6333DBD3850502B3 SALIDA 39
E4F9A4AB52CED8134C649BF319EBCC90 SALIDA 40
B9DDD29AC6128A6CAB121E34A4C62B36 SALIDA 41
6FCDDAD898F2CE4EFF51294F5EAAF5C9 SALIDA 42
C9A6FE2BF4028080BEA6F7FC417BD7E3 SALIDA 43
6A2026846D8609D60F298A9C0673127F SALIDA 44
2CB25C005E26EFEA44336C4C97A4240B SALIDA 45
496967AB8680DDD73D09A0E4C7DCC8AA SALIDA 46
D5AF94DE93487D1F3A8C577CB84A66A4 SALIDA 47
84BDAC569CAE2828705F267CC8376E90 SALIDA 48
F7401DDA5AD5AB712B7EB5D10C6F99B6 SALIDA 49
1C9D54318539EBD4C3B5B7E37BF119F0 SALIDA 50
ACA572D65FB2764CFFD4A6ECA090EA0D SALIDA 51
36D9C627B8C2A886A10CCB36EAE3DFBB SALIDA 52
010EDBF5981E143A81D646E597A4A568 SALIDA 53
8DB44D538DC20CC2F40F3067FD298E60 SALIDA 54
930EB53BC71E6AC4B82972BDCD5AAFB3 SALIDA 55
6C42A81EDCBC9517CCD89C30C95597B4 SALIDA 56
DA389847AD06DF19D76EE119C71E1DD3 SALIDA 57
E018FDAE13D3118F9A5D1A647A3F0462 SALIDA 58
2AA65DB36264239D3846180FABDFAD20 SALIDA 59
1472163E9A4F780F1CEB44B07ECF4FDB SALIDA 60
C8273FDC8F3A9F72E91097614B62397C SALIDA 61
66C8427DCD733AAF7B3470CB7D976E3F SALIDA 62
146131CB17F1424D4F8DA91E6F80C1D0 SALIDA 63
2610D0AD83659081AE085266A88770DC SALIDA 64
38A2B5A974B0575C5D733917FB0D4570 SALIDA 65
E21D401EBC60DE20D6C486E4F39A588B SALIDA 66
E51D5F88C670B079C0CA1F0C2C4405A2 SALIDA 67
246A94788A642FB3D1B823C8762380C8 SALIDA 68
B80C391C5C41A4C3B30C68E0E3D7550F SALIDA 69
B77C4754FC64EB9A1154A9AF0BB1F21C SALIDA 70
FB554DE520D159A06BF219FC7F34A02F SALIDA 71
A89FBA152D76B4927BEED160DDB76C57 SALIDA 72
5676EAB4A98D2E8473B3F3D46424247C SALIDA 73
4E8F068BD7EDE52A639036EC86C33568 SALIDA 74
F0193C4D7AFF1791EE4C07EB4A1824FC SALIDA 75
AC8686EECA9BA761AFE82D67B928C33F SALIDA 76
5FAF8573E33B145B6A369CD3606AB2C9 SALIDA 77
31587E9944AB1C16B844ECAD0DF2E7DA SALIDA 78
D017FECD91148ABA37F6F3068AA67D8A SALIDA 79
788EF2F021A73CBA2794B616078A8500 SALIDA 80
5D1EF20DCED6BCBC12131AC7C54788AA SALIDA 81
B3C8CF961FAF9EA05FDDE6D1E4D8F663 SALIDA 82
143075C70605861C7FAC6526199E459F SALIDA 83
A5AE12EADE9A87268D898BFC8FC0252A SALIDA 84
0924F7CF2E877A4819F5244A360DCEA9 SALIDA 85
3D9E9635AFCC3E291CC7AB3F27D1C99A SALIDA 86
9D80FEEBF87510E2B8FB98BB54FD788C SALIDA 87
5F9D1A082A1A37985F174002ECA01309 SALIDA 88
A390EBB1D1403930184A44B4876646E4 SALIDA 89
700FE918981C3195BB6C4BCB46B74E29 SALIDA 90
907984406F7BF2D17FB1EB15B673D747 SALIDA 91
C32A956DCFC875C2AC7C7CC8B8CC26E1 SALIDA 92
02646E2EBFA9B820CF8424E9B9B6EB51 SALIDA 93
621FDA3A5BBD54C6D3C685816BD4EAD8 SALIDA 94
D4E216040426DFAF18B152469BC5AC2F SALIDA 95
9D0635B9D33B6CDBD71F5D246EA17CC8 SALIDA 96
10ABAD1BD9BAE5448808765583A2CC1A SALIDA 97
6891889E16544E355FF65A793C39C9A8 SALIDA 98
CC735582E68072C163CD9DDF46B91279 SALIDA 99
C5C68B9AEEB7F878DF578EFA562F9574 SALIDA 100

10.5.5. Fichero 3: Entradas Encriptación con clave de 256 bits

000102030506070 80A0B0C0D0F10111214151617191A1B1C1E1F202123242526 CLAVE 1
834EADFCCAC7E1B30664B1ABA44815AB DATOS 1
28292A2B2D2E2F30323334353738393A3C3D3E3F41424344464748494B4C4D4E CLAVE 2
D9DC4DBA3021B05D67C0518F72B62BF1 DATOS 2
50515253555657585A5B5C5D5F60616264656667696A6B6C6E6F707173747576 CLAVE 3
A291D86301A4A739F7392173AA3C604C DATOS 3
78797A7B7D7E7F808283848587888 98A8C8D8E8F91929394969798999B9C9D9E CLAVE 4
4264B2696498DE4DF79788A9F83E9390 DATOS 4
A0A1A2A3A5A6A7A8AAABACADAFB0B1B2B4B5B6B7B9BABBBCBEBFC0C1C3C4C5C6 CLAVE 5
EE9932B3721804D5A83EF5949245B6F6 DATOS 5
C8C9CACBCDCECFD0D2D3D4D5D7D8D9DADCDDDEDFE1E2E3E4E6E7E8E9EBECEDEE CLAVE 6
E6248F55C5FDCBCA9CBBB01C88A2EA77 DATOS 6
F0F1F2F3F5F6F7F8FAFBFCFDFE0100020405060 7090A0B0C0E0F101113141516 CLAVE 7
B8358E41B9DFF65FD461D55A99266247 DATOS 7
18191A1B1D1E1F20222324252728292A2C2D2E2F31323334363738393B3C3D3E CLAVE 8
F0E2D72260AF58E21E015AB3A4C0D906 DATOS 8
404142434546474 84A4B4C4D4F50515254555657595A5B5C5E5F606163646566 CLAVE 9
475B8B823CE8893DB3C44A9F2A379FF7 DATOS 9
68696A6B6D6E6F7072737475777 8797A7C7D7E7F818283848687888 98B8C8D8E CLAVE 10
688F5281945812862F5F3076CF80412F DATOS 10
90919293959697989A9B9C9D9FA0A1A2A4A5A6A7A9AAABACAEAFB0B1B3B4B5B6 CLAVE 11
08D1D2BC750AF553365D35E75AFACEAA DATOS 11
B8B9BABBBDBEBFC0C2C3C4C5C7C8C9CACCCDCECFD1D2D3D4D6D7D8D9DBDCDDDE CLAVE 12
8707121F47CC3EFCECA5F9A8474950A1 DATOS 12
E0E1E2E3E5E6E7E8EAEBECEDEFF0F1F2F4F5F6F7F9FAFBFCFEFE010103040506 CLAVE 13
E51AA0B135DBA566939C3B6359A980C5 DATOS 13
08090A0B0D0E0F10121314151718191A1C1D1E1F21222324262728292B2C2D2E CLAVE 14
069A007FC76A459F98BAF917FEDF9521 DATOS 14
303132333536373 83A3B3C3D3F40414244454647494A4B4C4E4F505153545556 CLAVE 15
726165C1723FBCF6C026D7D00B091027 DATOS 15
58595A5B5D5E5F60626364656768696A6C6D6E6F71727374767778797B7C7D7E CLAVE 16
D7C544DE91D55CFCDE1F84CA382200CE DATOS 16
80818283858687888A8B8C8D8F90919294959697999A9B9C9E9FA0A1A3A4A5A6 CLAVE 17
FED3C9A161B9B5B2BD611B41DC9DA357 DATOS 17
A8A9AAABADAEAFB0B2B3B4B5B7B8B9BABCBDBEBFC1C2C3C4C6C7C8C9CBCCCDCE CLAVE 18
4F634CDC6551043409F30B635832CF82 DATOS 18
D0D1D2D3D5D6D7D8DADBDCDDDFE0E1E2E4E5E6E7E9EAEBECEEEFF0F1F3F4F5F6 CLAVE 19
109CE98DB0DFB36734D9F3394711B4E6 DATOS 19
70717273757677787A7B7C7D7F8081828485868 7898A8B8C8E8F909193949596 CLAVE 20
4EA6DFABA2D8A02FFDFFA89835987242 DATOS 20
98999A9B9D9E9FA0A2A3A4A5A7A8A9AAACADAEAFB1B2B3B4B6B7B8B9BBBCBDBE CLAVE 21
5AE094F54AF58E6E3CDBF976DAC6D9EF DATOS 21
C0C1C2C3C5C6C7C8CACBCCCDCFD0D1D2D4D5D6D7D9DADBDCDEDFE0E1E3E4E5E6 CLAVE 22
764D8E8E0F29926DBE5122E66354FDBE DATOS 22
E8E9EAEBEDEEEFF0F2F3F4F5F7F8F9FAFCFDFEFF0102030406070 8090B0C0D0E CLAVE 23
3F0418F888CDF29A982BF6B75410D6A9 DATOS 23
101112131516171 81A1B1C1D1F20212224252627292A2B2C2E2F303133343536 CLAVE 24
E4A3E7CB12CDD56AA4A75197A9530220 DATOS 24
38393A3B3D3E3F40424344454748494A4C4D4E4F51525354565758595B5C5D5E CLAVE 25
211677684AAC1EC1A160F44C4EBF3F26 DATOS 25
606162636566676 86A6B6C6D6F707172747576777 97A7B7C7E7F808183848586 CLAVE 26
D21E439FF749AC8F18D6D4B105E03895 DATOS 26
88898A8B8D8E8F909293949597989 99A9C9D9E9FA1A2A3A4A6A7A8A9ABACADAE CLAVE 27
D9F6FF44646C4725BD4C0103FF5552A7 DATOS 27
B0B1B2B3B5B6B7B8BABBBCBDBFC0C1C2C4C5C6C7C9CACBCCCECFD0D1D3D4D5D6 CLAVE 28
0B1256C2A00B976250CFC5B0C37ED382 DATOS 28
D8D9DADBDDDEDFE0E2E3E4E5E7E8E9EAECEDEEEFF1F2F3F4F6F7F8F9FBFCFDFE CLAVE 29
B056447FFC6DC4523A36CC2E972A3A79 DATOS 29
000102030506070 80A0B0C0D0F10111214151617191A1B1C1E1F202123242526 CLAVE 30
5E25CA78F0DE55802524D38DA3FE4456 DATOS 30
28292A2B2D2E2F30323334353738393A3C3D3E3F41424344464748494B4C4D4E CLAVE 31
A5BCF4728FA5EAAD8567C0DC24675F83 DATOS 31
50515253555657585A5B5C5D5F60616264656667696A6B6C6E6F707173747576 CLAVE 32
814E59F97ED84646B78B2CA022E9CA43 DATOS 32
78797A7B7D7E7F808283848587888 98A8C8D8E8F91929394969798999B9C9D9E CLAVE 33
15478BEEC58F4775C7A7F5D4395514D7 DATOS 33
A0A1A2A3A5A6A7A8AAABACADAFB0B1B2B4B5B6B7B9BABBBCBEBFC0C1C3C4C5C6 CLAVE 34
253548FFCA461C67C8CBC78CD59F4756 DATOS 34
C8C9CACBCDCECFD0D2D3D4D5D7D8D9DADCDDDEDFE1E2E3E4E6E7E8E9EBECEDEE CLAVE 35
FD7AD8D73B9B0F8CC41600640F503D65 DATOS 35
F0F1F2F3F5F6F7F8FAFBFCFDFE0100020405060 7090A0B0C0E0F101113141516 CLAVE 36
06199DE52C6CBF8AF954CD65830BCD56 DATOS 36
18191A1B1D1E1F20222324252728292A2C2D2E2F31323334363738393B3C3D3E CLAVE 37
F17C4FFE48E44C61BD891E257E725794 DATOS 37
404142434546474 84A4B4C4D4F50515254555657595A5B5C5E5F606163646566 CLAVE 38
9A5B4A402A3E8A59BE6BF5CD8154F029 DATOS 38
68696A6B6D6E6F7072737475777 8797A7C7D7E7F818283848687888 98B8C8D8E CLAVE 39
79BD40B91A7E07DC939D441782AE6B17 DATOS 39
90919293959697989A9B9C9D9FA0A1A2A4A5A6A7A9AAABACAEAFB0B1B3B4B5B6 CLAVE 40

D8CEAAF8976E5FBE1012D8C84F323799 DATOS 40
B8B9BABBBDBEBFC0C2C3C4C5C7C8C9CACCCDCECFD1D2D3D4D6D7D8D9DBDCDDDE CLAVE 41
3316E2751E2E388B083DA23DD6AC3FBE DATOS 41
E0E1E2E3E5E6E7E8EAEBECEDEFF0F1F2F4F5F6F7F9FAFBFCFEFE010103040506 CLAVE 42
8B7CFBE37DE7DCA793521819242C5816 DATOS 42
08090A0B0D0E0F10121314151718191A1C1D1E1F21222324262728292B2C2D2E CLAVE 43
F23F033C0EEBF8EC55752662FD58CE68 DATOS 43
30313233353637383A3B3C3D3F40414244454647494A4B4C4E4F505153545556 CLAVE 44
59EB34F6C8BDBACC5FC6AD73A59A1301 DATOS 44
58595A5B5D5E5F60626364656768696A6C6D6E6F71727374767778797B7C7D7E CLAVE 45
DCDE8B6BD5CF7CC22D9505E3CE81261A DATOS 45
80818283858687888A8B8C8D8F90919294959697999A9B9C9E9FA0A1A3A4A5A6 CLAVE 46
E33CF7E524FED781E7042FF9F4B35DC7 DATOS 46
A8A9AAABADAEAFB0B2B3B4B5B7B8B9BABCBDBEBFC1C2C3C4C6C7C8C9CBCCCDCE CLAVE 47
27963C8FACDF73062867D164DF6D064C DATOS 47
D0D1D2D3D5D6D7D8DADBDCDDDFE0E1E2E4E5E6E7E9EAEBECEEEFF0F1F3F4F5F6 CLAVE 48
77B1CE386B551B995F2F2A1DA994EEF8 DATOS 48
F8F9FAFBFDFEFE00020304050708090A0C0D0E0F11121314161718191B1C1D1E CLAVE 49
F083388B013679EFCF0BB9B15D52AE5C DATOS 49
20212223252627282A2B2C2D2F30313234353637393A3B3C3E3F404143444546 CLAVE 50
C5009E0DAB55DB0ABDB636F2600290C8 DATOS 50
48494A4B4D4E4F5052535455758595A5C5D5E5F61626364666768696B6C6D6E CLAVE 51
7804881E26CD532D8514D3683F00F1B9 DATOS 51
70717273757677787A7B7C7D7F80818284858687898A8B8C8E8F909193949596 CLAVE 52
46CDDCD73D1EB53E675CA012870A92A3 DATOS 52
98999A9B9D9E9FA0A2A3A4A5A7A8A9AAACADAEAFB1B2B3B4B6B7B8B9BBBCBDBE CLAVE 53
A9FB44062BB07FE130A8E8299EACB1AB DATOS 53
C0C1C2C3C5C6C7C8CACBCCCDCFD0D1D2D4D5D6D7D9DADBDCDEDFE0E1E3E4E5E6 CLAVE 54
2B6FF8D7A5CC3A28A22D5A6F221AF26B DATOS 54
E8E9EAEBEDEEEFF0F2F3F4F5F7F8F9FAFCFDFEFF0102030406070809 0B0C0D0E CLAVE 55
1A9527C29B8ADD4B0E3E656DBB2AF8B4 DATOS 55
10111213151617181A1B1C1D1F20212224252627292A2B2C2E2F303133343536 CLAVE 56
7F99CF2C75244DF015EB4B0C1050AEAE DATOS 56
38393A3B3D3E3F404243444454748494A4C4D4E4F51525354565758595B5C5D5E CLAVE 57
E84FF85B0D9454071909C1381646C4ED DATOS 57
60616263656667686A6B6C6D6F70717274757677797A7B7C7E7F808183848586 CLAVE 58
89AFD40F99521280D5399B12404F6DB4 DATOS 58
88898A8B8D8E8F90929394959798999A9C9D9E9FA1A2A3A4A6A7A8A9ABACADAE CLAVE 59
A09EF32DBC5119A35AB7FA38656F0329 DATOS 59
B0B1B2B3B5B6B7B8BABBBCBDBFC0C1C2C4C5C6C7C9CACBCCCECFD0D1D3D4D5D6 CLAVE 60
61773457F068C376C7829B93E696E716 DATOS 60
D8D9DADBDDDEDFE0E2E3E4E5E7E8E9EAECEDEEEFF1F2F3F4F6F7F8F9FBFCFDFE CLAVE 61
A34F0CAE726CCE41DD498747D891B967 DATOS 61
000102030506070 80A0B0C0D0F10111214151617191A1B1C1E1F202123242526 CLAVE 62
856F59496C7388EE2D2B1A27B7697847 DATOS 62
28292A2B2D2E2F30323334353738393A3C3D3E3F41424344464748494B4C4D4E CLAVE 63
CB090C593EF7720BD95908FB93B49DF4 DATOS 63
50515253555657585A5B5C5D5F60616264656667696A6B6C6E6F707173747576 CLAVE 64
A0AC75CD2F1923D460FC4D457AD95BAF DATOS 64
78797A7B7D7E7F80828384858788898A8C8D8E8F91929394969798999B9C9D9E CLAVE 65
2A2B282974777689E8E9EEEF525D5C5F DATOS 65
A0A1A2A3A5A6A7A8AAABACADAFB0B1B2B4B5B6B7B9BABBBCBEBFC0C1C3C4C5C6 CLAVE 66
909192939390919E0F0E09089788898A DATOS 66
C8C9CACBCDCECFD0D2D3D4D5D7D8D9DADCDDDEDFE1E2E3E4E6E7E8E9EBECEDEE CLAVE 67
777675748D8E8F907170777649464744 DATOS 67
F0F1F2F3F5F6F7F8FAFBFCFDFE01000204050607090A0B0C0E0F101113141516 CLAVE 68
717073720605040B2D2C2B2A05FAFBF9 DATOS 68
18191A1B1D1E1F20222324252728292A2C2D2E2F31323334363738393B3C3D3E CLAVE 69
64656667FEFDFCC31B1A1D1CA5AAABA8 DATOS 69
40414243454647484A4B4C4D4F50515254555657595A5B5C5E5F606163646566 CLAVE 70
DBDAD9D86A696867B5B4B3B2C8D7D6D5 DATOS 70
68696A6B6D6E6F707273747577778797A7C7D7E7F818283848687888898B8C8D8E CLAVE 71
5C5D5E5FE3E0E1FE31303736333C3D3E DATOS 71
90919293959697989A9B9C9D9FA0A1A2A4A5A6A7A9AAABACAEAFB0B1B3B4B5B6 CLAVE 72
545556574B484946737275745 46B6A69 DATOS 72
B8B9BABBBDBEBFC0C2C3C4C5C7C8C9CACCCDCECFD1D2D3D4D6D7D8D9DBDCDDDE CLAVE 73
ECEDEEEFC6C5C4BB56575051F5FAFBF8 DATOS 73
E0E1E2E3E5E6E7E8EAEBECEDEFF0F1F2F4F5F6F7F9FAFBFCFEFE010103040506 CLAVE 74
464744452724252AC9C8CFCED2CDCCCF DATOS 74
08090A0B0D0E0F10121314151718191A1C1D1E1F21222324262728292B2C2D2E CLAVE 75
E6E7E4E54142435C878681801C131211 DATOS 75
30313233353637383A3B3C3D3F40414244454647494A4B4C4E4F505153545556 CLAVE 76
72737071CFCCCDC2F9F8FFFE710E0F0C DATOS 76
58595A5B5D5E5F60626364656768696A6C6D6E6F71727374767778797B7C7D7E CLAVE 77
505152537370714EC3C2C5C4010E0F0C DATOS 77
80818283858687888A8B8C8D8F90919294959697999A9B9C9E9FA0A1A3A4A5A6 CLAVE 78
A8A9AAAB5C5F5E51AEAFA8A93D222320 DATOS 78
A8A9AAABADAEAFB0B2B3B4B5B7B8B9BABCBDBEBFC1C2C3C4C6C7C8C9CBCCCDCE CLAVE 79
DEDFDCDDF6F5F4EB10111617FEF1F0F3 DATOS 79
D0D1D2D3D5D6D7D8DADBDCDDDFE0E1E2E4E5E6E7E9EAEBECEEEFF0F1F3F4F5F6 CLAVE 80
BDBCBFBE5E5D5C530B0A0D0CFAC5C4C7 DATOS 80
F8F9FAFBFDFEFE00020304050708090A0C0D0E0F11121314161718191B1C1D1E CLAVE 81
8A8B8889050606F8F4F5F2F3636C6D6E DATOS 81

20212223252627282A2B2C2D2F30313234353637393A3B3C3E3F404143444546 CLAVE 82
A6A7A4A54D4E4F40B2B3B4B539262724 DATOS 82
48494A4B4D4E4F505253545555758595A5C5D5E5F61626364666768696B6C6D6E CLAVE 83
9C9D9E9FE9EAEBF40E0F08099B949596 DATOS 83
707172737576777787A7B7C7D7F8081828485868789A8B8C8E8F909193949596 CLAVE 84
2D2C2F2E1013121DCCCDCACBED121310 DATOS 84
98999A9B9D9E9FA0A2A3A4A5A7A8A9AAACADAEAFB1B2B3B4B6B7B8B9BBBCBDBE CLAVE 85
F4F5F6F7EDEEEFD0EAEBECEDF7F8F9FA DATOS 85
C0C1C2C3C5C6C7C8CACBCCCDCFD0D1D2D4D5D6D7D9DADBDCDEDFE0E1E3E4E5E6 CLAVE 86
3D3C3F3E282B2A2573727574150A0B08 DATOS 86
E8E9EAEBEDEEEFF0F2F3F4F5F7F8F9FAFCFDFEFF01020304060708090B0C0D0E CLAVE 87
B6B7B4B5F8FBFAE5B4B5B2B3A0AFAEAD DATOS 87
10111213151617181A1B1C1D1F20212224252627292A2B2C2E2F303133343536 CLAVE 88
B7B6B5B4989B9A95878681809BA4A5A6 DATOS 88
38393A3B3D3E3F40424344454748494A4C4D4E4F51525354565758595B5C5D5E CLAVE 89
A8A9AAABE5E6E798E9E8EFEE4748494A DATOS 89
606162636566667686A6B6C6D6F70717274757677797A7B7C7E7F808183848586 CLAVE 90
ECEDEEEFD9DADBD4B9B8BFBE657A7B78 DATOS 90
88898A8B8D8E8F90929394959798999A9C9D9E9FA1A2A3A4A6A7A8A9ABACADAE CLAVE 91
7F7E7D7C696A6B74CACBCCCD929D9C9F DATOS 91
B0B1B2B3B5B6B7B8BABBBCBDBFC0C1C2C4C5C6C7C9CACBCCCECFD0D1D3D4D5D6 CLAVE 92
08090A0B0605040BFFFEF9F8B9C6C7C4 DATOS 92
D8D9DADBDDDEDFE0E2E3E4E5E7E8E9EAECEDEEEFF1F2F3F4F6F7F8F9FBFCFDFE CLAVE 93
08090A0BF1F2F3CCFCFDFAFB68676665 DATOS 93
000102030506070 80A0B0C0D0F10111211451617191A1B1C1E1F202123242526 CLAVE 94
CACBC8C93A393837050403020D121310 DATOS 94
28292A2B2D2E2F30323334353738393A3C3D3E3F41424344464748494B4C4D4E CLAVE 95
E9E8EBEA8281809F8F8E8988343B3A39 DATOS 95
505152535556575 85A5B5C5D5F606162646566676 96A6B6C6E6F707173747576 CLAVE 96
515053524645444BD0D1D6D7340B0A09 DATOS 96
78797A7B7D7E7F808283848587888 98A8C8D8E8F91929394969798999B9C9D9E CLAVE 97
42434041ECEFEE1193929594C6C9C8CB DATOS 97
A0A1A2A3A5A6A7A8AAABACADAFB0B1B2B4B5B6B7B9BABBBCBEBFC0C1C3C4C5C6 CLAVE 98
EFEEEDECC2C1C0CF76777071455A5B58 DATOS 98
C8C9CACBCDCECFD0D2D3D4D5D7D8D9DADCDDDEDFE1E2E3E4E6E7E8E9EBECEDEE CLAVE 99
5F5E5D5C3F3C3D221D1C1B1A19161714 DATOS 99
F0F1F2F3F5F6F7F8FAFBFCFDFE0100020405060 7090A0B0C0E0F101113141516 CLAVE 100
000102034142434C1C1D1A1B8D727371 DATOS 100

10.5.6. Fichero 3: Salidas Encriptación con clave de 256 bits

1946DABF6A03A2A2C3D0B05080AED6FC SALIDA 1
5ED301D747D3CC715445EBDEC62F2FB4 SALIDA 2
6585C8F43D13A6BEAB6419FC5935B9D0 SALIDA 3
2A5B56A596680FCC0E05F5E0F151ECAE SALIDA 4
F5D6FF414FD2C6181494D20C37F2B8C4 SALIDA 5
85399C01F59FFFB5204F19F8482F00B8 SALIDA 6
92097B4C88A041DDF98144BC8D22E8E7 SALIDA 7
89BD5B73B356AB412AEF9F76CEA2D65C SALIDA 8
2536969093C55FF9454692F2FAC2F530 SALIDA 9
07FC76A872843F3F6E0081EE9396D637 SALIDA 10
E38BA8EC2AA741358DCC93E8F141C491 SALIDA 11
D028EE23E4A89075D0B03E868D7D3A42 SALIDA 12
8CD9423DFC459E547155C5D1D522E540 SALIDA 13
080E9517EB1677719ACF728086040AE3 SALIDA 14
7C1700211A3991FC0ECDED0AB3E576B0 SALIDA 15
DABCBCC855839251DB51E224FBE87435 SALIDA 16
68D56FAD0406947A4DD27A7448C10F1D SALIDA 17
DA9A11479844D1FFEE24BBF3719A9925 SALIDA 18
5E4BA572F8D23E738DA9B05BA24B8D81 SALIDA 19
A115A2065D667E3F0B883837A6E903F8 SALIDA 20
3E9E90DC33EAC2437D86AD30B137E66E SALIDA 21
01CE82D8FBCDAE824CB3C48E495C3692 SALIDA 22
0C9CFF163CE936FAAF083CFD3DEA3117 SALIDA 23
5131BA9BD48F2BBA85560680DF504B52 SALIDA 24
9DC503BBF09823AEC8A977A5AD26CCB2 SALIDA 25
9A6DB0C0862E506A9E397225884041D7 SALIDA 26
430BF9570804185E1AB6365FC6A6860C SALIDA 27
3525EBC02F4886E6A5A3762813E8CE8A SALIDA 28
07FA265C763779CCE224C7BAD671027B SALIDA 29
E8B72B4E8BE243438C9FFF1F0E205872 SALIDA 30
109D4F999A0E11ACE1F05E6B22CBCB50 SALIDA 31
45A5E8D4C3ED58403FF08D68A0CC4029 SALIDA 32
196865964DB3D417B6BD4D586BCB7634 SALIDA 33
60436AD45AC7D30D99195F815D98D2AE SALIDA 34
BB07A23F0B61014B197620C185E2CD75 SALIDA 35
5BC0B2850129C854423AFF0751FE343B SALIDA 36

7541A78F96738E6417D2A24BD2BECA40 SALIDA 37
B0A303054412882E464591F1546C5B9E SALIDA 38
778C06D8A355EEEE214FCEA14B4E0EEF SALIDA 39
09614206D15CBACE63227D06DB6BEEBB SALIDA 40
41B97FB20E427A9FDBBB358D9262255D SALIDA 41
C1940F703D845F957652C2D64ABD7ADF SALIDA 42
D2D44FCDAE5332343366DB297EFCF21B SALIDA 43
EA8196B79DBE167B6AA9896E287EED2B SALIDA 44
D6B0B0C4BA6C7DBE5ED467A1E3F06C2D SALIDA 45
EC51EB295250C22C2FB01816FB72BCAE SALIDA 46
ADED6630A07CE9C7408A155D3BD0D36F SALIDA 47
697C9245B9937F32F5D1C82319F0363A SALIDA 48
AAD5AD50C6262AAEC30541A1B7B5B19C SALIDA 49
7D34B893855341EC625BD6875AC18C0D SALIDA 50
7EF05105440F83862F5D780E88F02B41 SALIDA 51
C377C06403382061AF2C9C93A8E70DF6 SALIDA 52
1DBDB3FFDC052DACC83318853ABC6DE5 SALIDA 53
69A6EAB00432517D0BF483C91C0963C7 SALIDA 54
0797F41DC217C80446E1D514BD6AB197 SALIDA 55
9DFD76575902A637C01343C58E011A03 SALIDA 56
ACF4328AE78F34B9FA9B459747CC2658 SALIDA 57
B0479AEA12BAC4FE2384CF98995150C6 SALIDA 58
9DD52789EFE3FFB99F33B3DA5030109A SALIDA 59
ABBB755E4621EF8F1214C19F649FB9FD SALIDA 60
DA27FB8174357BCE2BED0E7354F380F9 SALIDA 61
C59A0663F0993838F6E5856593BDC5EF SALIDA 62
ED60B264B5213E831607A99C0CE5E57E SALIDA 63
E50548746846F3EB77B8C520640884ED SALIDA 64
28282CC7D21D6A2923641E52D188EF0C SALIDA 65
0DFA5B02ABB18E5A815305216D6D4F8E SALIDA 66
7359635C0EECEFE31D673395FB46FB99 SALIDA 67
73C679F7D5AEF2745C9737BB4C47FB36 SALIDA 68
B192BD472A4D2EAFB786E97458967626 SALIDA 69
0EC327F6C8A2B147598CA3FDE61DC6A4 SALIDA 70
FC418EB3C41B859B38D4B6F646629729 SALIDA 71
30249E5AC282B1C981EA64B609F3A154 SALIDA 72
5E6E08646D12150776BB43C2D78A9703 SALIDA 73
FAEB3D5DE652CD3447DCEB343F30394A SALIDA 74
A8E88706823F6993EF80D05C1C7B2CF0 SALIDA 75
8CED86677E6E00A1A1B15968F2D3CCE6 SALIDA 76
9FC7C23858BE03BDEBB84E90DB6786A9 SALIDA 77
B4FBD65B33F70D8CF7F1111AC4649C36 SALIDA 78
C5C32D5ED03C4B53CC8C1BD0EF0DBBF6 SALIDA 79
D1A7F03B773E5C212464B63709C6A891 SALIDA 80
6B7161D8745947AC6950438EA138D028 SALIDA 81
FD47A9F7E366EE7A09BC508B00460661 SALIDA 82
00D40B003DC3A0D9310B659B98C7E416 SALIDA 83
EEA4C79DCC8E2BDA691F20AC48BE0717 SALIDA 84
E78F43B11C204403E5751F89D05A2509 SALIDA 85
D0F0E3D1F1244BB979931E38DD1786EF SALIDA 86
042E639DC4E1E4DDE7B75B749EA6F765 SALIDA 87
BC032FDD0EFE29503A980A7D07AB46A8 SALIDA 88
0C93AC949C0DA6446EFFB86183B6C910 SALIDA 89
E0D343E14DA75C917B4A5CEC4810D7C2 SALIDA 90
0EAFB821748408279B937B626792E619 SALIDA 91
FA1AC6E02D23B106A1FEF18B274A553F SALIDA 92
0DADFE019CD12368075507DF33C1A1E9 SALIDA 93
3A0879B414465D9FFBAF86B33A63A1B9 SALIDA 94
62199FADC76D0BE1805D3BA0B7D914BF SALIDA 95
1B06D6C5D333E742730130CF78E719B4 SALIDA 96
F1F848824C32E9DCDCBF21580F069329 SALIDA 97
1A09050CBD684F784D8E965E0782F28A SALIDA 98
79C2969E7DED2BA7D088F3F320692360 SALIDA 99
091A658A2F7444C16ACCB669450C7B63 SALIDA 100

10.5.7. Fichero 4: Entradas Desencriptación con clave de 128 bits

80000000000000000000000000000000 CLAVE 1
0edd33d3c621e546455bd8ba1418bec8 DATOS 1
c0000000000000000000000000000000 CLAVE 2
4bc3f883450c113c64ca42e1112a9e87 DATOS 2
e0000000000000000000000000000000 CLAVE 3
72a1da770f5d7ac4c9ef94d822affd97 DATOS 3
f0000000000000000000000000000000 CLAVE 4
970014d634e2b7650777e8e84d03ccd8 DATOS 4
f8000000000000000000000000000000 CLAVE 5
f17e79aed0db7e279e955b5f493875a7 DATOS 5

fc000000000000000000000000000000 CLAVE 6
9ed5a75136a940d0963da379db4af26a DATOS 6
fe000000000000000000000000000000 CLAVE 7
c4295f83465c7755e8fa364bac6a7ea5 DATOS 7
ff000000000000000000000000000000 CLAVE 8
b1d758256b28fd850ad4944208cf1155 DATOS 8
ff800000000000000000000000000000 CLAVE 9
42ffb34c743de4d88ca38011c990890b DATOS 9
ffc000000000000000000000000000000 CLAVE 10
9958f0ecea8b2172c0c1995f9182c0f3 DATOS 10
ffe000000000000000000000000000000 CLAVE 11
956d7798fac20f82a8823f984d06f7f5 DATOS 11
fff000000000000000000000000000000 CLAVE 12
a01bf44f2d16be928ca44aaf7b9b106b DATOS 12
fff800000000000000000000000000000 CLAVE 13
b5f1a33e50d40d103764c76bd4c6b6f8 DATOS 13
fffc00000000000000000000000000000 CLAVE 14
2637050c9fc0d4817e2d69de878aee8d DATOS 14
fffe00000000000000000000000000000 CLAVE 15
113ecbe4a453269a0dd26069467fb5b5 DATOS 15
ffff00000000000000000000000000000 CLAVE 16
97d0754fe68f11b9e375d070a608c884 DATOS 16
ffff80000000000000000000000000000 CLAVE 17
c6a0b3e998d05068a5399778405200b4 DATOS 17
fffffc0000000000000000000000000 CLAVE 18
df556a33438db87bc41b1752c55e5e49 DATOS 18
fffffe0000000000000000000000000 CLAVE 19
90fb128d3a1af6e548521bb962bf1f05 DATOS 19
fffff00000000000000000000000000 CLAVE 20
26298e9c1db517c215fadfb7d2a8d691 DATOS 20
fffff80000000000000000000000000 CLAVE 21
a6cb761d61f8292d0df393a279ad0380 DATOS 21
ffffffc0000000000000000000000000 CLAVE 22
12acd89b13cd5f8726e34d44fd486108 DATOS 22
ffffffe0000000000000000000000000 CLAVE 23
95b1703fc57ba09fe0c3580febdd7ed4 DATOS 23
fffffff0000000000000000000000000 CLAVE 24
de11722d893e9f9121c381becc1da59a DATOS 24
fffffff8000000000000000000000000 CLAVE 25
6d114ccb27bf391012e8974c546d9bf2 DATOS 25
fffffffc000000000000000000000000 CLAVE 26
5ce37e17eb4646ecfac29b9cc38d9340 DATOS 26
fffffffe000000000000000000000000 CLAVE 27
18c1b6e2157122056d0243d8a165cddb DATOS 27
ffffffff000000000000000000000000 CLAVE 28
99693e6a59d1366c74d823562d7e1431 DATOS 28
ffffffff800000000000000000000000 CLAVE 29
6c7c64dc84a8bba758ed17eb025a57e3 DATOS 29
fffffffc000000000000000000000000 CLAVE 30
e17bc79f30eaab2fac2cbbe3458d687a DATOS 30
fffffffe000000000000000000000000 CLAVE 31
1114bc2028009b923f0b01915ce5e7c4 DATOS 31
ffffffff000000000000000000000000 CLAVE 32
9c28524a16a1e1c1452971caa8d13476 DATOS 32
ffffffff800000000000000000000000 CLAVE 33
ed62e16363638360fdd6ad62112794f0 DATOS 33
fffffffc000000000000000000000000 CLAVE 34
5a8688f0b2a2c16224c161658ffd4044 DATOS 34
fffffffe000000000000000000000000 CLAVE 35
23f710842b9bb9c32f26648c786807ca DATOS 35
ffffffff000000000000000000000000 CLAVE 36
44a98bf11e163f632c47ec6a49683a89 DATOS 36
ffffffff800000000000000000000000 CLAVE 37
0f18aff94274696d9b61848bd50ac5e5 DATOS 37
fffffffc000000000000000000000000 CLAVE 38
82408571c3e2424540207f833b6dda69 DATOS 38
fffffffe000000000000000000000000 CLAVE 39
303ff996947f0c7d1f43c8f3027b9b75 DATOS 39
ffffffff000000000000000000000000 CLAVE 40
7df4daf4ad29a3615a9b6ece5c99518a DATOS 40
ffffffff800000000000000000000000 CLAVE 41
c72954a48d0774db0b4971c526260415 DATOS 41
fffffffffc00000000000000000000 CLAVE 42
1df9b76112dc6531e07d2cfda04411f0 DATOS 42
fffffffffe00000000000000000000 CLAVE 43
8e4d8e699119e1fc87545a647fb1d34f DATOS 43
fffffffff000000000000000000000 CLAVE 44
e6c4807ae11f36f091c57d9fb68548d1 DATOS 44
fffffffff800000000000000000000 CLAVE 45
8ebf73aad49c82007f77a5c1ccec6ab4 DATOS 45
fffffffffc00000000000000000000 CLAVE 46
4fb288cc2040049001d2c7585ad123fc DATOS 46
fffffffffe00000000000000000000 CLAVE 47

04497110efb9dceb13e2b13fb4465564 DATOS 47
ffffffffff00000000000000000000 CLAVE 48
75550e6cb5a88e49634c9ab69eda0430 DATOS 48
ffffffffff80000000000000000000 CLAVE 49
b6768473ce9843ea66a81405dd50b345 DATOS 49
ffffffffffc0000000000000000000 CLAVE 50
cb2f430383f9084e03a653571e065de6 DATOS 50
ffffffffffe0000000000000000000 CLAVE 51
ff4e66c07bae3e79fb7d210847a3b0ba DATOS 51
fffffffffff0000000000000000000 CLAVE 52
7b90785125505fad59b13c186dd66ce3 DATOS 52
fffffffffff8000000000000000000 CLAVE 53
8b527a6aebdaec9eaef8eda2cb7783e5 DATOS 53
fffffffffffc000000000000000000 CLAVE 54
43fdaf53ebbc9880c228617d6a9b548b DATOS 54
fffffffffffe000000000000000000 CLAVE 55
53786104b9744b98f052c46f1c850d0b DATOS 55
fffffffffffff000000000000000000 CLAVE 56
b5ab3013dd1e61df06cbaf34ca2aee78 DATOS 56
fffffffffffff800000000000000000 CLAVE 57
7470469be9723030fdcc73a8cd4fbb10 DATOS 57
ffffffffffffc00000000000000000 CLAVE 58
a35a63f5343ebe9ef8167bcb48ad122e DATOS 58
ffffffffffffe00000000000000000 CLAVE 59
fd8687f0757a210e9fdf181204c30863 DATOS 59
fffffffffffff00000000000000000 CLAVE 60
7a181e84bd5457d26a88fbae96018fb0 DATOS 60
fffffffffffff80000000000000000 CLAVE 61
653317b9362b6f9b9e1a580e68d494b5 DATOS 61
ffffffffffffc0000000000000000 CLAVE 62
995c9dc0b689f03c45867b5faa5c18d1 DATOS 62
ffffffffffffe0000000000000000 CLAVE 63
77a4d96d56dda398b9aabecfc75729fd DATOS 63
fffffffffffff0000000000000000 CLAVE 64
84be19e053635f09f2665e7bae85b42d DATOS 64
fffffffffffff8000000000000000 CLAVE 65
32cd652842926aea4aa6137bb2be2b5e DATOS 65
ffffffffffffc000000000000000 CLAVE 66
493d4a4f38ebb337d10aa84e9171a554 DATOS 66
ffffffffffffe000000000000000 CLAVE 67
d9bff7ff454b0ec5a4a2a69566e2cb84 DATOS 67
fffffffffffff000000000000000 CLAVE 68
3535d565ace3f31eb249ba2cc6765d7a DATOS 68
fffffffffffff800000000000000 CLAVE 69
f60e91fc3269eecf3231c6e9945697c6 DATOS 69
ffffffffffffffc00000000000000 CLAVE 70
ab69cfadf51f8e604d9cc37182f6635a DATOS 70
ffffffffffffffe00000000000000 CLAVE 71
7866373f24a0b6ed56e0d96fcdafb877 DATOS 71
ffffffffffffff00000000000000 CLAVE 72
1ea448c2aac954f5d812e9d78494446a DATOS 72
ffffffffffffff80000000000000 CLAVE 73
acc5599dd8ac02239a0fef4a36dd1668 DATOS 73
ffffffffffffffc0000000000000 CLAVE 74
d8764468bb103828cf7e1473ce895073 DATOS 74
ffffffffffffffe0000000000000 CLAVE 75
1b0d02893683b9f180458e4aa6b73982 DATOS 75
fffffffffffffff0000000000000 CLAVE 76
96d9b017d302df410a937dcdb8bb6e43 DATOS 76
fffffffffffffff8000000000000 CLAVE 77
ef1623cc44313cff440b1594a7e21cc6 DATOS 77
ffffffffffffffc000000000000 CLAVE 78
284ca2fa35807b8b0ae4d19e11d7dbd7 DATOS 78
ffffffffffffffe000000000000 CLAVE 79
f2e976875755f9401d54f36e2a23a594 DATOS 79
fffffffffffffff000000000000 CLAVE 80
ec198a18e10e532403b7e20887c8dd80 DATOS 80
fffffffffffffff800000000000 CLAVE 81
545d50ebd919e4a6949d96ad47e46a80 DATOS 81
ffffffffffffffffc00000000000 CLAVE 82
dbdfb527060e0a71009c7bb0c68f1d44 DATOS 82
ffffffffffffffffe00000000000 CLAVE 83
9cfa1322ea33da2173a024f2ff0d896d DATOS 83
fffffffffffffffff00000000000 CLAVE 84
8785b1a75b0f3bd958dcd0e29318c521 DATOS 84
fffffffffffffffff80000000000 CLAVE 85
38f67b9e98e4a97b6df030a9fcdd0104 DATOS 85
ffffffffffffffffc0000000000 CLAVE 86
192afffb2c880e82b05926d0fc6c448b DATOS 86
ffffffffffffffffe0000000000 CLAVE 87
6a7980ce7b105cf530952d74daaf798c DATOS 87
fffffffffffffffff0000000000 CLAVE 88
ea3695e1351b9d6858bd958cf513ef6c DATOS 88

ffffffffffffffff8000000000 CLAVE 89
6da0490ba0ba0343b935681d2cce5ba1 DATOS 89
ffffffffffffffffc000000000 CLAVE 90
f0ea23af08534011c60009ab29ada2f1 DATOS 90
ffffffffffffffffe000000000 CLAVE 91
ff13806cf19cc38721554d7c0fcdcd4b DATOS 91
fffffffffffffffff000000000 CLAVE 92
6838af1f4f69bae9d85dd188dcdf0688 DATOS 92
fffffffffffffffff800000000 CLAVE 93
36cf44c92d550bfb1ed28ef583ddf5d7 DATOS 93
ffffffffffffffffc00000000 CLAVE 94
d06e3195b5376f109d5c4ec6c5d62ced DATOS 94
ffffffffffffffffe00000000 CLAVE 95
c440de014d3d610707279b13242a5c36 DATOS 95
fffffffffffffffff00000000 CLAVE 96
f0c5c6ffa5e0bd3a94c88f6b6f7c16b9 DATOS 96
ffffffffffffffff80000000 CLAVE 97
3e40c3901cd7effc22bffc35dee0b4d9 DATOS 97
ffffffffffffffffc0000000 CLAVE 98
b63305c72bedfab97382c406d0c49bc6 DATOS 98
ffffffffffffffffe0000000 CLAVE 99
36bbaab22a6bd4925a99a2b408d2dbae DATOS 99
fffffffffffffffff0000000 CLAVE 100
307c5b8fcd0533ab98bc51e27a6ce461 DATOS 100

10.5.8. Fichero 4: Salidas Desencriptación con clave de 128 bits

Todas las salidas esperadas tienen valor cero.

10.5.9. Fichero 5: Entradas Desencriptación con clave de 192 bits

800000000000000000000000000000000000000000000000 CLAVE 1
de885dc87f5a92594082d02cc1e1b42c DATOS 1
c000000000000000000000000000000000000000000000000 CLAVE 2
132b074e80f2a597bf5febd8ea5da55e DATOS 2
e000000000000000000000000000000000000000000000000 CLAVE 3
6eccedf8de592c22fb81347b79f2db1f DATOS 3
f000000000000000000000000000000000000000000000000 CLAVE 4
180b09f267c45145db2f826c2582d35c DATOS 4
f800000000000000000000000000000000000000000000000 CLAVE 5
edd807ef7652d7eb0e13c8b5e15b3bc0 DATOS 5
fc000000000000000000000000000000000000000000000000 CLAVE 6
9978bcf8dd8fd72241223ad24b31b8a4 DATOS 6
fe000000000000000000000000000000000000000000000000 CLAVE 7
5310f654343e8f27e12c83a48d24ff81 DATOS 7
ff000000000000000000000000000000000000000000000000 CLAVE 8
833f71258d53036b02952c76c744f5a1 DATOS 8
ff800000000000000000000000000000000000000000000000 CLAVE 9
eba83ff200cff9318a92f8691a06b09f DATOS 9
ffc000000000000000000000000000000000000000000000000 CLAVE 10
ff620ccbe9f3292abdf2176b09f04eba DATOS 10
ffe000000000000000000000000000000000000000000000000 CLAVE 11
7ababc4b3f516c9aafb35f4140b548f9 DATOS 11
fff000000000000000000000000000000000000000000000000 CLAVE 12
aa187824d9c4582b0916493ecbde8c57 DATOS 12
fff800000000000000000000000000000000000000000000000 CLAVE 13
1c0ad553177fd5ea1092c9d626a29dc4 DATOS 13
fffc000000000000000000000000000000000000000000000000 CLAVE 14
a5dc46c37261194124ecaebd680408ec DATOS 14
fffe000000000000000000000000000000000000000000000000 CLAVE 15
e4f2f2ae23e9b10bacfa58601531ba54 DATOS 15
ffff000000000000000000000000000000000000000000000000 CLAVE 16
b7d67cf1a1e91e8ff3a57a172c7bf412 DATOS 16
ffff800000000000000000000000000000000000000000000000 CLAVE 17
26706be06967884e847d137128ce47b3 DATOS 17
ffffc000000000000000000000000000000000000000000000000 CLAVE 18
b2f8b409b0585909aad3a7b5a219072a DATOS 18
ffffe000000000000000000000000000000000000000000000000 CLAVE 19
5e4b7bff0290c78344c54a23b722cd20 DATOS 19
fffff000000000000000000000000000000000000000000000000 CLAVE 20

07093657552d4414227ce161e9ebf7dd DATOS 20
ffffff80000000000000000000000000000000000000000000000000 CLAVE 21
e1af1e7d8bc225ed4dffb771ecbb9e67 DATOS 21
fffffc00000000000000000000000000000000000000000000000000 CLAVE 22
ef6555253635d8432156cfd9c11b145a DATOS 22
fffffe00000000000000000000000000000000000000000000000000 CLAVE 23
fb4035074a5d4260c90cbd6da6c3fceb DATOS 23
fffffe00000000000000000000000000000000000000000000000000 CLAVE 24
446ee416f9ad1c103eb0cc96751c88e1 DATOS 24
fffffe80000000000000000000000000000000000000000000000000 CLAVE 25
198ae2a4637ac0a7890a8fd1485445c9 DATOS 25
fffffc00000000000000000000000000000000000000000000000000 CLAVE 26
562012ec8faded0825fb2fa70ab30cbd DATOS 26
fffffe00000000000000000000000000000000000000000000000000 CLAVE 27
cc8a64b46b5d88bf7f247d4dbaf38f05 DATOS 27
ffffff00000000000000000000000000000000000000000000000000 CLAVE 28
a168253762e2cc81b42d1e5001762699 DATOS 28
ffffff80000000000000000000000000000000000000000000000000 CLAVE 29
1b41f83b38ce5032c6cd7af98cf62061 DATOS 29
fffffc00000000000000000000000000000000000000000000000000 CLAVE 30
61a89990cd1411750d5fb0dc988447d4 DATOS 30
fffffe00000000000000000000000000000000000000000000000000 CLAVE 31
b5accc8ed629edf8c68a539183b1ea82 DATOS 31
ffffff00000000000000000000000000000000000000000000000000 CLAVE 32
b16fa71f846b81a13f361c43a851f290 DATOS 32
ffffff80000000000000000000000000000000000000000000000000 CLAVE 33
4fad6efdff5975aee7692234bcd54488 DATOS 33
fffffc00000000000000000000000000000000000000000000000000 CLAVE 34
ebfdb05a783d03082dfe5fdd80a00b17 DATOS 34
fffffe00000000000000000000000000000000000000000000000000 CLAVE 35
eb81b584766997af6ba5529d3bdd8609 DATOS 35
ffffff00000000000000000000000000000000000000000000000000 CLAVE 36
0cf4ff4f49c8a0ca060c443499e29313 DATOS 36
ffffff80000000000000000000000000000000000000000000000000 CLAVE 37
cc4ba8a8e029f8b26d8afff9df133bb6 DATOS 37
fffffc00000000000000000000000000000000000000000000000000 CLAVE 38
fefebf64360f38e4e63558f0ffc550c3 DATOS 38
fffffe00000000000000000000000000000000000000000000000000 CLAVE 39
12ad98cbf725137d6a8108c2bed99322 DATOS 39
ffffff00000000000000000000000000000000000000000000000000 CLAVE 40
6afaa996226198b3e2610413ce1b3f78 DATOS 40
ffffff80000000000000000000000000000000000000000000000000 CLAVE 41
2a8ce6747a7e39367828e290848502d9 DATOS 41
fffffc00000000000000000000000000000000000000000000000000 CLAVE 42
223736e8b8f89ca1e37b6deab40facf1 DATOS 42
fffffe00000000000000000000000000000000000000000000000000 CLAVE 43
c0f797e50418b95fa6013333917a9480 DATOS 43
ffffff00000000000000000000000000000000000000000000000000 CLAVE 44
a758de37c2ece2a02c73c01fedc9a132 DATOS 44
ffffff80000000000000000000000000000000000000000000000000 CLAVE 45
3a9b87ae77bae706803966c66c73adbd DATOS 45
fffffc00000000000000000000000000000000000000000000000000 CLAVE 46
d365ab8df8ffd782e358121a4a4fc541 DATOS 46
fffffe00000000000000000000000000000000000000000000000000 CLAVE 47
c8dcd9e6f75e6c36c8daee0466f0ed74 DATOS 47
ffffff00000000000000000000000000000000000000000000000000 CLAVE 48
c79a637beb1c0304f14014c037e736dd DATOS 48
ffffff80000000000000000000000000000000000000000000000000 CLAVE 49
105f0a25e84ac930d996281a5f954dd9 DATOS 49
fffffc00000000000000000000000000000000000000000000000000 CLAVE 50
42e4074b2927973e8d17ffa92f7fe615 DATOS 50
fffffe00000000000000000000000000000000000000000000000000 CLAVE 51
4fe2a9d2c1824449c69e3e0398f12963 DATOS 51
ffffff00000000000000000000000000000000000000000000000000 CLAVE 52
b7f29c1e1f62847a15253b28a1e9d712 DATOS 52
ffffff80000000000000000000000000000000000000000000000000 CLAVE 53
36ed5d29b903f31e8983ef8b0a2bf990 DATOS 53
fffffc00000000000000000000000000000000000000000000000000 CLAVE 54
27b8070270810f9d023f9dd7ff3b4aa2 DATOS 54
fffffe00000000000000000000000000000000000000000000000000 CLAVE 55
94d46e155c1228f61d1a0db4815ecc4b DATOS 55
ffffff00000000000000000000000000000000000000000000000000 CLAVE 56
ca6108d1d98071428eeceef1714b96dd DATOS 56
ffffff80000000000000000000000000000000000000000000000000 CLAVE 57
dc5b25b71b6296cf73dd2cdcac2f70b1 DATOS 57
fffffc00000000000000000000000000000000000000000000000000 CLAVE 58
44aba95e8a06a2d9d3530d2677878c80 DATOS 58
fffffe00000000000000000000000000000000000000000000000000 CLAVE 59
a570d20e89b467e8f5176061b81dd396 DATOS 59
ffffff00000000000000000000000000000000000000000000000000 CLAVE 60
758f4467a5d8f1e7307dc30b34e404f4 DATOS 60
ffffff80000000000000000000000000000000000000000000000000 CLAVE 61
bcea28e9071b5a2302970ff352451bc5 DATOS 61

ffffffffffffc0000000000000000000000000000000 CLAVE 62
7523c00bc177d331ad312e09c9015c1c DATOS 62
ffffffffffffe0000000000000000000000000000000 CLAVE 63
ccac61e3183747b3f5836da21a1bc4f4 DATOS 63
fffffffffffff0000000000000000000000000000000 CLAVE 64
707b075791878880b44189d3522b8c30 DATOS 64
fffffffffffff8000000000000000000000000000000 CLAVE 65
7132d0c0e4a07593cf12ebb12be7688c DATOS 65
ffffffffffffc0000000000000000000000000000000 CLAVE 66
effbac1644deb0c784275fe56e19ead3 DATOS 66
ffffffffffffe0000000000000000000000000000000 CLAVE 67
a005063f30f4228b374e2459738f26bb DATOS 67
fffffffffffff0000000000000000000000000000000 CLAVE 68
29975b5f48bb68fcbbc7cea93b452ed7 DATOS 68
fffffffffffff8000000000000000000000000000000 CLAVE 69
cf3f2576e2afedc74bb1ca7eeec1c0e7 DATOS 69
ffffffffffffc0000000000000000000000000000000 CLAVE 70
07c403f5f966e0e3d9f296d6226dca28 DATOS 70
ffffffffffffe0000000000000000000000000000000 CLAVE 71
c8c20908249ab4a34d6dd0a31327ff1a DATOS 71
fffffffffffff0000000000000000000000000000000 CLAVE 72
c0541329ecb6159ab23b7fc5e6a21bca DATOS 72
fffffffffffff8000000000000000000000000000000 CLAVE 73
7aa1acf1a2ed9ba72bc6deb31d88b863 DATOS 73
ffffffffffffc0000000000000000000000000000000 CLAVE 74
808bd8eddabb6f3bf0d5a8a27be1fe8a DATOS 74
ffffffffffffe0000000000000000000000000000000 CLAVE 75
273c7d7685e14ec66bbb96b8f05b6ddd DATOS 75
fffffffffffff0000000000000000000000000000000 CLAVE 76
32752eefc8c2a93f91b6e73eb07cca6e DATOS 76
fffffffffffff8000000000000000000000000000000 CLAVE 77
d893e7d62f6ce502c64f75e281f9c000 DATOS 77
ffffffffffffc0000000000000000000000000000000 CLAVE 78
8dfd999be5d0cfa35732c0ddc88ff5a5 DATOS 78
ffffffffffffe0000000000000000000000000000000 CLAVE 79
02647c76a300c3173b841487eb2bae9f DATOS 79
fffffffffffff0000000000000000000000000000000 CLAVE 80
172df8b02f04b53adab028b4e01acd87 DATOS 80
fffffffffffff8000000000000000000000000000000 CLAVE 81
054b3bf4998aeb05afd87ec536533a36 DATOS 81
ffffffffffffc0000000000000000000000000000000 CLAVE 82
3783f7bf44c97f065258a666cae03020 DATOS 82
ffffffffffffe0000000000000000000000000000000 CLAVE 83
aad4c8a63f80954104de7b92cede1be1 DATOS 83
fffffffffffff0000000000000000000000000000000 CLAVE 84
cbfe61810fd5467ccdacb75800f3ac07 DATOS 84
fffffffffffff8000000000000000000000000000000 CLAVE 85
830d8a2590f7d8e1b55a737f4af45f34 DATOS 85
ffffffffffffffc0000000000000000000000000000000 CLAVE 86
fffcd4683f858058e74314671d43fa2c DATOS 86
ffffffffffffffe0000000000000000000000000000000 CLAVE 87
523d0babbb82f46ebc9e70b1cd41ddd0 DATOS 87
fffffffffffffff0000000000000000000000000000000 CLAVE 88
344aab37080d7486f7d542a309e53eed DATOS 88
ffffffffffffffff8000000000000000000000000000000 CLAVE 89
56c5609d0906b23ab9caca816f5dbebd DATOS 89
ffffffffffffffc0000000000000000000000000000000 CLAVE 90
7026026eedd91adc6d831cdf9894bdc6 DATOS 90
ffffffffffffffe0000000000000000000000000000000 CLAVE 91
88330baa4f2b618fc9d9b021bf503d5a DATOS 91
fffffffffffffff0000000000000000000000000000000 CLAVE 92
fc9e0ea22480b0bac935c8a8ebefcdcf DATOS 92
fffffffffffffff8000000000000000000000000000000 CLAVE 93
29ca779f398fb04f867da7e8a44756cb DATOS 93
ffffffffffffffc0000000000000000000000000000000 CLAVE 94
51f89c42985786bfc43c6df8ada36832 DATOS 94
ffffffffffffffe0000000000000000000000000000000 CLAVE 95
6ac1de5fb8f21d874e91c53b560c50e3 DATOS 95
fffffffffffffff0000000000000000000000000000000 CLAVE 96
03aa9058490eda306001a8a9f48d0ca7 DATOS 96
fffffffffffffff8000000000000000000000000000000 CLAVE 97
e34ec71d6128d4871865d617c30b37e3 DATOS 97
ffffffffffffffffc0000000000000000000000000000 CLAVE 98
14be1c535b17cabd0c4d93529d69bf47 DATOS 98
ffffffffffffffffe0000000000000000000000000000 CLAVE 99
c9ef67756507beec9dd3862883478044 DATOS 99
fffffffffffffffff0000000000000000000000000000 CLAVE 100
40e231fa5a5948ce2134e92fc0664d4b DATOS 100

10.5.10. Fichero 5: Salidas Desencriptación con clave de 192 bits

Todas las salidas esperadas tienen valor cero.

10.5.11. Fichero 6: Entradas Desencriptación con clave de 256 bits

8000000000000000000000000000000000000000000000000000000000000000 CLAVE 1
e35a6dcb19b201a01ebcfa8aa22b5759 DATOS 1
c0000000000000000000000000000000000000000000000000000000000000000 CLAVE 2
b29169cdcf2d83e838125a12ee6aa400 DATOS 2
e0000000000000000000000000000000000000000000000000000000000000000 CLAVE 3
d8f3a72fc3cdf74dfaf6c3e6b97b2fa6 DATOS 3
f0000000000000000000000000000000000000000000000000000000000000000 CLAVE 4
1c777679d50037c79491a94da76a9a35 DATOS 4
f8000000000000000000000000000000000000000000000000000000000000000 CLAVE 5
9cf4893ecafa0a0247a898e040691559 DATOS 5
fc0000000000000000000000000000000000000000000000000000000000000000 CLAVE 6
8fbb413703735326310a269bd3aa94b2 DATOS 6
fe0000000000000000000000000000000000000000000000000000000000000000 CLAVE 7
60e32246bed2b0e859e55c1cc6b26502 DATOS 7
ff0000000000000000000000000000000000000000000000000000000000000000 CLAVE 8
ec52a212f80a09df6317021bc2a9819e DATOS 8
ff8000000000000000000000000000000000000000000000000000000000000000 CLAVE 9
f23e5b600eb70dbccf6c0b1d9a68182c DATOS 9
ffc0000000000000000000000000000000000000000000000000000000000000000 CLAVE 10
a3f599d63a82a968c33fe26590745970 DATOS 10
ffe0000000000000000000000000000000000000000000000000000000000000000 CLAVE 11
d1ccb9b1337002cbac42c520b5d67722 DATOS 11
fff0000000000000000000000000000000000000000000000000000000000000000 CLAVE 12
cc111f6c37cf40a1159d00fb59fb0488 DATOS 12
fff8000000000000000000000000000000000000000000000000000000000000000 CLAVE 13
dc43b51ab609052372989a26e9cdd714 DATOS 13
fffc0000000000000000000000000000000000000000000000000000000000000000 CLAVE 14
4dcede8da9e2578f39703d4433dc6459 DATOS 14
fffe0000000000000000000000000000000000000000000000000000000000000000 CLAVE 15
1a4c1c263bbccfafc11782894685e3a8 DATOS 15
ffff0000000000000000000000000000000000000000000000000000000000000000 CLAVE 16
937ad84880db50613423d6d527a2823d DATOS 16
ffff8000000000000000000000000000000000000000000000000000000000000000 CLAVE 17
610b71dfc688e150d8152c5b35ebc14d DATOS 17
ffffc0000000000000000000000000000000000000000000000000000000000000000 CLAVE 18
27ef2495dabf323885aab39c80f18d8b DATOS 18
ffffe0000000000000000000000000000000000000000000000000000000000000000 CLAVE 19
633cafea395bc03adae3a1e2068e4b4e DATOS 19
fffff0000000000000000000000000000000000000000000000000000000000000000 CLAVE 20
6e1b482b53761cf631819b749a6f3724 DATOS 20
fffff8000000000000000000000000000000000000000000000000000000000000000 CLAVE 21
976e6f851ab52c771998dbb2d71c75a9 DATOS 21
fffffc0000000000000000000000000000000000000000000000000000000000000000 CLAVE 22
85f2ba84f8c307cf525e124c3e22e6cc DATOS 22
fffffe0000000000000000000000000000000000000000000000000000000000000000 CLAVE 23
6bcca98bf6a835fa64955f72de4115fe DATOS 23
ffffff0000000000000000000000000000000000000000000000000000000000000000 CLAVE 24
2c75e2d36eebd65411f14fd0eb1d2a06 DATOS 24
ffffff8000000000000000000000000000000000000000000000000000000000000000 CLAVE 25
bd49295006250ffca5100b6007a0eade DATOS 25
ffffffc0000000000000000000000000000000000000000000000000000000000000000 CLAVE 26
a190527d0ef7c70f459cd3940df316ec DATOS 26
ffffffe0000000000000000000000000000000000000000000000000000000000000000 CLAVE 27
bbd1097a62433f79449fa97d4ee80dbf DATOS 27
fffffff0000000000000000000000000000000000000000000000000000000000000000 CLAVE 28
07058e408f5b99b0e0f061a1761b5b3b DATOS 28
fffffff8000000000000000000000000000000000000000000000000000000000000000 CLAVE 29
5fd1f13fa0f31e37fabde328f894eac2 DATOS 29
fffffffc0000000000000000000000000000000000000000000000000000000000000000 CLAVE 30
fc4af7c948df26e2ef3e01c1ee5b8f6f DATOS 30
fffffffe0000000000000000000000000000000000000000000000000000000000000000 CLAVE 31
829fd7208fb92d44a074a677ee9861ac DATOS 31
ffffffff0000000000000000000000000000000000000000000000000000000000000000 CLAVE 32
ad9fc613a703251b54c64a0e76431711 DATOS 32
ffffffff8000000000000000000000000000000000000000000000000000000000000000 CLAVE 33
33ac9eccc4cc75e2711618f80b1548e8 DATOS 33
ffffffffc0000000000000000000000000000000000000000000000000000000000000000 CLAVE 34
2025c74b8ad8f4cda17ee2049c4c902d DATOS 34

fffffffe0000000000000000000000000000000000000000000000000000000000 CLAVE 35
f85ca05fe528f1ce9b790166e8d551e7 DATOS 35
ffffffff0000000000000000000000000000000000000000000000000000000000 CLAVE 36
6f6238d8966048d4967154e0dad5a6c9 DATOS 36
ffffffff8000000000000000000000000000000000000000000000000000000000 CLAVE 37
f2b21b4e7640a9b3346de8b82fb41e49 DATOS 37
fffffffc0000000000000000000000000000000000000000000000000000000000 CLAVE 38
f836f251ad1d11d49dc344628b1884e1 DATOS 38
fffffffe0000000000000000000000000000000000000000000000000000000000 CLAVE 39
077e9470ae7abea5a9769d49182628c3 DATOS 39
ffffffff0000000000000000000000000000000000000000000000000000000000 CLAVE 40
e0dcc2d27fc9865633f85223cf0d611f DATOS 40
ffffffff8000000000000000000000000000000000000000000000000000000000 CLAVE 41
be66cfea2fecd6bf0ec7b4352c99bcaa DATOS 41
fffffffc0000000000000000000000000000000000000000000000000000000000 CLAVE 42
df31144f87a2ef523facdcf21a427804 DATOS 42
fffffffe0000000000000000000000000000000000000000000000000000000000 CLAVE 43
b5bb0f5629fb6aae5e1839a3c3625d63 DATOS 43
fffffffff0000000000000000000000000000000000000000000000000000000000 CLAVE 44
3c9db3335306fe1ec612bdbfae6b6028 DATOS 44
fffffffff8000000000000000000000000000000000000000000000000000000000 CLAVE 45
3dd5c34634a79d3cfcc8339760e6f5f4 DATOS 45
fffffffffc0000000000000000000000000000000000000000000000000000000000 CLAVE 46
82bda118a3ed7af314fa2ccc5c07b761 DATOS 46
fffffffffe0000000000000000000000000000000000000000000000000000000000 CLAVE 47
2937a64f7d4f46fe6fea3b349ec78e38 DATOS 47
fffffffff0000000000000000000000000000000000000000000000000000000000 CLAVE 48
225f068c28476605735ad671bb8f39f3 DATOS 48
fffffffff8000000000000000000000000000000000000000000000000000000000 CLAVE 49
ae682c5ecd71898e08942ac9aa89875c DATOS 49
fffffffffc0000000000000000000000000000000000000000000000000000000000 CLAVE 50
5e031cb9d676c3022d7f26227e85c38f DATOS 50
fffffffffe0000000000000000000000000000000000000000000000000000000000 CLAVE 51
a78463fb064db5d52bb64bfef64f2dda DATOS 51
ffffffffff0000000000000000000000000000000000000000000000000000000000 CLAVE 52
8aa9b75e784593876c53a00eae5af52b DATOS 52
ffffffffff8000000000000000000000000000000000000000000000000000000000 CLAVE 53
3f84566df23da48af692722fe980573a DATOS 53
ffffffffffc0000000000000000000000000000000000000000000000000000000000 CLAVE 54
31690b5ed41c7eb42a1e83270a7ff0e6 DATOS 54
ffffffffffe0000000000000000000000000000000000000000000000000000000000 CLAVE 55
77dd7702646d55f08365e477d3590eda DATOS 55
fffffffffff0000000000000000000000000000000000000000000000000000000000 CLAVE 56
4c022ac62b3cb78d739cc67b3e20bb7e DATOS 56
fffffffffff8000000000000000000000000000000000000000000000000000000000 CLAVE 57
092fa137ce18b5dfe7906f550bb13370 DATOS 57
ffffffffffc0000000000000000000000000000000000000000000000000000000000 CLAVE 58
3e0cdadf2e68353c0027672c97144dd3 DATOS 58
ffffffffffe0000000000000000000000000000000000000000000000000000000000 CLAVE 59
d8c4b200b383fc1f2b2ea677618a1d27 DATOS 59
fffffffffff0000000000000000000000000000000000000000000000000000000000 CLAVE 60
11825f99b0e9bb3477c1c0713b015aac DATOS 60
fffffffffff8000000000000000000000000000000000000000000000000000000000 CLAVE 61
f8b9fffb5c187f7ddc7ab10f4fb77576 DATOS 61
fffffffffffc0000000000000000000000000000000000000000000000000000000000 CLAVE 62
ffb4e87a32b37d6f2c8328d3b5377802 DATOS 62
fffffffffffe0000000000000000000000000000000000000000000000000000000000 CLAVE 63
d276c13a5d220f4da9224e74896391ce DATOS 63
ffffffffffff0000000000000000000000000000000000000000000000000000000000 CLAVE 64
94efe7a0e2e031e2536da01df799c927 DATOS 64
ffffffffffff8000000000000000000000000000000000000000000000000000000000 CLAVE 65
8f8fd822680a85974e53a5a8eb9d38de DATOS 65
ffffffffffffc0000000000000000000000000000000000000000000000000000000000 CLAVE 66
e0f0a91b2e45f8cc37b7805a3042588d DATOS 66
ffffffffffffe0000000000000000000000000000000000000000000000000000000000 CLAVE 67
597a6252255e46d6364dbeeda31e279c DATOS 67
ffffffffffffff0000000000000000000000000000000000000000000000000000000000 CLAVE 68
f51a0f694442b8f05571797fec7ee8bf DATOS 68
ffffffffffffff8000000000000000000000000000000000000000000000000000000000 CLAVE 69
9ff071b165b5198a93dddeebc54d09b5 DATOS 69
ffffffffffffffc0000000000000000000000000000000000000000000000000000000000 CLAVE 70
c20a19fd5758b0c4bc1a5df89cf73877 DATOS 70
ffffffffffffffe0000000000000000000000000000000000000000000000000000000000 CLAVE 71
971201663071l9ca2280e9315668e96f DATOS 71
fffffffffffffff0000000000000000000000000000000000000000000000000000000000 CLAVE 72
4b3b9f1e099c2a09dc091e90e4f18f0a DATOS 72
fffffffffffffff8000000000000000000000000000000000000000000000000000000000 CLAVE 73
eb040b891d4b37f6851f7ec219cd3f6d DATOS 73
ffffffffffffffc0000000000000000000000000000000000000000000000000000000000 CLAVE 74
9f0fdec08b7fd79aa39535bea42db92a DATOS 74
ffffffffffffffe0000000000000000000000000000000000000000000000000000000000 CLAVE 75
2e70f168fc74bf911df240bcd2cef236 DATOS 75
fffffffffffffff0000000000000000000000000000000000000000000000000000000000 CLAVE 76

462ccd7f5fd1108dbc152f3cacad328b DATOS 76
ffffffffffffff8000000000000000000000000000000000000000000000000 CLAVE 77
a4af534a7d0b643a01868785d86dfb95 DATOS 77
ffffffffffffffc0000000000000000000000000000000000000000000000000 CLAVE 78
ab980296197e1a5022326c31da4bf6f3 DATOS 78
ffffffffffffffe0000000000000000000000000000000000000000000000000 CLAVE 79
f97d57b3333b6281b07d486db2d4e20c DATOS 79
fffffffffffffff0000000000000000000000000000000000000000000000000 CLAVE 80
f33fa36720231afe4c759ade6bd62eb6 DATOS 80
fffffffffffffff8000000000000000000000000000000000000000000000000 CLAVE 81
fdcfac0c02ca538343c68117e0a15938 DATOS 81
fffffffffffffffc0000000000000000000000000000000000000000000000000 CLAVE 82
ad4916f5ee5772be764fc027b8a6e539 DATOS 82
fffffffffffffffe0000000000000000000000000000000000000000000000000 CLAVE 83
2e16873e1678610d7e14c02d002ea845 DATOS 83
ffffffffffffffff0000000000000000000000000000000000000000000000000 CLAVE 84
4e6e627c1acc51340053a8236d579576 DATOS 84
ffffffffffffffff8000000000000000000000000000000000000000000000000 CLAVE 85
ab0c8410aeeead92feec1eb430d652cb DATOS 85
ffffffffffffffffc0000000000000000000000000000000000000000000000000 CLAVE 86
e86f7e23e835e114977f60e1a592202e DATOS 86
ffffffffffffffffe0000000000000000000000000000000000000000000000000 CLAVE 87
e68ad5055a367041fade09d9a70a794b DATOS 87
fffffffffffffffff0000000000000000000000000000000000000000000000000 CLAVE 88
0791823a3c666bb6162825e78606a7fe DATOS 88
fffffffffffffffff8000000000000000000000000000000000000000000000000 CLAVE 89
dcca366a9bf47b7b868b77e25c18a364 DATOS 89
fffffffffffffffffc0000000000000000000000000000000000000000000000000 CLAVE 90
684c9efc237e4a442965f84bce20247a DATOS 90
fffffffffffffffffe0000000000000000000000000000000000000000000000000 CLAVE 91
a858411ffbe63fdb9c8aa1bfaed67b52 DATOS 91
ffffffffffffffffff0000000000000000000000000000000000000000000000000 CLAVE 92
04bc3da2179c3015498b0e03910db5b8 DATOS 92
ffffffffffffffffff8000000000000000000000000000000000000000000000000 CLAVE 93
40071eeab3f935dbc25d00841460260f DATOS 93
ffffffffffffffffffc0000000000000000000000000000000000000000000000000 CLAVE 94
0ebd7c30ed2016e08ba806ddb008bcc8 DATOS 94
ffffffffffffffffffe0000000000000000000000000000000000000000000000000 CLAVE 95
15c6becf0f4cec7129cbd22d1a79b1b8 DATOS 95
fffffffffffffffffff0000000000000000000000000000000000000000000000000 CLAVE 96
0aeede5b91f721700e9e62edbf60b781 DATOS 96
fffffffffffffffffff8000000000000000000000000000000000000000000000000 CLAVE 97
266581af0dcfbed1585e0a242c64b8df DATOS 97
ffffffffffffffffffc0000000000000000000000000000000000000000000000000 CLAVE 98
6693dc911662ae473216ba22189a511a DATOS 98
ffffffffffffffffffe0000000000000000000000000000000000000000000000000 CLAVE 99
7606fa36d86473e6fb3a1bb0e2c0adf5 DATOS 99
fffffffffffffffffff0000000000000000000000000000000000000000000000000 CLAVE 100
112078e9e11fbb78e26ffb8899e96b9a DATOS 100

10.5.12. Fichero 6: Salidas Desencriptación con clave de 256 bits

Todas las salidas esperadas tienen valor cero.