UNIVERSIDAD NACIONAL DEL SUR

TESIS DE MAGÍSTER EN INGENIERÍA

Diseño de tres arquitecturas para un módulo criptográfico AES

Ing. Paola Anabella Ceminari

Bahía Blanca Argentina

2021

Prefacio

Esta Tesis se presenta como parte de los requisitos para optar al grado Académico de Magister en Ingeniería, de la Universidad Nacional del Sur y no ha sido presentada previamente para la obtención de otro título en esta Universidad u otra. La misma contiene los resultados obtenidos en investigaciones llevadas a cabo en el Departamento de Ingeniería Eléctrica y de Computadoras durante el período comprendido entre agosto de 2014 y Junio de 2019, bajo la dirección del Dr. Pablo Mandolesi.

Resumen

La seguridad de la información cumple un rol cada vez más importante en la actualidad. Esto motivó al Centro de Micro y Nanoelectrónica del Bicentenario (CMNB) del Instituto Nacional de Tecnología Industrial (INTI) a incorporar a su biblioteca de bloques de Propiedad Intelectual (IP) un módulo criptográfico capaz de ser integrado en cualquier sistema de mayor complejidad en el que se requiera confidencialidad. Dentro de la diversidad de algoritmos criptográficos existentes se decidió implementar AES (*Advanced Encryption Standard*) por ser ampliamente usado hoy en día y estar libre de licencias o regalías.

Cada una de las aplicaciones impone distintos requisitos en el módulo de cifrado. El diseño de un mismo módulo AES que cumpla con los requisitos de todo el rango de aplicaciones posibles no es factible en la práctica. En este trabajo se propone el diseño de tres arquitecturas para un módulo de cifrado en bloques AES, mediante distintas técnicas de diseño de circuitos integrados digitales. Estas arquitecturas se denominan *básica*, *compacta* y *pipeline*, cada una de ellas está destinada a un rango de aplicaciones distinto y su diseño se orienta a la implementación en FPGA.

Abstract

Information security plays an increasingly important role today. This motivated CMNB to incorporate a cipher module to its IP library, capable of being integrated in a more complex system that needs confidenciality. Within the diversity of existing cryptographic algorithms, AES (*Advanced Encryption Standard*) was chosen to be implemented, since it is widely used today and is free of licenses or royalties.

Each application imposes different requirements on the encryption module. Designing a single AES module that meets the requirements of the full range of possible applications is not feasible in practice. In this work, the design of three architectures for an AES block encryption module is proposed, using different digital integrated circuit design techniques. These architectures are called *basic*, *compact* and *pipeline*, each one intended for a particular range of applications and their design is oriented towards FPGA implementation.

Agradecimientos

Quisiera agradecer en primer lugar al Dr. Pablo Mandolesi, guía fundamental en el camino para llegar a esta Tesis. Gracias por los consejos, el apoyo y el acompañamiento. Gracias a los jurados, Dres. Alfonso Chacón Rodriguez, Joel Gak Szollosy y Pedro Julián por sus enriquecedoras correcciones.

Gracias también a todas las personas con las trabajé en el Centro de Micro y Nanoelectrónica del INTI, por enseñarme tanto y compartir momentos que siempre quedarán en mi corazón. Especialmente al Dr. Martín Di Federico, por introducirme en el mundo del diseño digital, y al Dr. Ariel Arelovich por su contribución en el desarrollo del modelo de referencia utilizado durante las simulaciones.

También quisiera agradecer a la Dra. Laura Rueda, profesora adjunta del Departamento de Matemática, cuya ayuda desinteresada fue de suma importancia para comprender el álgebra de campos finitos, totalmente desconocida para mí antes de comenzar con esta Tesis. Gracias a los profesores de los distintos cursos realizados durante el transcurso de la Maestría, gracias a la Uiversidad Nacional del Sur por brindar educación de calidad.

Gracias también a mi familia: Marcela, Chichito, Juli y Sofi por el apoyo constante y el incentivo en los momentos en los que la rutina y el trabajo hicieron que pierda el foco. Gracias a mis abuelos y tíos, que siempre están ahí. Una mención especial a Mati, mi compañero de vida, gracias por ser incondicional y por los mates que hicieron más llevaderas las tardes de redacción.

Por último quisiera también agradecer a mis amigas y amigos, a los de siempre y a los más nuevos. Gracias por darme la posibilidad de compartir la vida y celebrar juntos los momentos de alegría, que con ustedes al lado se sienten aún mejor.

Índice General

Índice de Figuras

Índice de Tablas

Abreviaturas

AES	**A**dvanced **E**ncryption **S**tandard
ANSII	**A**gence **N**ationale de la **S**écurité des **S**ystèmes d'**I**nformation
CBC	**C**ipher **B**lock **C**haining Mode
CFB	**C**ipher **F**eed**B**ack Mode
CLB	**C**onfigurable **L**ogic **B**lock
DES	**D**ata **E**ncryption **S**tandard
ECB	**E**lectronic **C**ode**B**ook Mode
ECC	**E**lliptic **C**urve **C**ryptography
FPGA	**F**ield **P**rogrammable **G**ate **A**rray
FSM	**F**inite **S**tate **M**achine
IDEA	**I**nternational **D**ata **E**ncryption **A**lgorithm
LUT	**L**ook**U**p **T**able
NIST	**N**ational **I**nstitute of **S**tandards and **T**echnology
NSA	**N**ational **S**ecurity **A**gency
OFB	**O**utput **F**eed**B**ack
RC4	**R**ivest **C**ipher 4
RC6	**R**ivest **C**ipher 6
RSA	**R**ivest **S**hamir and **A**dleman
SHA	**S**ecure **H**ash **A**lgotithm
SPN	**S**ubstitution **P**ermutation **N**etwork

Capítulo 1

Introducción

La criptografía es una disciplina que estudia diferentes métodos para transformar un mensaje en un criptograma. Generalmente se utilizan algoritmos que, en relación a una variable denominada *llave*, vuelve al mensaje incomprensible e indescifrable para quien no tiene conocimiento de la llave. La función es reversible: la aplicación de un algoritmo de descifrado al criptograma devuelve el mensaje original. La seguridad de los algoritmos de cifrado depende en gran medida de la dificultad que tiene un potencial intruso para obtener el valor de la llave.

La criptografía moderna en sus orígenes fue mayormente utilizada en la trasmisión de mensajes militares. Durante la Primera y Segunda Guerras Mundiales se desarrollaron y mejoraron algoritmos de cifrado, así como técnicas de análisis necesarias para corromper la seguridad de los algoritmos (rama de la criptografía denominada cripotanálisis). A pesar de que uno de los mayores incentivos para el desarrollo de la criptografía fue la actividad militar, los avances tecnológicos implican la necesidad de contar con herramientas de seguridad en diversas actividades de la vida cotidiana, aunque muchas veces no lo notemos. Tanto las conversaciones por Whatsapp como las transferencias bancarias y las conexiones de un dispositivo a internet mediante WiFi, solo para mencionar algunos ejemplos, hacen uso de sistemas criptográficos para proteger la información personal de los usuarios.

El espectro de aplicaciones de los sistemas criptográficos es cada vez mayor. Un claro ejemplo es la tecnología denominada Internet de las Cosas (IoT), que se basa en la

interconexión de millones de dispositivos a través de internet. En este escenario el cifrado se vuelve indispensable para mantener la seguridad de un sistema.

El desarrollo del algoritmo de cifrado *Advanced Encryption Standard (AES)* fue impulsado por el Instituto Nacional de Estándares y Tecnología de Estados Unidos (NIST), que en enero de 1997 anunció el comienzo de una iniciativa para reemplazar al *Data Encryption Standard (DES)*.

El proceso de selección de AES fue abierto: distintos grupos de investigación propusieron sus algoritmos y la evaluación de seguridad y eficiencia necesarias para elegir a uno de ellos fue realizada por el NIST en conjunto con investigadores y aficionados a la criptografía, quienes llevaron a cabo diferentes pruebas sobre los algoritmos propuestos. En 1999 se anunciaron los cinco finalistas: MARS, RC6, Rijndael, Serpent y Twofish. Finalmente, en octubre del año 2000, el algoritmo Rijndael (Daemen y Rijmen, 1998) fue proclamado ganador.

Durante las evaluaciones llevadas a cabo en el proceso de selección de AES, (Ichikawa, Kasuya y Matsui, 2000; Elbirt, Yip y Paar, 2001; Chodowiec, 2002), uno de los factores considerados fue el desempeño de los algoritmos al implementarlos utilizando distintas plataformas: software, FPGA o ASIC. Estos tres enfoques presentan distintas características.

- En cuanto al desempeño, tanto ASICs como FPGAs soportan procesamiento paralelo y tamaño de palabra variable; mientras que para las implementaciones en software el procesamiento paralelo está limitado por la cantidad de estructuras internas con las que cuenta el microprocesador y el nivel de paralelismo de las instrucciones; además de que en un microprocesador el ancho de palabra es fijo.
- En cuanto a la velocidad, un circuito implementado en FPGA es generalmente más lento que su implementación en ASIC (considerando que son fabricados utilizando transistores del mismo tamaño), debido a los retardos introducidos por la lógica requerida en el primero para la reconfiguración.
- Con respecto al proceso de desarrollo, las implementaciones en software son menos costosas y cuentan con un ciclo de diseño corto; mientras que para ASIC el tiempo requerido y los costos son mayores. Los FPGA representan una relación

de compromiso, debido a que en este caso tanto el costo como el tiempo de diseño son moderados.

- Un punto que debe considerarse al implementar un módulo de cifrado es la resistencia a ataques. En este sentido, los ASIC presentan la mayor fortaleza, mientras que los FPGA están limitados y las aplicaciones en software son más vulnerables que las dos anteriores.

Las evaluaciones llevados a cabo durante la selección comprobaron que Rijndael, el algoritmo seleccionado como estándar AES, puede ser implementado en forma eficiente sobre cualquiera de las tres plataformas mencionadas.

1.1. Motivación

El hecho de que la seguridad de la información sea tan importante y se requiera en diversas aplicaciones motivó al Centro de Micro y Nanoelectrónica del Bicentenario (CMNB) del Instituto Nacional de Tecnología Industrial (INTI) a incorporar a su biblioteca de bloques de Propiedad Intelectual (IP) un módulo criptográfico capaz de ser integrado en cualquier sistema de mayor complejidad en el que se requiera confidencialidad. Dentro de la diversidad de algoritmos criptográficos existentes se decidió implementar AES (*Advanced Encryption Standard*) por ser ampliamente usado hoy en día y estar libre de licencias o regalías.

AES se utiliza actualmente en distintos ámbitos que requieren seguridad, como telefonía a través de internet (VoIP), manejo de archivos (WinZip y RAR, BitLocker, FileVault, y CipherShed), VPNs (*Virtual Private Networks*, que admiten el uso de conexiones de internet públicas para conectarse a redes más seguras) y en la protección WPA2 y WPA3 de redes WiFi. En cuanto a IoT, AES se utiliza en los estándares de comunicación LoraWan, IEEE 802.15.4, Sigfox y ZWave. La finalidad de este algoritmo en todos los casos es proveer confidencialidad a una comunicación. Aun así, cada una de estas aplicaciones requiere distintos requisitos en el módulo de cifrado. El diseño de un mismo módulo AES que cumpla con los requisitos impuestos por todo el rango de aplicaciones posibles no es factible en la práctica. En este trabajo se propone el diseño de tres arquitecturas para un módulo de cifrado en bloques AES, mediante distintas técnicas de diseño de circuitos integrados digitales.

1.2. Objetivos

El objetivo principal de este trabajo es el estudio y aplicación de diferentes técnicas de diseño digital utilizando lenguaje de descripción de *hardware* (HDL). Este objetivo global se ve plasmado en la implementación de tres módulos que tienen una misma funcionalidad (cifrado AES), pero poseen diferentes características en cuanto a desempeño y área. Estas arquitecturas se denominan *básica*, *compacta* y *pipeline*, cada una de ellas está destinada a un rango de aplicaciones distinto:

- Básica: esta versión presenta una relación de compromiso media entre área y velocidad, apropiada para aplicaciones como voz sobre IP (VoIP).
- Compacta: minimiza el área y los recursos necesarios por el sistema de cifrado. Por este motivo, esta implementación está destinada principalmente a dispositivos inalámbricos, como pueden ser nodos IoT.
- Pipeline: prioriza el desempeño del sistema criptográfico, lo que la hace adecuada para sistemas de mayor velocidad, como Wi-Fi.

Es importante destacar que las técnicas estudiadas y utilizadas para lograr este objetivo no se limitan a módulos criptográficos, sino que pueden ser utilizadas en forma general al diseñar bloques digitales con distinta funcionalidad.

Considerando que los FPGA son rápidos, admiten paralelismo e implican un ciclo de diseño corto y flexible se decidió implementar los tres bloques en este tipo de plataforma. Desde el punto de vista de la implementación de un bloque de cifrado, los FPGA tienen como ventaja, además, la disponibilidad de bloques de memoria RAM y la posibilidad de ajustar fácilmente las operaciones lógicas que conforman un algoritmo de este tipo (principalmente desplazamientos, sustituciones y operaciones XOR) a la estructura de los CLBs (bloques lógicos configurables) presentes en los arreglos programables modernos.

Los tres módulos desarrollados serán destinados a una biblioteca de bloques IP, de manera que puedan ser utilizados en sistemas de mayor complejidad que requieran el cifrado de datos. Con esta finalidad se decidió incorporar a los tres bloques propuestos una interfaz AMBA AHB.

Los objetivos particulares que se proponen son:

- Estudiar las características generales de los algoritmos de cifrado.
- Estudiar en detalle el estándar de cifrado en bloques AES y analizar las implementaciones reportadas en la bibliografía. Este objetivo implica además el estudio de las operaciones de álgebra de campos finitos que se llevan a cabo durante el proceso de cifrado.
- Diseñar tres arquitecturas que cumplan con el estándar AES y presenten distintas características en cuanto a desempeño y área. Describirlas en lenguaje HDL utilizando diferentes técnicas de diseño digital y haciendo un uso eficiente de los recursos disponibles en los FPGA.
- Realizar un modelo de referencia que permita generar vectores de prueba.
- Verificar mediante simulaciones el correcto funcionamiento de las tres arquitecturas propuestas, utilizando como referencia los pares entrada-salida generados mediante el modelo de referencia.
- Una vez verificada la funcionalidad de las arquitecturas, incorporar una interfaz AMBA AHB a cada una de ellas. Este objetivo implica el estudio de este protocolo de comunicación, ampliamente usado hoy en día para el desarrollo de SoCs.
- Sintetizar las arquitecturas para FPGA, utilizando diferentes estrategias de síntesis disponibles en la herramienta de Xilinx.

1.3. Organización de la tesis

Este documento se encuentra organizado de la siguiente manera:

- En el Capítulo 2 se recorre brevemente la historia de la criptografía y se presenta la clasificación de los algoritmos de cifrado actuales y su estructura. Esta base teórica es necesaria para una mejor comprensión de las operaciones que componen el algoritmo de cifrado AES.
- En el Capítulo 3 se describen en detalle las operaciones que componen tanto el cifrado como el descifrado en el estándar AES. En este capítulo también se detallan los modos de operación básicos para un módulo de cifrado en bloques

- En el Capítulo 4 se detalla cuáles son los parámetros a tener en cuenta durante el diseño de un módulo de cifrado AES. Para cada uno de estos parámetros se analizan sus variantes y se establece cuáles de ellos serán constantes para las tres arquitecturas propuestas y por qué. Por otro lado, para aquellos parámetros que no son fijos se analizan las diferentes opciones y su impacto.
- En los Capítulos 5, 6 y 7 se presenta el diseño de las arquitecturas (*Básica, Pipeline* y *Compacta* respectivamente. Para cada de una de ellas se presenta la estructura utilizada y el detalle de implementación para cada ronda de cifrado, así como la interfaz AMBA para comunicarse con un maestro que controla su operación. En los tres casos se presentan también los resultados obtenidos al simular los bloques.
- En el Capítulo 8 se presentan los resultados obtenidos al sintetizar cada uno de los módulos propuestos, utilizando la herramienta Vivado de Xilinx.
- En el Capítulo 9 se presentan las conclusiones.
- El trabajo cuenta además con dos apéndices. El primero de ellos describe en forma resumida el álgebra de campos finitos, que es la base teórica de las operaciones algebraicas que se desarrollan durante el cifrado o descifrado. Por otro lado, un segundo apéndice brinda información adicional sobre el bus de comunicaciones AMBA AHB. En el mismo se detalla el temporizado correspondiente a las operaciones de lectura y escritura en este tipo de bus.

Capítulo 2

Introducción a la criptografía

La utilización de estrategias para mantener la confidencialidad de información sensible es tan antigua como la comunicación en sí misma. La palabra criptografía proviene del griego (criptos = oculto y grafo = escritura, literalmente *escritura oculta*) y se define como la aplicación de diferentes técnicas para alterar una representación lingüística de ciertos mensajes con el fin de hacerlos ininteligibles a receptores no autorizados.

El cifrado o encriptación de un mensaje consiste en la alteración de su contenido, de manera que sólo pueda ser interpretado correctamente por receptores autorizadas. Los elementos que intervienen en este proceso se ilustran en la Figura 2.1:

- Texto plano: mensaje a cifrar.
- Llave: secuencia de caracteres que funciona a modo de "contraseña", también se la denomina *clave* y se usa como operando durante los procesos de cifrado y descifrado. Si se conoce el algoritmo y la cantidad de posibles llaves es reducida, para un posible atacante es suficiente con intentar con cada uno de estos posibles valores de llave hasta obtener el mensaje. Este tipo de ataque es el más básico y se denomina como *ataque de fuerza bruta*. En los algoritmos actuales las llaves utilizadas tienen una cantidad de bits suficiente como para que un ataque de este tipo lleve años, haciéndolo irrealizable en la práctica.
- Texto cifrado: mensaje cifrado, también denominado criptograma.

FIGURA 2.1: Cifrado y descifrado de un mensaje.

Una de las técnicas más antiguas conocidas es el cifrado Caesar, que era utilizado por el emperador romano Julio Cesar para comunicarse en forma secreta con sus colaboradores. Este método consiste en la aplicación de un desplazamiento de tres letras hacia la izquierda en el alfabeto. De esta manera, cada aparición de la letra D en el texto es reemplazada por la A; cada aparición de la letra E es reemplazada por la B, y así sucesivamente.

Texto plano: ABCDEFGHIJKLMNOPQRSTUVWXYZ

Texto cifrado: XYZABCDEFGHIJKLMNOPQRSTUVW

Este tipo de cifrado evolucionó y surgieron métodos más robustos en los que el desplazamiento en el alfabeto (que corresponde a la llave en este tipo de algoritmo) no es fijo para todos los caracteres. Por ejemplo, el cifrado de Vigenère (Battista, 1553) al que se lo conocía como el código indescifrable (*le chiffre indéchiffrable* en francés). Sin embargo, trescientos años después se publicó el primer libro en el que se presentó el criptoanálisis de este algoritmo (Kasiski, 1863). El método Kasiski consiste en determinar la longitud de la clave en un cifrado Vigenère a partir de la búsqueda de palabras repetidas en el texto cifrado, lo cual significa casi con toda probabilidad que dichas palabras no sólo eran la misma antes del cifrado sino que además la clave coincidió en la misma posición en ambas ocurrencias.

En el año 1883, el criptógrafo holandés Auguste Kerckhoffs enunció seis axiomas fundamentales relativos a las propiedades deseables de un sistema criptográfico:

1. Si el sistema no es teóricamente irrompible, al menos debe serlo en la práctica.
2. La efectividad del sistema no debe depender de que su diseño permanezca en secreto.

3. La clave debe ser fácilmente memorizable de manera que no haya que recurrir a notas escritas.
4. Los criptogramas deberán dar resultados alfanuméricos.
5. El sistema debe ser operable por una única persona.
6. El sistema debe ser fácil de utilizar.

El segundo de ellos, denominado como principio de Kerckhoffs, es de suma importancia aún en la actualidad. El mismo enuncia que la seguridad de un sistema de este tipo sólo depende de la confidencialidad de la llave (Kerckhoffs, 1883). La intención de Kerckhoffs era demostrar que el método utilizado para cifrar una comunicación debe ser seguro aun si un potencial adversario conoce con detalle el algoritmo. El cumplimiento de este principio en la actualidad está dado en el hecho de que muchos de los algoritmos de cifrado más utilizados son de conocimiento público. De esta forma es posible estandarizarlos y aumentar su confiabilidad, ya que se los estudia en profundidad y se comprueba su seguridad ante distintos ataques.

Hasta mediados del siglo XX, la criptografía fue consideraba más bien un arte que una ciencia, debido a que los algoritmos de cifrado no contaban con un respaldo teórico sino que se basaban en el ingenio y la creatividad. Generalmente se suele relacionar el inicio de la criptografía moderna con el estudio realizado por Claude Shannon (Shannon, 1949), quien realizó desarrollos matemáticos relacionados a la criptografía y definió que los objetivos de la criptografía son confidencialidad y autenticidad. En su trabajo, Shannon incorporó los conceptos de confusión y difusión:

- Confusión: relación, matemáticamente compleja, entre el texto cifrado y la llave utilizada. La complejidad de esta relación evita que la llave pueda ser deducida a partir del texto cifrado. El mecanismo empleado para conseguir la confusión es la sustitución, que consiste en sustituir cada elemento del texto plano por otro u otros elementos en el texto cifrado.
- Difusión: influencia de un símbolo perteneciente al texto plano sobre múltiples símbolos del texto cifrado. Un alto grado de difusión en un algoritmo implica que si se cambia un bit en el texto plano, cambia la mayor cantidad posible de bits en el texto cifrado. La difusión se logra generalmente mediante permutaciones en los caracteres de un mensaje durante el cifrado.

Además del desarrollo teórico de los algoritmos de cifrado, otro de los hitos que marcó el comienzo de la criptografía moderna fue el uso masivo de este tipo de técnicas. Mientras que en el pasado su uso se limitaba a instituciones gubernamentales o militares, actualmente se requieren comunicaciones seguras en operaciones cotidianas llevadas a cabo por millones de personas en todo el mundo, como transacciones bancarias, servicios de comunicación, uso de tarjetas de crédito, correo electrónico, entre otras.

Los avances tecnológicos dieron paso a nuevos requerimientos en los sistemas criptográficos. La criptografía moderna ofrece las herramientas necesarias para establecer una comunicación segura (que hoy en día no sólo abarca la ofuscación de un mensaje), sus cuatro objetivos son confidencialidad, integridad, autenticación y no repudio. Estos términos se definen en el SGSI (Sistema de Gestión de Seguridad de la Información), concepto central sobre el que se construye el estándar denominado ISO 27001.

- Confidencialidad: según la definición propuesta en las normas ISO, confidencialidad es una propiedad de la información que pretende garantizar el acceso sólo a las personas autorizadas. Esto se logra a partir del cifrado de un mensaje.
- Control de integridad: el propósito del control de integridad es garantizar que la información no sea alterada, eliminada o destruida por entidades o personas no autorizadas. Esto se logra mediante la detección de alteraciones en un mensaje durante su transmisión, ya sean producidas en forma accidental o intencional.
- Autenticación: identificación de las partes que conforman la comunicación. Es la técnica mediante la cual se comprueba que cada emisor y receptor es quien se supone que es y no un impostor. Generalmente la autenticación es el paso previo al establecimiento de una conexión entre dos entidades para intercambio de llaves.
- No Repudio: asociación de cada transacción a la identidad que la llevó a cabo. El servicio de Seguridad de No Repudio o irrenunciabilidad está estandarizado en la norma ISO-7498-2. El emisor no puede negar qué envió porque el destinatario tiene pruebas del envío. Por otro lado, el receptor no puede negar que recibió el mensaje porque el emisor tiene pruebas de la recepción.

2.1. Clasificación de los algoritmos de cifrado

Los algoritmos de cifrado se clasifican de acuerdo a la forma en la que operan sobre el texto plano y a la utilización de una o dos llaves durante una comunicación segura. El diagrama de la Figura 2.2 ilustra esta clasificación general y cuenta con un ejemplo para cada tipo. En forma general, los algoritmos de cifrado se dividen en tres categorías: simétricos, asimétricos (o de llave privada y pública, respectivamente) y funciones *hash*. Las tres categorías cumplen distintas funciones dentro de un sistema seguro, razón por la cual generalmente se requiere una combinación de estos métodos para cumplir con los cuatro objetivos de la criptografía mencionados con anterioridad.

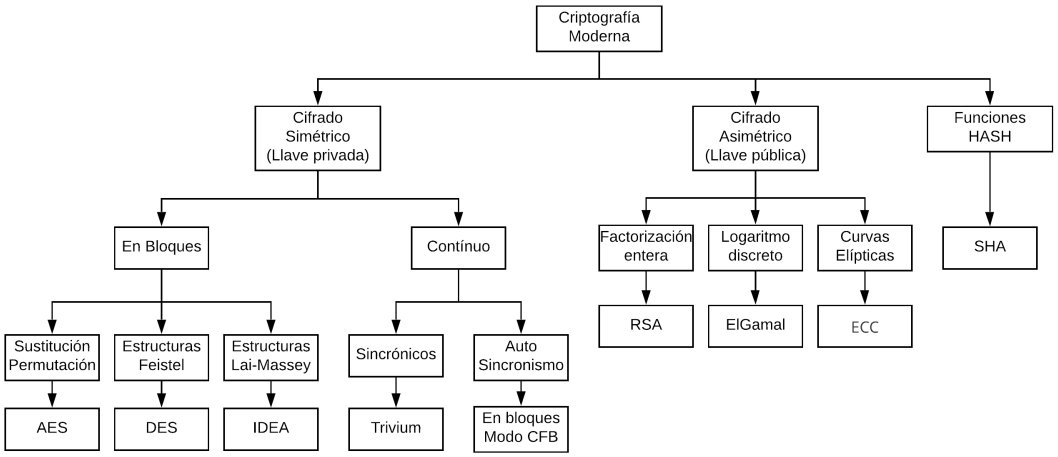


FIGURA 2.2: Clasificación de los algoritmos de cifrado.

2.1.1. Cifrado simétrico o de llave privada

En este tipo de cifrado, las partes que conforman la comunicación segura tienen conocimiento de una llave secreta que es usada tanto para cifrar como para descifrar mensajes. En la Figura 2.3 se ilustra este tipo de cifrado para dos entes A y B que desean compartir información confidencial. Se puede observar que ambos tienen conocimiento de la llave privada. La comunicación de esta llave entre las partes es crítica y se realiza en forma previa al intercambio de información. Este proceso debe realizarse utilizando un medio seguro para sostener la confidencialidad de la comunicación.

Los algoritmos de cifrado simétrico son rápidos, razón por la cual suelen ser utilizados para cifrar/descifrar grandes volúmenes de datos. Además de proveer confidencialidad

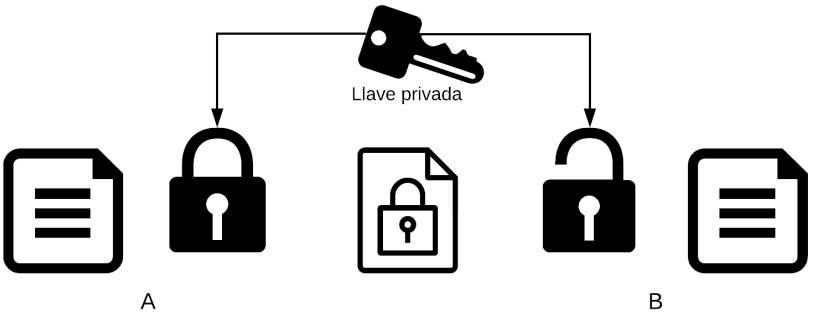


FIGURA 2.3: Cifrado simétrico.

en forma individual, suelen ser utilizados como elementos fundamentales en la construcción de bloques de autenticación de mensajes, control de integridad o funciones *hash* (Menezes, 1996). Dependiendo de la forma en la que procesan el texto plano se pueden dividir en dos clases: en bloques y continuos, ambas ilustradas en la Figura 2.4. En la misma, k representa la llave, x el texto plano e y el texto cifrado.

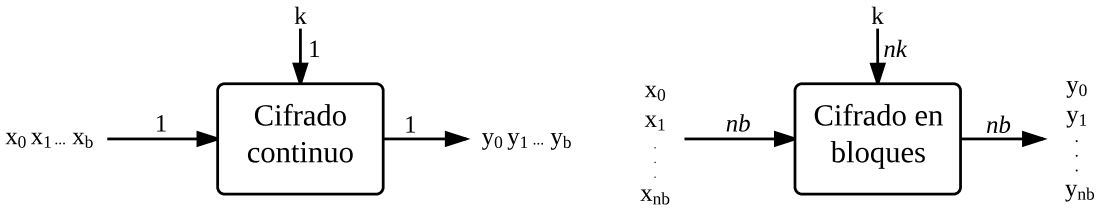


FIGURA 2.4: Cifrado simétrico continuo y en bloques.

Cifrado continuo o de flujo

En los algoritmos de cifrado continuo cada bit del texto plano es cifrado en forma independiente, utilizando una operación de cifrado que varía con el tiempo.

En los algoritmos de cifrado continuo sincrónicos la llave se genera en forma independiente del texto plano o cifrado. En este caso, debe existir un correcto sincronismo entre emisor y receptor (utilización de la misma llave y operación en cada momento) para que la comunicación se efectúe en forma correcta. Si por ejemplo se agrega o pierde uno de los bits del texto cifrado durante su transmisión, se modifica el estado del receptor y, como consecuencia, se pierde el sincronismo.

Por otro lado, en los algoritmos de cifrado continuo de auto-sincronismo la llave es función de su valor anterior y de caracteres previos en el texto cifrado. En este caso, si se insertan o eliminan bits en el texto cifrado la comunicación recupera el sincronismo en forma automática luego de un número fijo de errores en la obtención de bits de texto plano. Dentro de los algoritmos de cifrado continuo más populares se pueden mencionar Rabbit (Boesgaard y col., 2003), Trivium (Canniere y Preneel, 2005), HC-256 (Wu, 2004), entre otros.

Cifrado en bloques

En los algoritmos de cifrado en bloques el texto plano se divide en bloques de n_b bits. Cada uno de estos bloques es cifrado en forma individual y utilizando la misma transformación, obteniendo para cada uno un bloque de texto cifrado de n_b bits. Por lo general, se utiliza una misma llave secreta para cifrar múltiples bloques de texto plano. El valor de n_b suele ser mayor a 64. Si este valor es muy pequeño el algoritmos resulta más vulnerable a ataques basados en análisis estadísticos, ya que en un mensaje dado se repiten con mayor frecuencia los bloques de texto cifrado obtenidos. Por otro lado, un valor de n_b demasiado grande aumenta la complejidad del algoritmo que transforma un bloque de texto plano en un bloque de texto cifrado. Existen diferentes técnicas que se utilizan al momento de subdividir el texto plano en bloques de n_b bits, las mismas de denominan *Modos de operación* y están estandarizados. Su descripción se detalla en el capítulo siguiente.

La mayoría de los algoritmos de cifrado en bloques opera sobre cada bloque de texto plano mediante la aplicación reiterada de una transformación invertible. El valor obtenido en cada iteración suele denominarse *estado*. Cada iteración, denominada *ronda*, tiene dos entradas de n_b bits:

- El resultado obtenido en la iteración anterior.
- Una llave de ronda, que se usa como llave en la transformación. Las llaves de ronda requeridas son generadas a partir de la llave secreta mediante un algoritmo de expansión.

Los algoritmos de cifrado en bloques que poseen esta arquitectura se denominan *iterados*. La cantidad de rondas representa una relación de compromiso entre velocidad

y seguridad. Existen distintos tipos de cifrado en bloques iterado, dependiendo de la arquitectura que presentan las transformaciones de ronda (Figura 2.5):

- **Redes de sustitución-permutación (SPN)**

  En el inicio de cada ronda se combina el dato de entrada con la llave de ronda mediante una operación sencilla, generalmente una *XOR*. A continuación, cada ronda presenta una etapa de sustitución seguida de una etapa de permutación. El objetivo de estas etapas es aportar confusión y difusión, respectivamente.

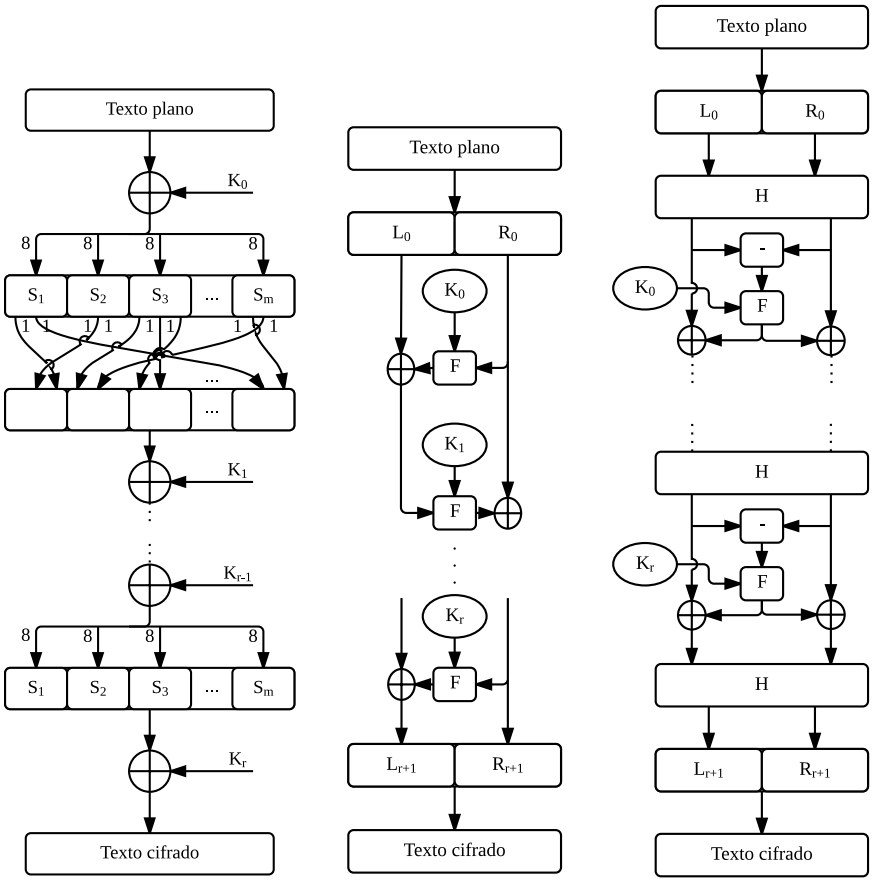
  La capa de sustitución típicamente utiliza *look-up tables* u otro mecanismo para transformar un dato. Esta transformación debe ser biyectiva para asegurar la existencia de su operación inversa (utilizada en el proceso de descifrado). Una sustitución segura cambia, aproximadamente, la mitad de sus bits de salida al modificar sólo un bit a su entrada. Esta última propiedad se basa en el efecto avalancha enunciado por Feistel, 1973. En el mismo se expone que en un sistema seguro, el cambio de un bit en la entrada de la ronda n debe afectar a todos los bits de salida de la ronda $n + x$, siendo el valor de x menor a la cantidad total de rondas.

  La capa de permutación toma la salida de la capa de sustitución y permuta sus bits. La salida de esta etapa es la entrada a la próxima ronda.

- **Estructuras Feistel**

  Cada bloque de texto plano es dividido en dos sub-bloques de igual longitud, denominados L y R. La transformación de ronda (F) se aplica sobre uno ellos (R). Al final de cada ronda, los sub-bloques son intercambiados. Una de las ventajas que presenta este tipo de arquitectura al compararla con las redes de sustitución-permutación, es que la transformación de ronda no necesita ser invertible. De esta manera, durante la implementación en hardware de una arquitectura de este tipo es posible compartir recursos entre un módulo de cifrado y uno de descifrado.

- **Estructuras Lai-Massey** Ofrecen características de seguridad similares a las que brindan las estructuras Feistel. Cada bloque de texto plano se divide en dos sub-bloques de igual longitud. A continuación, la transformación de ronda (F) es aplicada sobre la diferencia entre ambos. El resultado obtenido es combinado con ambas mitades mediante una operación XOR. Opcionalmente, también es posible



(a) Red sustitución-permutación (b) Estructura de Feistel (c) Estructura de Lai-Massey

FIGURA 2.5: Estructuras de cifrado en bloques iterado.

aplicar una transformación (H) a cada uno de los sub-bloques antes de calcular la diferencia entre ellos.

Los algoritmos de cifrado en bloques más conocidos son AES (Daemen y Rijmen, 1998), DES (NIST, 1999), 3DES (NIST, 2012b), Blowfish (Schneier, 1994), Serpent (Anderson, Biham y Knudsen, 1998), IDEA (Lai, 1992), entre otros.

A pesar de ofrecer seguridad y eficiencia, el cifrado simétrico presenta puntos débiles, principalmente en cuanto al intercambio de llaves. Debido a que la llave debe ser conocida sólo por las entidades autorizadas, su transporte y almacenamiento representa un factor crítico, más aún considerando que es común la modificación de esta llave con cierta frecuencia a fin de aumentar la seguridad de la comunicación. Otro de los factores que debe ser considerado es que los algoritmos de cifrado simétrico no ofrecen servicios que cumplan con los cuatros objetivos principales de la criptografía mencionados con

anterioridad. En particular, estos algoritmos no autentican la identidad de las entidades que participan en la comunicación ni comprueban la integridad de los mensajes recibidos.

2.1.2. Cifrado asimétrico o de llave pública

Este tipo de cifrado permite a dos entidades comunicarse en forma segura sin necesidad de compartir una llave secreta entre ellas, tal como se ilustra en la Figura 2.6. Fue desarrollado por Diffie y Hellman (Diffie y Hellman, 1976) y se basa en el uso de operaciones matemáticas simples para las cuales el cálculo de su valor inverso es difucultoso, como la factorización y la exponencial modular. Este tipo de operaciones son generalmente llamadas unidireccionales.

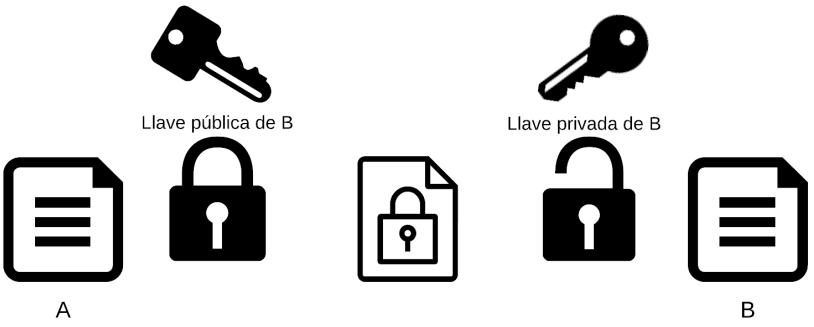


FIGURA 2.6: Cifrado asimétrico.

En un esquema de cifrado asimétrico, cada una de las entidades posee dos llaves, una pública y una privada, que están matemáticamente relacionadas entre sí mediante una operación unidireccional. De esta manera, para cada entidad resulta computacionalmente simple generar su par de llaves, mientras que es prácticamente imposible determinar la llave secreta a partir de la pública. Teniendo en cuenta estas consideraciones, la llave pública puede ser divulgada libremente sin comprometer la seguridad de la comunicación. La relación entre llave pública y privada es tal que si una de ellas es utilizada para cifrar un texto plano, la otra es requerida para descifrar el texto cifrado correspondiente.

Dentro de los algoritmos de cifrado asimétrico más usados se encuentran RSA (Rivest, Shamir y Adleman, 1978), Diffie-Hellman (Diffie y Hellman, 1976), ElGamal (Elgamal, 1985), ECC (Koblitz, 1987; Miller, 1985), entre otros. Debido al uso de llaves públicas y privadas, este tipo de algoritmos provee servicios que el cifrado simétrico no puede

brindar, como no repudio y autenticación. En la práctica, la complejidad computacional es elevada y se refleja en el tiempo que insume el proceso de cifrado o descifrado. Este último puede ser cientos, o incluso miles, de veces mayor que para los algoritmos simétricos. Por este motivo, los algoritmos asimétricos generalmente son usados para realizar la distribución e intercambio de llaves secretas o para proveer autenticación mediante firmas digitales.

2.1.3. Funciones *hash*

Las funciones *hash* son algoritmos que, a diferencia del cifrado simétrico y asimétrico, no utilizan una llave. Básicamente mapean una entrada de longitud arbitraria a una cadena de caracteres de longitud fija.

FIGURA 2.7: Función Hash.

La salida de una función *hash* es una representación compacta de la entrada, razón por la cual estas funciones son utilizadas con frecuencia en medios de autenticación, como firmas digitales y control de integridad de mensajes. Su relación entrada-salida hace que sea prácticamente imposible hallar el valor del mensaje original a partir de su *hash* y modificar el contenido del mensaje original sin que cambie su *hash*. Las funcionas *hash* más usadas en la actualidad se denominan SHA, cuyas variantes más recientes pertenecen a la familia SHA-3.

2.1.4. Importancia de los distintos tipos de cifrado

Como se vio en las secciones anteriores, existen múltiples tipos de algoritmos que se utilizan para cifrar mensajes. Cada uno de ellos está optimizado para distintas operaciones indispensables en un canal de comunicación seguro. Con el fin de visualizar las

ventajas que presenta cada tipo de cifrado y el uso que generalmente se les da, a continuación se muestra un ejemplo básico. En el mismo, un usuario denominado Alice desea comunicarse en forma segura con otro usuario denominado Bob.

1. **Intercambio de llaves mediante cifrado asimétrico**. Alice cifra la llave secreta usando la llave pública de Bob, mediante un algoritmo asimétrico. Bob descifra este mensaje usando su llave privada.

2. **Confidencialidad** Una vez que ambas partes tienen conocimiento de la llave secreta, la comunicación entre ellos se realiza mediante un algoritmo simétrico.

3. **Integridad, autenticación y no repudio** Además del texto cifrado, Alice envía el valor *hash* del mensaje (firma digital) mediante cifrado asimétrico; en este caso el cifrado se realiza utilizando su llave privada. En el otro extremo, Bob descifra el *hash* mediante la llave pública de Alice y lo compara con el valor obtenido al calcular la función *hash* del mensaje recibido mediante el algoritmo simétrico. Si ambos son iguales, Bob verifica que el mensaje no fue alterado durante su transmisión, y que fue enviado por Alice.

El ejemplo mostrado permite visualizar que, para cumplir con los cuatro objetivos principales de la criptografía en forma eficiente, se requiere una combinación de distintos algoritmos criptográficos.

Capítulo 3

Descripción del algoritmo AES

Advanced Encryption Standard es un algoritmo de cifrado simétrico en bloques. El Instituto Nacional de Estándares y Tecnología de Estados Unidos (NIST) publicó este algoritmo como estándar en 2001. Originalmente estaba destinado a la protección de información sensible en instituciones gubernamentales, pero con el paso del tiempo se transformó en un estándar de facto global.

AES consiste en una red de sustitución-permutación con una descripción algebraica relativamente simple que puede ser eficientemente implementada en diversas plataformas. Desde su publicación, se realizaron numerosos estudios para obtener ataques que corrompan su seguridad. En (Bogdanov, Khovratovich y Rechberger, 2011) se muestra un método para recuperar la lleve secreta de AES con un retardo aproximadamente cuatro veces menor al de un ataque de fuerza bruta. De todas maneras, este ataque no deja de ser teórico, ya que con la capacidad de cómputo actual tomaría billones de años completarlo. Los únicos ataques exitosos en contra de AES en la práctica toman ventaja de ciertas debilidades en la forma en la que se implementa el algoritmo, pero no implican una debilidad del algoritmo en sí mismo.

En este capítulo se describe en detalle el algoritmo de cifrado AES, poniendo énfasis en las transformaciones que componen cada ronda. A continuación, se describe el proceso de expansión mediante el cual se obtienen las llaves de ronda a partir de la llave secreta. Finalmente, se describen algunos de los modos de operación más comúnmente utilizados.

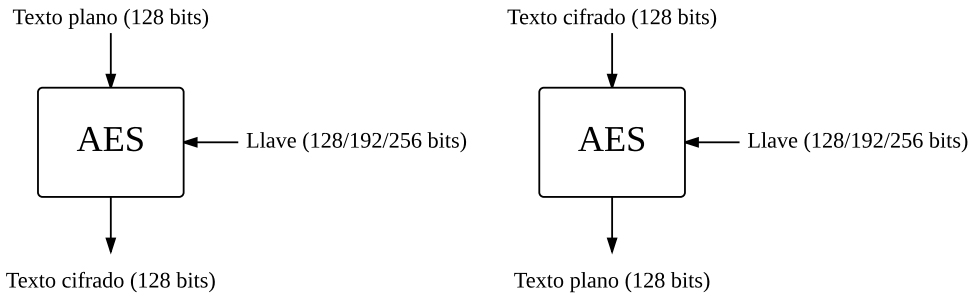
3.1. Origen y características generales

En enero de 1997, el Instituto Nacional de Estándares y Tecnología de Estados Unidos (NIST) anunció el comienzo de una iniciativa para el desarrollo de un estándar de cifrado en bloques, denominado *Advanced Encryption Standard (AES)*, con la finalidad de reemplazar al *Data Encryption Standard (DES)*, ampliamente usado hasta ese momento. El proceso de selección fue abierto y la evaluación de seguridad y eficiencia fue realizada en conjunto con investigadores y aficionados a la criptografía, quienes llevaron a cabo diferentes pruebas sobre los algoritmos propuestos. En 1999 se anunciaron los cinco finalistas: MARS (Burwick, 1998), RC6 (Rivest y col., 1998), Rijndael (Daemen y Rijmen, 1998), Serpent (Anderson, Biham y Knudsen, 1998) y Twofish (Schneier y col., 1998). Finalmente, en octubre del año 2000 el algoritmo Rijndael fue anunciado como el ganador y en 2001 se lo aprobó como estándar federal en Estados Unidos. La única diferencia entre la especificación original de Rijndael y el estándar publicado por NIST (NIST, 2001a) es el rango de valores aceptados para la longitud del bloque de datos de entrada y la llave. Mientras que Rijndael acepta cualquier longitud múltiplo de 32 bits (entre 128 y 256) para estos parámetros, en AES se fija la longitud del bloque de datos en 128 bits y la llave admite tres valores posibles: 128, 192 o 256 bits.

En el año 2003 la Agencia de Seguridad Nacional de Estados Unidos (NSA) anunció la aprobación el uso de AES para cifrar documentos clasificados, utilizando cualquier llave para el nivel *SECRET* y llaves de 192 o 256 bits para el nivel *TOP SCRET*. Esta fue la primera vez que se admitió el uso de un algoritmo público para proteger este tipo de información (Paar y Pelzl, 2010).

AES es un algoritmo de cifrado que posee las siguientes características:

- Es *simétrico* porque utiliza la misma llave para cifrar y descifrar un mensaje.
- Es *en bloques* porque opera sobre bloques de una longitud fija de bits n_b. NIST fijó como uno de los requisitos que n_b sea de 128 bits (Figura 3.1).
- Es *iterado* porque el cifrado consiste en la aplicación reiterada de una transformación que está compuesta de una *Red de Sustitución-Permutación*.
- Admite tres diferentes tamaños de llave: 128, 192 ó 256 bits. Esta condición también fue impuesta por NIST durante la convocatoria.



(a) Cifrado (b) Descifrado

FIGURA 3.1: Entradas y salidas de AES.

Al ser un algoritmo iterado, tanto la operación de cifrado como la de descifrado consisten en la aplicación reiterada de una transformación que está compuesta de una Red de Sustitución-Permutación. Cada una de las transformaciones aplicadas se denomina *ronda* y los resultados parciales obtenidos en cada una de ellas se denomina *estado*. En el caso particular de AES, la cantidad de rondas N_r depende del tamaño de la llave, tal como se muestra en la Tabla 3.1.

TABLA 3.1: Cantidad de rondas según longitud de la llave

Longitud de llave	Cantidad de rondas (N_r)
128 bits	10
192 bis	12
256 bits	14

La primer ronda tiene como entrada el bloque de texto plano de 128 bits, mientras que las rondas siguientes tienen como entrada el resultado de la anterior. Además, cada una de las rondas también requiere como parámetro de entrada una *llave de ronda* de 128 bits. Todas las llaves de ronda requeridas durante el proceso de cifrado o descifrado son generadas a partir de la llave privada que define la comunicación simétrica y que solo conocen los entes autorizados. El estándar define tres algoritmos de expansión distintos, uno para cada tamaño de llave admisible.

Los estados intermedios son bloques de datos de 128 bits, y suelen representarse como matriz de 4×4 bytes para facilitar la visualización de las distintas operaciones que se llevan a cabo sobre ellos durante el cifrado y descifrado. La Ecuación 3.1 muestra la manera en la que se forma la matriz de estado S. En la misma se observa que B_F, el byte más significativo de un dato de 128 bits, ocupa la posición $(1, 1)$ de la matriz S. Los bytes siguientes se van ubicando ocupando las filas subsiguientes hasta completar la primer

columna. Una vez completa se comienza a completar la segunda, y así sucesivamente. El byte menos significativo B_0 ocupa la posición $(4, 4)$.

$$B_F B_E B_D B_C B_B B_A B_9 B_8 B_7 B_6 B_5 B_4 B_3 B_2 B_1 B_0 \quad \Updownarrow \quad \begin{bmatrix} S_{0,0} & S_{0,1} & S_{0,2} & S_{0,3} \\ S_{1,0} & S_{1,1} & S_{1,2} & S_{1,3} \\ S_{2,0} & S_{2,1} & S_{2,2} & S_{2,3} \\ S_{3,0} & S_{3,1} & S_{3,2} & S_{3,3} \end{bmatrix} = \begin{bmatrix} B_F & B_B & B_7 & B_3 \\ B_E & B_A & B_6 & B_2 \\ B_D & B_9 & B_5 & B_1 \\ B_C & B_8 & B_4 & B_0 \end{bmatrix}. \tag{3.1}$$

3.2. Cifrado

El cifrado de un mensaje consta de una iteración de una transformación denominada *ronda inicial*, seguida de $N_r - 1$ iteraciones de una *ronda general* y por último una iteración de una *ronda final* (Figura 3.2). Cada una de estas rondas tienen dos entradas: el *estado* y una *llave de ronda*, ambas de 128 bits.

La ronda inicial consta simplemente de una operación XOR entre el bloque de texto plano y la primer llave de ronda. Como se verá en detalle en la descripción de los algoritmos de expansión, la llave de ronda inicial coincide con los 128 primeros bits de la llave privada. Por otro lado, la ronda general está compuestas por una red de Sustitución-Permutación cuyas etapas se denominan de la siguiente manera:

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

La ronda final es similar a las rondas generales, salvo que no cuenta con la transformación MixColumns. A continuación, se detalla la operación de cada una de las etapas que conforman una ronda general.

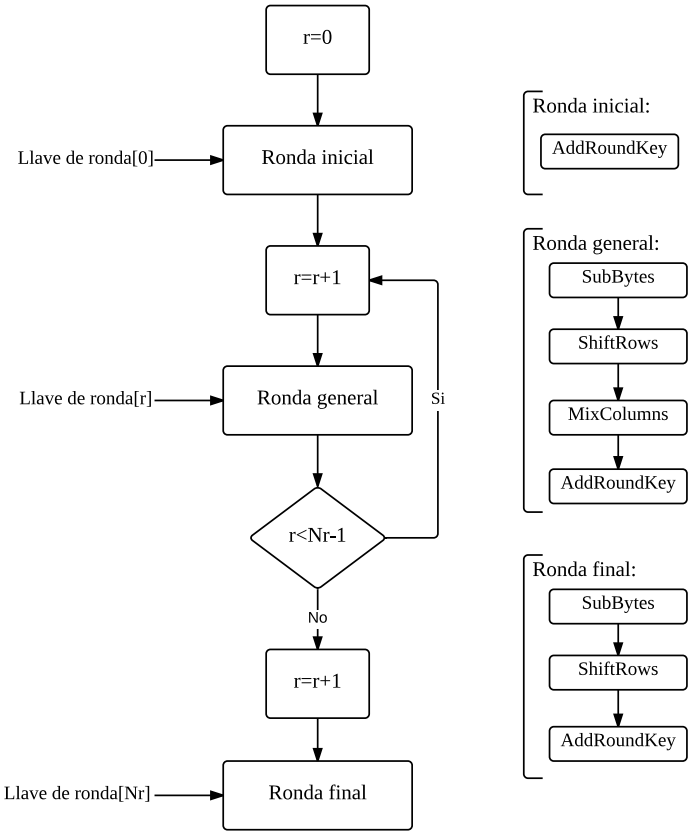


FIGURA 3.2: Diagrama de actividades que representa el cifrado en AES.

SubBytes

Esta transformación corresponde a la capa de sustitución en la red Sustitución-Permutación, y es la única operación no lineal en el estándar AES. Se basa en la aplicación de una misma transformación (denominada S-BOX) a cada uno de los bytes que conforman la matriz de estado. Su objetivo es mapear cada uno de los 256 valores posibles en forma biyectiva mediante una expresión algebraica que es compleja (aporta confusión) y minimiza la amplitud de la correlación entre entrada y salida.

Matemáticamente, la S-BOX utilizada puede ser vista como una inversión en el campo finito $GF\{2^8\}$ [1], usando el polinomio irreducible $x^8+x^4+x^3+x+1$, seguida de un mapeo afín. Este último no tiene impacto en las propiedades de no linealidad pero incrementa la complejidad de la expresión y evita la presencia de puntos fijos y opuestos en la transformación, es decir, ningún valor es mapeado a sí mismo o a su valor negado.

[1] En el Apéndice A se presenta una introducción al álgebra de campos finitos, base de la mayoría de las transformaciones de ronda en AES.

Existen dos maneras de implementar una S-BOX: mediante la implementación de las operaciones aritméticas en campos finitos que la componen, o mediante el uso de *lookup tables*. En la Figura 3.3 y Tabla 3.2 se muestran ambas implementaciones para la S-BOX utilizada en AES. En la *lookup table*, la columna se determina a partir del *nibble* menos significativo y la fila a partir del *nibble* más significativo de un byte. El dato en dicha ubicación de la tabla es el resultado obtenido al transformar el byte utilizado para indexar la tabla.

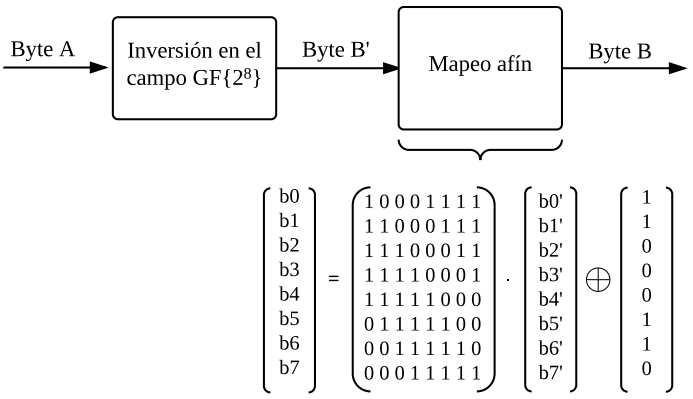


FIGURA 3.3: Implementación de la S-BOX mediante álgebra de campos finitos.

TABLA 3.2: Implementación de la S-BOX mediante una *lookup table*.

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

ShiftRows

Esta operación consiste en el desplazamiento de las filas de la matriz de estado. Cada una de ellas es desplazada una cantidad distinta de bytes, aumentando la resistencia del sistema ante ataques relacionados al criptoanálisis lineal y diferencial.

Como se observa en la Ecuación 3.2, la primer fila no es desplazada, la segunda es desplazada en forma circular un byte a la izquierda, la tercera es desplazada en forma circular dos bytes a la izquierda, y la última tres bytes a la izquierda. Esta operación mejora las características de difusión del cifrado.

$$\begin{bmatrix} S'_{0,0} & S'_{0,1} & S'_{0,2} & S'_{0,3} \\ S'_{1,0} & S'_{1,1} & S'_{1,2} & S'_{1,3} \\ S'_{2,0} & S'_{2,1} & S'_{2,2} & S'_{2,3} \\ S'_{3,0} & S'_{3,1} & S'_{3,2} & S'_{3,3} \end{bmatrix} = \begin{bmatrix} S_{0,0} & S_{0,1} & S_{0,2} & S_{0,3} \\ S_{1,1} & S_{1,2} & S_{1,3} & S_{1,0} \\ S_{2,2} & S_{2,3} & S_{2,0} & S_{2,1} \\ S_{3,3} & S_{3,0} & S_{3,1} & S_{3,2} \end{bmatrix} \tag{3.2}$$

MixColumns

Consiste en una operación lineal que se aplica sobre cada una de las columnas que conforman la matriz de estado. Esta transformación realiza el mayor aporte de difusión al sistema criptográfico. Matemáticamente corresponde a la multiplicación entre polinomios sobre el campo finito $GF\{2^8\}$. Cada columna se considera como un polinomio cuyos coeficientes pertenecen al campo $GF\{2^8\}$ y se multiplica, módulo el polinomio irreducible x^4+1, con un polinomio constante definido en el estándar como $03x^3+01x^2+01x+02$. La elección del mismo se realizó considerando que la transformación debe ser invertible. En la Ecuación 3.3 se observa esta operación en forma matricial para una de las columnas de la matriz de estado.

$$\begin{bmatrix} S'_{0,j} \\ S'_{1,j} \\ S'_{2,j} \\ S'_{3,j} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \times \begin{bmatrix} S_{0,j} \\ S_{1,j} \\ S_{2,j} \\ S_{3,j} \end{bmatrix} \tag{3.3}$$

Si se ve a cada ronda como una red de sustitución-permutación, la capa de permutación está dada por las transformaciones ShiftRows y MixColumns. Las mismas están definidas en una manera tal que la variación de uno de los bytes de la matriz de estado se propaga a todos los bytes del estado obtenido dos rondas después, tal como se ilustra en la Figura 3.4.

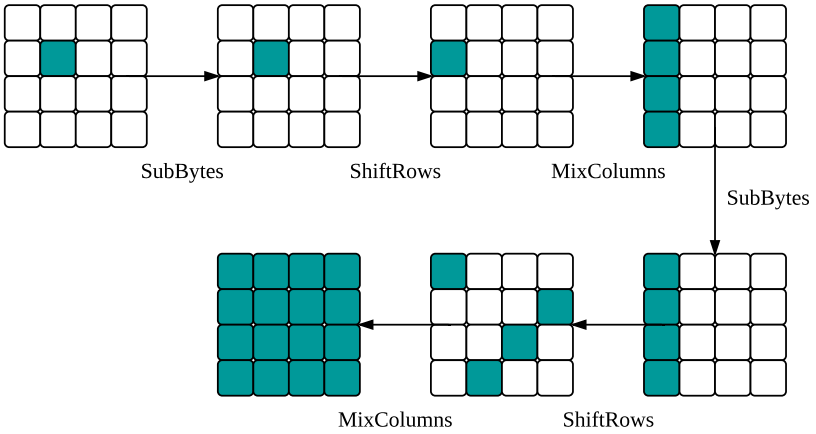


FIGURA 3.4: Efecto de la variación de un byte de la matriz de estado durante el transcurso de dos rondas generales.

AddRoundKey

Durante esta operación se realiza la XOR, bit a bit, entre la matriz de estado de datos y la matriz de estado de la llave de ronda correspondiente.

3.3. Descifrado

Durante el proceso de descifrado (Figura 3.5) se aplican transformaciones de ronda inversas a las utilizadas durante el cifrado y en un orden diferente.

En primer lugar, la ronda inicial en este caso consiste en una operación XOR, bit a bit, entre la matriz de estado de datos (texto cifrado) y la matriz de estado correspondiente a la última llave de ronda generada por el algoritmo de expansión de llave.

La ronda general está compuesta por la aplicación de transformaciones inversas, en el siguiente orden:

- InvShiftRows.
- InvSubBytes
- AddRoundKey.
- InvMixColumns.

En forma análoga al cifrado, la ronda final es similar a las rondas generales, salvo que no cuenta con la transformación InvMixColumns.

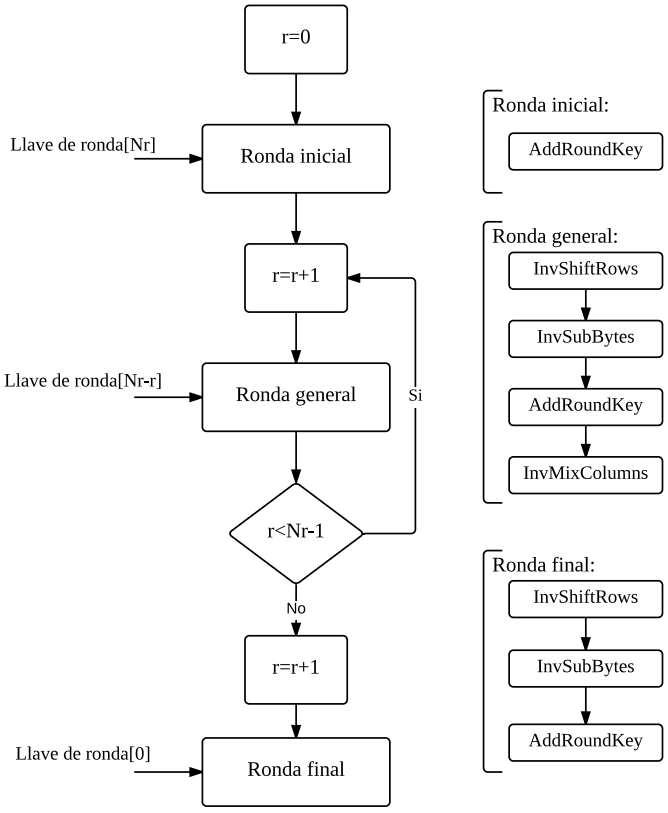


FIGURA 3.5: Diagrama de actividades que representa el descifrado en AES.

La especificación de cada una de las transformaciones es tal que invierte la conversión realizada durante el cifrado, como se detalla a continuación.

InvShiftRows

Esta operación consiste en el desplazamiento de las filas de la matriz de estado. Como se observa en la Ecuación 3.4, la primer fila no es desplazada, la segunda es desplazada en forma circular un byte a la derecha, la tercera es desplazada en forma circular dos bytes a la derecha, y la última tres bytes a la derecha.

$$\begin{bmatrix} S'_{0,0} & S'_{0,1} & S'_{0,2} & S'_{0,3} \\ S'_{1,0} & S'_{1,1} & S'_{1,2} & S'_{1,3} \\ S'_{2,0} & S'_{2,1} & S'_{2,2} & S'_{2,3} \\ S'_{3,0} & S'_{3,1} & S'_{3,2} & S'_{3,3} \end{bmatrix} = \begin{bmatrix} S_{0,0} & S_{0,1} & S_{0,2} & S_{0,3} \\ S_{1,3} & S_{1,0} & S_{1,1} & S_{1,2} \\ S_{2,2} & S_{2,3} & S_{2,0} & S_{2,1} \\ S_{3,1} & S_{3,2} & S_{3,3} & S_{3,0} \end{bmatrix} \tag{3.4}$$

InvSubBytes

Esta transformación consiste en la aplicación de la S-BOX inversa a cada uno de los bytes que conforman la matriz de estado.

Matemáticamente, la S-BOX inversa puede ser vista como la aplicación de la inversa del mapeo afín utilizado durante el cifrado, seguida del cálculo de la inversa multiplicativa en el campo $GF\{2^8\}$, usando el polinomio irreducible $x^8 + x^4 + x^3 + x + 1$.

Existen dos formas de implementar una S-BOX inversa: mediante la implementación de las operaciones aritméticas en campos finitos que la componen, o mediante el uso de *lookup tables*. En la Figura 3.6 y Tabla 3.3 se muestran ambas implementaciones para la S-BOX inversa utilizada en AES. En la *lookup table*, la columna se determina a partir del *nibble* menos significativo, y la fila a partir del *nibble* más significativo de un byte. El dato en dicha ubicación de la tabla es el resultado obtenido al transformar el byte utilizado para indexar la tabla.

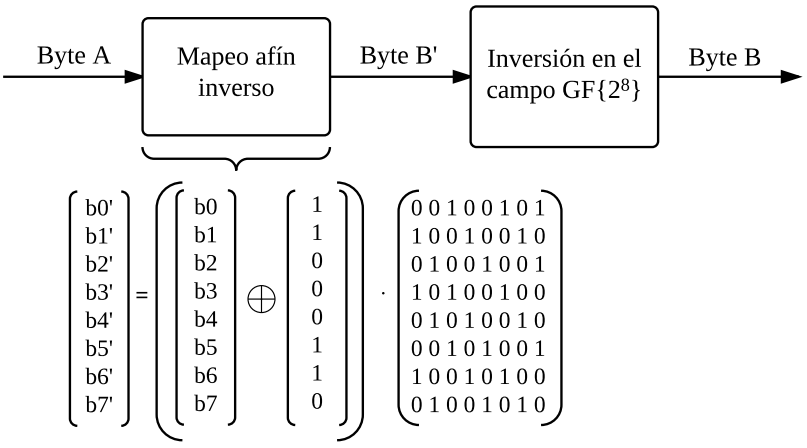


FIGURA 3.6: Implementación de la S-BOX inversa mediante álgebra de campos finitos.

InvMixColumns

Esta transformación es la inversa de MixColumns. Matemáticamente corresponde a la multiplicación entre polinomios sobre el campo finito $GF\{2^8\}$. Cada columna se considera como un polinomio cuyos coeficientes pertenecen al campo $GF\{2^8\}$ y se multiplica, módulo el polinomio irreducible $x^4 + 1$, con el polinomio $0bx^3 + 0dx^2 + 09x + 0e$ (inversa

TABLA 3.3: Implementación de la S-BOX inversa mediante una *lookup table*.

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

del polinomio utilizado en el cifrado). En la Ecuación 3.5 se observa esta operación en forma matricial para una de las columnas de la matriz de estado.

$$\begin{bmatrix} S'_{0,j} \\ S'_{1,j} \\ S'_{2,j} \\ S'_{3,j} \end{bmatrix} = \begin{bmatrix} 0e\ 0b\ 0d\ 09 \\ 09\ 0e\ 0b\ 0d \\ 0d\ 09\ 0e\ 0b \\ 0b\ 0d\ 09\ 0e \end{bmatrix} \times \begin{bmatrix} S_{0,j} \\ S_{1,j} \\ S_{2,j} \\ S_{3,j} \end{bmatrix} \tag{3.5}$$

AddRoundKey

Debido a que en el campo finito $GF\{2^8\}$ la suma equivale a la resta, la inversa de AddRoundKey es también la XOR bit a bit entre la matriz de estado y la llave de ronda.

3.4. Expansión de llave

Durante cada una de las rondas de cifrado o descifrado se utiliza como argumento una llave, denominada llave de ronda, que es derivada a partir de la llave secreta a través de un algoritmo de expansión. Este último depende de la longitud de la llave secreta y es recursivo, es decir, cada llave de ronda depende del valor de la llave de ronda anterior. Esta dependencia entre rondas es no lineal debido a la presencia de la transformación SubBytes en el algoritmo de expansión. Además, en el mismo se incorpora una constante

cuyo valor es distinto para cada una de las rondas, garantizando que el valor de cada llave de ronda sea diferente al anterior. A continuación se describe el procedimiento de expansión, considerando cada una de las tres longitudes de llave secreta admitidas en AES.

Expansión para una llave de 128 bits

Para llaves de 128 bits, la cantidad de rondas necesarias en AES es 10. Por lo tanto, se requiere la obtención de 11 llaves de ronda a partir de la llave secreta (una para la ronda inicial y luego una para cada una de las 10 rondas siguientes). Estas 11 llaves de ronda pueden ser almacenadas en un arreglo de 44 palabras de 32 bits cada una.

El proceso de expansión se observa en la Figura 3.7. En primer lugar, se almacena la llave secreta en las primeras cuatro palabras del arreglo de salida. Las palabras subsiguientes se computan de la siguiente manera:

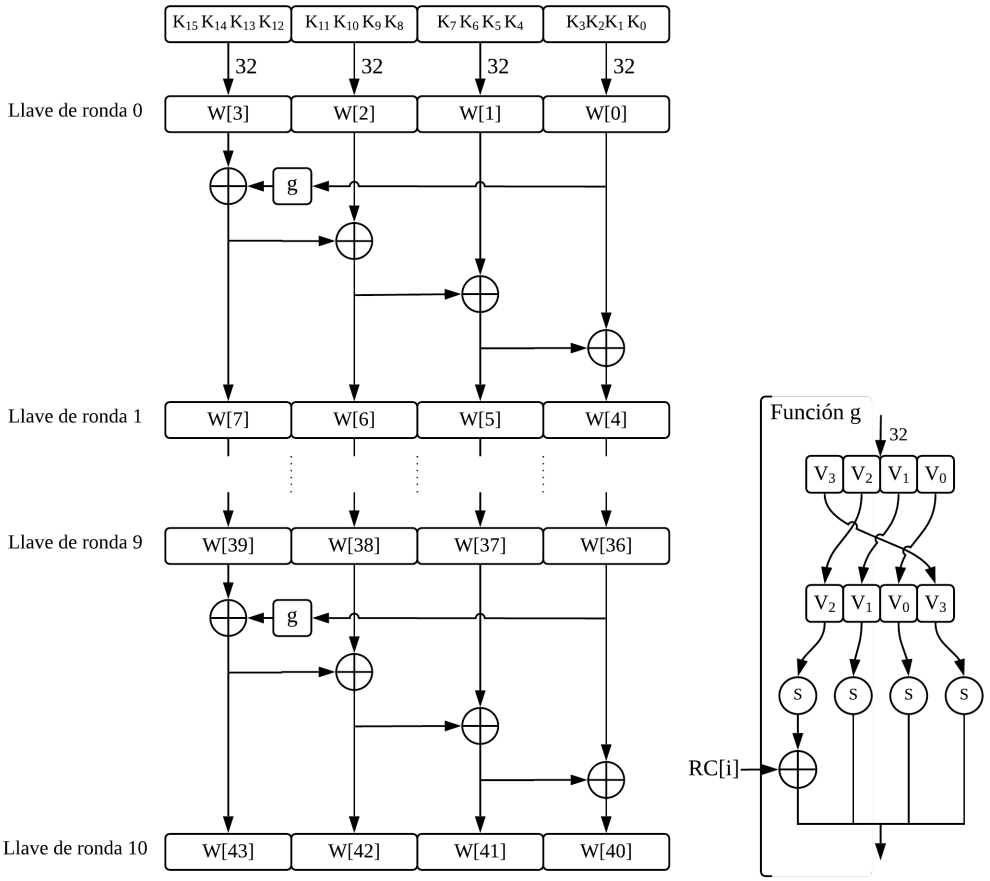


FIGURA 3.7: Algoritmo de expansión para una llave de 128 bits.

- La palabra más significativa de cada llave de ronda se determina mediante la siguiente ecuación:

$$W[i] = W[i-4] \oplus g(W[i-7]). \tag{3.6}$$

- Para el resto de las palabras, rige la siguiente expresión:

$$W[i] = W[i-4] \oplus W[i+1]. \tag{3.7}$$

La función $g(\cdot)$ es no lineal. Consiste en un desplazamiento de los cuatro bytes que componen la palabra, seguido de una sustitución usando la misma S-BOX que en el proceso cifrado. Por último, se realiza la función XOR entre el byte más significativo del valor obtenido y una constante de ocho bits, denominada RC, cuyo valor es distinto en cada iteración de la expansión:

$$\begin{aligned}
RC[1] &= x^0 = (00000010)^0 = 00000001 \\
RC[2] &= x^1 = (00000010)^1 = 00000010 \\
RC[3] &= x^2 = (00000010)^1 = 00000100 \\
RC[4] &= x^3 = (00000010)^1 = 00001000 \\
RC[5] &= x^4 = (00000010)^1 = 00010000 \\
RC[6] &= x^5 = (00000010)^1 = 00100000 \\
RC[7] &= x^6 = (00000010)^1 = 01000000 \\
RC[8] &= x^7 = (00000010)^1 = 10000000 \\
RC[9] &= x^8 = (00000010)^1 = 00011011 \\
RC[10] &= x^9 = (00000010)^9 = 00110110.
\end{aligned}$$

Expansión para una llave de 192 bits

Para llaves de 192 bits, la cantidad de rondas necesarias en AES es 12. Por lo tanto, se requiere la obtención de 13 llaves de ronda a partir de la llave secreta. Dado que el bloque de datos es siempre de 128 bits, cada una de las llaves de ronda calculadas debe ser también de 128 bits. Como consecuencia, el resultado de la expansión es un arreglo de 52 palabras de 32 bits cada una. Cada iteración de la expansión computa seis palabras de este arreglo, por lo que en total se requieren nueve iteraciones para completarlo.

Como se muestra en la Figura 3.8, la manera en la que se computa cada valor del arreglo es análoga a la presentada para el caso de llaves de 128 bits.

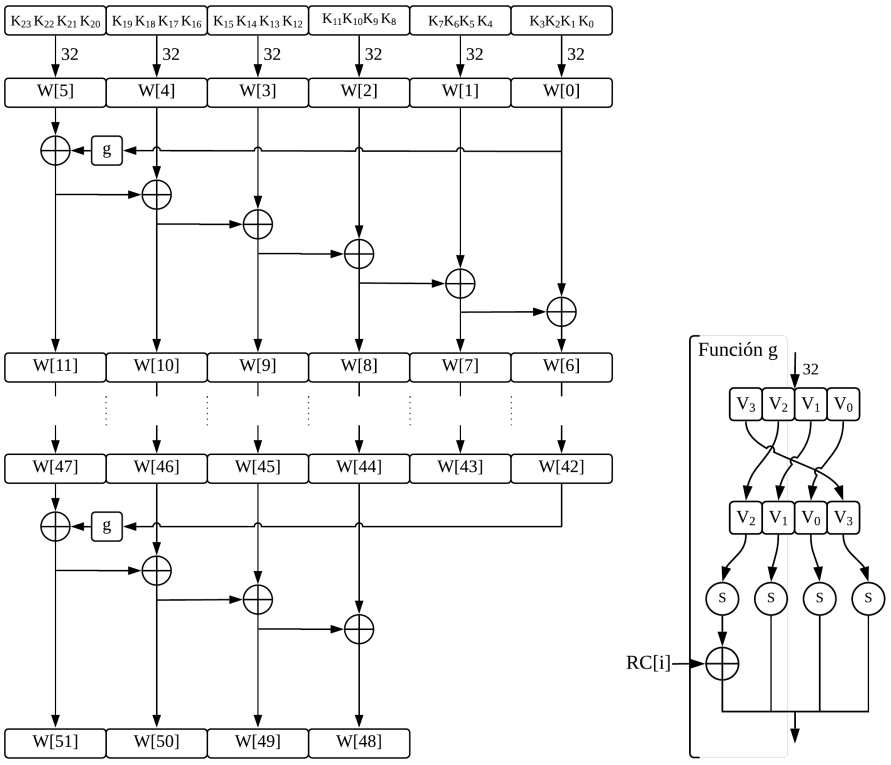


FIGURA 3.8: Algoritmo de expansión para una llave de 192 bits.

Expansión para una llave de 256 bits

Para llaves de 256 bits, la cantidad de rondas necesarias en AES es 14. Por lo tanto, se requiere la obtención de 15 llaves de ronda a partir de la llave secreta. Debido a que el bloque de datos es siempre de 128 bits, cada una de las llaves de ronda calculadas debe ser también de 128 bits. Como consecuencia, el resultado de la expansión es un arreglo de 60 palabras de 32 bits cada una. Cada iteración de la expansión computa ocho palabras de este arreglo, por lo que en total se requieren nueve iteraciones para completarlo. Como se muestra en la Figura 3.9, en este caso se incorpora otra función no lineal denominada $h(\cdot)$ que aplica a cada byte la S-BOX usada en $g(\cdot)$.

Independientemente del algoritmo de expansión usado, existen dos formas de manejar las llaves de ronda:

- **Precómputo** La totalidad de las llaves de ronda son generadas en forma previa al proceso de cifrado o descifrado y quedan a disposición en un arreglo de memoria

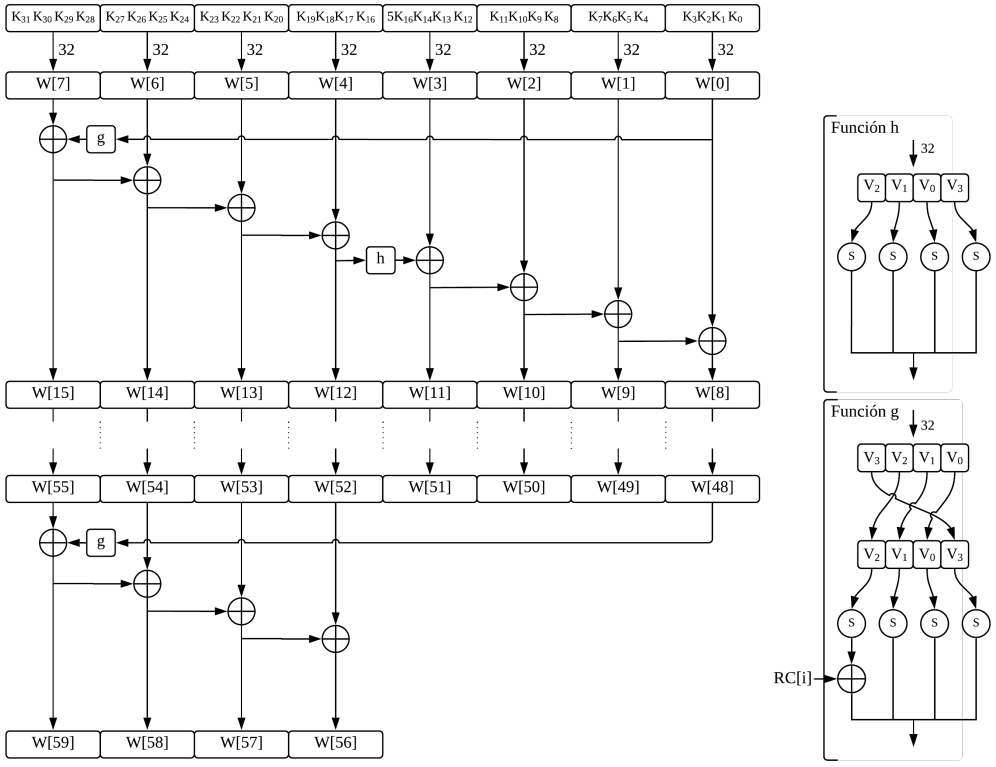


FIGURA 3.9: Algoritmo de expansión para una llave de 256 bits.

para ser utilizadas cuando se requieren. Este enfoque es generalmente usado en casos en los que la llave secreta no se modifica con frecuencia, ya que de esta forma se realizan múltiples operaciones de cifrado o descifrado por cada cálculo de llaves de ronda. El precómputo de las llaves de ronda mejora la velocidad del sistema a expensas de un mayor requerimiento de memoria. En casos en los que la llave secreta cambia con frecuencia este enfoque no es conveniente.

- **On-the-fly** Las llaves de ronda son generadas en forma paralela al proceso de cifrado a medida que se las requiere. Debido a que para el proceso de descifrado la secuencia de las rondas es inversa, se requiere en un primer momento el valor de la llave de ronda usada durante la ronda final del cifrado. Este valor, debido a la recursividad de la expansión, depende de todas las llaves de ronda utilizadas durante el cifrado. Por lo tanto, al implementar este criterio de expansión, se obtienen menores requerimientos de memoria a expensas de un retardo mayor para el descifrado que para el cifrado.

3.5. Modos de operación

Los modos de operación definen el enfoque que se utiliza al enviar al módulo de cifrado en bloques un texto plano cuya longitud es mayor a la longitud del bloque n_b. Los diferentes modos de operación están definidos en recomendaciones emitidas por NIST (NIST, 2001b), (NIST, 2007). La utilización de estos modos permite la construcción de distintos mecanismos criptográficos a partir de un módulo de cifrado en bloques, como funciones *hash*, autenticación, generadores de números aleatorios, así como también cifrado continuo. Las pruebas de conformidad con la recomendación son llevadas a cabo dentro del Programa de Validación de Modos Criptográficos (CMVP). Los cinco modos de operación básicos son:

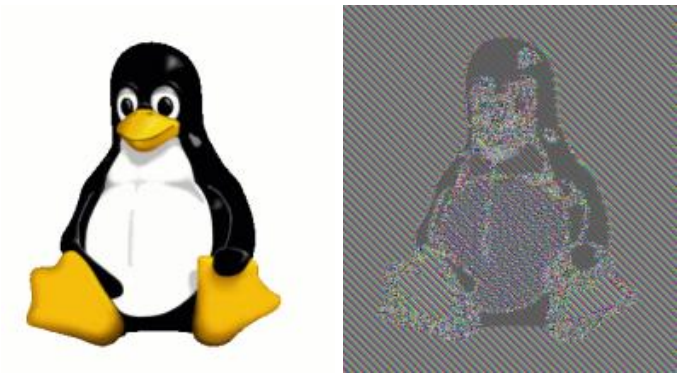
- Electronic Codebook (ECB).
- Cipher Block Chaining (CBC)
- Cipher Feedback (CFB)
- Output Feedback(OFB)
- Counter (CTR)

Electronic Codebook Mode (ECB)

En este modo de operación el texto plano se divide en segmentos cuya longitud es igual a la longitud del bloque, n_b. Cada uno de estos segmentos es cifrado o descifrado en forma independiente. La ventaja principal que presenta este modo es que las transformaciones de ronda, tanto para el cifrado como para el descifrado, pueden ser paralelizadas. Además, en caso de producirse errores en la transmisión solo se ven afectados los bloques damnificados, sin afectar a los demás.

La debilidad más importante de este modo es que el texto cifrado preserva las características estadísticas del texto plano, debido a que dos bloques de texto plano iguales resultan en dos textos cifrados iguales. Este efecto es visualizado en la Figura 3.10, que muestra el cifrado de cada uno de los píxeles de una imagen en modo ECB.

De esta manera, un posible atacante puede reconocer si un mismo mensaje fue enviado más de una vez, observando sólo el texto cifrado. Además, este modo no presenta técnicas



(a) Imagen original (b) Resultado del cifrado

FIGURA 3.10: Cifrado de una imagen en modo ECB.

de preservación de integridad. Las debilidades mencionadas no afectan al algoritmo criptográfico en sí mismo, por lo que es importante destacar que no pueden ser revertidas aumentando la cantidad de bits de llave.

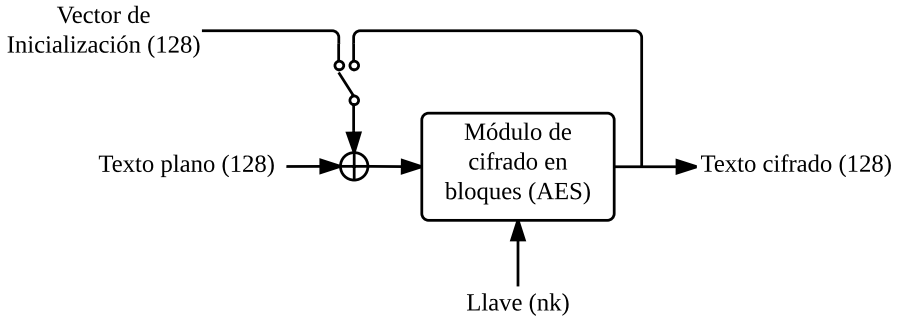
Cipher Block Chaining Mode (CBC)

En este modo, la entrada al módulo de cifrado en bloques es el resultado de la operación XOR entre un bloque del texto plano y el texto cifrado obtenido al cifrar bloque anterior. Al realizar el cifrado o descifrado del primer bloque de datos se utiliza un vector de inicialización (VI), como puede observarse en la Figura 3.11.

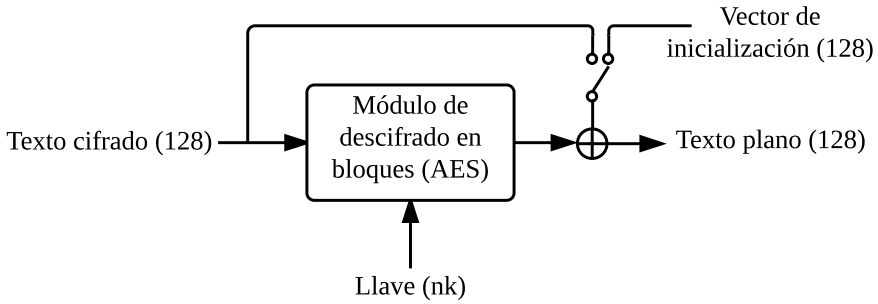
Debido a que el procesamiento de cada etapa depende del resultado anterior, en este modo de operación no es posible paralelizar las transformaciones de ronda en el proceso de cifrado. De todos modos, las transformaciones inversas que componen el descifrado son paralelizables, siempre que los textos cifrados correspondientes a cada bloque se encuentran disponibles en forma simultánea.

En la mayoría de los casos es deseable que los vectores de inicialización sean usados una sola vez, con el fin de obtener resultados diferentes al cifrar más de una vez un mismo texto plano. De esta manera es posible evitar el reconocimiento de patrones en el texto cifrado. Este modo no presenta técnicas de preservación de integridad.

El modo CBC puede ser utilizado como herramienta para generar un código de autenticación de mensajes (MAC por su sigla en inglés *Message Authentication Code*) a partir de un módulo de cifrado en bloques. Para calcular el MAC de un mensaje m utilizando



(a) Cifrado



(b) Descifrado

FIGURA 3.11: Esquema de cifrado y descifrado en modo CBC.

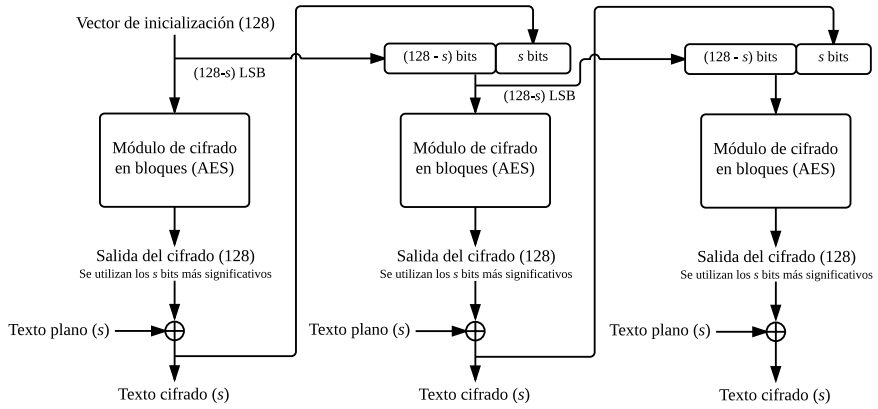
esta técnica, se cifra el mensaje en modo CBC, utilizando el valor cero como vector de inicialización, y se mantiene el último bloque. Debido a que el cifrado del último bloque depende de todos los anteriores, un receptor puede verificar de esta manera si el mensaje fue alterado durante su transmisión.

Cipher Feedback Mode (CFB)

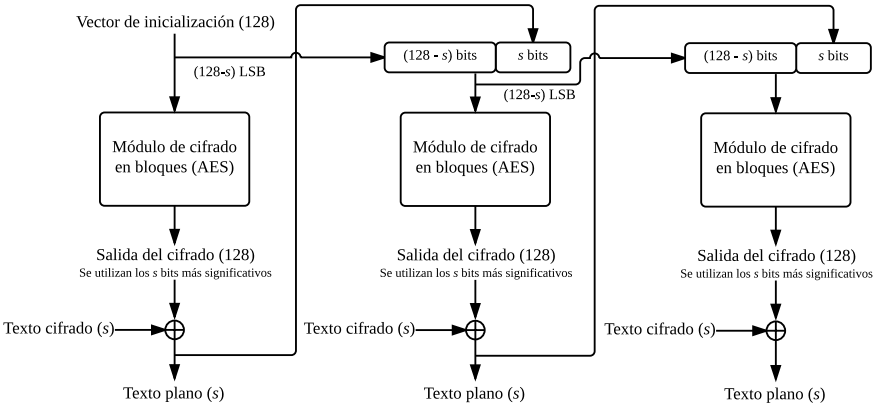
En este modo se requiere a modo de parámetro un valor de tipo entero, denominado s, que define la longitud de cada segmento del texto plano y cifrado. El valor de este parámetro puede ser mayor o igual a 1 y menor o igual a n_b.

La primer entrada al módulo criptográfico es el vector de inicialización. Los s bits más significativos del resultado son operados mediante una XOR con el primer segmento de texto plano. El resultado obtenido es concatenado con los $b - s$ bits menos significativos del vector de inicialización, formando la segunda entrada al bloque. Este esquema se muestra en la Figura 3.12.

Debido a que en este modo el cifrado del texto plano se realiza mediante una operación XOR, el mismo se trata un cifrado continuo de auto sincronismo en el que el valor de la



(a) Cifrado



(b) Descifrado

FIGURA 3.12: Esquema de cifrado y descifrado en modo CFB.

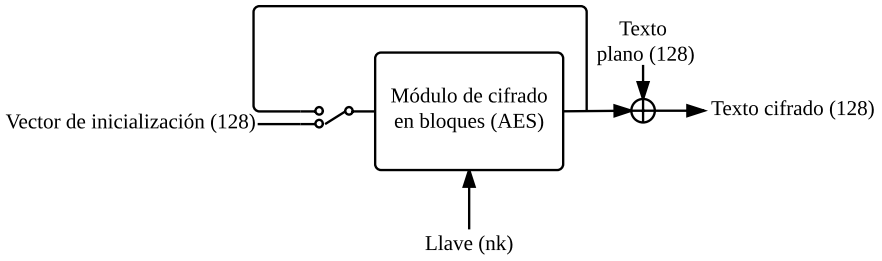
llave depende del último texto cifrado. El proceso de descifrado es exactamente igual al de cifrado debido a que la operación XOR y su inversa son iguales.

En este modo las transformaciones de ronda del cifrado no son paralelizabes, ya que el cifrado de un bloque depende del resultado obtenido al cifrar el bloque anterior. De todos modos, las transformaciones de ronda inversas presentes en el descifrado son paralelizables siempre y cuando los textos cifrados correspondientes a cada bloque se encuentran disponibles en forma simultánea.

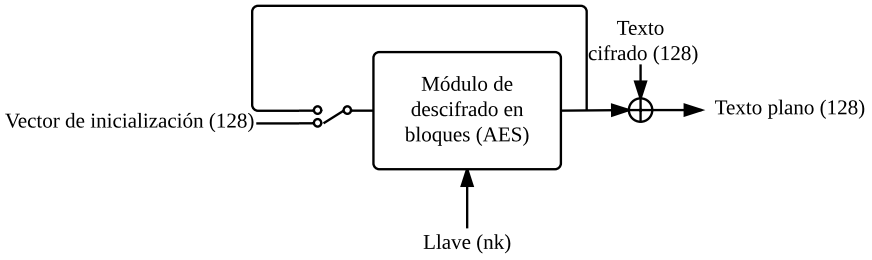
En la mayoría de los casos es deseable que los vectores de inicialización sean usados una sola vez, de manera de obtener resultados diferentes al cifrar un mismo texto plano en múltiples ocasiones. De esta manera es posible evitar el reconocimiento de patrones en el texto cifrado. Este método tampoco cuenta con técnicas de preservación de integridad.

Output Feedback (OFB)

En este modo, se realiza el cifrado en bloque de un valor de entrada. Luego, se realiza la XOR entre el resultado obtenido y el segmento de texto plano. En el primer paso, la entrada al proceso de cifrado es el vector de inicialización, mientras que en los pasos subsiguientes la entrada corresponde al valor de salida del cifrado en bloque en el paso anterior. Para el último segmento de texto plano, cuya longitud no necesariamente igual a la longitud del bloque, se usan los bits más significativos de la salida del cifrado en bloque para realizar la XOR con el segmento de texto plano.



(a) Cifrado



(b) Descifrado

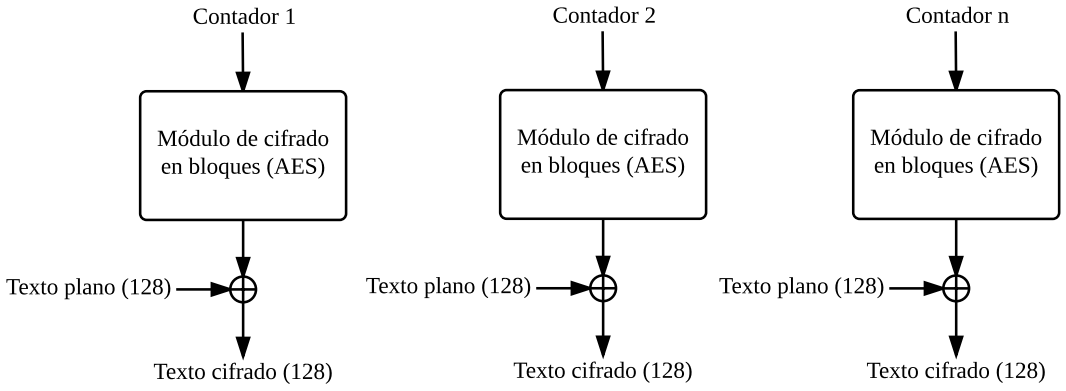
FIGURA 3.13: Esquema de cifrado y descifrado en modo OFB.

Counter Mode (CTR)

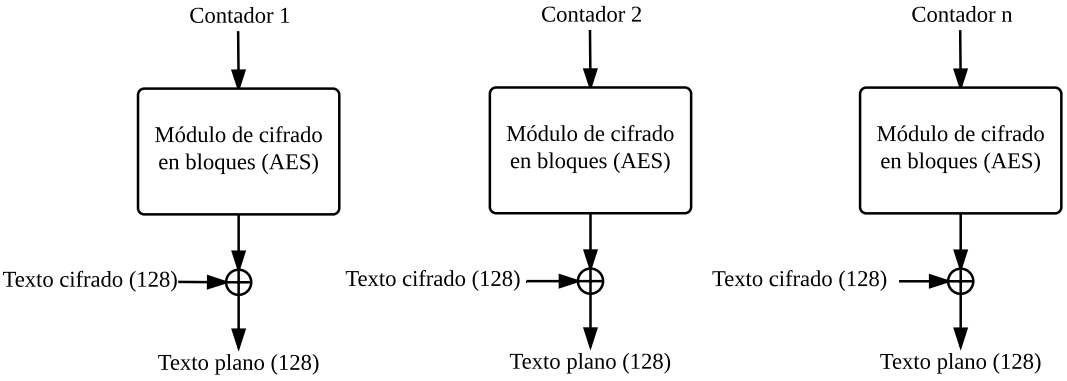
Este modo consiste en la aplicación de cifrado en bloques a un conjunto de valores de entrada, denominados contadores. Las secuencias de salida producidas son operadas por una función XOR junto con un segmento de texto plano. La secuencia de contadores debe ser tal que cada entrada al módulo de cifrado en bloques deber ser distinta a las demás, para todos los mensajes que se cifran bajo la misma llave. Para el último segmento de texto plano, cuya longitud es u, no necesariamente igual a la longitud del bloque, se usan los u bits más significativos de la salida del cifrado en bloque para realizar la XOR con el segmento de texto plano.

Esta configuración corresponde a un esquema de cifrado continuo sincrónico, debido a que el valor de la llave no depende del texto plano ni del texto cifrado.

El proceso de descifrado es exactamente igual al de cifrado. En este modo, las operaciones de cifrado y descifrado pueden ser paralelizadas. Además, los valores de salida de los distintos pasos de cifrado en bloque pueden ser generados en forma previa a que el texto plano esté disponible.



(a) Cifrado



(b) Descifrado

FIGURA 3.14: Esquema de cifrado y descifrado en modo CTR.

Capítulo 4

Parámetros de diseño y estructura general de las arquitecturas propuestas

Las implementaciones en hardware de un módulo de cifrado simétrico, así como de cualquier otro bloque digital, pueden ser evaluadas mediante distintas cifras de mérito como desempeño, área y latencia.

- Latencia: la latencia se define como el tiempo que se requiere para procesar los datos de entrada y obtener un resultado. En el caso del cifrado en bloques, corresponde al tiempo que transcurre desde que se ingresa un bloque de texto plano hasta que se obtiene a la salida el bloque de texto cifrado.
- Desempeño: cantidad de datos procesados por unidad de tiempo (generalmente la unidad de tiempo utilizada es un segundo). En el caso de cifrado en bloques, el desempeño corresponde a la cantidad de bloques de texto cifrado obtenidos a la salida del módulo en un segundo. Suele medirse en bits, bytes o paquetes por segundo.
- Área: cantidad de recursos utilizados por el módulo (compuertas lógicas en ASICs o bloques lógicos configurables (CLBs) en FPGAs).

Cada aplicación tiene sus requerimientos de latencia, desempeño y área. Es tarea de la persona a cargo del diseño de un bloque digital optimizarlo mediante la correcta elección

de la arquitectura general, tecnología (en caso de ser posible) y la forma en la que se implementa cada una de las operaciones que se llevan a cabo, para poder cumplir con los requerimientos de la aplicación.

El objetivo principal de este trabajo es el desarrollo de distintas arquitecturas para un procesador criptográfico AES. A pesar de que todas las arquitecturas implementan el mismo algoritmo, durante el diseño de cada una de ellas se utilizan determinadas técnicas de diseño digital para que presenten distintas características en cuanto a desempeño, velocidad y área, haciéndolas acordes para distintas aplicaciones. De esta manera, a pesar de que para un bloque de texto plano determinado las tres devuelven el mismo bloque de texto cifrado, la velocidad de procesamiento, el tiempo que se demora en obtener una salida y la forma en la que se procesan los datos difiere para cada una de ellas. Las tres arquitecturas propuestas en este trabajo se denominan *básica*, *pipeline* y *compacta*.

Tabla 4.1: Comparación entre las arquitecturas propuestas.

	Básica	Compacta	Pipeline
Prioridad	Relación media entre área/velocidad	Minimizar área	Maximizar desempeño
Aplicación	VoIP (Voip-info.org, 2018)	IoT (Alliance, 2018)	WiFi (IEEE 802.1)
Desempeño	15-64 Kbps	0.3-50 Kbps	11-1300 Mbps

En este capítulo se describen en primer lugar las distintos parámetros que se deben definir al diseñar un módulo de cifrado en bloques y se presentan alternativas para cada uno de ellos, evaluando su impacto en las figuras de mérito mencionadas. Luego, se define la estructura general que presentan las arquitecturas propuestas y los parámetros de diseño que comparten, así como aquellos que son diferentes para cada una.

4.1. Manejo de la llave secreta

Como se vio en el capítulo anterior, el estándar de NIST define que la llave secreta en AES puede ser de 128, 192 o 256 bits.

El proceso de expansión necesario para obtener las llaves de ronda puede realizarse de dos maneras: en paralelo con el cifrado, obteniendo cada llave de ronda a medida que es requerida, o calculando la totalidad de las llaves de ronda en forma previa al proceso de

cifrado. Estos esquemas se denominan *on-the-fly* y precálculo, respectivamente. Ambos representan una relación de compromiso entre velocidad y cantidad de recursos.

El precálculo de llaves presenta ventajas en cuanto a la velocidad con la que se dispone de las llaves de ronda durante el cifrado (solo se requiere leer una memoria para tener cada llave de ronda a disposición), pero requiere los medios suficientes para almacenar la totalidad de las llaves de ronda una vez generadas.

Por otro lado, la expansión *on-the-fly* es útil en los casos en los que no se cuenta con la memoria necesaria para almacenar las llaves, pero representa una desventaja en el proceso de descifrado ya que en este caso las llaves se requieren en un orden inverso al del cifrado. Esto implica que en la primer ronda es necesario contar con la última llave de ronda, que depende de todas las anteriores debido a la recursividad del algoritmo de expansión. Si las llaves de obtienen *on-the-fly*, el proceso de expansión para cada ronda de descifrado es más largo (y por ende demora más tiempo) que para el cifrado, ya que en este último la expansión es de una sola llave por ronda.

La elección entre estos esquemas también depende de la frecuencia con la que se modifica la llave secreta. Si la misma se mantiene constante durante múltiples operaciones de cifrado, una expansión *on-the-fly* implica la realización de la misma secuencia de operaciones utilizando los mismos operandos durante cada una de las iteraciones de cifrado, resultando en una implementación poco eficiente. Por el otro lado, en el caso extremo de que la llave secreta sea distinta para cada bloque de texto plano, una expansión con precálculo seria aún más ineficiente ya que para cada operación de cifrado se realiza el proceso completo de expansión y, además, se almacenan los resultados intermedios para ser utilizados una sola vez.

4.2. Modos de operación

Un módulo de cifrado en bloques AES generalmente es operado utilizando alguno de los modos de operación recomendados por NIST: ECB, CBC, CFB, OFB o CTR. El modo utilizado es uno de los factores que determinan la elección de la arquitectura adecuada para un sistema de cifrado en bloques.

En los modos no realimentados (ECB, CTR), el cifrado de cada bloque puede ser llevado a cabo en forma independiente de los demás. Esto implica la posibilidad de paralelizar las transformaciones de ronda para aumentar el desempeño del sistema. Por el contrario, en los modos realimentados (CBC, CFB y OFB) no es posible comenzar a cifrar un bloque de datos hasta que finaliza el proceso para el bloque anterior.

4.3. Bus interno e interfaz con el entorno

Generalmente, el ancho del bus interno de los módulos de cifrado en bloques es de 8, 32 ó 128 bits, dependiendo de los requerimientos de la aplicación y la disponibilidad de recursos. La interfaz entre el bloque de cifrado y su entorno puede ser realizada de diversas maneras. Sin embargo, es recomendable utilizar algún tipo de interfaz que sea estándar y facilite la reutilización del bloque de cifrado.

El uso de interfaces estándar permite la creación de bibliotecas de componentes reutilizables, simplifica el proceso de integración y facilita la verificación del diseño.

Los protocolos de interconexión AMBA (*Advanced Microcontroller Bus Architecture*) fueron originalmente desarrollados por ARM en 1996. La primer versión incluye dos buses: ASB (*Advanced System Bus*) y APB (*Advanced Peripheral Bus*). Con el tiempo los protocolos fueron actualizándose y se transformaron en estándar de facto para realizar interconexiones en chip. Este conjunto de protocolos de comunicación es independiente de la tecnología, posee una documentación completa y no requiere el pago de *royalties* para su utilización.

La versión 5 de la especificación AMBA define los siguientes protocolos de comunicación:

- AXI5, AXI5-Lite y ACE5 (ARM, 2017)
- Advanced High-performance Bus (AHB5, AHB-Lite) (ARM, 2015)
- CHI Coherent Hub Interface (CHI) (ARM, 2014)
- Distributed Translation Interface (DTI)(ARM, 2018b)
- Generic Flash Bus (GFB) (ARM, 2018a)

4.4. Implementación de cada una de las transformaciones de ronda.

El algoritmo de cifrado AES está compuesto por una ejecución iterada de transformaciones que componen una red de Sustitución-Permutación. Estas transformaciones son básicamente desplazamientos, operaciones algebraicas sobre campos finitos y sustituciones.

Existen distintas alternativas al implementar las transformaciones de ronda que componen el algoritmo de cifrado AES. En esta sección se describen los enfoques más comúnmente hallados en publicaciones y bibliografía.

4.4.1. SubBytes e InvSubBytes:

Estas transformaciones de ronda pueden ser implementadas mediante lógica combinacional que representa las operaciones en campos finitos que conforman las transformaciones, o mediante LUTs. Entre ellas existe una relación de compromiso entre área y velocidad: mientras que las LUT son de rápido acceso, la lógica combinacional presenta una disminución en el área.

A partir de la publicación del estándar AES, la comunidad científica se esforzó en la obtención de implementaciones de las transformaciones SubBytes e InvSubBytes cada vez más compactas. En (Canright, 2005), (Satoh y col., 2001) y (Rijmen, 2000) se propone la utilización de isomorfismos entre los campos $GF\{2^8\}$, $GF\{2^4\}$, $GF\{2^2\}$ y $GF\{2\}$ para simplificar el cálculo de la inversa multiplicativa de un elemento de $GF\{2^8\}$. Este método es utilizado frecuentemente para la implementación de estas transformaciones en ASICs.

4.4.2. MixColumns e InvMixColumns

Como se mencionó con anterioridad, esta operación corresponde a la multiplicación entre polinomios sobre el campo finito $GF\{2^8\}$. En la Ecuación 4.1 se observa la transformación MixColumns en forma matricial. Existen diversas estrategias para implementar la lógica que conforma esta operación; la mayoría de ellas consiste en expresar los factores

de la multiplicación en función de elementos más simples, como XORs y multiplicaciones por el elemento {02}, descripta en detalle en el Apéndice A.

$$\begin{bmatrix} S'_{0,j} \\ S'_{1,j} \\ S'_{2,j} \\ S'_{3,j} \end{bmatrix} = \begin{bmatrix} 02\ 03\ 01\ 01 \\ 01\ 02\ 03\ 01 \\ 01\ 01\ 02\ 03 \\ 03\ 01\ 01\ 02 \end{bmatrix} \times \begin{bmatrix} S_{0,j} \\ S_{1,j} \\ S_{2,j} \\ S_{3,j} \end{bmatrix} \tag{4.1}$$

$$S'_{0,j} = \{02\}S_{0,j} \oplus \{03\}S_{1,j} \oplus S_{2,j} \oplus S_{3,j} \tag{4.2}$$

$$S'_{1,j} = S_{0,j} \oplus \{02\}S_{1,j} \oplus \{03\}S_{2,j} \oplus S_{3,j} \tag{4.3}$$

$$S'_{2,j} = S_{0,j} \oplus S_{1,j} \oplus \{02\}S_{2,j} \oplus \{03\}S_{3,j} \tag{4.4}$$

$$S'_{3,j} = \{03\}S_{0,j} \oplus S_{1,j} \oplus S_{2,j} \oplus \{02\}S_{3,j} \tag{4.5}$$

La multiplicación entre un elemento cualquiera del campo finito $GF\{2^8\}$ y el elemento {02} puede ser implementada mediante un desplazamiento y tres operaciones XOR (Standaert y col., 2003). En el presente trabajo esta operación se denominada como `porX` y se representa en los diagramas como un recuadro con este nombre, indicando una implementación como la que se observa en la Figura 4.1.

El producto de un elemento por {03} puede ser obtenido mediante el producto de dicho elemento por {02}, seguido de una operación XOR entre el resultado de esta operación y el elemento original.

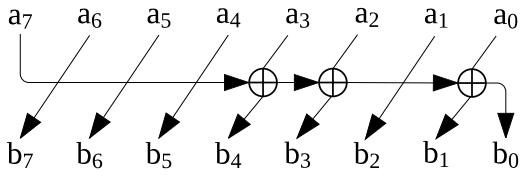


FIGURA 4.1: Multiplicación por el elemento {02} en $GF\{2^8\}$.

Una implementación intuitiva de la transformación MixColumns puede verse en la Figura 4.2. En la misma, se generan las multiplicaciones por {02} y {03} para cada uno de los bytes que componen una columna de la matriz de estado. A continuación, se opera con estos factores para obtener la transformación de ronda. En la figura, por razones de claridad, sólo se muestra el cálculo completo del primer byte de la transformación.

El cálculo de los tres bytes restantes de cada columna de la matriz de estado se realiza operando en forma análoga, según las Ecuaciones 4.3, 4.4 y 4.5. Estas últimas surgen de la multiplicación matricial dada en la Ecuación 4.1. Esta implementación requiere en total 144 compuertas XOR para realizar la transformación de una columna, y su camino crítico es el retardo de cuatro compuertas XOR.

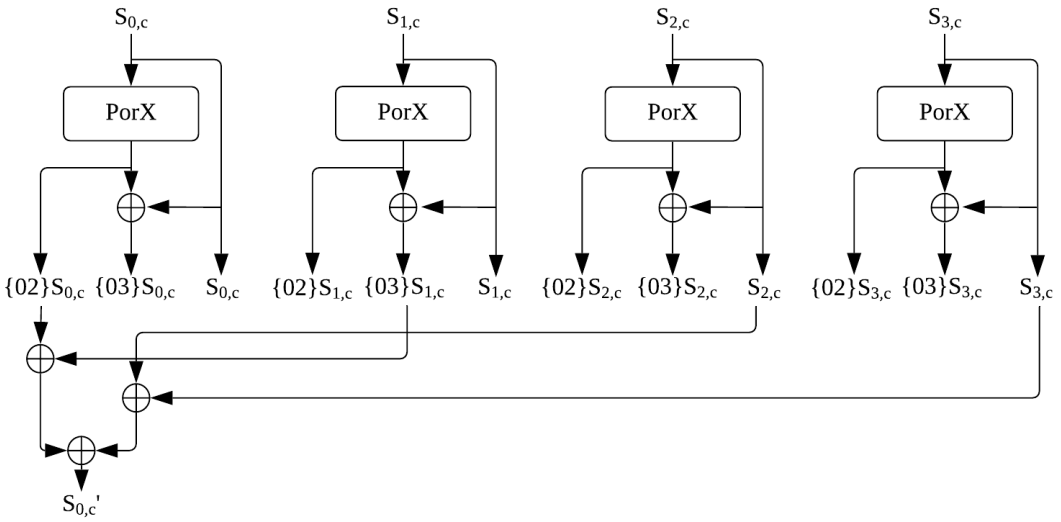


FIGURA 4.2: Posible implementación de la transformación MixColumns.

Otra implementación posible (Lu y Tseng, 2002) se obtiene al agrupar las expresiones que conforman la transformación de ronda, obteniendo como resultado las Ecuaciones 5.1, 5.2, 5.3 y 5.4. En la Figura 4.3 se observa la implementación para el primer byte de la columna. La misma puede ser fácilmente extendida para hallar la transformación de los tres bytes restantes. Esta implementación requiere un total de 144 compuertas XOR para realizar la transformación de una columna, y su camino crítico es el retardo de tres compuertas XOR.

$$S'_{0,j} = \{02\}(S_{0,j} \oplus S_{1,j}) \oplus S_{1,j} \oplus S_{2,j} \oplus S_{3,j} \tag{4.6}$$

$$S'_{1,j} = S_{0,j} \oplus \{02\}(S_{1,j} \oplus S_{2,j}) \oplus S_{2,j} \oplus S_{3,j} \tag{4.7}$$

$$S'_{2,j} = S_{0,j} \oplus S_{1,j} \oplus \{02\}(S_{2,j} \oplus S_{3,j}) \oplus S_{3,j} \tag{4.8}$$

$$S'_{3,j} = S_{0,j} \oplus S_{1,j} \oplus S_{2,j} \oplus \{02\}(S_{3,j} \oplus S_{0,j}) \tag{4.9}$$

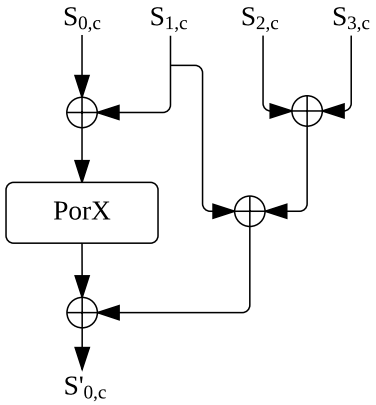


FIGURA 4.3: Posible implementación de la transformación MixColumns, agrupando operandos.

Por otro lado en (Noo-Intara, 2004) se propone una implementación adecuada para aquellos casos en los que se prioriza minimizar la cantidad de recursos utilizados. En esta implementación, ilustrada en la Figura 4.4 para una sola columna, se obtiene un coeficiente por cada ciclo de reloj. En este caso se cuenta con un registro de desplazamiento en el que se almacenan los cuatro bytes de una columna de la matriz de estados. El primer elemento del registro se multiplica por el coeficiente 02, el segundo por 03 y el tercer y cuarto se multiplican por 01. En cada ciclo de reloj se desplaza el registro y se obtiene un nuevo byte de la columna resultante. Se requieren cuatro ciclos de reloj para completar la transformación. Para transformar la matriz de estados completa se requieren cuatro instancias de esta transformación (una por cada columna), o bien instanciar solo una y requerir 16 ciclos de reloj.

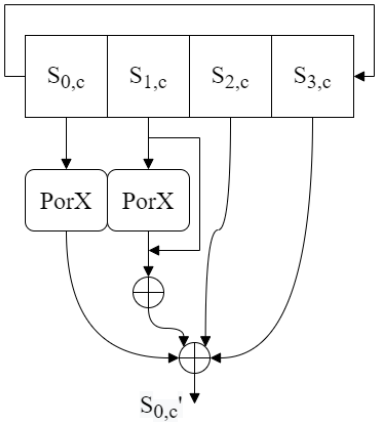


FIGURA 4.4: Posible implementación de la transformación MixColumns, reducida en área.

La transformación inversa, InvMixColumns, está definida por le expresión matricial de la Ec. 4.10.

$$\begin{bmatrix} S'_{0,j} \\ S'_{1,j} \\ S'_{2,j} \\ S'_{3,j} \end{bmatrix} = \begin{bmatrix} 0e\ 0b\ 0d\ 09 \\ 09\ 0e\ 0b\ 0d \\ 0d\ 09\ 0e\ 0b \\ 0b\ 0d\ 09\ 0e \end{bmatrix} \times \begin{bmatrix} S_{0,j} \\ S_{1,j} \\ S_{2,j} \\ S_{3,j} \end{bmatrix} \tag{4.10}$$

$$S'_{0,j} = \{0e\}S_{0,j} \oplus \{0b\}S_{1,j} \oplus \{0d\}S_{2,j} \oplus \{09\}S_{3,j} \tag{4.11}$$

$$S'_{1,j} = \{09\}S_{0,j} \oplus \{0e\}S_{1,j} \oplus \{0b\}S_{2,j} \oplus \{0d\}S_{3,j} \tag{4.12}$$

$$S'_{2,j} = \{0d\}S_{0,j} \oplus \{09\}S_{1,j} \oplus \{0e\}S_{2,j} \oplus \{0b\}S_{3,j} \tag{4.13}$$

$$S'_{3,j} = \{0b\}S_{0,j} \oplus \{0d\}S_{1,j} \oplus \{09\}S_{2,j} \oplus \{0e\}S_{3,j} \tag{4.14}$$

Las distintas implementaciones de InvMixColumns surgen de la capacidad de expresar la multiplicación entre un elemento cualquiera del campo finito $GF\{2^8\}$, denominado X, y los elementos $\{09\}$, $\{0b\}$, $\{0d\}$ y $\{0e\}$ en función de multiplicaciones por el elemento $\{02\}$, como se muestra en las Ecuaciones 4.15, 4.16, 4.17 y 4.18.

$$\{09\}X = \{02\}(\{02\}(\{02\}X)) \oplus X \tag{4.15}$$

$$\{0b\}X = \{02\}(\{02\}(\{02\}X)) \oplus \{03\}X \tag{4.16}$$

$$\{0d\}X = \{02\}(\{02\}(\{02\}X)) \oplus \{05\}X \tag{4.17}$$

$$\{0e\}X = \{02\}(\{02\}(\{02\}X)) \oplus \{06\}X \tag{4.18}$$

Estas expresiones resultan en la implementación de InvMixColumns que se muestra en la Figura 4.5. La misma requiere un total de 368 compuertas XOR para realizar la transformación de una columna y su camino crítico es el retardo de seis compuertas XOR.

En forma análoga a la transformación MixColumns, es posible obtener mejoras en la implementación de InvMixColumns agrupando algunos de los términos en las expresiones matemáticas de la transformación de cada byte. El resultado se muestra en las Ecuaciones 4.19, 4.20, 4.21 y 4.22. La implementación resultante se ilustra en la Figura 4.6 para uno de los bytes de la columna. La transformación de los tres bytes restantes se implementa en forma análoga. Este enfoque requiere de 320 compuertas XOR para realizar la transformación de una columna y su camino crítico es el retardo de siete compuertas

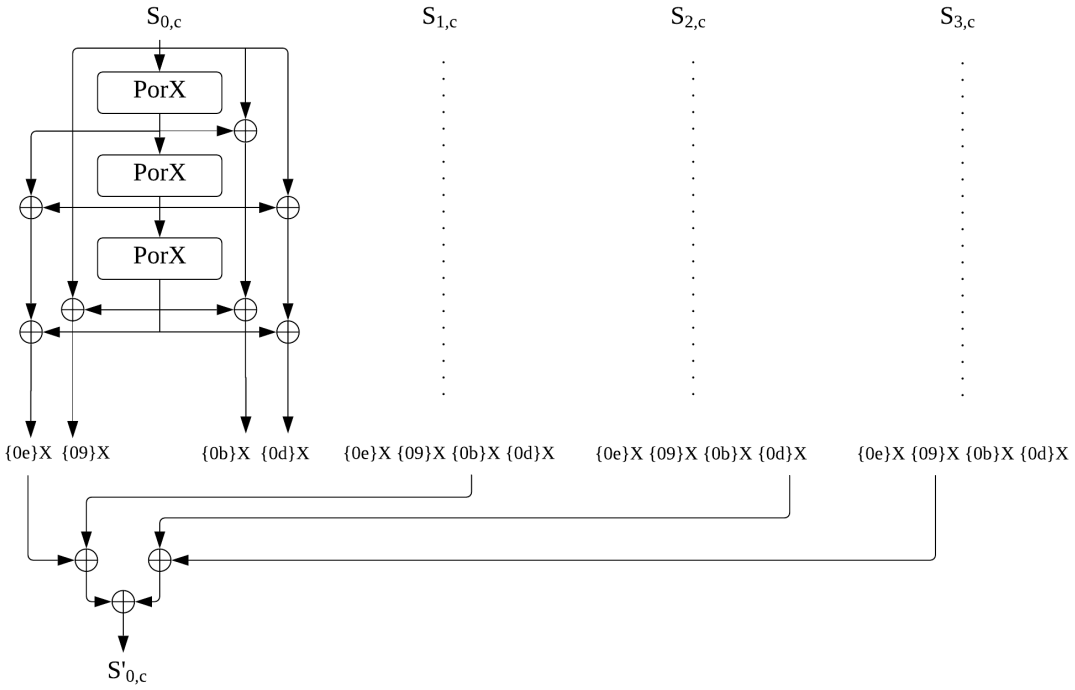


FIGURA 4.5: Posible implementación de la transformación InvMixColumns.

XOR.

$$S'_{0,c} = \{04\}(\{02\}(S_{0,c} \oplus S_{1,c}) \oplus \{02\}(S_{2,c} \oplus S_{3,c}) \oplus (S_{0,c} \oplus S_{2,c})) \oplus \{02\}(S_{0,c} \oplus S_{1,c}) \oplus S_{1,c} \oplus S_{2,c} \oplus S_{3,c} \quad (4.19)$$

$$S'_{1,c} = \{04\}(\{02\}(S_{1,c} \oplus S_{2,c}) \oplus \{02\}(S_{3,c} \oplus S_{0,c}) \oplus (S_{1,c} \oplus S_{3,c})) \oplus \{02\}(S_{1,c} \oplus S_{2,c}) \oplus S_{2,c} \oplus S_{3,c} \oplus S_{0,c} \quad (4.20)$$

$$S'_{2,c} = \{04\}(\{02\}(S_{2,c} \oplus S_{3,c}) \oplus \{02\}(S_{0,c} \oplus S_{1,c}) \oplus (S_{2,c} \oplus S_{0,c})) \oplus \{02\}(S_{2,c} \oplus S_{3,c}) \oplus S_{3,c} \oplus S_{0,c} \oplus S_{1,c} \quad (4.21)$$

$$S'_{3,c} = \{04\}(\{02\}(S_{3,c} \oplus S_{0,c}) \oplus \{02\}(S_{1,c} \oplus S_{2,c}) \oplus (S_{3,c} \oplus S_{1,c})) \oplus \{02\}(S_{3,c} \oplus S_{0,c}) \oplus S_{0,c} \oplus S_{1,c} \oplus S_{2,c} \quad (4.22)$$

En los casos en los que se prioriza minimizar recursos, InvMixColumns puede ser implementada mediante un registro de desplazamiento y una operación de multiplicación por cada uno de los coeficientes (0e, ob, od y 09), tal como se ilustra en la Figura 4.7. Al igual que para MixColumns, el registro desplaza los bytes en cada ciclo de reloj y se requieren cuatro desplazamientos para obtener una columna de la matriz de estados.

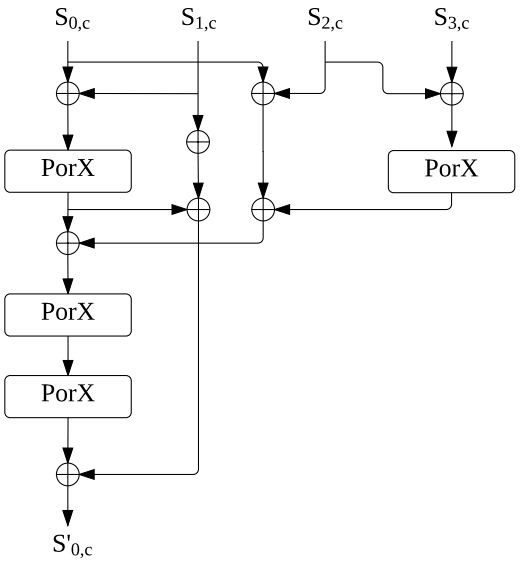


FIGURA 4.6: Posible implementación de la transformación InvMixColumns, agrupando operandos.

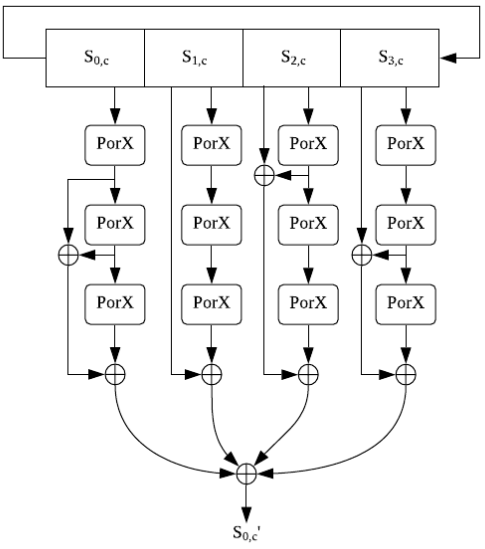


FIGURA 4.7: Posible implementación de la transformación InvMixColumns, reducida en área.

4.5. Manejo de iteraciones

El manejo de iteraciones define la cantidad de bits que se procesan en simultáneo durante cada ciclo de reloj. Este parámetro de diseño depende de la cantidad de instancias de las transformaciones de ronda y la cantidad de bits que procesa cada una de ellas. En esta sección se analizan tres posibles enfoques para el módulo de cifrado AES: básico, desenroscado y *pipeline*.

4.5.1. Enfoque básico

En un diseño iterado, el enfoque básico consiste en la implementación una sola instancia de la lógica sobre la que se itera (ronda) de manera que se ejecuta una iteración por cada ciclo de reloj.

Para el caso de un módulo de cifrado en bloques, la secuencia de transformaciones que componen una ronda se implementan mediante lógica combinacional. Como se observa en la Figura 4.8, la entrada a esta lógica combinacional que implementa una ronda está controlada por un multiplexor. De esta manera, durante la primer ronda la entrada a la lógica es un nuevo bloque de texto plano mientras que para las rondas subsiguientes se ingresa el resultado obtenido en la ronda anterior. El registro de 128 bits actualiza su valor en cada ciclo de reloj, controlando el flujo de información para que se ejecute una ronda por cada ciclo de reloj.

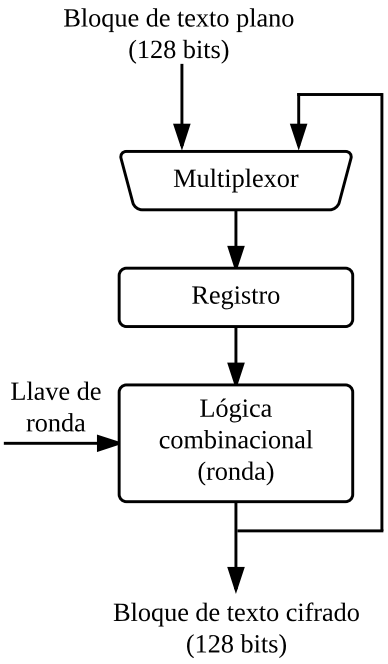


FIGURA 4.8: Enfoque básico.

Las características de este enfoque son:

- El número de ciclos de reloj requeridos para cifrar un bloque de datos es igual al número de rondas (Nr).

- El mínimo valor para el período de reloj, T_{clk}, depende del retardo de la lógica combinacional correspondiente a una ronda (t_r), los tiempos de establecimiento y propagación del registro (t_s y t_p respectivamente) y el tiempo de propagación del multiplexor (t_m).

$$Tclk_b = t_r + t_s + t_p + t_m \tag{4.23}$$

- La latencia y el desempeño son:

$$L_b = N_r \times T_{clk}, \tag{4.24}$$

$$D_b = \frac{128 \text{ bits}}{N_r \times T_{clk}} \tag{4.25}$$

La frecuencia máxima de reloj ($\frac{1}{Tclk_b}$) está prácticamente definida por el retardo de la lógica combinacional, que corresponde a la suma de los retardos de cada una de las transformaciones que componen una ronda: AddRoundKey, SubBytes, ShiftRows y MixColumns. Como se mencionó en las secciones anteriores, este valor depende de la manera en la que se implementa cada una de ellas.

4.5.2. Enfoque desenroscado

El enfoque desenroscado se basa en la implementación de múltiples instancias de la lógica sobre la que se itera, con el objetivo llevar a cabo múltiples iteraciones de ronda por cada ciclo de reloj, reduciendo la cantidad de iteraciones sobre la misma. En este caso se llevan a cabo K rondas de cifrado durante un ciclo de reloj, siendo K un divisor de N_r, Figura 4.9.

Las características de este enfoque son:

- El número de ciclos de reloj necesarios para cifrar un bloque es $\frac{N_r}{K}$.
- Debido a que se implementan K instancias de la lógica correspondiente a una ronda, el área en este caso es aproximadamente K veces mayor al de el enfoque básico.
- El período mínimo del reloj es aproximadamente K veces el período mínimo para el enfoque básico.

$$Tclk_{lu} = K \times t_r + t_s + t_p + t_m. \tag{4.26}$$

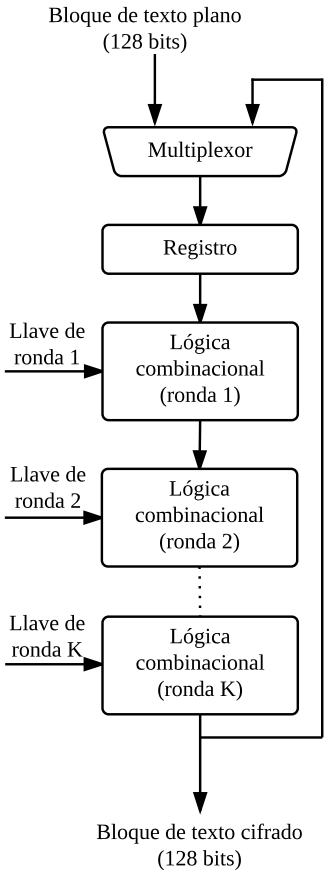


FIGURA 4.9: Enfoque desenroscado.

- La relación entre el desempeño de este enfoque y el obtenido para el básico es

$$D_{lu} = D_b \times \frac{(1+\tau)}{(1+\tau/K)}, \tag{4.27}$$

donde

$$\tau = \frac{t_s + t_p + t_m}{t_r}. \tag{4.28}$$

En el caso que $K = N_r$ se dice que el desenroscado es completo. En este caso el multiplexor a la entrada deja de ser necesario.

Debe tenerse en cuenta que en este enfoque se requieren K llaves de ronda por cada ciclo de reloj. Esto implica la necesidad de calcularlas en forma paralela en caso de considerarse un esquema de expansión *on-the-fly*, o bien dejarlas a disposición en memoria (*precálculo*).

Dado el pequeño aumento en el desempeño a expensas de un incremento notable en el área (proporcional a K), la elección de este enfoque sólo es justificable para las aplicaciones en las que se utiliza el cifrado en uno de los modos realimentados y el incremento de área pueda ser aceptable.

4.5.3. Enfoque Pipeline

En los enfoques anteriores no es posible ingresar un nuevo bloque de texto plano hasta que finaliza el cifrado del bloque anterior. Para ciertas aplicaciones esta restricción puede resultar ineficiente, ya que a la salida se obtiene un bloque de texto plano cada una cierta cantidad de ciclos de reloj (salvo en la estructura desenroscada, en la que cuando $K = N_r$ se obtiene un bloque de texto cifrado por cada ciclo de reloj, pero con un reloj K veces más lento).

Para un módulo iterado como AES, el enfoque *pipeline* consiste en la implementación de K (siendo K divisor de N_r) rondas interconectadas mediante registros, Figura 4.10. De esta manera, en cada ciclo de reloj se ejecutan K rondas en simultáneo. Si $K = N_R$ se dice que el *pipeline* es completo. En ese caso se puede procesar K bloques de texto plano en paralelo y el multiplexor de entrada deja de ser necesario.

Las características de este enfoque son:

- El número de ciclos de reloj necesarios para cifrar un bloque es N_r.
- Debido a que se implementan K instancias de la lógica correspondiente a una ronda, el área en este caso es aproximadamente K veces mayor al del enfoque básico (es un poco mayor al de la estructura desenroscada para un mismo valor de K, debido a los registros extras).
- El período mínimo del reloj es igual al período mínimo para el enfoque básico. Su valor depende principalmente del retardo de la lógica combinacional que implementa las transformaciones de ronda.

$$Tclk_p = t_r + t_s + t_p + t_m. \tag{4.29}$$

- El desempeño en este caso es K veces mayor al obtenido en el enfoque básico

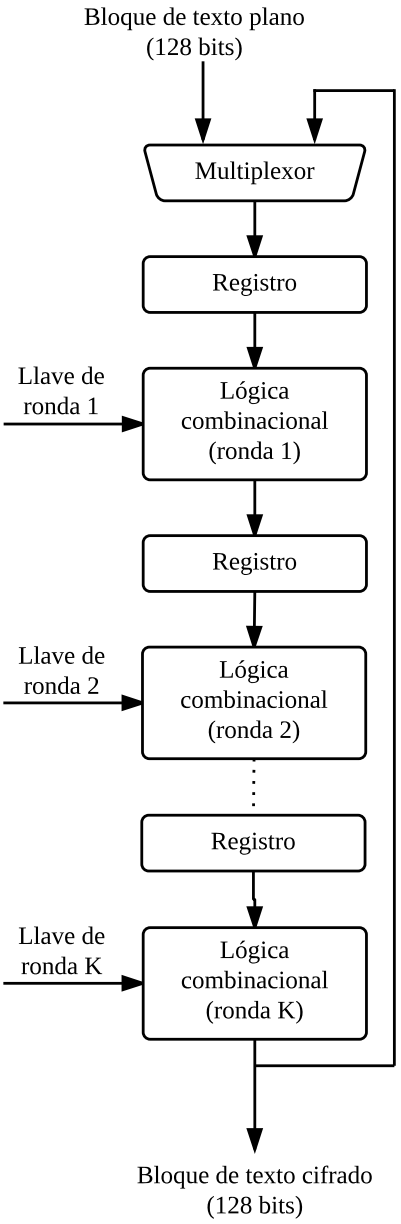


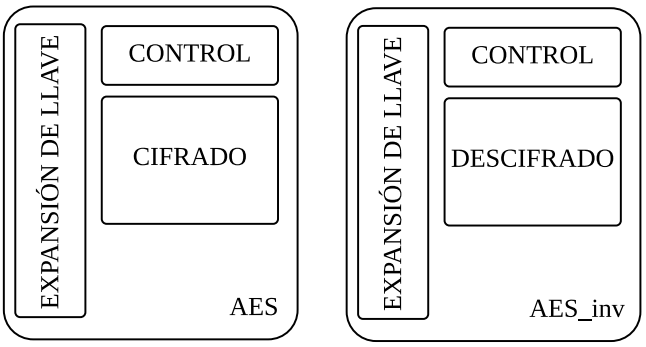
FIGURA 4.10: Enfoque pipeline.

$$D_p = K \times D_b \tag{4.30}$$

En este enfoque también se requieren de K llaves en simultáneo, por lo que el *precálculo* resulta en la mejor opción para el manejo de la expansión de llaves.

4.6. Estructura general de las arquitecturas propuestas

A pesar de ser diseñadas para cumplir con distintas especificaciones, las tres arquitecturas propuestas en este trabajo (*básica*, *compacta* y *pipeline*) poseen una misma estructura modular que consta de tres bloques con función específica: un bloque dedicado a la expansión de llave, un bloque de cifrado o descifrado y un bloque de control, tal como se ilustra en la Figura 4.11. A pesar de que cada uno de estos bloques se implementa de distintas maneras para cada una de la arquitecturas, la estructura modular se mantiene.



(a) Cifrado (b) Descifrado

FIGURA 4.11: Diagrama en bloques para las arquitecturas de cifrado y descifrado.

- Expansión de llave: este bloque genera once llaves de ronda a partir de una llave secreta, mediante el algoritmo de expansión detallado en la sección 3.4.
- Cifrado/Descifrado: estos bloques realizan el proceso de cifrado o descifrado. Además, cuentan con un arreglo en el que se almacenan las llaves de ronda generadas en el bloque de expansión en los casos en los que se realiza un precálculo de las mismas.
- Control: bloque que controla la interfaz con el entorno y la comunicación entre los bloques de expansión y cifrado o descifrado. Es responsable de que se envíen datos a los bloques de expansión y cifrado sólo en los momentos en los que los mismos se encuentran en estado ocioso, de manera de evitar sobre-escrituras y errores. Además, controla que el cifrado o descifrado de un dato comience sí y solo sí la expansión de llave fue realizada con éxito y el arreglo en que se almacenan las llaves de ronda está completo en caso de realizarse un precálculo de las mismas.

Estos tres bloques se interconectan entre sí mediante señales de dato y validación que permiten sincronizar la comunicación entre ellos y con el exterior, como se ilustra en la Figura 4.12. En la misma, las flechas anchas representan buses de datos. La Tabla 5.1 describe la funcionalidad de cada una de las señales que componen la interacción entre los bloques de cifrado, expansión de llave y control y están presentes en las tres arquitecturas propuestas. Para los buses de datos la cantidad de bits difiere ya que tanto las arquitecturas *básica* como *pipeline* cuentan con buses de 128 bits, mientras que para *compacta* estos buses son de 32 bits. El esquema de comunicación utilizado consiste en la presencia de una señal de validación (de un bit) asociada a cada una de las señales de datos. De esta manera, se le informa al bloque que recibe una señal de datos en qué ciclos de reloj la información es válida y debe ser leída.

Tabla 4.2: Entradas, salidas y señales internas en las arquitecturas propuestas

Puerto	Sentido	Tamaño	Descripción
CLK	entrada	1 bit	Reloj
RST	entrada	1 bit	Reset sincrónico activo con nivel lógico alto
data_in	entrada	128/32 bits	Puerto mediante el cual se ingresa la llave secreta o un bloque de texto plano
plain_valid	entrada	1 bit	Mediante un nivel lógico alto indica que el puerto *data_in* contiene un bloque de texto plano válido
key_valid	entrada	1 bit	Mediante un nivel lógico alto indica que el puerto *data_in* contiene una llave secreta válida
cipher_text	salida	128/32 bits	Puerto mediante el cual el módulo AES entrega un bloque de texto cifrado
cipher_text_valid	salida	1 bit	Mediante un nivel lógico alto indica que el puerto *cipher_text* contiene un bloque de texto cifrado válido
cipher_ready	salida	1 bit	Mediante un nivel lógico alto indica que el módulo AES está listo para recibir un nuevo bloque de datos, ya sea texto plano o llave

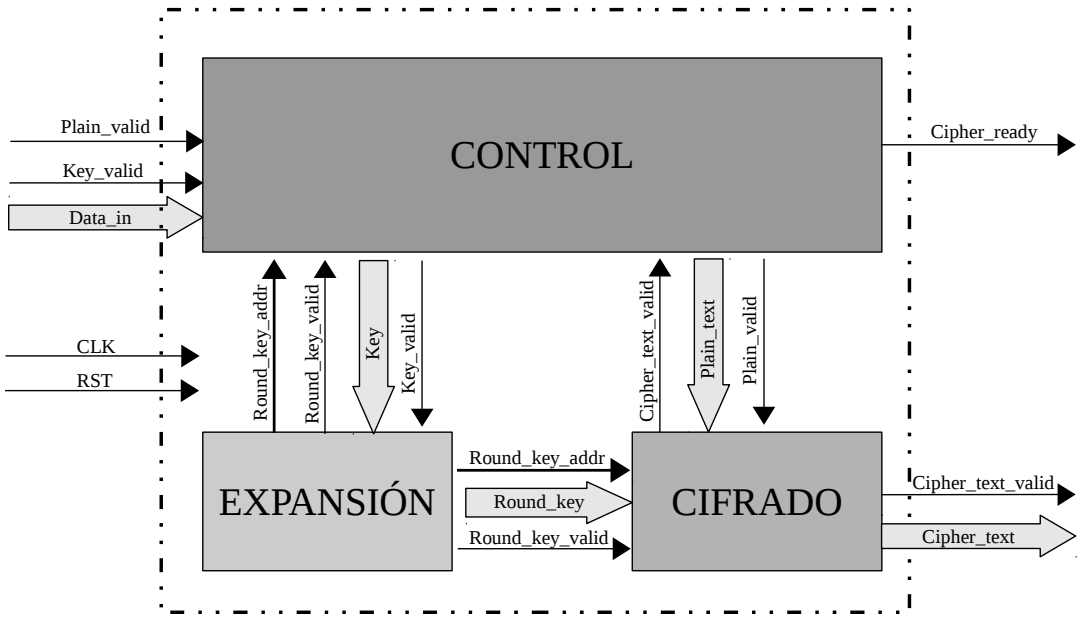


FIGURA 4.12: Estructura general de las arquitecturas propuestas. Las flechas angostas representan señales de un bit, y las más anchas representan buses de datos.

Independientemente de las diferencias en cuanto a la estructura utilizada al implementar el cifrado o descifrado, las tres arquitecturas propuestas presentan ciertas características en común, que responden a algunas de las especificaciones presentadas en las secciones previas:

- Longitud de la llave de 128 bits. La llave utilizada en los tres casos es de este tamaño, ya que según distintas recomendaciones emitidas por entidades de seguridad como NSA (National Security Agency) (NSA, 2009), ANSII (Agence Nationale de la Sécurité des Systèmes d'information) (ANSII, 2014), NIST (National Institute of Standards and Technology) (NIST, 2012a), ECRYPT (European Network of Excellence in Cryptology) (ECRYPT, 2012), las llaves de 128 bits continuarán siendo seguras para el cifrado simétrico hasta el año 2030, como mínimo.
- Modo de operación ECB. La elección de este modo de operación se debe a que es la base para todos los demás modos de operación. Su extensión para implementar cualquiera de los demás modos, en caso de ser requerido por una aplicación en particular, es sencilla; la lógica adicional consiste frecuentemente en funciones XOR y registros. Otro factor que se consideró es que AES-128 en modo ECB es la configuración más utilizada en publicaciones para realizar comparaciones entre distintas implementaciones del algoritmo.

- Interfaz entre el bloque de cifrado y su entorno: AMBA AHB.

  Para que un diseño sea versátil y pueda ser integrado en bloques de mayor complejidad, es sumamente importante que cuente con una interfaz estándar que facilite su comunicación con otros componentes. En este trabajo se decidió utilizar una interfaz AMBA AHB[1] es distinta para las tres arquitecturas propuestas.

- Implementación de la transformación de ronda SubBytes: Debido a que las tres arquitecturas de AES se sintetizaron en FPGA, se decidió implementar las transformaciones SubBytes e InvSubBytes mediante el uso de LUTs, ya que este tipo de plataforma cuenta con bloques de memoria que pueden ser fácilmente utilizados con este propósito, sin alterar la cantidad de bloques lógicos configurables disponibles.

La diferencia entre las arquitecturas *básica*, *pipeline* y *compacta* radica en la manera en la que se implementan los sub-bloques de expansión, cifrado y control, principalmente los de cifrado o descifrado. Para *básica* y *pipeline* se implementaron técnicas como las ilustradas en las Figuras 4.8 y 4.10, respectivamente; mientras que en *compacta* se comparten recursos para llevar a cabo una ronda.

[1] En el Apéndice B se describen las transacciones básicas establecidas en el protocolo.
La implementación de este tipo de interfaz se realizó mediante un envoltorio o *wrapper* que adapta las señales del estándar a las de los módulos de cifrado desarrollados. La estructura de este envoltorio

La Tabla 4.3 resume las estrategias de diseño que se utilizaron en cada una de ellas. En los capítulos siguientes se describe en detalle el desarrollo las tres arquitecturas propuestas en este trabajo.

TABLA 4.3: Comparación entre las arquitecturas propuestas.

	Básica	Compacta	Pipeline
Manejo de llaves	Precálculo y *on-the-fly*	Precálculo y *on-the-fly*	Precálculo
Implementación de MixColumns	Agrupación de operandos(Lu y Tseng, 2002)	Reducida (Noo-Intara, 2004)	Agrupación de operandos (Lu y Tseng, 2002)
Manejo de iteraciones	Enfoque básico	Uso compartido de recursos	Pipeline

4.7. Entorno de simulación

El banco de prueba o *test bench* utilizado para verificar la funcionalidad de las estructuras diseñadas se basa en la metodología UVM (*Universal Verification Metodology*). Esta metodología fue estandarizada por Accellera (Accellera, 2018) y su utilización es ampliamente extendida en la industria, ya que admite la reutilización de gran parte del entorno de verificación. La estructure de UVM se basa en la composición de objetos que a lo largo de su simulación pasan por diferentes fases: creación, construcción, ejecución y reporte. Las clases requeridas se codifican en SystemVerilog.

En la Figura 4.13 se ilustra un diagrama del entorno utilizado. En el mismo se observa que en el archivo de mayor jerarquía, denominado `tb_top`, se instancia el diseño bajo testeo (DUT por su sigla en inglés) y se conecta sus puertos de entrada y salida a una interface. Esta interface es estimulada y monitoreada desde el agente, que forma parte del entorno instanciado en el test.

En este caso en particular el Agente cuenta también con un predictor, que recibe el mismo estímulo (texto plano y llave) que el DUT y genera el valor de texto cifrado esperado. Este bloque consiste en el modelo de referencia utilizado para determinar si las salidas del DUT son correctas. Cada vez que genera un nuevo texto cifrado, el predictor lo envía al scoreboard mediante un puerto de análisis (*analysis port*).

El driver es el encargado de estimular al DUT, enviando a la interface los valores de texto plano y llave indicados por la Secuencia. Estos valores se especifican para cada uno de los tests.

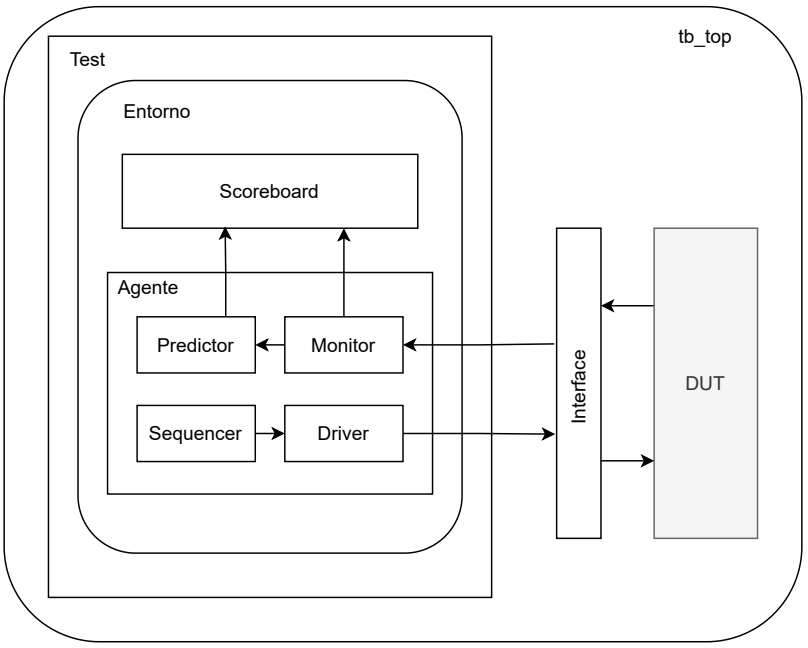


FIGURA 4.13: Entorno de simulación utilizando metodología UVM.

El Monitor es el encargado de observar las entradas y salidas del DUT. Por un lado, monitorea las entradas para determinar en qué momentos el DUT recibe un nuevo bloque de texto plano o llave secreta. Ante esta situación envía estos valores al predictor, mediante un Puerto de Análisis, de manera que el predictor reciba en todo momento los mismos estímulos que el DUT. El Monitor también observa la interface para determinar en qué momentos el DUT genera un nuevo bloque de texto cifrado. Ante esta situación envía el valor de texto cifrado al scoreboard, mediante un Puerto de Análisis, de manera que cada vez que el DUT genera un bloque de texto cifrado, el mismo sea comparado con la salida correspondiente del predictor.

El scoreboard compara el valor del bloque de texto cifrado que generó el DUT (detectado por el Monitor) con el valor de texto cifrado generado por el predictor. En caso de que no sean iguales se genera un error. El scoreboard cuenta con una cola que almacena los valores predichos a medida que los recibe. El objetivo es evitar pérdidas de información en caso de que un nuevo valor de predicción se reciba antes de que el monitor detecte un nuevo bloque de texto cifrado en la interface.

La utilización de esta metodología de verificación admite que el entorno utilizado para las tres arquitecturas sea el mismo. La implementación de la interface, driver y monitor

se adaptan a las diferencias que existen entre ellas. Por otro lado, los tests que se corren son los mismos y su descripción se mantiene para todas las arquitecturas.

4.7.1. Tests

Los tests que verifican la funcionalidad de las arquitecturas propuestas básicamente se dividen en dos categorías:

- Dirigidos: los tests dirigidos se basan en los vectores de prueba recomendados por NIST en su suite de validación para el algoritmo AES (NIST, 2002). Cada uno de estos tests incluye una secuencia de bloques de texto plano y llaves secreta que fueron diseñados específicamente para hallar defectos en la implementación del algoritmo de cifrado.
- Aleatorio: se incluye también un test en el que se genera en forma aleatoria una secuencia de 500 bloques de texto plano. En forma aleatoria se producen también cambios en la llave secreta entre ellos.

4.7.2. Verificación funcional del envoltorio AHB

Para los casos en los que se agrega un envoltorio AHB, además de adaptar la interface, driver y monitores se agregaron *assertions* de SystemVerilog (SVA por sus siglas en inglés) que verifican el correcto funcionamiento de un esclavo AHB.

El comportamiento de un esclavo AHB, según la especificación de AMBA AHB debe comprender las siguientes consideraciones:

- Las señales de control que devuelve el esclavo al maestro son HREADY y HRESP.
- HREADY indica la terminación o extensión de una transferencia (HREADY = 0 extiende la fase de datos de una transferencia).
- HRESP indica el éxito o falla de una transferencia. (HRESP = 0 indica OK y HRESP = 1 indica ERROR).
- Ante una transferencia de tipo IDLE los esclavos deben ignorar las señales de control y proveer una respuesta de tipo exitosa y sin ciclos de espera.

- Ante una transferencia de tipo BUSY los esclavos deben ignorar las señales de control y proveer una respuesta de tipo exitosa y sin ciclos de espera.

Estas restricciones en el comportamiento del esclavo AHB, junto con ciertos requerimientos de las arquitecturas propuestas, se expresan como *assertions*. El objetivo es chequear en todo momento las señales del protocolo de comunicación AHB y verificar que cumplan con determinadas características que son detalladas para cada arquitectura en los capítulos siguientes.

Capítulo 5

Desarrollo de la arquitectura básica

El diseño de la arquitectura *básica* prioriza obtener valores medios de desempeño y área, razón por la cual es apropiada para aplicaciones como VoIP. La estructura contiene tres bloques: control, cifrado y expansión de llave. El bloque de cifrado consta de la implementación de una ronda y el algoritmo AES se lleva a cabo al realizar $N_r = 10$ iteraciones sobre este hardware, como se ilustra en la Figura 5.1.

Cada bloque de texto plano de 128 bits que ingresa al bloque de cifrado debe atravesar la lógica que implementa la ronda inicial. El resultado de esta operación ingresa 10 veces a la lógica que implementa una ronda general en forma iterativa, una vez por cada ciclo de reloj.

El bloque de control es el encargado de asegurar que el cifrado de un bloque de texto plano comience solo si el módulo de cifrado está listo, es decir, si finalizó el proceso de expansión de llave y no se está cifrando un bloque de texto plano anterior.

La Tabla 5.1 presenta la descripción de la interfaz de esta arquitectura. En la misma se observa que los buses de datos internos y hacia el exterior son de 128 bits, de manera que en cada ciclo de reloj se procesa la matriz de estados completa.

Con respecto al ingreso de datos, se utiliza un solo bus para disminuir la cantidad de puertos de entrada. Mediante dos señales de control *plain_valid* y *key_valid* se indica si el dato ingresado corresponde a un bloque de texto plano o a la llave secreta. Internamente

estas dos señales se utilizan como habilitación para dos registros de 128 bits, uno que almacena el texto plano y es entrada al bloque de cifrado y otro que almacena la llave secreta y es entrada al bloque de expansión de llave.

Con respecto a las salidas, *cipher_text* contiene el resultado del bloque cifrado y su valor es válido en los ciclos de reloj en los cuales *cipher_text_valid* posee un nivel lógico alto. Por otro lado, *cipher_ready* posee un valor lógico alto en los ciclos de reloj en los que la arquitectura está lista para recibir un nuevo bloque de texto plano.

TABLA 5.1: Descripción de entradas y salidas de la arquitectura *básica*

Puerto	Sentido	Tamaño	Descripción
CLK	entrada	1 bit	Reloj
RST	entrada	1 bit	Reset sincrónico activo con nivel lógico alto
data_in	entrada	128 bits	Puerto mediante el cual se ingresa la llave secreta o un bloque de texto plano
plain_valid	entrada	1 bit	Mediante un nivel lógico alto indica que el puerto *data_in* contiene un bloque de texto plano válido
key_valid	entrada	1 bit	Mediante un nivel lógico alto indica que el puerto *data_in* contiene una llave secreta válida
cipher_text	salida	128 bits	Puerto mediante el cual el módulo AES entrega un bloque de texto cifrado
cipher_text_valid	salida	1 bit	Mediante un nivel lógico alto indica que el puerto *cipher_text* contiene un bloque de texto cifrado válido
cipher_ready	salida	1 bit	Mediante un nivel lógico alto indica que el módulo AES está listo para recibir un nuevo bloque de datos, ya sea texto plano o llave

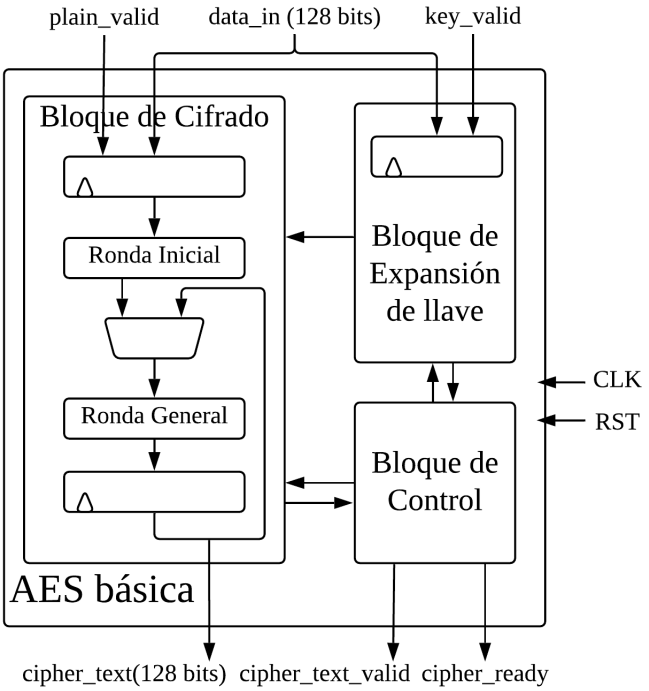


FIGURA 5.1: Diagrama en bloques de la arquitectura *básica*.

5.1. Cifrado, control y expansión de llave

5.1.1. Diseño del bloque de cifrado

El diseño del bloque de cifrado para esta arquitectura se realizó siguiendo un esquema de manejo de iteraciones básico, en el cual se lleva a cabo una ronda de cifrado por cada ciclo de reloj. En este esquema se instancia la lógica correspondiente a una ronda inicial (XOR entre el bloque de texto plano y la llave secreta), seguido de la lógica correspondiente a una ronda general (SubBytes + ShiftRows + MixColumns). La realimentación entre los puertos de salida y entrada de una ronda general se realiza a través de un registro de 128 bits que actualiza su valor en cada ciclo de reloj durante el proceso de cifrado, tal como se ilustra en la Figura 5.2.

En la Figura 5.2 también se observa la presencia de registros a la entrada y a la salida del bloque de cifrado. El registro de entrada se habilita en los ciclos de reloj en los que la señal de entrada *plain_valid* posee un nivel lógico alto, indicando que el dato presente en el bus de entrada *plain_text* es válido.

La cantidad de iteraciones se controla a partir de un contador denominado *ronda*. Este contador se reinicia a uno cada vez que ingresa un nuevo bloque de texto plano e incrementa su valor en los ciclos de reloj posteriores hasta llegar a once. En los ciclos de reloj en los que su valor es menor a diez se habilita el registro de realimentación que comunica la salida de la ronda general con su entrada, admitiendo de esta manera la ejecución de una ronda por cada ciclo de reloj. En los ciclos de reloj en los que *ronda* vale diez (última iteración) se omite la transformación MixColumns mediante el multiplexor de salida. Por otro lado, en este ciclo de reloj también se habilita el registro de salida, permitiendo que el bloque de texto cifrado se envíe al exterior mediante el bus *cipher_text*, junto con un nivel alto en la salida *cipher_text_valid* indicando que el valor presente en el bus es válido.

Como se mencionó en el capítulo anterior, existen diversas maneras de implementar las transformaciones que componen una ronda de cifrado AES. A continuación se detalla la forma en la que se implementan estas transformaciones en la arquitectura *básica*.

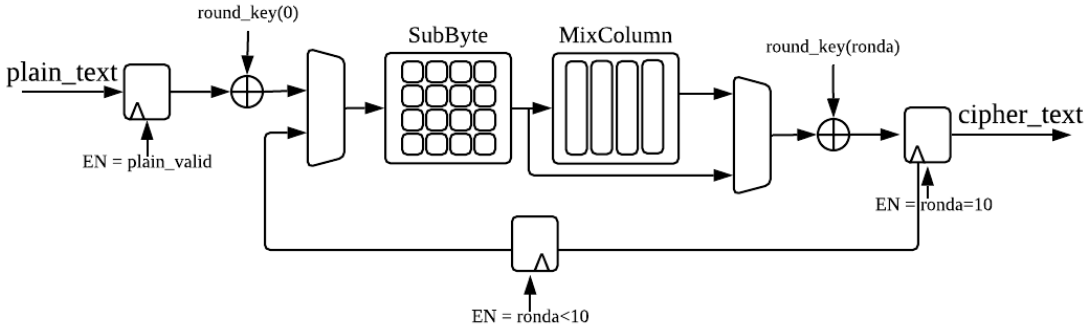


FIGURA 5.2: Diagrama del bloque de cifrado en la arquitectura *básica*.

SubBytes

La transformación de ronda SubBytes se implementa mediante el uso de LUTs. El bloque de cifrado contiene 16 instancias de la LUT que define esta operación (SBOX). De esta manera, se opera en paralelo sobre los 16 bytes que conforman el estado. Cada una de estas LUTs se describe en lenguaje VHDL como una sentencia de tipo *case* en la que se evalúa cada uno de los posibles valores del byte de entrada y se le asigna un valor determinado a la salida en cada caso en función de la Tabla 5.2

TABLA 5.2: Implementación de la S-BOX mediante una LUT.

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

ShiftRows

La transformación de ronda ShiftRows consiste en desplazamientos en las filas de la matriz de *estado*. Estos desplazamientos no requieren de una lógica en particular ya que

se implementan mediante la correcta interconexión entre la salida de las 16 LUTS que conforman SubBytes y la entrada de la lógica que implementa MixColumns.

MixColumns

La transformación de ronda MixColumns se implementa utilizando la estructura que se obtiene al agrupar las expresiones de la multiplicación matricial (Lu y Tseng, 2002) y que fue detallada en el capítulo anterior. El bloque de cifrado contiene cuatro instancias de la lógica que transforma una columna del estado, de manera que se transforman las cuatro columnas de la matriz en simultáneo.

Por comodidad se replican las Ecuaciones 5.1, 5.2, 5.3 y 5.4. La Figura 5.3, ilustra la lógica necesaria para operar sobre una columna de la matriz. La misma consta de operaciones XOR bit a bit entre bytes y bloques que implementan la multiplicación con el elemento $\{02\}$, tal como se detalló en el capítulo anterior.

$$S'_{0,j} = \{02\}(S_{0,j} \oplus S_{1,j}) \oplus S_{1,j} \oplus S_{2,j} \oplus S_{3,j} \tag{5.1}$$

$$S'_{1,j} = S_{0,j} \oplus \{02\}(S_{1,j} \oplus S_{2,j}) \oplus S_{2,j} \oplus S_{3,j} \tag{5.2}$$

$$S'_{2,j} = S_{0,j} \oplus S_{1,j} \oplus \{02\}(S_{2,j} \oplus S_{3,j}) \oplus S_{3,j} \tag{5.3}$$

$$S'_{3,j} = S_{0,j} \oplus S_{1,j} \oplus S_{2,j} \oplus \{02\}(S_{3,j} \oplus S_{0,j}) \tag{5.4}$$

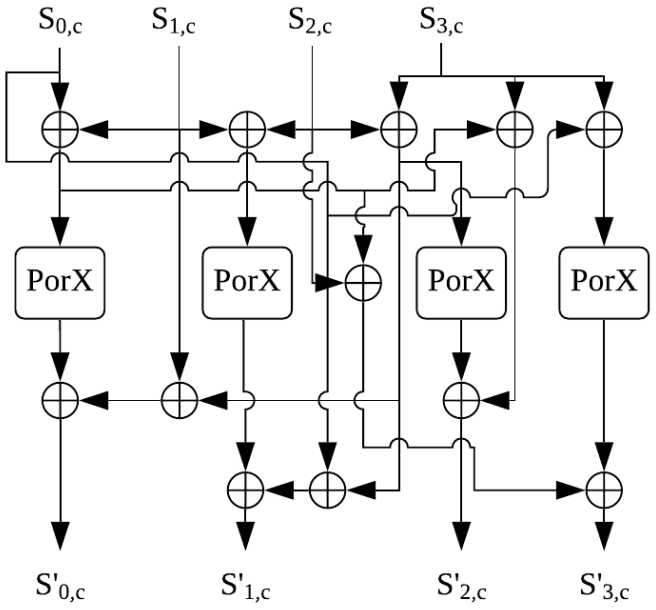


FIGURA 5.3: Implementación de la transformación MixColumns en la Arquitectura básica.

AddRoundKey

La transformación de ronda AddRoundKey se implementa mediante 16 instancias de la operación XOR entre dos bytes. De esta manera se procesan los 128 bits de la matriz de estado en simultáneo y se halla la XOR con la llave de ronda. Esta última proviene de un arreglo interno en caso de realizarse un precálculo de llaves, o directamente del bloque de expansión cuando se realiza una expansión *on-the-fly*.

La Figura 5.4 ilustra la manera en la que se implementa la ronda general. En la misma, se identifica con un valor hexadecimal a cada uno de los bytes del dato de 128 bits de entrada. Además, se identifica a los distintos tipos de transformaciones de ronda con distintos colores. Cada una de las instancias corresponde a las implementaciones mencionadas:

- Las 16 instancias que componen SubBytes corresponden a 16 LUTs como la mostrada en la Tabla 5.2.
- ShiftRows se implementa mediante la correcta interconexión entre los bytes que se obtienen en cada una de las LUTs y los bytes de entrada a MixColumns.
- MixColumns consta de 4 instancias de la lógica que procesa una columna. Cada una de estas instancias está compuesta de compuertas XOR y multiplicaciones por {02}, tal como se muestra en la Figura 5.3.
- AddRoundKey consta de 16 instancias de una compuerta XOR bit a bit entre dos bytes.

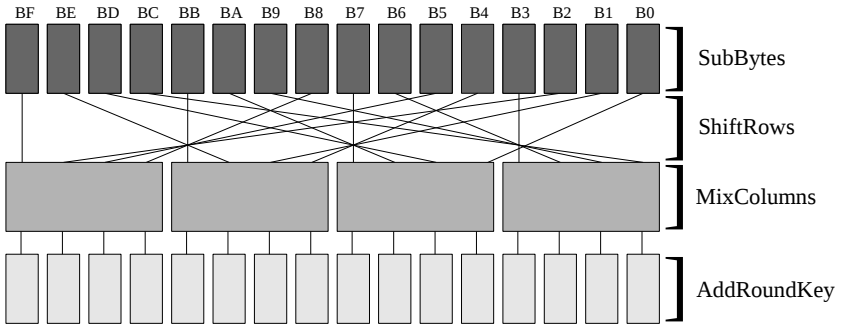


FIGURA 5.4: Esquema de una ronda general para la arquitectura *básica*.

5.1.2. Diseño del bloque de expansión de llave

Con respecto a la expansión de llave se realizaron dos versiones de la arquitectura *básica*, una en la que se precalculan las llaves de ronda y otra en la que se calculan a medida que se las requiere.

Precálculo

En esta versión de la arquitectura *básica* la expansión de llave es precalculada. Esto implica que la expansión se hace en forma previa al cifrado y no requiere estar en sincronismo con otros bloques. Con el fin de lograr una relación de compromiso entre el uso de recursos y el tiempo que demanda este bloque, se estableció como criterio de diseño que calcule una llave de ronda por cada ciclo de reloj. Para cumplir con esta restricción se utiliza una esquema de manejo de iteraciones básico, análogo al utilizado en el bloque de cifrado, obteniendo la estructura que se muestra en la Figura 5.5. En la misma se observa que se implementa la lógica correspondiente a una iteración del algoritmo de expansión y se lleva a cabo una iteración por cada ciclo de reloj mediante la realimentación entre sus puertos de salida y entrada.

La lógica que implementa una iteración de la expansión de llave opera sobre los cuatro *words* que componen su entrada. Como se mencionó en el Capítulo 3, esta lógica está compuesta por operaciones XOR bit a bit entre los *words* y la aplicación de la función $g(\cdot)$. Esa última posee una $SBOX$ idéntica a la utilizada en la transformación de ronda SubBytes, además de desplazamientos y una operación XOR con una constante llamada $RCON$, cuyo valor es diferente para cada iteración.

En la Figura 5.5 se observa la presencia de un registro de entrada y un registro de realimentación, ambos de 128 bits. El registro de entrada se habilita en los ciclos de reloj en los que la señal de entrada *key_valid* posee un nivel lógico alto, indicando que el dato presente en el bus de entrada *key* es válido. Por otro lado, el registro de realimentación actualiza su valor en cada ciclo de reloj durante el proceso de expansión de llave.

La cantidad de iteraciones se controla a partir de un contador denominado *ronda*. Este contador se reinicia a uno cada vez que ingresa una nueva llave e incrementa su valor en los ciclos de reloj posteriores hasta llegar a once. El registro de realimentación que comunica la salida y la entrada de la ronda de expansión se habilita en los ciclos de reloj

en los que el valor de ronda es menor a once, admitiendo de esta manera la expansión de una nueva llave por cada ciclo de reloj.

A medida que este bloque calcula las llaves de ronda, las envía al bloque de cifrado junto con una señal de habilitación (*key_out_valid*) y una dirección (*key_out_addr*) que identifica a cada llave. Esta dirección se utiliza en el bloque de cifrado para indexar las llaves de ronda en un arreglo de llaves denominado *key_table*. El bloque de cifrado accede a este arreglo de llaves de ronda en cada una de las iteraciones de cifrado.

Tanto la señal de habilitación del bus de salida, como la dirección que identifica a cada llave de ronda, son enviadas además al bloque de control para que este último pueda identificar el momento en el que finaliza el proceso de expansión de llave.

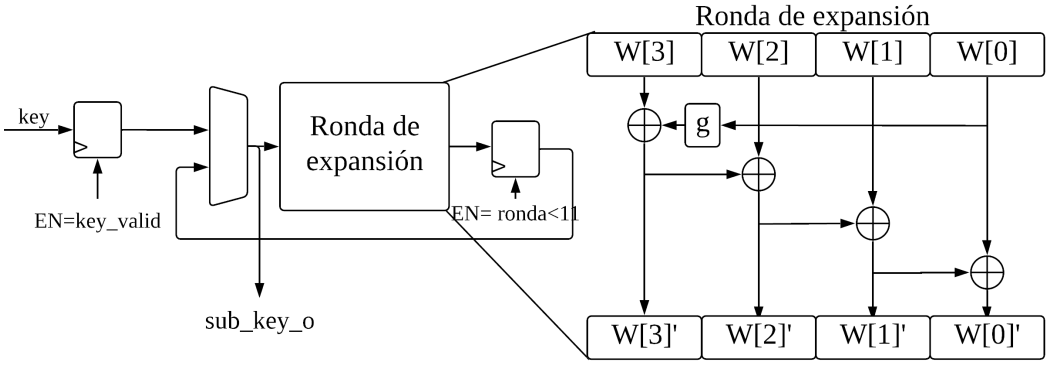


FIGURA 5.5: Diagrama del bloque de expansión de llave en la arquitectura *básica*.

on-the-fly

En una segunda versión de la arquitectura *básica* la expansión de llave se realiza en paralelo con el cifrado. De esta manera, el bloque de expansión obtiene la enésima llave de ronda ronda en el mismo ciclo de reloj en el que el bloque de cifrado lleva a cabo la enésima ronda de cifrado. La implementación del bloque de expansión es análoga a la presentada en la Figura 5.5, salvo que en este caso el valor de ronda de expansión se incrementa en sincronismo con el valor de ronda de cifrado. De esta manera deja de ser necesario el arreglo de llaves y el bloque de control prescinde del estado de expansión de llave.

El bloque de control registra el valor de la llave secreta en los ciclos de reloj en los que la entrada *key_valid* posee un nivel lógico alto. Este valor registrado es el que posteriormente envía al bloque de expansión de llave cada vez que ingresa un nuevo bloque de texto plano al módulo (ciclos de reloj en los cuales *plain_valid* posee un nivel lógico alto).

A pesar de que representa mejoras en cuanto a la cantidad de recursos necesarios para la expansión de llave, la desventaja de este tipo de enfoque es que el cálculo de llave se produce cada vez que se procesa un dato. De esta manera, en los casos en los que la llave secreta no se modifica con frecuencia el módulo de expansión halla las mismas once llaves de ronda cada vez que ingresa un nuevo bloque de texto plano.

5.1.3. Bloque de control

La función principal de este bloque es la generación y envío de señales de control hacia el exterior y a los bloques de cifrado y expansión. Su objetivo es asegurar el correcto flujo de la información, ignorando solicitudes en caso que se requiera el cifrado de un nuevo dato mientras se está llevando a cabo la expansión de llave o el cifrado de un dato anterior. Además, asegura que el cifrado de un dato comience sí y solo sí la expansión de llave fue llevada a cabo en el caso de utilizarse un esquema de expansión de llave con precálculo.

El bloque de control se implementa mediante la máquina de estados ilustrada en la Figura 5.6.

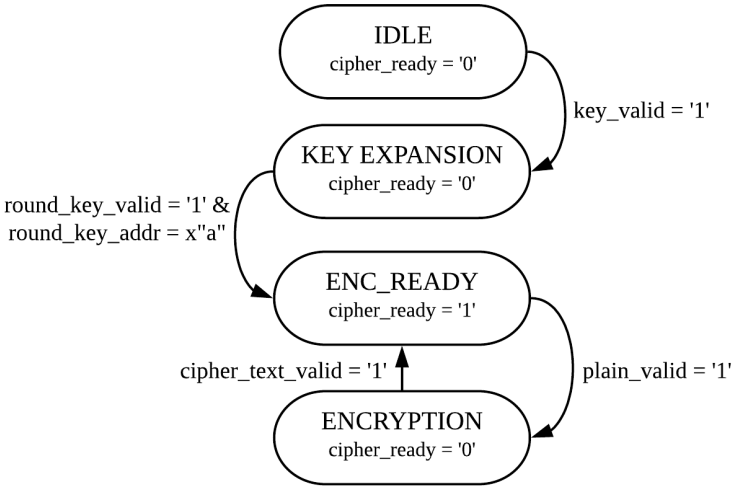


FIGURA 5.6: Máquina de estados en el bloque de control para la arquitectura *básica*.

- La máquina de estados permanece en estado ocioso IDLE desde que detecta un nivel alto en la entrada *reset* hasta que detecta un nivel alto en la entrada *key_valid*, indicando el ingreso de la llave secreta mediante el bus de datos. Una vez que se detecta el ingreso de la llave, la máquina de estados evoluciona a KEY_EXPANSION.
- La máquina de estados permanece en estado KEY_EXPANSION durante el transcurso de tiempo en el cual el bloque de expansión de llave genera las once llaves de ronda. Una vez que se genera la última llave de ronda (situación identificada a partir del valor de dirección de llave generada por el bloque de expansión) la máquina de estados de control pasa al estado ENC_READY. En la implementación en la que la expansión de llave se realiza *on-the-fly* el estado KEY_EXPANSION carece de utilidad. Por lo tanto, en ese caso la FSM evoluciona de IDLE a ENC_READY en forma directa cuando la señal de entrada *key_valid* posee nivel alto.
- En el estado ENC_READY el bloque de control da aviso al exterior de que el módulo de cifrado está listo para recibir un nuevo bloque de texto plano. Esta señalización se realiza mediante un nivel alto en la salida *cipher_ready*. Una vez que se recibe un bloque de texto plano, la máquina de estados evoluciona a ENCRYPTION y esta señal baja, indicando que el módulo de cifrado está ocupado.
- La máquina de estados permanece en estado ENCRYPTION durante el transcurso de tiempo en el cual el bloque de cifrado procesa el dato ingresado. Una vez que finaliza el cifrado (situación identificada a partir de un nivel lógico alto en la señal *cipher_text_valid* proveniente del bloque de cifrado) la máquina de estados regresa a ENC_READY, indicando la disponibilidad para recibir un nuevo bloque de texto plano.
- El ingreso de una nueva llave secreta puede realizarse en los estados IDLE o ENC_READY. Si se recibe una llave en otro momento se la ignora.
- El ingreso de un nuevo bloque de texto plano puede realizarse en el estado ENC_READY. Si se recibe un bloque de texto plano en otro momento se lo ignora.
- El dato enviado a los bloques de expansión de llave y cifrado se conecta directamente con el bus de datos de entrada *data_in*. Por otro lado, las señales de habilitación *key_valid* y *plain_valid* se envían a los bloques de expansión o cifrado solo si el módulo está listo para recibir una nueva llave o bloque de texto plano. Además, el bloque de control adapta estas señales de habilitación de manera que,

independientemente de la cantidad de ciclos de reloj que duren estas señales de entrada, los bloques de expansión y cifrado siempre reciban señales de habilitación con una duración fija de un ciclo de reloj.

5.2. Interfaz AMBA

El protocolo de comunicación AMBA AHB determina que los esclavos deben tener las entradas y salidas detalladas en la Tabla 5.3. La adaptación de las señales de entrada

Tabla 5.3: Entradas y salidas de un esclavo AHB-Lite

Nombre	Fuente	Descripción
HCLK	Fuente de reloj	Reloj global del sistema
HRESETn	Controlador de reset	Reset global del sistema. Única señal activa con nivel lógico bajo.
HADDR[31:0]	Maestro	Bus de direcciones de 32 bits.
HBURST[2:0]	Maestro	Indica si una transferencia es simple o forma parte de una ráfaga.
HMASTLOCK	Maestro	Indica con nivel alto que la transferencia actual forma parte de una secuencia bloqueada.
HPROT[3:0]	Maestro	Indica si una transferencia forma parte de búsqueda de *opcode* o acceso de datos, y si está en modo de acceso privilegiado o de usuario.
HSIZE[2:0]	Maestro	Indica el tamaño de la transferencia.
HTRANS[1:0]	Maestro	Indica el tipo de transferencia (IDLE, BUSY, NONSEQUENTIAL o SEQUNTIAL).
HWDATA[31:0] [1]	Maestro	Bus que transfiere datos desde el maestro hacia los esclavos durante transferencias de escritura.
HWRITE	Maestro	Indica la dirección de una transferencia. Es decir, si se trata de una lectura o escritura.
HSEL	Decodificador	Selección de los esclavos.
HREADY	Multiplexor	Indica mediante un nivel alto que la transferencia anterior está completa.
HRDATA[31:0] [1]	Esclavo	Bus que transfiere datos desde los esclavos hacia el maestro durante transferencias de lectura.
HREADYOUT	Esclavo	Indica con un nivel lógico alto la finalización de una transferencia. Esta señal es usada por el esclavo para extender una transferencia, mediante un nivel bajo.
HRESP	Esclavo	Provee información adicional al maestro sobre el estado de una transferencia.

[1]El ancho de este bus no está restringido a 32 bits. El protocolo admite 8, 16, 32, 64, 128, 256, 512 y 1024 bits.

y salida del módulo de cifrado para que cumpla con los requerimientos del protocolo

de comunicación AMBA AHB se realiza a partir de un envoltorio, tal como se ilustra en la Figura 5.7. Este último se implementa en forma independiente, de manera que su utilización sea opcional.

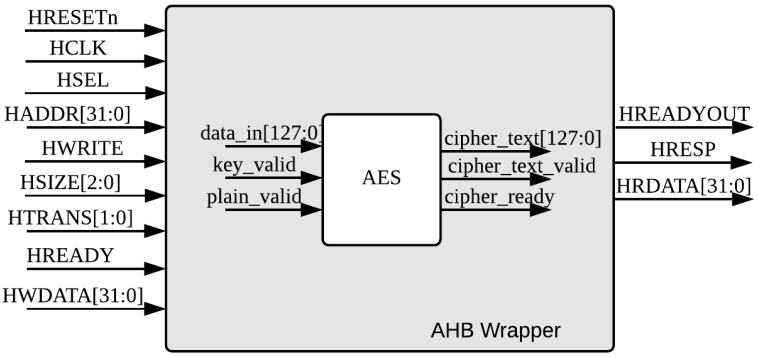


FIGURA 5.7: Envoltorio AHB para la arquitectura *básica*.

El envoltorio para la arquitectura *básica* cuenta con buses de datos HRDATA y HWDATA de 32 bits. Debido a que en esta arquitectura del módulo AES los buses de datos de entrada y salida *data_in* y *cipher_text* son de 128 bits, el envío de datos desde un Maestro AHB hacia el módulo AES y viceversa se realiza mediante la escritura y lectura de registros internos de 32 bits.

El envoltorio cuenta con doce registros internos de 32 bits. Ocho registros se utilizan para el envío de texto plano y llave desde el Maestro AHB hacia el módulo de cifrado, mientras que los cuatro restantes se utilizan para el envío de texto cifrado hacia el Maestro AHB. Cada uno de estos registros posee una dirección específica y se escriben y leen utilizando el protocolo de comunicación definido para AHB Lite. La Tabla 7.3 muestra el nombre de cada uno de estos registros, su conexión interna a los buses de datos del módulo de cifrado y su dirección.

Con la finalidad de ingresar una nueva llave secreta al módulo de cifrado, el Maestro AHB escribe los cuatro registros de llave en forma consecutiva, siguiendo el protocolo de comunicación especificado en el estándar y detallado en el Apéndice B de este trabajo. El Maestro envía la dirección del registro a escribir a través del bus de direcciones HADDR, junto con un nivel alto en la señal de escritura HWRITE y en la señal de selección HSEL (esta última se genera a partir de la dirección y tiene un nivel alto cuando el maestro se comunica con el módulo de cifrado). El envoltorio lee estas señales y escribe los registros internos siguiendo el protocolo AHB. Una vez que se escribe el

TABLA 5.4: Registros definidos en el envoltorio AHB-Lite para la arquitectura básica.

Nombre	Dato	Dirección
reg_key_3	Key[127:96]	x00000000
reg_key_2	Key[95:64]	x00000004
reg_key_1	Key[63:32]	x00000008
reg_key_0	Key[31:0]	x0000000c
reg_plain_3	Plain_text[127:96]	x00000010
reg_plain_2	Plain_text[95:64]	x00000014
reg_plain_1	Plain_text[63:32]	x00000018
reg_plain_0	Plain_text[31:0]	x0000001c
reg_ciph_3	Cipher_text[127:96]	x00000020
reg_ciph_2	Cipher_text[95:64]	x00000024
reg_ciph_1	Cipher_text[63:32]	x00000028
reg_ciph_0	Cipher_text[31:0]	x0000002c

registro correspondiente al *word* menos significativo (*reg_key_0*), el envoltorio envía el contenido de la llave (concatenación de los cuatro registros de llave) a través del bus de datos *data_in* del módulo AES y activa la señal de habilitación que indica la presencia de una llave secreta válida (*key_valid*).

Con la finalidad de ingresar un bloque de texto plano al módulo de cifrado, el Maestro AHB escribe los cuatro registros de texto plano en forma consecutiva, siguiendo el protocolo de comunicación especificado en el estándar y detallado en el Apéndice B de este trabajo. El Maestro envía la dirección del registro a escribir a través del bus de direcciones HADDR, junto con un nivel alto en la señal de escritura HWRITE y en la señal de selección HSEL (esta última se genera a partir de la dirección y tiene un nivel alto cuando el maestro se comunica con el módulo de cifrado). El envoltorio lee estas señales y escribe los registros internos siguiendo el protocolo AHB. Una vez que se escribe el registro correspondiente al *word* menos significativo (*reg_plain_0*), el envoltorio envía el contenido del texto plano (concatenación de los cuatro registros de texto plano) a través del bus de datos *data_in* del módulo AES y activa la señal de habilitación que indica la presencia de un bloque de texto plano válido (*plain_valid*).

Debido a que en ambos casos el envoltorio genera las señales de habilitación y envía los datos al módulo de cifrado en el momento en el que se escribe el registro menos significativo, es crucial realizar la escritura de este registro luego de haber escrito correctamente los tres registros restantes que componen el dato (no es importante el orden de escritura de estos tres registros, solo que el último en escribirse sea el menos significativo). De esta manera se asegura que el dato que llega al módulo de cifrado sea el adecuado.

La señal de salida HREADYOUT del envoltorio se mantiene en nivel lógico bajo durante los procesos de cifrado y expansión de llave, indicando al maestro AHB que el módulo de cifrado está ocupado y no puede recibir un nuevo dato. Una vez que el proceso de cifrado finaliza, el envoltorio utiliza la salida *cipher_text_valid* del módulo de cifrado como habilitación para escribir los cuatro registros correspondientes al texto cifrado. Estos registros pueden ser accedidos por el maestro AHB utilizando el protocolo de lectura y mantienen su valor hasta que se obtiene un nuevo bloque de texto cifrado. La señal *cipher_text_valid* puede ser utilizada además como interrupción para dar aviso al Maestro AHB de que los registros de salida contienen un dato válido.

5.3. Verificación funcional

Utilizando la herramienta XSim, incorporada en el entorno de Vivado, se simularon tanto los tests dirigidos como el test aleatorio de 500 muestras. La totalidad de los tests corridos pasaron, obteniendo resultados favorables.

A continuación se presentan formas de onda que ilustran el proceso de expansión de llave y cifrado para la arquitectura básica.

Expansión de llave - precálculo

La expansión de llave comienza con la recepción de la llave secreta de 128 bits a través de la entrada *data_i*, junto con un pulso de nivel alto en la entrada *key_valid*, indicando la presencia de la llave en el bus de datos, tal como se observa en la Figura 5.8.

En primer lugar, la presencia de una llave en el bus de datos produce una evolución en la FSM de control, cuyo estado pasa de *IDLE* a *KEY_EXPANSION*. Este bloque de control verifica que está listo para recibir una nueva llave y se encarga de enviar la llave presente en *data_in* y el pulso *key_valid* al bloque de expansión de llave.

Una vez que el bloque de expansión recibe una nueva llave secreta válida, reinicia el valor del contador de iteraciones *ronda* y comienza a calcular una llave de ronda por cada ciclo de reloj, utilizando el algoritmo de expansión detallado en secciones anteriores. En la Figura 5.8 se observa que el bloque envía tres señales hacia los bloques de cifrado y control:

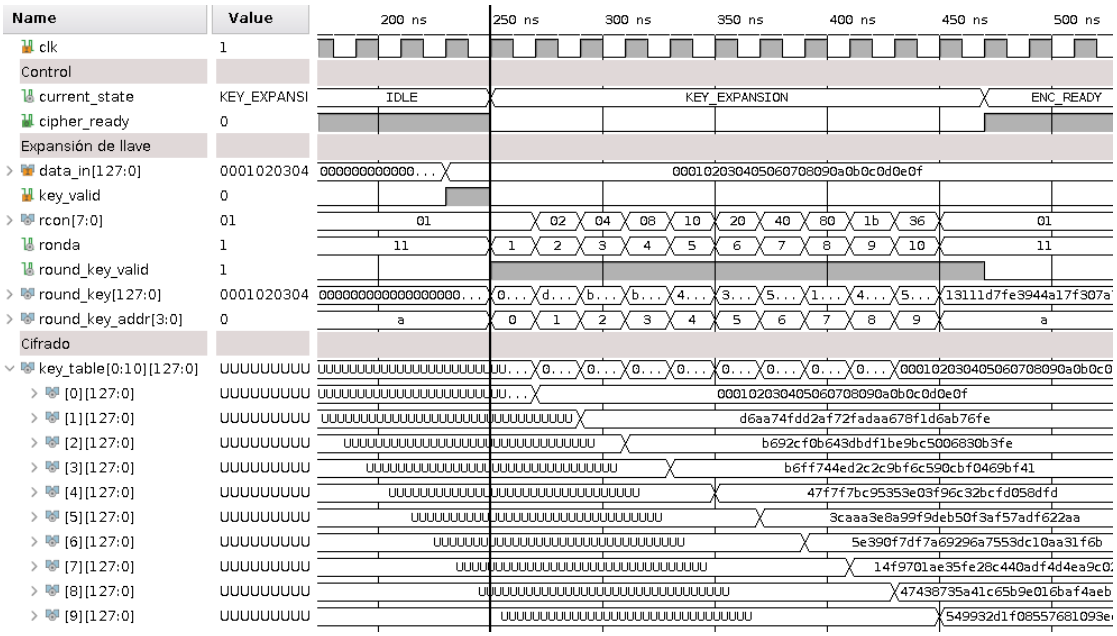


FIGURA 5.8: Expansión de llave para la arquitectura *básica*.

- *round_key*: salida de 128 bits que corresponde al valor de las llaves de ronda.
- *round_key_valid*: salida de un bit que indica con un nivel lógico alto que los datos presentes en el bus *round_key* son válidos.
- *round_key_addr*: salida de 4 bits identifica a cada llave presente en el bus *round_key*.

El bloque de cifrado lee estas tres señales y las utiliza para almacenar las once llaves de ronda en el arreglo *key_table*. Por otro lado, el bloque de control lee el valor de las señales *round_key_valid* y *round_key_addr* para determinar el momento en el que se envía la última llave de ronda. Ante esta situación su FSM evoluciona a ENC_READY.

En la Figura 5.8 se observa además que la señal de control de iteraciones *ronda* permanece fija con valor once en los momentos en los que el bloque de expansión está ocioso y se reinicia a uno cuando recibe una llave. Su valor se incrementa en los ciclos de reloj en los cuales es menor a once, controlando de esta forma la cantidad de iteraciones. También se observa que la salida *cipher_ready*, generada por el bloque de control, tiene un nivel lógico alto una vez que el proceso de expansión finaliza, indicando que el módulo AES está listo para recibir un bloque de texto plano para ser cifrado. El tiempo requerido por la arquitectura *básica* para realizar el proceso de expansión de llave es de once ciclos de reloj.

Cifrado

Una vez que finaliza el proceso de expansión de llave, el bloque AES indica mediante un nivel lógico alto en la señal *cipher_ready* que está listo para recibir un nuevo dato. Al identificar esta situación, el maestro envía a través del puerto *data_in* uno de los bloques de texto plano que obtiene del archivo de texto correspondiente, junto con un pulso en la entrada *plain_text_valid* para indicar que se trata de un dato válido, tal como se ilustra en la Figura 5.9.

La identificación de un dato válido en la entrada provoca la evolución de la FSM de control al estado *ENCRYPTION*. En este estado, la salida *cipher_ready* regresa a nivel lógico bajo, indicando que el módulo de cifrado está ocupado y no puede recibir un nuevo dato en su entrada.

El bloque de cifrado registra el bloque de texto plano cuando detecta un nivel alto en su entrada *plain_valid* y comienza a operar con él. En primer lugar le aplica la transformación de ronda inicial (XOR con la llave secreta). Al resultado de esta operación se le aplican las transformaciones de la ronda general (SubBytes + ShiftRows + MixColumns + AddRoundKey) en forma iterada, una vez por cada ciclo de reloj. Las señales *round_input* y *round_out* corresponden a la entrada y salida de cada iteración de ronda. En la Figura 5.9 se observa que la entrada a cada ronda corresponde a la salida de la ronda anterior, salvo la primer ronda para la cual la entrada corresponde al bloque de texto plano.

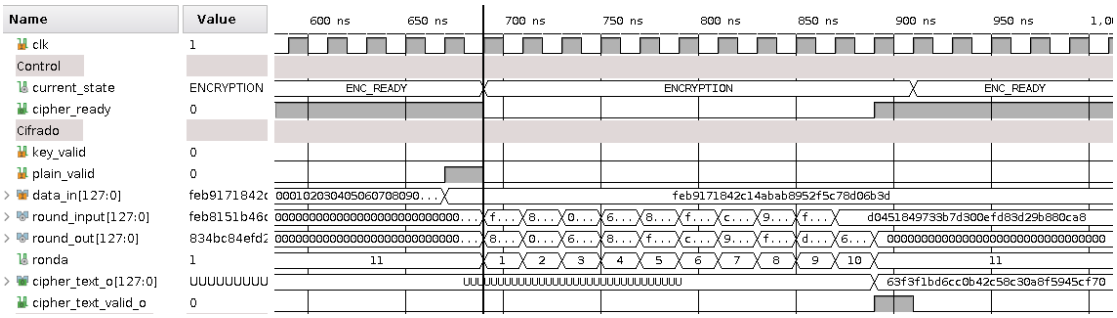


FIGURA 5.9: Cifrado para la arquitectura básica.

En forma análoga al bloque de expansión, la señal de control de iteraciones *ronda* permanece fija con valor once en los momentos en los que el bloque de cifrado está ocioso y se reinicia a uno cuando recibe un bloque de texto plano. Su valor se incrementa en los

ciclos de reloj en los cuales su valor es menor a once. Esta señal tiene distintas finalidades en el módulo de cifrado:

- Se utiliza como índice para acceder a la tabla de llaves de ronda *key_table*.
- Controla la cantidad de veces que el dato a la salida de las transformaciones de ronda es realimentado a la entrada (cantidad de iteraciones).
- Permite identificar la última ronda, en la que la transformación MixColumns debe ser omitida y el valor de texto cifrado se envía al exterior.

Una vez que se llevaron a cabo las diez rondas, el bloque de datos de 128 bits obtenido se envía a través del puerto *cipher_text* junto con un pulso en la señal *cipher_text_valid* que indica la presencia de un dato válido a la salida, este pulso tiene una duración de un ciclo de reloj.

La Figura 5.10 ilustra el proceso de cifrado para la implementación de la arquitectura básica en la cual se realiza la expansión de llave en paralelo al cifrado. En la misma se observa que en este caso la FSM evoluciona directamente al estado ENC_READY cuando detecta un valor alto en la señal de entrada *key_valid*. Por otro lado, la entrada *round_key* (salida del bloque de expansión de llave) se actualiza en cada ciclo de reloj del cifrado.

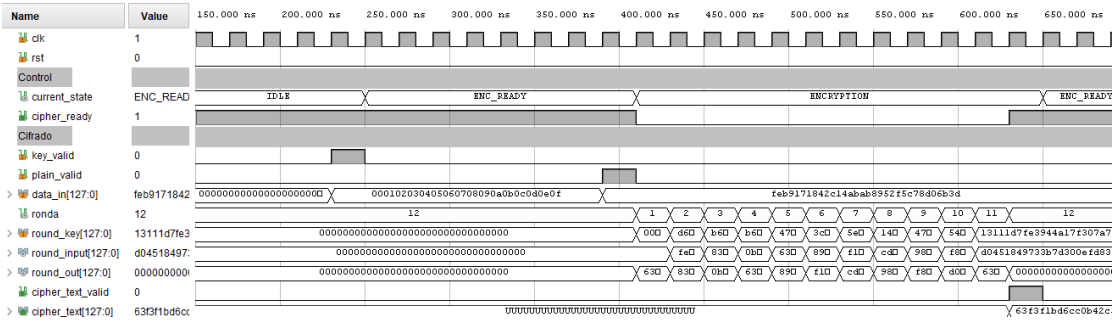


FIGURA 5.10: Cifrado para la arquitectura básica con expansión de llave *on-the-fly*.

En ambos casos, Figuras 5.9 y 5.10, se aprecia que el tiempo requerido por la arquitectura *básica* para procesar un bloque de texto plano (128 bits) es de once ciclos de reloj.

Interfaz AMBA AHB

La interfaz AMBA se implementó mediante un envoltorio que puede ser agregado a los módulo de cifrado en forma opcional. Este envoltorio contiene doce registros de 32 bits:

cuatro de ellos corresponden a la llave secreta, cuatro al bloque de texto plano y cuatro al bloque de texto cifrado. De esta manera, el maestro AHB se comunica con el módulo de cifrado mediante la escritura y lectura de los registros.

El banco de prueba o *Test Bench* en este caso es similar al presentado en las secciones anteriores, salvo que se modificó la estructura del maestro para que se comunique mediante una interfaz AHB. El maestro envía cada dato de 128 bits (llave o texto plano) mediante la escritura de cuatro registros internos del envoltorio, de 32 bits cada uno.

En primer lugar, el maestro AHB envía la llave secreta (se utilizó la misma que en las pruebas anteriores porque fue la utilizada en el modelo de referencia) y luego comienza a enviar de a uno los bloques de texto plano que obtiene del archivo de entradas. Por otro lado, en el banco de prueba se comparan los resultados obtenidos por el bloque de cifrado con los valores almacenados en el archivo de bloques de texto cifrado, tal como se hacía en las pruebas anteriores.

La Figura 5.11 muestra la comunicación entre el maestro AHB y la arquitectura *básica* durante la expansión de llave. En la misma se observa que el maestro escribe los cuatro registros correspondientes a la llave secreta (*reg_key_3*, *reg_key_2*, *reg_key_1* y *reg_key_0*), colocando sus direcciones en el bus HADDR y solicitando la escritura mediante un nivel alto en HSEL y HWRITE. Dado que la llave secreta junto con su señal de habilitación se envían al módulo de cifrado una vez que se escribe el registro *reg_key_0*, es imprescindible que los tres registros restantes hayan sido escritos en forma previa.

Se observa además que la escritura de los registros cumple con el temporizado definido en el estándar AHB:

- En el primer ciclo de reloj de una transferencia, denominado *fase de dirección*, se establece la dirección del registro a escribir en el bus HADDR y las señales de control HWRITE, HSEL, HSIZE y HTRANS.
- En el ciclo siguiente, denominado *fase de dato*, se envía el dato a ser escrito en el registro definido en la fase anterior. La señal de salida HREADYOUT se utiliza para extender la fase de datos de una operación en caso de ser necesario, indicando que el esclavo necesita un tiempo adicional para completarla.

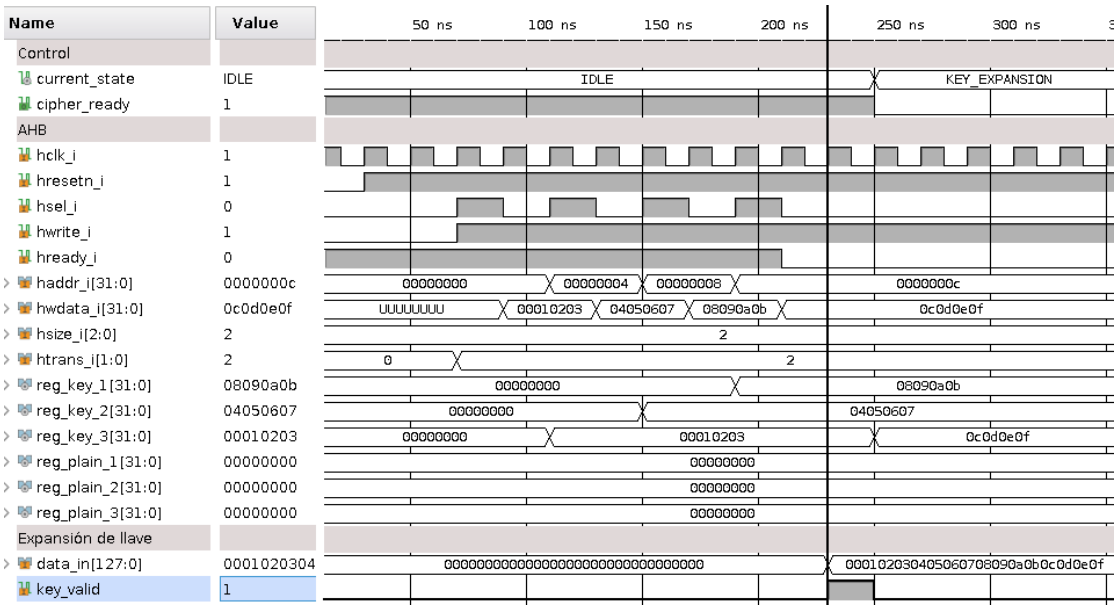


FIGURA 5.11: Expansión de llave mediante interfaz AMBA.

El protocolo AHB admite simultaneidad entre la fase de datos de una transacción y la fase de direcciones de la siguiente (denominado *pipeline* entre transacciones). En este caso el maestro implementado no hace este tipo de paralelismo, motivo por el cual la señal HSEL oscila entre nivel lógico alto y bajo, indicando que en la fase de datos no se recibe una dirección válida para la próxima transacción. De todas formas, el envoltorio admite ambos tipos de transferencias, con paralelismo y sin paralelismo.

En el envoltorio propuesto se utiliza la señal *HREADYOUT* para indicar al maestro AHB que el módulo de cifrado está ocupado durante la expansión de llave y no puede recibir nuevos datos de entrada. A esta señal se le asigna un valor lógico bajo en la fase de datos de la escritura del registro *reg_key_0* y regresa a nivel alto cuando se detecta un flanco ascendente en la señal *cipher_ready*, es decir, cuando finaliza la exposición y el módulo de cifrado está listo para recibir un nuevo dato.

Una vez que el módulo de cifrado vuelve a estar listo, el maestro AHB comienza a enviar bloques de texto plano mediante la escritura de los registros *reg_plain_3*, *reg_plain_2*, *reg_plain_1* y *reg_plain_0*. Esta operación se lleva a cabo colocando sus direcciones en el bus HADDR y solicitando la escritura mediante un nivel alto en HWRITE y HSEL, tal como se ilustra en la Figura 5.12. El bloque de texto plano junto con su señal de habilitación se envían al módulo de cifrado una vez que se escribe el registro *reg_plain_0*, por lo que es imprescindible que los tres registros restantes hayan sido escritos en forma previa.

En forma análoga a la expansión de llave, se utiliza la señal *HREADYOUT* para indicar al maestro AHB que el módulo está ocupado durante el cifrado del bloque de texto plano ingresado y no puede recibir nuevos datos de entrada. A esta señal se le asigna un valor lógico bajo en la fase de datos de la escritura del registro *reg_plain_0* y regresa a nivel alto cuando se detecta un flanco ascendente en la señal *cipher_ready*, es decir, cuando finaliza el cifrado y el módulo está listo para recibir un nuevo dato.

Una vez finalizado el proceso de cifrado, la señal *cipher_text_valid* se utiliza como habilitación para escribir en forma simultánea los cuatro registros de texto cifrado (*reg_ciph_3*, *reg_ciph_2*, *reg_ciph_1* y *reg_ciph_0*). Esta operación se ilustra en la Figura 5.12.

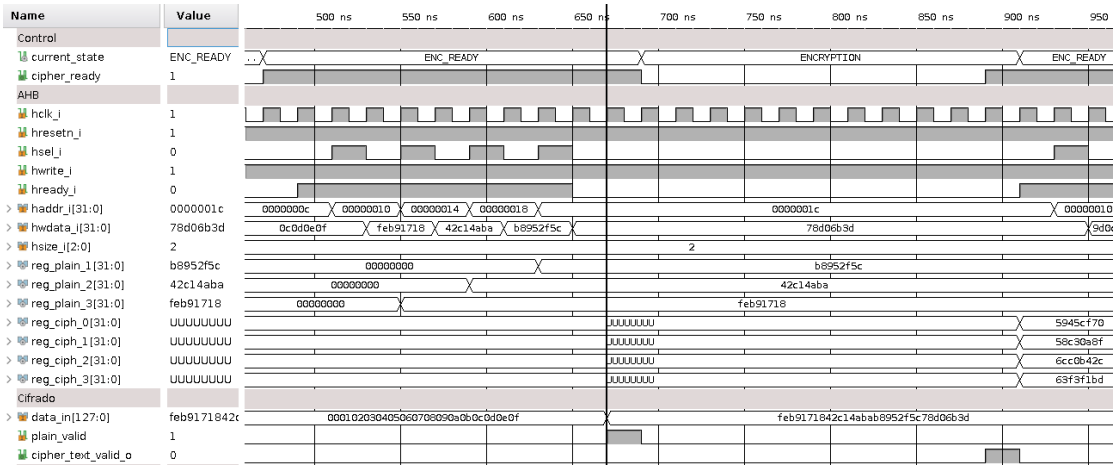


FIGURA 5.12: Cifrado mediante interfaz AMBA.

En las Figuras 5.11 y 5.12 se observa que los registros menos significativos de los datos de entrada (*reg_key_0* y *reg_plain_0*) no se almacenan en forma interna en el envoltorio. Esto se debe a que en los ciclos de reloj en los que se deberían escribir estos registros se envía el dato de 128 bits al módulo de expansión o cifrado. El dato que se envía es la concatenación entre los tres registros almacenados y el dato presente en el bus de datos HWDATA. De esta manera se ahorran recursos y tiempo, ya que no es necesario esperar a que los cuatro registros estén escritos para poder enviar el dato completo al bloque de cifrado.

$$data\_i = reg\_key\_3 \,\&\, reg\_key\_2 \,\&\, reg\_key\_1 \,\&\, HWDATA$$

$$data\_i = reg\_plain\_3 \,\&\, reg\_plain\_2 \,\&\, reg\_plain\_1 \,\&\, HWDATA$$

Assertions de la interfaz AMBA AHB

Las *assertions* incluidas en la arquitectura básica chequean las siguientes características del envoltorio AHB:

- Ante una transferencia de tipo IDLE debe ignorar las señales de control y proveer una respuesta de tipo exitosa y sin ciclos de espera.
- Ante una transferencia de tipo BUSY debe ignorar las señales de control y proveer una respuesta de tipo exitosa y sin ciclos de espera.
- Tanto la operación de cifrado como la expansión de llave insertan ciclos de espera en las transferencias. Esto implica que HREADYOUT debe irse a 0 en la fase de datos al escribir el registro en el que se almacena el *word* menos significativo de llave y texto plano.
- El tiempo en el que la señal HREADYOUT puede estar en 0 tiene una cota máxima, dada por la latencia de la arquitectura básica.
- Es necesario que HSIZE esté configurado en forma acorde. Dado que el tamaño de los registros es de 32 bits esto implica que HSIZE debe ser 010.
- Los registros en los que se almacena el valor de texto cifrado son de solo lectura. En caso que el master intente escribirlos, el esclavo indica una condición de error en su salida HRESP.

5.4. Latencia

La siguiente tabla resume los valores de latencia para los procesos de expansión de llave y cifrado en la arquitectura básica.

Tabla 5.5: Valores de latencia para la arquitectura básica.

Proceso	Ciclos de reloj
Expansión de llave	11
Cifrado	11

Capítulo 6

Desarrollo de la arquitectura pipeline

El diseño de la arquitectura pipeline prioriza obtener un mayor desempeño. Como se mencionó con anterioridad, esta arquitectura es apropiada para aplicaciones más exigentes, como Wi-Fi, en las que se requiere el cifrado de grandes volúmenes de datos con un desempeño alto. La estructura contiene tres bloques: control, cifrado y expansión de llave. El bloque de cifrado consta de la implementación en hardware de las once rondas requeridas para cifrar un bloque de texto plano, como se ilustra en la Figura 6.1. En la misma los registros se representan con líneas oscuras.

Cada bloque de texto plano de 128 bits que ingresa al bloque de cifrado atraviesa la lógica que implementa la ronda inicial y el resultado de esta operación ingresa a la primer ronda general. Durante el proceso de cifrado, en cada flanco ascendente del reloj se actualizan las entradas de cada instancia de la ronda general, de manera que el resultado de una ronda se convierte en entrada de la ronda siguiente. Cada ronda recibe datos solo en los ciclos de reloj en los que la ronda anterior tiene un salida válida. De esta manera es posible ingresar al módulo de cifrado un bloque de texto plano por ciclo de reloj, sin necesidad de esperar a que finalice el cifrado de los bloques anteriores.

El desempeño de esta arquitectura es máximo cuando se ingresa un bloque de texto plano por cada ciclo de reloj y la estructura interna del bloque plano está completa, es decir, todas las instancias de las rondas están operando en paralelo. La Tabla 5.1 presenta la descripción de la interfaz de esta arquitectura. En la misma se observa que

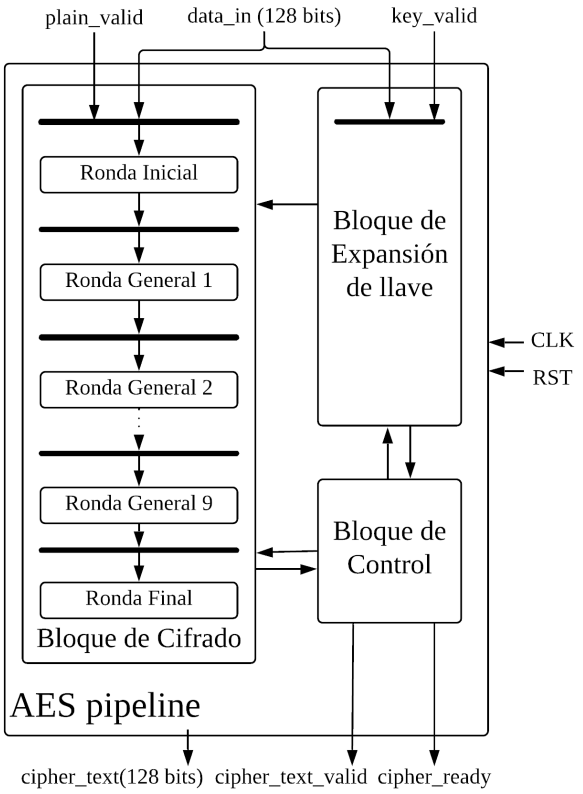


FIGURA 6.1: Diagrama en bloques de la arquitectura pipeline.

los buses de datos internos y hacia el exterior son de 128 bits, de manera que en cada ciclo de reloj se procesa la matriz de estados completa.

Con respecto al ingreso de datos, se utiliza un solo bus para disminuir la cantidad de puertos de entrada. Mediante dos señales de control *plain_valid* y *key_valid* se indica si el dato ingresado corresponde a un bloque de texto plano o a la llave secreta. Internamente estas dos señales se utilizan como habilitación para dos registros de 128 bits, uno que almacena el texto plano y es entrada a la ronda inicial en el bloque de cifrado y otro que almacena la llave secreta en el bloque de expansión de llave.

Con respecto a las salidas, *cipher_text* contiene el resultado del bloque cifrado y su valor es válido en los ciclos de reloj en los cuales *cipher_text_valid* posee un nivel lógico alto. Por otro lado, *cipher_ready* posee un valor lógico alto en los ciclos de reloj en los que la arquitectura está lista para recibir un nuevo bloque de texto plano.

TABLA 6.1: Descripción de entradas y salidas de la arquitectura pipeline

Puerto	Sentido	Tamaño	Descripción
CLK	entrada	1 bit	Reloj
RST	entrada	1 bit	Reset sincrónico activo con nivel lógico alto
data_in	entrada	128 bits	Puerto mediante el cual se ingresa la llave secreta o un bloque de texto plano
plain_valid	entrada	1 bit	Mediante un nivel lógico alto indica que el puerto *data_in* contiene un bloque de texto plano válido
key_valid	entrada	1 bit	Mediante un nivel lógico alto indica que el puerto *data_in* contiene una llave secreta válida
cipher_text	salida	128 bits	Puerto mediante el cual el módulo AES entrega un bloque de texto cifrado
cipher_text_valid	salida	1 bit	Mediante un nivel lógico alto indica que el puerto *cipher_text* contiene un bloque de texto cifrado válido
cipher_ready	salida	1 bit	Mediante un nivel lógico alto indica que el módulo AES está listo para recibir un nuevo bloque de datos, ya sea texto plano o llave

6.1. Cifrado, control y expansión de llave

Diseño del bloque de cifrado

El diseño del bloque de cifrado para esta arquitectura se realizó siguiendo un esquema de manejo de iteraciones pipeline, en el cual se admite la ejecución de todas las rondas requeridas para el cifrado en simultáneo, cada una para un bloque de texto plano diferente. En este esquema se instancia la lógica correspondiente a una ronda inicial (XOR entre el bloque de texto plano y la llave secreta), seguido de nueve instancias de la la lógica correspondiente a una ronda general (SubBytes + ShiftRows + MixColumns + AddRoundKey) y una instancia de la ronda final (SubBytes + ShiftRows + AddRoundKey), tal como se ilustra en la Figura 6.2. El desplazamiento de los datos para conectar la salida de una etapa a la entrada de la etapa siguiente se realiza a través de registros de 128 bits. Estos registros actualizan su valor en los ciclos de reloj en los cuales hay un dato válido a la salida de la ronda anterior.

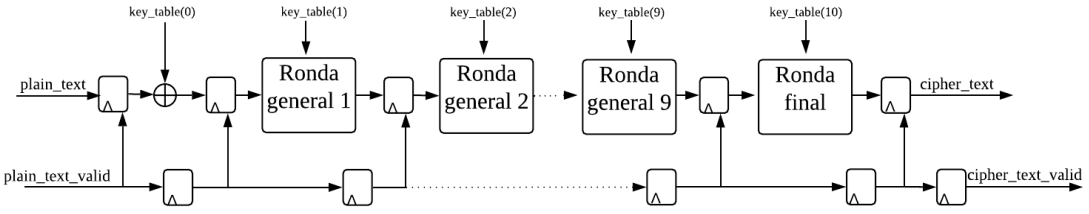


FIGURA 6.2: Implementación del bloque de cifrado en la arquitectura pipeline.

En la Figura 6.2 también se observa la presencia de registros de 128 bits a la entrada y a la salida del bloque de cifrado. El registro de entrada se habilita en los ciclos de reloj en los que la señal de entrada *plain_valid* posee un nivel lógico alto, indicando que el dato presente en el bus de entrada *plain_text* es válido.

Por otro lado, la habilitación de los registros que conectan la salida de una ronda con la entrada de la ronda siguiente se generan a partir de una segunda cadena de registros de desplazamiento (*shift registers*). Estos registros desplazan el valor de la entrada *plain_vaid* en cada ciclo de reloj. De esta manera, la señal *plain_vaid* "acompaña" al dato a medida que avanza por las distintas rondas. Los registros que controlan el flujo del dato durante el cifrado conectan la salida de una ronda con la entrada de la ronda siguiente solo en los ciclos de reloj en los que la señal de habilitación proveniente del *shift register* tiene un nivel alto, indicando la presencia de un dato válido en su entrada.

La salida de la última etapa del *shift register* corresponde a la salida *cipher_text_valid*, indicando que el dato presente en la salida del bloque de cifrado es válida. Las salidas *cipher_text* y *cipher_text_valid* tienen una duración de un ciclo de reloj. En los casos en los que se ingresa un bloque de texto plano por cada ciclo de reloj y todas las rondas instanciadas están activas, el desempeño es máximo y se obtienen bloques de texto cifrado a la salida en forma consecutiva, uno por ciclo de reloj.

Con respecto a implementación las transformaciones de ronda, se utilizó el mismo enfoque de la arquitectura básica. Los detalles de diseño de estas implementaciones se describen en el capítulo anterior.

La Figura 6.3 ilustra la manera en la que se implementa la ronda general. En la misma, se identifica con un valor hexadecimal a cada uno de los bytes del dato de 128 bits que ingresa a la ronda. Además, se identifica a las transformaciones de ronda con distintos colores. Cada una de las instancias corresponde a las implementaciones mencionadas:

- Las 16 instancias que componen SubBytes corresponden a 16 LUTs como la mostrada en la Tabla 5.2.
- ShiftRows se implementa mediante la correcta interconexión entre los bytes que se obtienen en cada una de las LUTs y los bytes de entrada a MixColumns.
- MixColumns consta de 4 instancias de la lógica que procesa una columna.

- AddRoundKey consta de 16 instancias de una compuerta XOR bit a bit entre dos bytes.

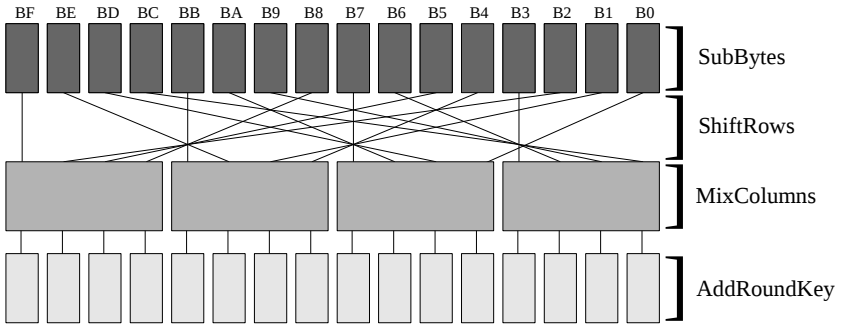


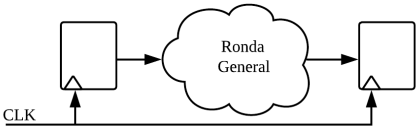
FIGURA 6.3: Esquema de una ronda general para la arquitectura pipeline.

Con el fin de mejorar la velocidad máxima de trabajo del bloque de cifrado se propone también realizar una estructura pipeline dentro de cada ronda. En este esquema es posible aumentar la frecuencia de reloj al disminuir el retardo de la lógica combinacional entre registros, tal como se ilustra en la Figura 6.4.

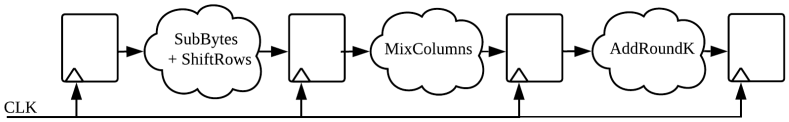
En la Figura 6.4-*a* se ilustra el caso en el que la lógica combinacional entre registros corresponde a una ronda general. En este caso la frecuencia máxima de reloj queda definida por el retardo de la lógica combinacional, que es la suma del retardo de cada una de las transformaciones que la componen: SubBytes, ShiftRows, MixColumns y AddRoundKey.

Por otro lado, si se agregan registros entre las transformaciones como se ilustra en la Figura 6.4-*b* , la frecuencia máxima de reloj queda definida por la transformación cuyo retardo es mayor. La transformación ShiftRows se coloca en un mismo bloque combinacional con otra transformación porque simplemente consiste en el correcto ruteo entre la salida de SubBytes y la entrada de MixColumns y no insume un tiempo significante.

Duarante el proceso de síntesis se puede obtener una aproximación del retardo de cada una de las transformaciones. Un análisis de estos tiempos permite determinar cuál es el mejor esquema de división de las transformaciones.



(a) Pipeline entre rondas



(b) Pipeline entre rondas y dentro de la ronda

FIGURA 6.4: *pipeline* entre rondas y dentro de las rondas.

Bloque de expansión de llave

En la arquitectura pipeline el esquema de expansión de llave adecuado es el precálculo, debido a que en esta arquitectura el bloque de cifrado requiere las once llaves de ronda en forma simultánea. La implementación propuesta para el bloque de expansión es la misma que la presentada para el caso de la arquitectura básica, cuyo diseño se detalla en el capítulo anterior.

Bloque de control

Este bloque se encarga de generar y enviar señales de control hacia el exterior y a los bloques de cifrado y expansión. Su objetivo es asegurar el correcto flujo de la información, ignorando solicitudes en caso que se requiera el cifrado de un nuevo dato mientras se está llevando a cabo la expansión de llave. Además, asegura que el cifrado de un dato comience sí y solo sí la expansión de llave fue llevada a cabo.

El bloque de control se implementa mediante la máquina de estados ilustrada en la Figura 6.5.

- La máquina de estados permanece en estado ocioso IDLE desde que detecta un nivel alto en la entrada *reset* hasta que detecta un nivel alto en la entrada *key_valid*, indicando el ingreso de la llave secreta mediante el bus de datos. Una vez que se detecta el ingreso de la llave, la máquina de estados evoluciona a KEY_EXPANSION.
- La máquina de estados permanece en estado KEY_EXPANSION durante el transcurso de tiempo en el cual el bloque de expansión de llave genera las once llaves

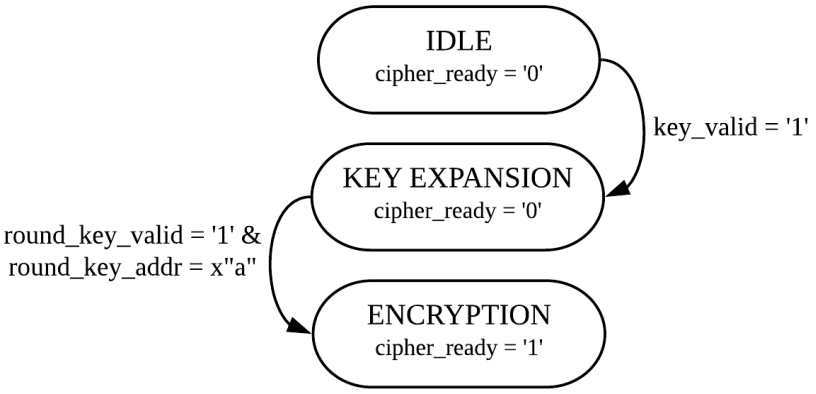


FIGURA 6.5: Máquina de estados en el sub-bloque de control para la arquitectura pipeline.

de ronda. Una vez que se genera la última llave de ronda (situación identificada a partir del valor de dirección de llave generada por el bloque de expansión) la máquina de estados de control pasa al estado ENCRYPTION.

- En estado ENCRYPTION, el bloque de cifrado está listo para recibir nuevos bloques de texto plano. Debido a la estructura interna del bloque de cifrado, en la arquitectura pipeline es posible ingresar un bloque de texto plano por cada ciclo de reloj. Por este motivo, la salida *cipher_ready* se mantiene en nivel alto en este estado.

- El ingreso de una nueva llave secreta puede realizarse solo en el estado IDLE. Si se recibe una llave en otro momento se la ignora. Esto implica que una vez que la máquina de estados está en el estado ENCRYPTION, es necesario enviar un reset general para hacer que regrese a estado IDLE y poder modificar la llave. Esta decisión de diseño se basa en que, con el objetivo de evitar resultados de cifrado erróneos, el pipeline del bloque de cifrado debe estar vacío al momento de modificar el valor de la llave secreta.

- El ingreso de un nuevo bloque de texto plano puede realizarse en el estado ENCRYPION. Si se recibe un bloque de texto plano en otro momento se lo ignora.

- El dato enviado a los bloques de expansión de llave y cifrado es el mismo y se conecta directamente con el bus de datos de entrada *data_in*. Por otro lado, las señales de habilitación *key_valid* y *plain_valid* se envían solo en los estados IDLE o ENCRYPION. La señal *key_valid* que recibe el bloque de expansión se adapta en el bloque de control para que tenga una duración de un ciclo de reloj. Por otro lado, *plain_valid* se envía al bloque de cifrado tal y como ingresa, ya que en esta

arquitectura puede tener una duración mayor, indicando que se ingresan varios bloques de texto plano en forma consecutiva, uno por cada ciclo de reloj.

6.2. Interfaz AHB

Para la arquitectura pipeline no es posible utilizar buses de datos de 32 bits como se hizo para la arquitectura básica, ya que en ese caso se podría ingresar como máximo un bloque de texto plano cada cuatro ciclos de reloj, siendo poco eficiente y desaprovechando los beneficios de la arquitectura pipeline.

Por este motivo, para la arquitectura pipeline se considera que los buses de datos HWDATA y HRDATA son de 128 bits, de manera que el módulo de cifrado pueda recibir un bloque de texto plano por cada ciclo de reloj y así poder utilizar eficientemente su estructura interna. De esta manera, el envoltorio realiza una conexión directa entre los buses de datos de AMBA AHB y los buses de datos del módulo de cifrado AES; mientras que las señales de habilitación que indican la presencia de un dato válido se generan en el envoltorio a partir de las señales de control del protocolo AMBA. El envío de la llave secreta y los bloques de texto plano a través de la interfaz AHB se realiza a través de la escritura de dos registros de 128 bits, cuya dirección se especifica en la Tabla 6.2

TABLA 6.2: Registros definidos en el envoltorio AHB-Lite para la arquitectura pipeline.

Nombre	Dato	Dirección
reg_key	Key[127:0]	x0000001C
reg_plain	Plain_text[127:0]	x0000000C

Debido a su estructura interna, la arquitectura pipeline obtiene un dato válido a su salida por cada ciclo de reloj siempre que su estructura interna esté completa. Bajo este esquema resulta imposible almacenar los bloques de texto cifrado en registros (como se hizo la arquitectura básica), debido a que un solo ciclo de reloj no es suficiente tiempo para que el maestro pueda leer este registro siguiendo el protocolo de lectura AHB. Una alternativa que soluciona este inconveniente es el almacenamiento de los bloques de texto cifrado en una estructura de memoria, como podría ser una FIFO, cuya habilitación de escritura está conectada a la señal *cipher_text_valid* y que pueda ser accedida posteriormente por el Maestro AHB.

Con la finalidad de ingresar una nueva llave secreta al módulo de cifrado, el Maestro AHB escribe el registro de llave siguiendo el protocolo de comunicación especificado en el estándar y detallado en el Apéndice B de este trabajo. El Maestro envía la dirección del registro a escribir a través del bus de direcciones HADDR, junto con un nivel alto en la señal de escritura HWRITE y en la señal de selección HSEL (esta última se genera a partir de la dirección y tiene un nivel alto cuando el maestro se comunica con el módulo de cifrado). El envoltorio lee estas señales y activa la señal de habilitación que indica la presencia de una llave secreta válida (*key_valid*) al detectar que el maestro está solicitando la escritura del registro de llave. El envoltorio conecta directamente la entrada *data_in* del bloque de cifrado al bus de datos HWDATA.

Con la finalidad de ingresar un bloque de texto plano al módulo de cifrado, el Maestro AHB escribe el registro de texto plano siguiendo el protocolo de comunicación especificado en el estándar y detallado en el Apéndice B de este trabajo. El Maestro envía la dirección del registro a escribir a través del bus de direcciones HADDR, junto con un nivel alto en la señal de escritura HWRITE y en la señal de selección HSEL (esta última se genera a partir de la dirección y tiene un nivel alto cuando el maestro se comunica con el módulo de cifrado). El envoltorio lee estas señales y activa la señal de habilitación que indica la presencia de un bloque de texto plano válido (*plain_valid*) cuando detecta que el maestro está solicitando la escritura del registro de texto plano. El envoltorio conecta directamente la entrada *data_in* del bloque de cifrado al bus de datos HWDATA. Esto implica que en este caso se envía directamente el dato y la señal que indica su validez, sin necesidad de almacenarlo internamente en el envoltorio, como sucede en la arquitectura básica.

Durante el proceso de expansión de llave la señal de salida HREADYOUT del envoltorio se mantiene en nivel lógico bajo, indicando al maestro AHB que el módulo de cifrado está ocupado. En la arquitectura pipeline la señal HREADYOUT se mantiene en nivel alto durante el proceso de cifrado, indicando que el bloque de cifrado está listo para recibir un nuevo bloque de texto plano.

6.3. Verificación funcional

Utilizando la herramienta XSim, incorporada en el entorno de Vivado, se simularon tanto los tests dirigidos como el test aleatorio de 500 muestras. La totalidad de los tests corridos pasaron, obteniendo resultados favorables.

A continuación se presentan formas de onda que ilustran el proceso de expansión de llave y cifrado para la arquitectura pipeline.

Expansión de llave

La expansión de llave comienza con la recepción de la llave secreta de 128 bits a través de la entrada *data_i*, junto con un pulso de nivel alto en la entrada *key_valid*, indicando la presencia de la llave en el bus de datos, tal como se observa en la Figura 6.6.

En primer lugar, la presencia de una llave en el bus de datos produce un cambio en la FSM de control, que evoluciona de *IDLE* a *KEY_EXPANSION*. Este bloque de control verifica que está listo para recibir una nueva llave y se encarga de enviar la llave presente en *data_in* y el pulso *key_valid* al bloque de expansión de llave.

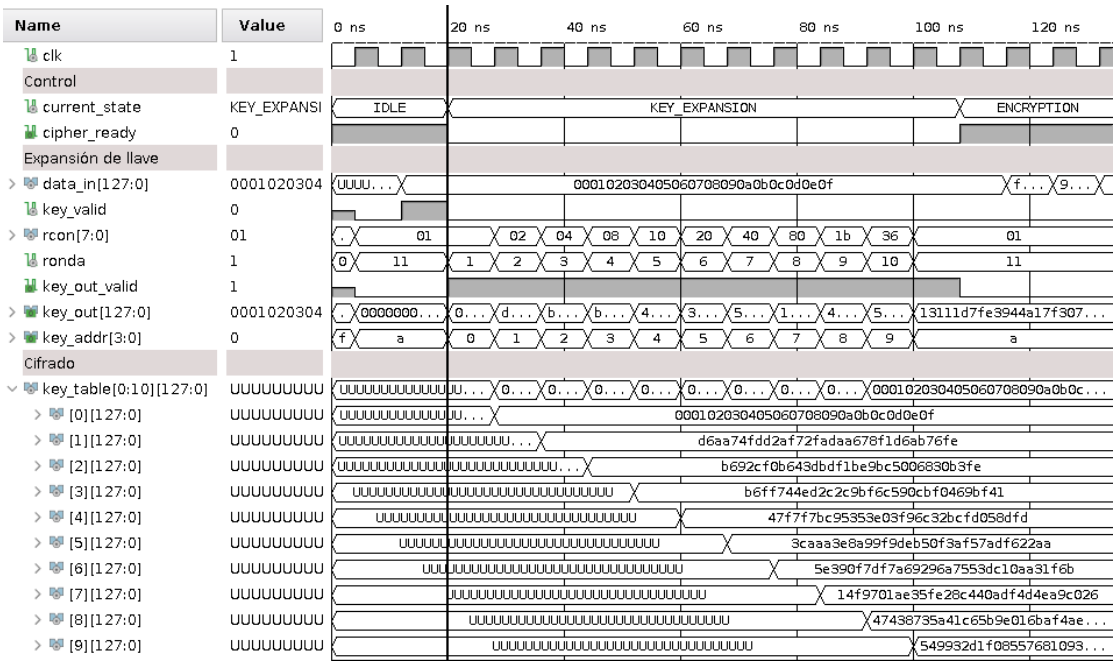


FIGURA 6.6: Expansión de llave para la arquitectura pipeline.

Una vez que el bloque de expansión recibe una nueva llave secreta válida, reinicia el valor del contador de iteraciones *ronda* y comienza a calcular una llave de ronda por

cada ciclo de reloj, utilizando el algoritmo de expansión. En la Figura se observa que el bloque envía tres señales hacia los bloques de cifrado y control:

- *key_outy*: valor de las llaves de ronda.
- *key_out_valid*: salida de un bit que indica con un nivel lógico alto que los datos presentes en el bus *round_key* son válidos.
- *key_addr*: salida de cuatro bits identifica a cada llave presente en el bus *round_key*.

El bloque de cifrado lee estas tres señales y las utiliza para almacenar las once llaves de ronda en el arreglo *key_table*. Por otro lado, el bloque de control lee el valor de las señales *round_key_valid* y *round_key_addr* para determinar el momento en el que se envía la última llave de ronda. Ante esta situación su FSM evoluciona a ENCRYPTION.

En la Figura 6.6 se observa además que la señal de control de iteraciones *ronda* permanece fija con valor once en los momentos en los que el bloque de expansión está ocioso y se reinicia a uno cuando recibe una llave. Su valor se incrementa en uno en los ciclos de reloj en los cuales su valor es menor a once, controlando de esta forma la cantidad de iteraciones. También se observa que la salida *cipher_ready*, generada por el bloque de control, tiene un nivel lógico alto una vez que el proceso de expansión finaliza, indicando que el módulo AES está listo para recibir un bloque de texto plano para ser cifrado. El tiempo requerido por la arquitectura *pipeline* para realizar el proceso de expansión de llave es de once ciclos de reloj.

Cifrado

Una vez que finaliza el proceso de expansión de llave, el bloque AES indica que está listo para recibir un nuevo dato mediante un nivel lógico alto en la señal *cipher_ready*. Al identificar esta situación, el maestro envía a través del puerto *data_in* uno de los bloques de texto plano que obtiene del archivo de texto correspondiente, junto con un pulso en la entrada *plain_text_valid* para indicar que se trata de un dato válido, tal como se ilustra en la Figura 6.7.

La identificación de un dato válido en la entrada provoca la evolución de la FSM de control al estado *ENCRYPTION*. En este estado, la salida *cipher_ready* posee un nivel lógico alto, indicando que el módulo de cifrado puede recibir un nuevo dato en su entrada.

Esto hace que el maestro continúe enviando un bloque de texto plano válido por cada ciclo de reloj, hasta alcanzar el final del archivo de texto.

El bloque de cifrado registra cada bloque de texto plano al detectar un nivel alto en su entrada *plain_valid* y comienza a operar con él. En el primer ciclo de reloj le aplica la transformación de ronda inicial (XOR con la llave secreta). Al resultado de esta operación se le aplican las transformaciones de las nueve rondas generales (SubBytes + ShiftRows + MixColumns + AddRoundKey), una por cada ciclo de reloj. Las señales *round_input* y *round_out* corresponden a la entrada y salida de cada ronda. En la Figura 6.7 se observa que la entrada a cada ronda corresponde a la salida de la ronda anterior y se actualiza en los ciclos de reloj en los que la etapa correspondiente del *shift_register input_valid_sr* tiene un valor lógico alto.

Once ciclos de reloj después del envío del primer bloque de texto plano, su procesamiento finaliza y en el puerto de salida *cipher_text* se presenta el primer bloque cifrado, junto con la señal de habilitación *cipher_text_valid*. A partir de ese momento, a la salida se obtiene un bloque de texto cifrado por cada ciclo de reloj. De esta manera la arquitectura pipeline tiene la misma latencia que la arquitectura básica, pero su desempeño es diez veces mayor, siempre y cuando se le ingrese un bloque de texto plano por cada ciclo de reloj. Esto implica que, la arquitectura pipeline es adecuada para aplicaciones en las que se cifran múltiples datos y en forma consecutiva, ya que si se le envían bloques de texto plano en forma individual y esporádica su desempeño no presenta mejoras.

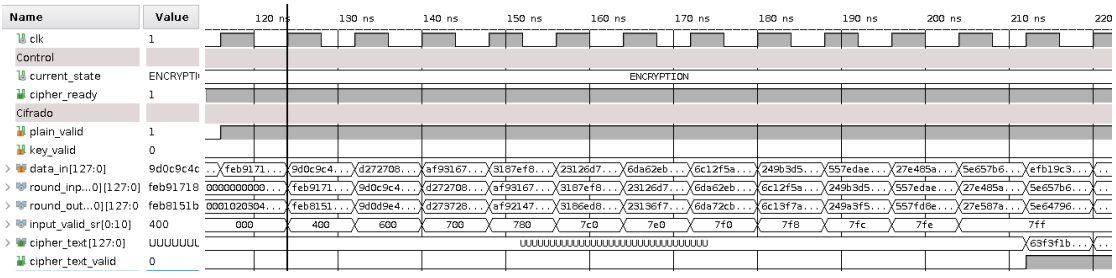


FIGURA 6.7: Cifrado en arquitectura *pipeline*.

Interfaz AMBA AHB

El envoltorio AHB para la arquitectura pipeline cuenta con buses de datos de 128 bits, con el fin de admitir la escritura de un bloque de datos por ciclo de reloj y así hacer un uso óptimo de la estructura paralela que ofrece esta arquitectura.

El banco de prueba o *Test Bench* en este caso es similar al presentado en las secciones anteriores, salvo que se modificó la estructura del maestro para que se comunique mediante una interfaz AHB. El maestro envía cada dato de 128 bits (llave o texto plano) mediante la escritura un registro.

En primer lugar, el maestro AHB envía la llave secreta (se utilizó la misma que en las pruebas anteriores porque fue la utilizada en el modelo de referencia), y luego comienza a enviar de a uno los bloques de texto plano que obtiene del archivo de entradas. Por otro lado, en el banco de prueba se comparan los resultados obtenidos por el bloque de cifrado con los valores almacenados en el archivo de bloques de texto cifrado, tal como se realizó en las pruebas anteriores.

La Figura 6.8 muestra la comunicación entre el maestro AHB y la arquitectura *pipeline* durante la expansión de llave. En la misma se observa que el maestro escribe el registro correspondientes a la llave secreta, colocando su dirección en el bus HADDR y solicitando la escritura mediante un nivel alto en HSEL y HWRITE.

Se observa además que la escritura del registro cumple con el temporizado definido en el estándar AHB. Este protocolo admite simultaneidad entre la fase de datos de una transacción y la fase de direcciones de la siguiente (denominado pipeline entre transacciones). En este caso el maestro implementado hace este tipo de paralelismo y por eso la señal HSEL se mantiene en nivel lógico alto, indicando que en la fase de datos se recibe una dirección válida para la próxima transacción. El envoltorio admite ambos tipos de transferencias, con paralelismo y sin paralelismo.

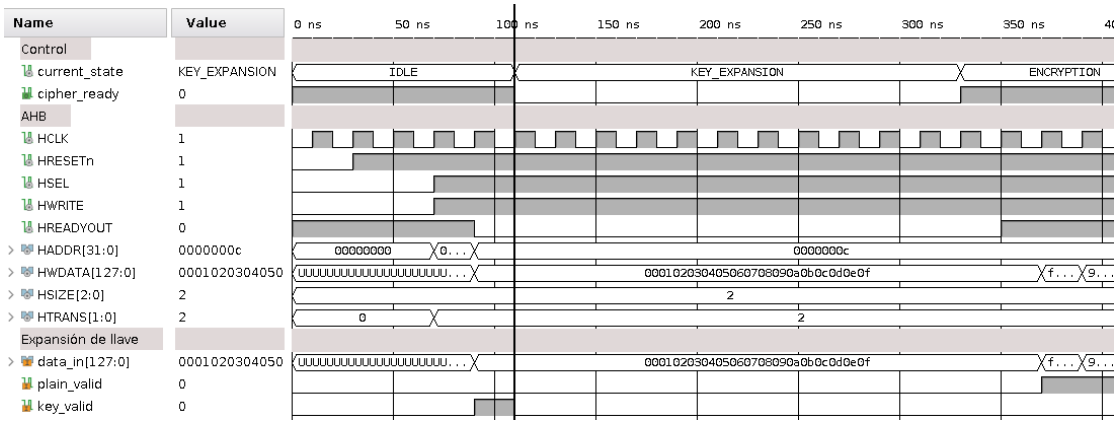


FIGURA 6.8: Expansión de llave mediante AMBA en la arquitectura pipeline.

En el envoltorio propuesto se utiliza la señal HREADYOUT para indicar al maestro AHB que el módulo de cifrado está ocupado durante la expansión de llave y no puede

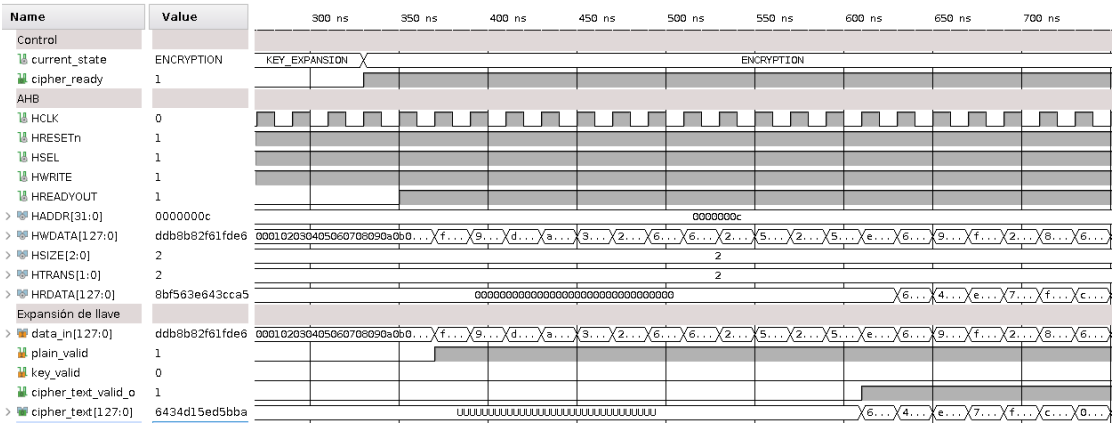


FIGURA 6.9: Cifrado mediante AMBA en la arquitectura pipeline.

recibir nuevos datos de entrada. Esta señal baja en la fase de datos de la escritura del registro *reg_key* y regresa a nivel alto cuando se detecta un flanco ascendente en la señal *cipher_ready*, es decir, cuando finaliza la expansión y el módulo de cifrado está listo para recibir bloques de texto plano.

Una vez que el módulo de cifrado vuelve a estar listo, el maestro AHB comienza a enviar bloques de texto plano mediante la escritura del registro de 128 bits *reg_plain*. Esta operación se lleva a cabo colocando sus direcciones en el bus HADDR y solicitando la escritura mediante un nivel alto en HWRITE y HSEL, tal como se ilustra en la Figura 6.9.

Debido a que el módulo de cifrado puede recibir un bloque de texto plano por cada ciclo de reloj, el envoltorio mantiene la salida HREADYOUT en nivel lógico alto durante el proceso de cifrado.

Pipeline intra e inter rondas

Se simuló también la arquitectura pipeline agregando un registro adicional dentro de las rondas generales. De esta forma se divide la lógica combinacional de una ronda en dos bloques con menor retardo, uno que incluye las transformaciones SubBytes y ShiftRows y otro que incluye MixColumns y AddRoundKey, tal como ilustra la Figura 6.4-*b*.

Las simulaciones obtenidas para el proceso de cifrado en este caso se presentan en la Figura 6.10. En la misma se observa que la latencia del cifrado es de 21 ciclos de reloj, debido a los diez registros que se agregaron (uno en cada ronda general). Para controlar el desplazamiento de los datos entre estos registros se adicionaron también diez registros

en el registro de desplazamiento que desplaza a la señal de entrada *plain_valid*, de manera que esta señal atraviese la misma cantidad de registros que el dato, tal como ocurre en la versión anterior.

Debido a la presencia de una mayor cantidad de registros, en este caso el módulo de cifrado puede procesar 20 bloques de texto plano en paralelo.

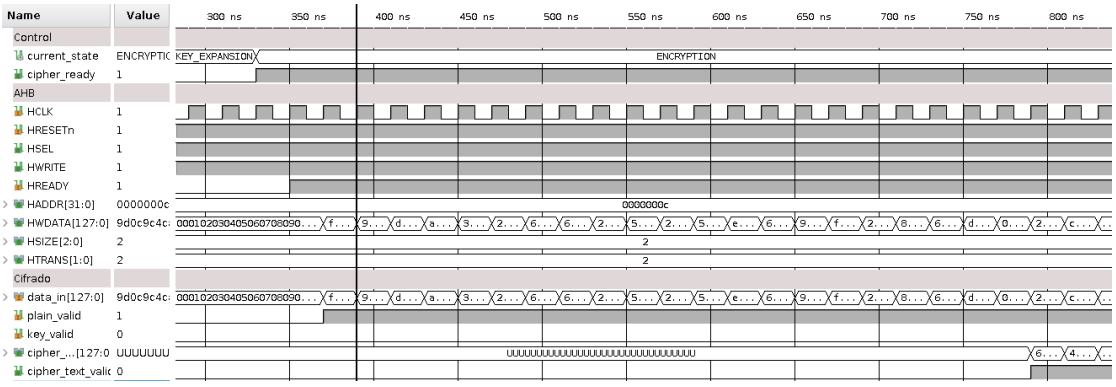


FIGURA 6.10: Cifrado mediante AMBA en la arquitectura pipeline con paralelismo en las rondas.

Assertions de la interfaz AMBA AHB

Las *assertions* incluidas en la arquitectura pipeline chequean las siguientes características del envoltorio AHB:

- Ante una transferencia de tipo IDLE debe ignorar las señales de control y proveer una respuesta de tipo exitosa y sin ciclos de espera.
- Ante una transferencia de tipo BUSY debe ignorar las señales de control y proveer una respuesta de tipo exitosa y sin ciclos de espera.
- Solo la expansión de llave inserta ciclos de espera en las transferencias. Esto implica que HREADYOUT debe irse a 0 en la fase de datos al escribir el registro en el que se almacena el *word* menos significativo de llave.
- El tiempo en el que la señal HREADYOUT puede estar en 0 tiene una cota máxima, dada por la latencia de la expansión de llave.
- Es necesario que HSIZE esté configurado en forma acorde. Dado que el tamaño de los registros es de 128 bits esto implica que HSIZE debe ser 100.

- Los registros en los que se almacena el valor de texto cifrado son de solo lectura. En caso que el master intente escribirlos, el esclavo indica una condición de error en su salida HRESP.

6.4. Latencia

La siguiente tabla resume los valores de latencia para los procesos de expansión de llave y cifrado en la arquitectura pipeline.

Tabla 6.3: Valores de latencia para la arquitectura pipeline.

Proceso	Ciclos de reloj
Expansión de llave	11
Cifrado	11
Cifrado pipeline inter-rondas	21

Capítulo 7

Desarrollo de la arquitectura compacta

En forma análoga a las arquitecturas básica y pipeline, la estructura de la arquitectura compacta contiene tres bloques: control, cifrado y expansión de llave. En este caso el bloque de cifrado consta de la implementación en hardware de la lógica requerida para operar sobre una columna de la matriz de estados, por lo que se requieren cuatro iteraciones sobre esta lógica para completar la ejecución de una ronda.

La Figura 7.1 ilustra la arquitectura compacta. En la misma los buses de datos son de 32 bits y se utiliza una memoria para almacenar la matriz de estado actual y la siguiente. En cada ciclo de reloj se lee un byte de cada fila de la matriz de estados actual, conformando un dato de 32 bits que es entrada a la lógica combinacional que implementa la transformación de una columna. En el próximo ciclo de reloj se almacena el resultado en el sector de memoria que corresponde a la matriz de estados siguiente.

La Tabla 7.1 presenta la descripción de la interfaz de esta arquitectura. Con respecto al ingreso de datos, se utiliza un único bus de 32 bits para disminuir la cantidad de puertos de entrada. Mediante dos señales de control *plain_valid* y *key_valid* se indica si el *word* ingresado forma parte de un bloque de texto plano o de la llave secreta. Internamente *key_valid* se utiliza como habilitación en cuatro registros que almacenan los 128 bits de la llave secreta y son entrada al bloque de expansión de llave; mientras que *plain_valid* se utiliza como señal de selección en un multiplexor que determina si los 32 bits que se escriben en memoria corresponden a un *word* de un nuevo bloque de texto plano o de un

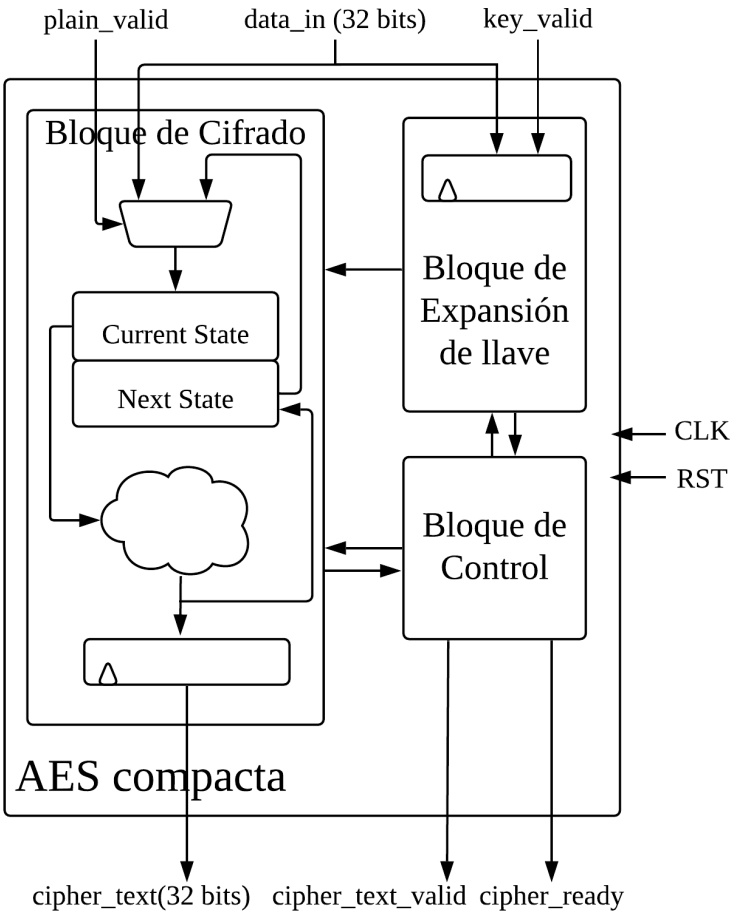


FIGURA 7.1: Diagrama en bloques de la arquitecura compacta.

bloque que está siendo cifrado, es decir, si proviene del bus de datos de entrada *data_in* o de de la salida de la lógica combinacional que implementa la transformación de una columna. Debido a que los bloques de texto plano y las llaves secretas son de 128 bits, se requieren cuatro escrituras de 32 bits para ingresarlos al módulo de cifrado. Las señales que validan el bus de datos (*plain_valid* o *pkey_valid*) deben tener un nivel alto durante el ingreso de los cuatro *words*.

Con respecto a las salidas, *cipher_text* contiene un *word* del bloque de texto cifrado y su valor es válido en los ciclos de reloj en los cuales *cipher_text_valid* posee un nivel lógico alto. Por otro lado, *cipher_ready* posee un valor lógico alto en los ciclos de reloj en los que la arquitectura está lista para recibir un nuevo bloque de texto plano. Debido a que los bloques de texto cifrado son de 128 bits, su envío hacia el exterior se realiza a través de cuatro ciclos de reloj en los que se envían las cuatro columnas de la matriz de estados. La salida *cipher_text_valid* se mantiene en alto durante estos cuatro ciclos de reloj.

TABLA 7.1: Descripción de entradas y salidas de la arquitectura compacta

Puerto	Sentido	Tamaño	Descripción
CLK	entrada	1 bit	Reloj
RST	entrada	1 bit	Reset sincrónico activo con nivel lógico alto
data_in	entrada	32 bits	Puerto mediante el cual se ingresa la llave secreta o un bloque de texto plano
plain_valid	entrada	1 bit	Mediante un nivel lógico alto indica que el puerto *data_in* contiene un bloque de texto plano válido
key_valid	entrada	1 bit	Mediante un nivel lógico alto indica que el puerto *data_in* contiene una llave secreta válida
cipher_text	salida	32 bits	Puerto mediante el cual el módulo AES entrega un bloque de texto cifrado
cipher_text_valid	salida	1 bit	Mediante un nivel lógico alto indica que el puerto *cipher_text* contiene un bloque de texto cifrado válido
cipher_ready	salida	1 bit	Mediante un nivel lógico alto indica que el módulo AES está listo para recibir un nuevo bloque de datos, ya sea texto plano o llave

7.1. Cifrado, control y expansión de llave

7.1.1. Diseño del bloque de cifrado

Debido a que en cada ronda del algoritmo se requieren como entrada los 128 bits obtenidos en la ronda anterior y que en la arquitectura compacta se opera con un *word* por ciclo de reloj, es necesario que esta arquitectura cuente con un medio en el que se almacenen los *words* que ya fueron transformados, así como los que están a la espera de ser procesados.

La estructura utilizada para el bloque de cifrado en la arquitectura compacta se basa en el uso de cuatro memorias de un byte de ancho de palabra y 8 direcciones. Cada una de estas memorias almacena una fila de la matriz de estados y las 8 direcciones se dividen en dos bancos: uno correspondiente a la matriz de estados y otro correspondiente al resultado de ronda (Chodowiec y Gaj, 2003).

Cada vez que ingresa un nuevo texto plano al bloque de cifrado se almacena su contenido en las memorias (una fila de la matriz de estados en cada memoria). Para que el bloque funcione en forma adecuada es importante que se ingresen los cuatro *words* del texto plano en orden, de más a menos significativo.

El proceso de cifrado comienza una vez que el bloque de texto plano es ingresado y almacenado en memoria en su totalidad. Durante la ejecución de una ronda se lee un byte de cada memoria (banco de estado actual) para conformar la palabra de 32 bits a la cual se le aplican las transformaciones de ronda. El resultado de esta transformación

se almacena en el banco de memoria de estado siguiente en el próximo ciclo de reloj, en simultáneo con la lectura de los próximos datos en el banco de estado actual. Una vez finalizada cada ronda los bancos intercambian su rol, es decir, en la próxima ronda se lee el banco que anteriormente almacenaba el resultado de la transformación y los nuevos resultados se almacenan en el banco que anteriormente almacenaba el estado actual. En este esquema de cifrado, la transformación ShiftRows se ejecuta mediante el direccionamiento de las memorias en forma adecuada durante su lectura.

A modo de ejemplo, en la Figura 7.2 se ilustran los valores de las cuatro memorias en el momento en el que se encuentra almacenado un bloque de texto plano para el cual cada uno de sus bytes se identifica con un valor hexadecimal, siendo B_F el más significativo y B_0 el menos significativo. La ejecución de una ronda de cifrado incluye las siguientes acciones:

- En el primer ciclo de reloj se leen los bytes B_F, B_A, B_5 y B_0. Los 8 bits leídos en cada memoria (32 bits en total) ingresan a la lógica combinatoria que implementa la transformación de una columna.
- En el segundo ciclo de reloj se almacena el resultado obtenido al procesar los cuatro bytes leídos en el ciclo anterior (B_F', B_A', B_5' y B_0' en la Figura) en la primer dirección del banco de memoria que está vacío. En forma simultánea, en este ciclo de reloj se leen los bytes B_B, B_6, B_1 y B_C.
- En el tercer ciclo de reloj se almacena en la segunda dirección del banco de memoria que está vacío el resultado obtenido al procesar el *word* leído en el ciclo anterior. En forma simultánea, en este ciclo de reloj se leen los bytes B_7, B_2, B_D y B_8.
- En el cuarto ciclo de reloj se almacena en la tercera dirección del banco de memoria que está vacío el resultado obtenido al procesar el *word* leído en el ciclo anterior. En forma simultánea, en este ciclo de reloj se leen los bytes B_3, B_E, B_9 y B_4.
- En el quinto ciclo de reloj se almacena en la cuarta dirección del banco de memoria que está vacío el resultado obtenido al procesar el *word* leído en el ciclo anterior.
- En el sexto ciclo de reloj comienza a ejecutarse la segunda ronda, en este caso el procedimiento es análogo al del primer ciclo de reloj, salvo que se lee el banco de memoria que fue escrito durante la ronda anterior.

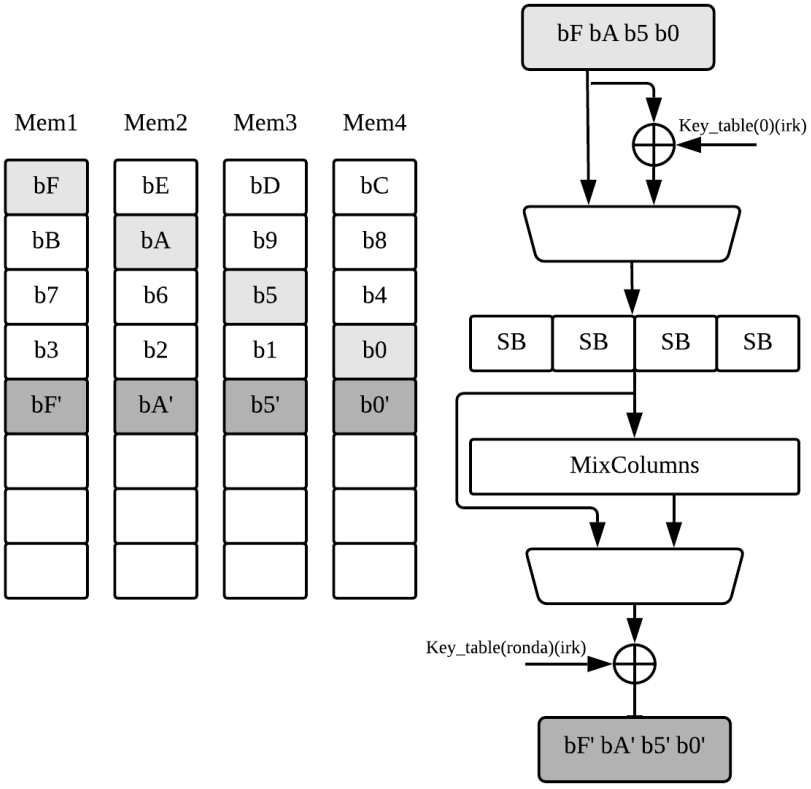


FIGURA 7.2: Bloque de cifrado en la arquitectura compacta.

Debido a que en la mayoría de los ciclos de reloj se realizan operaciones de lectura y escritura en simultáneo sobre las cuatro memorias, es necesario que las mismas sean *dual-port*. Se destaca además que la diferencia en el direccionamiento entre los cuatro bloques de memoria al ser leídos corresponde a la implementación de la transformación ShiftRows en esta arquitectura. Por otro lado, para el caso de la escritura el direccionamiento es el mismo para los cuatro bloques, ya que se almacena una de las columnas de la matriz de estados resultante luego de la ejecución de una ronda de cifrado.

Durante el quinto ciclo de reloj de cada ronda no se realizan operaciones de lectura, ya que es necesario esperar a que finalice la escritura del último byte para comenzar a ejecutar la ronda siguiente. Una vez que finaliza cada ronda, se intercambian los roles de ambos bancos y se repiten las operaciones de lectura y escrituras descritas.

La lógica combinatoria que implementa la ronda y está conectada directamente al puerto de salida de los bloques de memoria consta de cuatro instancias de la LUT que implementa la S-BOX de la transformación SubBytes, una instancia de la transformación

MixColumns que opera sobre una columna y cuatro XOR de 8 bits que implementan AddRoundKey.

Con respecto a MixColumns, para la arquitectura compacta se proponen dos implementaciones posibles:

- Una primer implementación en la que se cuenta con un registro de desplazamiento en el que se almacenan los cuatro bytes de una columna de la matriz de estados. El primer elemento del registro se multiplica por el coeficiente 02, el segundo por 03 y el tercer y cuarto se multiplican por 01. En cada ciclo de reloj se desplaza el registro y se obtiene un nuevo byte de la columna resultante, tal como se detalla en el Capítulo 4.
- Una segunda implementación en la que se expresan las ecuaciones en función de la multiplicación por el elemento *02*. Esta opción arroja como resultado la columna completa en un mismo ciclo de reloj y es la misma que se utiliza para las arquitecturas básica y pipeline.

Además de las cuatro memorias de fila y la lógica que implementa la transformación de un *word*, el bloque de cifrado también cuenta con una señal interna *ronda* que se utiliza para controlar la cantidad de iteraciones. Su valor se incrementa en el último ciclo de reloj de cada ronda y se utiliza como señal de selección en ambos multiplexores que se observan en la Figura 7.2, de manera que en la primer ronda se lleva a cabo también la ronda inicial y en la última ronda se ignora la transformación MixColumns. El valor de *ronda* se utiliza además para habilitar el registro de salida (*cipher_text)* sólo en la última ronda.

El bloque de cifrado cuenta también con la lógica necesaria para determinar cuál es la dirección que se escribe (iw) y lee (ir, ir_2, ir_3 e ir_4) en las memorias en cada ciclo de reloj, además de la dirección que determina cuál *word* de la llave de ronda debe leerse para realizar la XOR en la transformación AddRoundKey (ir_k1, ir_k2,ir_k3 e ir_k4) en caso de que se realice una expansión de llave con precálculo. Con la finalidad de controlar la evolución de estas direcciones, así como el del contador de ronda, se implementa una máquina de estados que determina la operación del bloque de cifrado y se ilustra en la Figura 7.3.

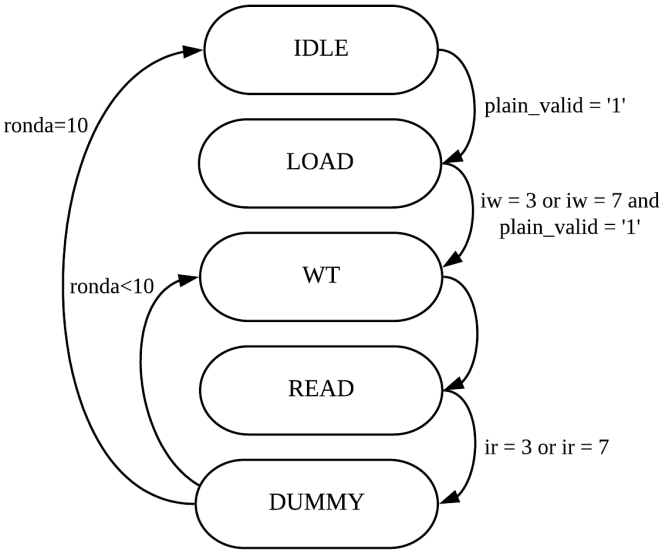


FIGURA 7.3: FSM en el bloque de cifrado en la arquitectura compacta.

- La máquina de estados permanece en estado IDLE desde que detecta un nivel alto en la entrada *reset* hasta que detecta un nivel alto en la entrada *plain_valid*, indicando el ingreso de un nuevo bloque de texto plano. Ante esta situación evoluciona a LOAD.
- Durante los ciclos de reloj en los que la máquina de estados permanece en estado LOAD se almacena el bloque de texto plano en las cuatro memorias, utilizando como dirección en cada una el valor almacenado en *iw*. Por este motivo en este estado se incrementa el contador *iw* cada vez que ingresa un nuevo *word* válido, mientras que los contadores *ir* y *ronda* se mantienen fijos. Al almacenar el cuarto y último *word* válido, la FSM evoluciona a WT.
- El estado WT es un estado de espera con una duración de un ciclo de reloj, esto se debe a que la lectura de la memoria de estado es sincrónica. En este estado se asigna el valor de dirección a cada memoria con el fin de leer la primer columna que debe ingresar a la lógica combinacional, de manera que en el próximo estado se cuente con este primer dato.
- Durante los ciclos de reloj en los que la máquina de estados permanece en estado READ se lee un byte de cada memoria en cada ciclo de reloj (direccionadas por *ir*, *ir*_2, *ir*_3 e *ir*_4). Los cuatro bytes leídos forman el *word* que ingresa a la lógica combinacional de transformación de ronda, tal como se mencionó con anterioridad. En este estado también se almacena el resultado de cada transformación de ronda en la memoria, en la dirección dada por *iw*. Por lo tanto, en este estado se habilitan

los contadores *iw* e *ir*. Las direcciones *ir_2*, *ir_3*, *ir_4* se generan a partir del valor de *ir*. En caso de utilizarse un esquema de precálculo para el manejo de las llaves de onda, los contadores *ir_k1*, *ir_k2*, *ir_k3* e *ir_k4*, también generados a partir del valor de *ir*, determinan qué *word* de la llave de ronda se utiliza en la transformación AddRoundKey. Una vez que se lee el último *word* de la matriz de estados la FSM evoluciona a DUMMY.

- La FSM permanece en estado DUMMY durante un ciclo de reloj, el objetivo de este estado es generar un ciclo de espera durante el cual se almacena el resultado de la última transformación de ronda en memoria. En este estado además se incrementa el valor del contador *ronda*. Si se trata de la última ronda, la FSM evoluciona a IDLE, en caso contrario regresa al estado WT para leer la matriz de estados recién generada y comenzar a ejecutar la ronda siguiente.
- Ambos contadores *iw* e *ir* evolucionan de 0 a 7 y una vez que alcanzan el valor 7 se reinician en 0. Durante el estado LOAD *iw* se incrementa de 0 a 3 mientras que *ir* se mantiene fijo en 0, de manera que cuando comienza la ejecución de las rondas *ir* comienza en 0, mientras que el primer resultado obtenido se almacena en la dirección 4.

7.1.2. Expansión de llave

El bloque de expansión en la arquitectura compacta cuenta con buses de 32 bits, de manera que se genera un *word* de cada llave por ciclo de reloj (Chodowiec y Gaj, 2003). En el mismo, ilustrado en la Figura 7.4, la llave secreta debe ingresarse de a un *word* por ciclo de reloj en forma consecutiva, comenzando siempre por el más significativo, notificando mediante la señal *key_in_valid* la presencia de un dato válido en el bus. El multiplexor de entrada habilita el ingreso de una nueva llave secreta cuando *key_in_valid* posee un nivel alto y permite la realimentación para calcular un nuevo *word* a partir de los anteriores una vez que inicia la expansión.

Internamente, el bloque de cifrado también cuenta con un contador *ronda* que identifica el valor de la ronda de expansión y se utiliza para controlar la cantidad de iteraciones. Su valor se reinicia en uno cada vez que ingresa una nueva llave y se incrementa hasta llegar a doce. Por otro lado, un contador denominado *i* identifica a qué *word* de la llave de ronda corresponden los 32 bits que ingresan al bloque de expansión. Este contador

se utiliza para determinar en qué casos debe aplicarse la función g y en cuáles no. El registro de desplazamiento permite el acceso al *word* calculado con tres ciclos de reloj de anterioridad.

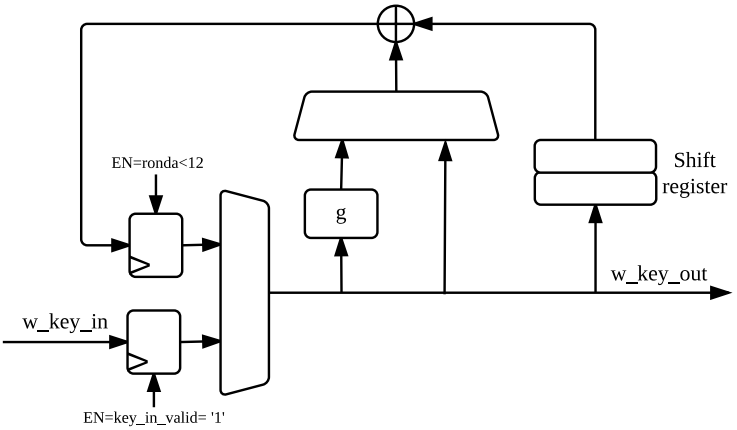


FIGURA 7.4: Expansión de llave en la arquitectura compacta.

Con respecto al manejo de llaves, se implementaron ambos enfoques para la arquitectura compacta: precálculo y *on-the-fly*.

- En forma análoga a las arquitecturas presentadas, en el enfoque precálculo las once llaves de ronda se almacenan en el bloque de cifrado, utilizando cuatro memorias de 32 bits de ancho de palabra y once direcciones cada una. En este caso el bloque de expansión comienza a operar cuando se detecta el ingreso de una nueva llave secreta, mediante un nivel alto la entrada *key_valid*. Durante la expansión envía en cada ciclo de reloj un *word* de llave, junto con un valor de dirección *key_addr* que permite al bloque de cifrado identificar en qué dirección de memoria se debe escribir, así como un valor *key_addr_i* que identifica el *word* que está enviando y permite al bloque de cifrado identificar en cuál de las cuatro memorias se debe escribir.
- Por otro lado, en el enfoque *on-the-fly* los *words* de las llaves de ronda se generan a medida que se requieren. En este esquema, el bloque de control almacena la llave secreta en cuatro registros de 32 bits y sincroniza el comienzo de la expansión con el inicio del cifrado al detectar la presencia de un nuevo bloque de texto plano mediante la señal *plain_valid*. En este caso los registros internos del bloque de expansión se actualizan al detectar un nivel alto en la señal *key_req* proveniente del bloque de cifrado. Además, las salidas *key_addr* y *key_addr_i* dejan de ser

necesarias, ya que en este esquema el bloque de cifrado obtiene la llave de ronda directamente del bus de salida del bloque de expansión.

7.1.3. Bloque de control

El bloque de control para la arquitectura compacta consiste en una FSM que tiene los mismos estados que la presentada para la arquitectura básica (IDLE, KEY_EXPANSION, ENC_READY y ENCRYPTION). La misma controla que el cifrado de un dato no comience hasta que la expansión de llave finaliza, y que se ignoren solicitudes de cifrado mientras otro bloque está siendo procesado. La diferencia en este caso es la cantidad de ciclos de reloj que se mantiene en cada estado. Para la implementación utilizando un esquema de expansión de llave *on-the-fly* el estado KEY_EXPANSION deja de ser necesario y la FSM evoluciona de IDLE a ENC_READY al detectar el ingreso de los cuatro *words* correspondientes a la llave secreta.

7.2. Interfaz AMBA

El envoltorio para la arquitectura compacta cuenta con buses de datos HRDATA y HWDATA de 32 bits. Debido a que en esta arquitectura del módulo AES los buses de datos de entrada y salida *data_in* y *cipher_text* poseen el mismo tamaño, la conexión entre el bus de datos de entrada del módulo de cifrado y el bus de datos HWDATA es directa. El envoltorio en este caso genera, a partir de las señales de control del protocolo AMBA, las señales de habilitación que indican la presencia de un dato válido. Por otro lado, para el texto cifrado se utilizan cuatro registros de 32 bits que almacenan el bloque de texto cifrado y pueden ser accedidos mediante la interfaz AHB. La Tabla 7.2 muestra las direcciones para cada uno de los registros.

Tabla 7.2: Registros definidos en el envoltorio AHB-Lite para la arquitectura compacta.

Nombre	Dato	Dirección
reg_key	Key[31:0]	x00000004
reg_plain	Plain_text[31:0]	x00000010
reg_cipher_text_0	Cipher_text_0[31:0]	x00000020
reg_cipher_text_1	Cipher_text_1[31:0]	x00000024
reg_cipher_text_2	Cipher_text_2[31:0]	x00000028
reg_cipher_text_3	Cipher_text_3[31:0]	x0000002C

Para ingresar una nueva llave secreta al módulo de cifrado, el maestro AHB escribe cuatro veces en forma consecutiva el registro de llave siguiendo el protocolo de comunicación especificado en el estándar y detallado en el Apéndice B de este trabajo. En cada iteración se escribe un *word* de la llave secreta, comenzando por el más significativo. El maestro envía la dirección del registro a escribir a través del bus de direcciones HADDR, junto con un nivel alto en la señal de escritura HWRITE y en la señal de selección HSEL (esta última se genera a partir de la dirección y tiene un nivel alto cuando el maestro se comunica con el módulo de cifrado). El envoltorio lee estas señales y activa la señal de habilitación que indica la presencia de una llave secreta válida (*key_valid*) cuando detecta que el maestro está solicitando la escritura del registro de llave. El envoltorio conecta en forma directa la entrada *data_in* del bloque de cifrado al bus de datos HWDATA. Esto implica que en este caso no es necesario almacenar el valor de la llave internamente en el envoltorio, como sucede en la arquitectura básica.

Con la finalidad de ingresar un bloque de texto plano al módulo de cifrado, el maestro AHB escribe cuatro veces en forma consecutiva el registro de texto plano siguiendo el protocolo de comunicación especificado en el estándar y detallado en el Apéndice B de este trabajo. En cada iteración se escribe un *word* del bloque de texto plano, comenzando por el más significativo. El maestro envía la dirección del registro a escribir a través del bus de direcciones HADDR, junto con un nivel alto en la señal de escritura HWRITE y en la señal de selección HSEL (esta última se genera a partir de la dirección y tiene un nivel alto cuando el maestro se comunica con el módulo de cifrado). El envoltorio lee estas señales y activa la señal de habilitación que indica la presencia de un bloque de texto plano válido (*plain_valid*) cuando detecta que el maestro está solicitando la escritura del registro de texto plano. El envoltorio conecta en forma directa la entrada *data_in* del bloque de cifrado al bus de datos HWDATA. Esto implica que en este caso no es necesario almacenar el valor del bloque de texto plano internamente en el envoltorio, como sucede en la arquitectura básica.

Durante el proceso de cifrado o expansión de llave la salida del envoltorio HREADYOUT se mantiene en nivel lógico bajo, indicando al maestro AHB que el módulo de cifrado está ocupado y no puede recibir un nuevo dato. Una vez que el proceso de cifrado finaliza, el envoltorio utiliza la salida *cipher_text_valid* del módulo de cifrado como habilitación para escribir los cuatro registros correspondientes al texto cifrado. Estos registros pueden ser accedidos por el maestro AHB utilizando el protocolo de lectura y mantienen su

valor hasta que se obtiene un nuevo bloque de texto cifrado. La señal *cipher_text_valid* puede ser utilizada además como interrupción para dar aviso al maestro AHB de que los registros de salida contienen un dato válido.

7.3. Verificación funcional

Utilizando la herramienta XSim, incorporada en el entorno de Vivado, se simularon tanto los tests dirigidos como el test aleatorio de 500 muestras. La totalidad de los tests corridos pasaron, obteniendo resultados favorables.

A continuación se presentan formas de onda que ilustran el proceso de expansión de llave y cifrado para la arquitectura compacta.

Expansión de llave - precálculo

En la implementación que considera esquema de manejo de llaves con precálculo se realiza la expansión completa antes de comenzar el cifrado. En la Figura 7.5 se observa el proceso de carga de llave secreta y posterior expansión para esta arquitectura.

En primer lugar, la FSM de control evoluciona de *IDLE* a *KEY_EXPANSION* ante la presencia de una llave en el bus de datos. Por otro lado, en el bloque de expansión de llave se reinicia el contador de ronda y comienza el proceso de expansión que obtiene un *word* de llave de ronda por cada ciclo de reloj, tal como se detalló en las secciones anteriores. En la Figura se observa que se envían cuatro señales hacia los bloques de cifrado y control:

- *key_out*: salida de 32 bits que corresponde a los distintos *words* de las llaves de ronda.
- *key_out_valid*: salida de un bit que indica con un nivel lógico alto que los datos presentes en el bus *key_out* son válidos.
- *key_addr*: salida de cuatro bits que identifica a cada llave de ronda.
- *key_addr_i*: salida de dos bits que identifica a cada *word* de la llave de ronda.

El bloque de cifrado lee estas cuatro señales y las utiliza para almacenar las llaves de ronda en las memorias de llave. Por otro lado, el bloque de control lee el valor de las señales *key_out_valid*, *key_addr* y *key_addr_i* para determinar el momento en el que se envía la última llave de ronda. Ante esta situación su FSM evoluciona a ENC_READY.

En la Figura 7.5 se observa además que la señal de control de iteraciones *ronda* permanece fija con valor doce en los momentos en los que el bloque de expansión está ocioso y se reinicia a uno cuando recibe una llave. Su valor se incrementa en los ciclos de reloj en los cuales se envía el último *word* de una llave de ronda, controlando de esta forma la cantidad de iteraciones. El tiempo requerido por la arquitectura compacta para realizar el proceso de expansión de llave es de 42 ciclos de reloj.

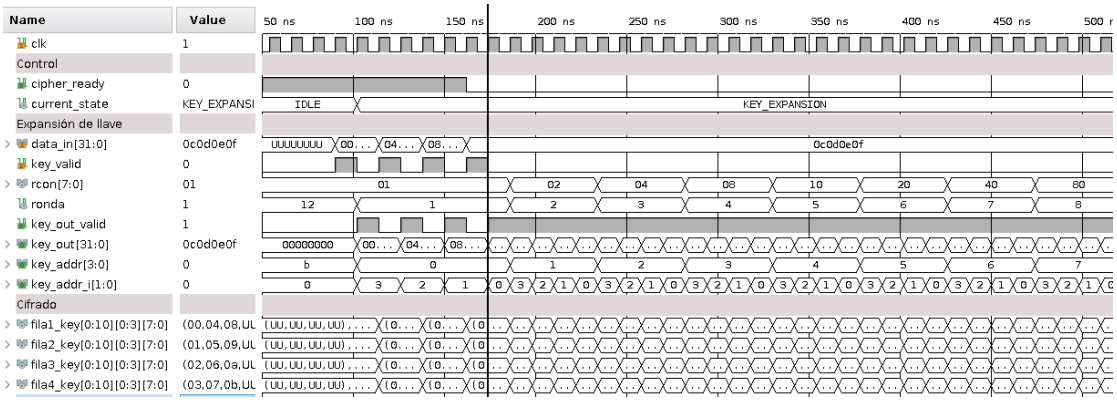


FIGURA 7.5: Expansión de llave para la arquitectura compacta - precálculo.

Expansión de llave - *on-the-fly*

La Figura 7.6 ilustra la carga de llave para las implementaciones de la arquitectura compacta en las que se realiza un manejo de llaves *on-the-fly*. En la misma se observa que el bloque de control almacena la llave secreta en el registro *secret_key* y que esta llave secreta es la que envía al bloque de expansión al mismo tiempo que envía el texto plano al bloque de cifrado. La Figura también muestra que, a pesar de no necesitar almacenar la totalidad de las llaves de ronda, el bloque de cifrado almacena la primer llave.

Considerando que la ronda inicial consiste en una operación XOR entre la primer llave de ronda y el bloque de texto plano, cada columna de la llave debe operar con la correspondiente columna del bloque de texto plano. Debido a la manera en la que se direccionan las memorias al leerse, en esta arquitectura se realiza una alteración en el

orden de las transformaciones de ronda ya que ShiftRows se lleva a cabo en forma previa a la ronda inicial. Por este motivo, es necesario que el bloque de cifrado almacene internamente el valor de la primer llave de ronda con el fin de ejecutar la ronda inicial en forma adecuada.

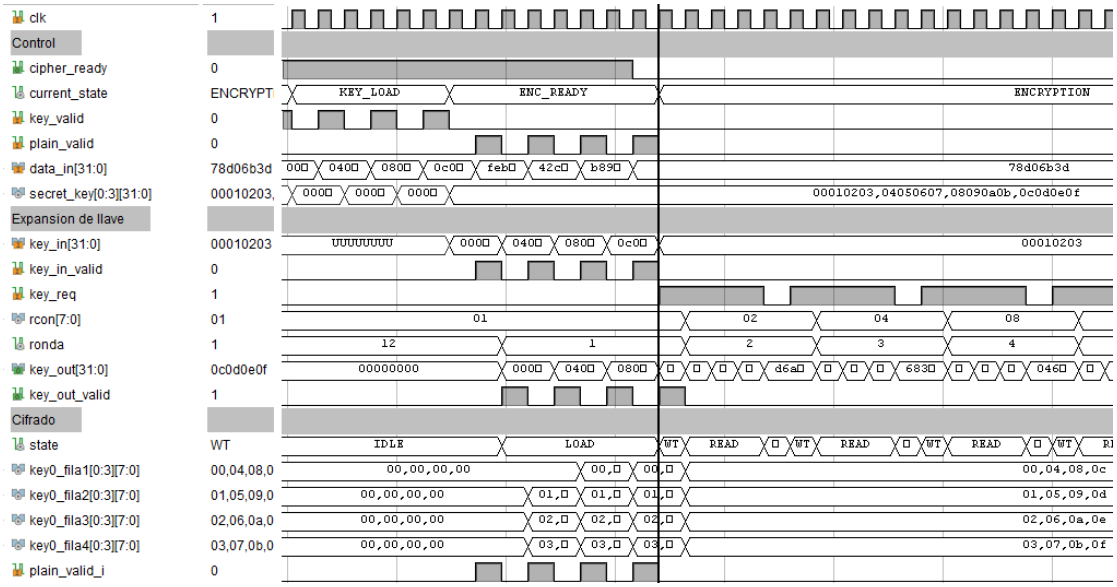


FIGURA 7.6: Expansión de llave para la arquitectura compacta - *on-the-fly*.

En este esquema de manejo de llaves, el bloque de expansión calcula un nuevo *word* de llave en los ciclos de reloj en los que su entrada *key_req* posee un nivel alto.

Cifrado

Una vez que finaliza el proceso de expansión de llave, el bloque AES presenta un nivel lógico alto en la señal *cipher_ready*, indicando que está listo para recibir un nuevo dato. Al identificar esta situación, el maestro envía a través del puerto *data_in* uno de los bloques de texto plano que obtiene del archivo de texto correspondiente, junto con un pulso en la entrada *plain_text_valid* para indicar que se trata de un dato válido, tal como se ilustra en la Figura 7.7. La misma muestra que los *words* del bloque de texto plano se cargan en orden, de más significativo a menos significativo y que la señal *cipher_ready* se mantiene en nivel alto hasta que se carga la última palabra de datos.

En la Figura se observa también el estado de la FSM del bloque de cifrado. La misma permanece en estado LOAD mientras se almacena el bloque de texto plano en las cuatro memorias y evoluciona a WT una vez que finaliza el proceso de carga. En el estado WT

se envía la primer dirección de lectura y evoluciona a READ, estado en el cual se lee y escribe la memoria.

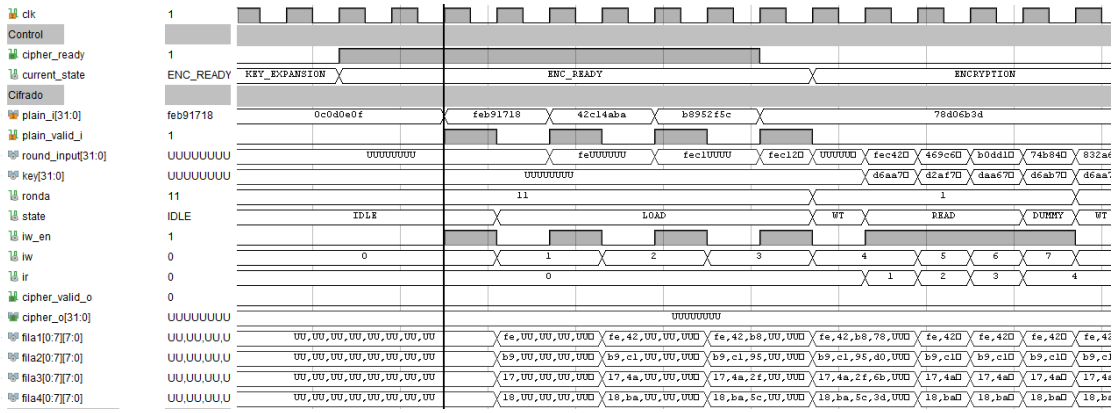


FIGURA 7.7: Carga de un bloque de texto plano en la arquitectura compacta.

En la Figura 7.8 se muestran las últimas iteraciones de ronda. En la misma se observa cómo la señal *cipher_ready* toma un valor lógico alto una vez que finaliza el proceso y las cuatro palabras del bloque de texto cifrado se envían por el puerto correspondiente junto con un nivel alto en *cipher_text_valid*. El tiempo requerido por la arquitectura compacta para realizar el proceso de cifrado es de 47 ciclos de reloj.

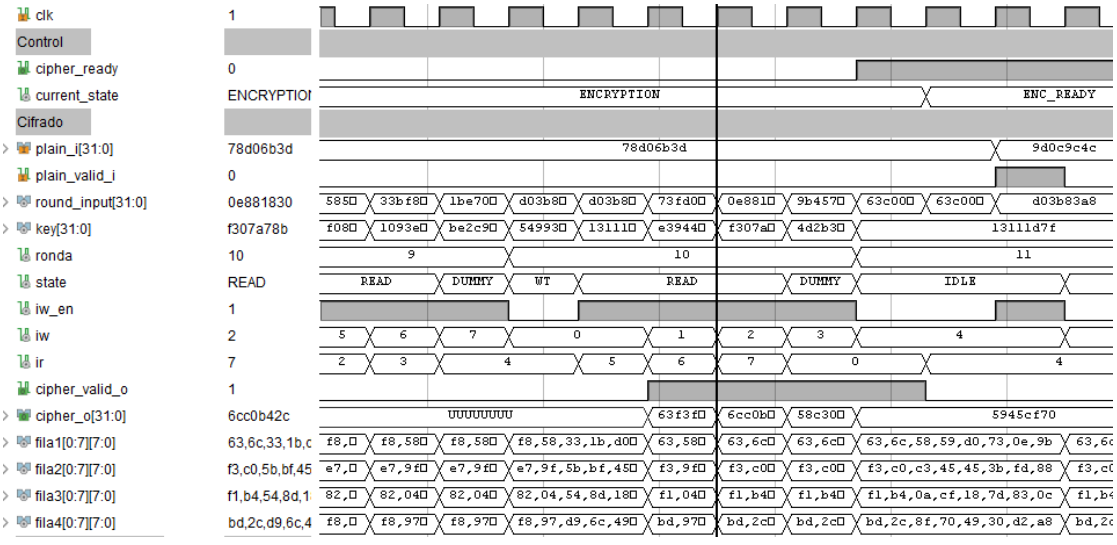


FIGURA 7.8: Obtención de un bloque de texto cifrado en la arquitectura compacta.

En una segunda versión del bloque de cifrado se implementó la transformación MixColumns utilizando la arquitectura compacta presentada en la sección 4.4 y que se ilustra en la Figura 4.4. En esta implementación, la transformación MixColumns obtiene un byte por cada ciclo de reloj, por lo que requiere cuatro ciclos de reloj para operar sobre una columna de la matriz de estados. Por esta razón en este caso se modificó la FSM

de control y se agregaron estados en los que se espera a que finalice esta transformación antes de leer una nueva columna de la matriz de estados, siempre y cuando el valor de ronda sea menor a diez (ya que en la décima ronda no se lleva a cabo esta transformación y por lo tanto puede leerse una columna por ciclo de reloj).

La Figura 7.9 ilustra una ronda de cifrado para esta implementación. En la misma se observa que en los ciclos de reloj en los que se lee una nueva columna de la matriz de estados, se envía una nueva entrada al bloque que ejecuta la transformación MixColumns junto con un pulso en la señal *mc_input_vld*. Cuando se detecta un nivel alto en la salida *mc_output_vld* de este bloque, el resultado se almacena en memoria y se incrementa el valor de la dirección de escritura *iw*. En ese mismo ciclo de reloj se envía al bloque MixColumns el valor de la siguiente columna de la matriz de estados. El valor de ronda incrementa una vez que las cuatro columnas de la matriz son transformadas y escritas en memoria.

La Figura 7.10 ilustra la ejecución de la última ronda, que no incluye la ejecución de la transformación MixColumns y el envío del dato cifrado junto con la señal *cipher_text_valid*. El tiempo requerido por la arquitectura compacta para realizar el proceso de cifrado cuando se implementa la transformación MixColumns en forma compacta es de 200 ciclos de reloj.

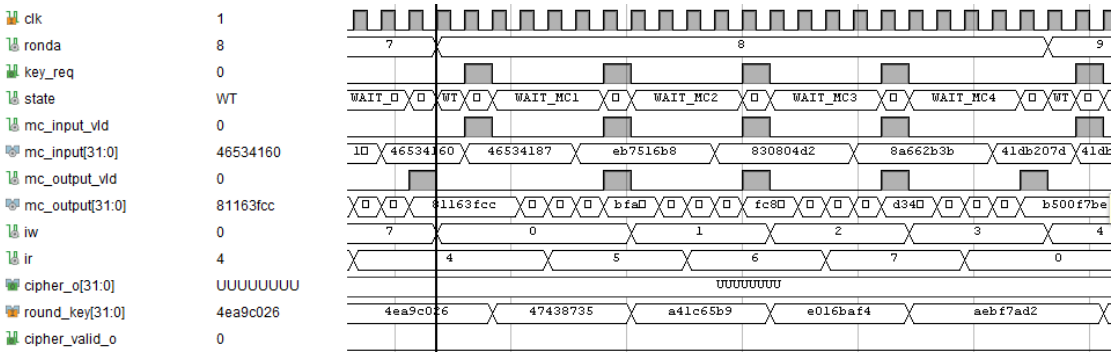


FIGURA 7.9: Ejecución de ronda de cifrado al implementar la versión compacta de MixColumns.

En las Figuras anteriores también puede observarse que las implementaciones simuladas presentan un manejo de llaves *on-the-fly* y que la señal que solicita un nuevo *word* de la llave de ronda (*key_req*) se activa en el mismo ciclo de reloj en el que se envía un nuevo *word* al bloque MixColumns, de esta manera se asegura que el *word* de llave está estable en el bus en el ciclo de reloj en el que la transformación MixColumns finaliza y se realiza AddRoundKey en forma combinatoria. Para las implementaciones en las que se

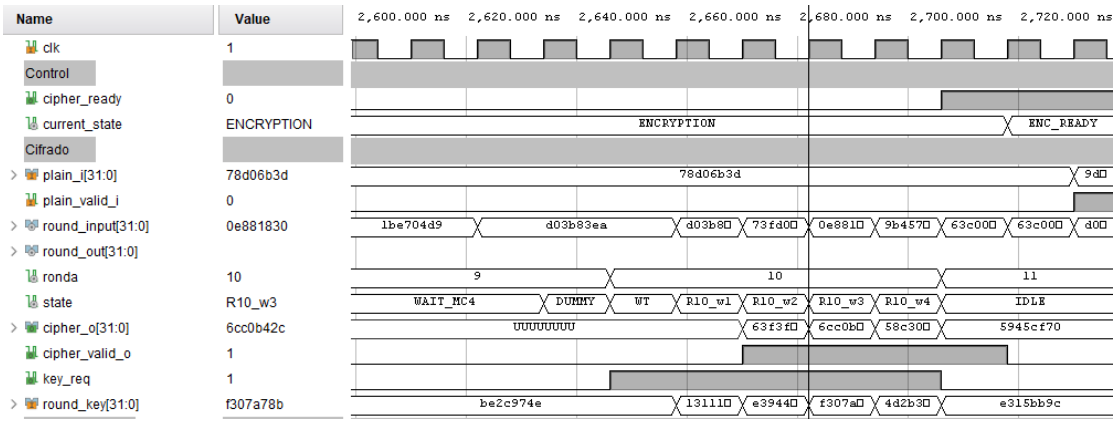


FIGURA 7.10: Obtención de un bloque de texto cifrado al implementar la versión compacta de MixColumns.

realiza un manejo de llave con precálculo el esquema de cifrado es el mismo, la diferencia radica en que en este caso las llaves de ronda se obtienen del arreglo interno denominado *key_table*.

Interfaz AMBA

El banco de prueba o *Test Bench* en este caso es similar al presentado en las secciones anteriores, salvo que se modificó la estructura del maestro para que se comunique mediante una interfaz AHB. El maestro envía cada dato de 128 bits (llave o texto plano) mediante cuatro escrituras de 32 bits cada una, utilizando las direcciones dadas para bloques de texto plano y llave secreta.

En primer lugar, el maestro AHB envía la llave secreta y luego comienza a enviar de a uno los bloques de texto plano que obtiene del archivo de entradas. Por otro lado, en el banco de prueba se comparan los resultados obtenidos por el bloque de cifrado con los valores almacenados en el archivo de bloques de texto cifrado.

La carga de llave secreta se realiza mediante cuatro escrituras consecutivas a la dirección de llave, una para cada *word* de la llave secreta. En la Fig 7.11 se muestra la carga de llave secreta. En la misma se observa que el maestro escribe cuatro veces en forma consecutiva la dirección correspondiente a la llave secreta, colocando su dirección en el bus HADDR y solicitando la escritura mediante un nivel alto en HSEL y HWRITE. En cada escritura el bus de datos contiene un *word* de la llave secreta, ordenados de más a menos significativo.

El protocolo AHB admite simultaneidad entre la fase de datos de una transacción y la fase de direcciones de la siguiente (denominado pipeline entre transacciones). En este caso el maestro implementado no hace este tipo de paralelismo, motivo por el cual la señal HSEL oscila entre nivel lógico alto y bajo, indicando que en la fase de datos no se recibe una dirección válida para la próxima transacción. De todas formas, el envoltorio admite ambos tipos de transferencias, con paralelismo y sin paralelismo.

En el envoltorio propuesto se utiliza la señal *HREADYOUT* para indicar al maestro AHB que el módulo de cifrado está ocupado durante la expansión de llave y no puede recibir nuevos datos de entrada. Esta señal baja en la fase de datos de la cuarta escritura del registro de llave y regresa a nivel alto cuando se detecta un flanco ascendente en la señal *cipher_ready*, es decir, cuando finaliza la exposición y el módulo de cifrado está listo para recibir un nuevo dato.

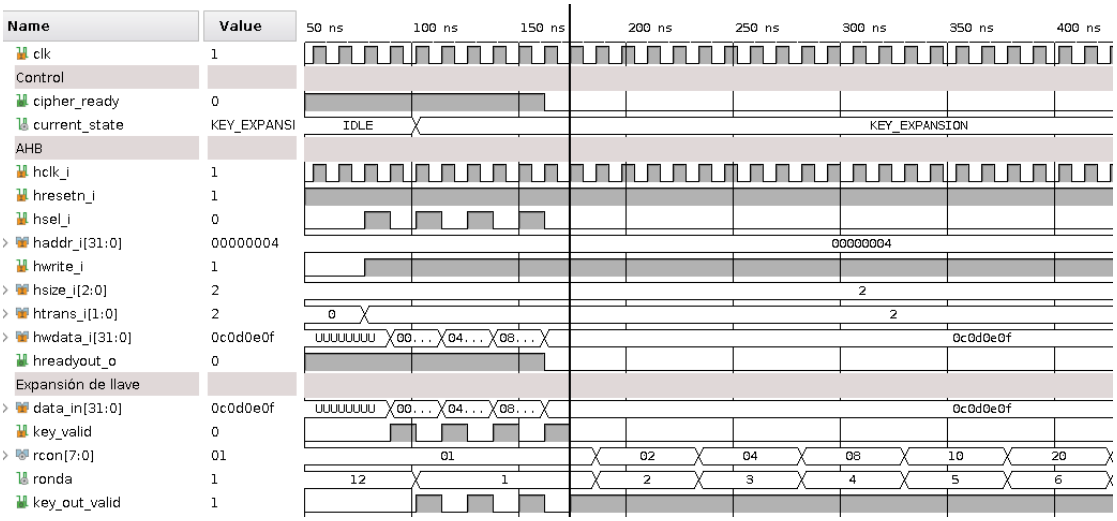


FIGURA 7.11: Carga y expansión de llave para arquitectura compacta mediante interfaz AHB.

Una vez que el módulo de cifrado vuelve a estar listo, el maestro AHB comienza a enviar bloques de texto plano mediante la escritura del registro de texto plano. Esta operación se lleva a cabo colocando su dirección en el bus HADDR y solicitando la escritura mediante un nivel alto en HWRITE y HSEL, tal como se ilustra en la Figura 7.12. En la misma se observa que el maestro escribe cuatro veces en forma consecutiva el registro *plain_reg*, colocando su dirección en el bus HADDR y solicitando la escritura mediante un nivel alto en HSEL y HWRITE. En cada escritura el bus de datos contiene un *word* del bloque de texto plano obtenido del archivo de texto, ordenados de más a menos significativo.

La escritura de este registro también admite tanto operaciones simples como pipeline en el bus AMBA.

En forma análoga a la expansión de llave, se utiliza la señal *HREADYOUT* para indicar al maestro AHB que el módulo está ocupado durante el cifrado del bloque de texto plano ingresado y no puede recibir nuevos datos de entrada. Esta señal baja en la fase de datos de la cuarta escritura del registro de texto plano y regresa a nivel alto cuando se detecta un flanco ascendente en la señal *cipher_ready*, es decir, cuando finaliza el cifrado y el módulo está listo para recibir un nuevo dato.

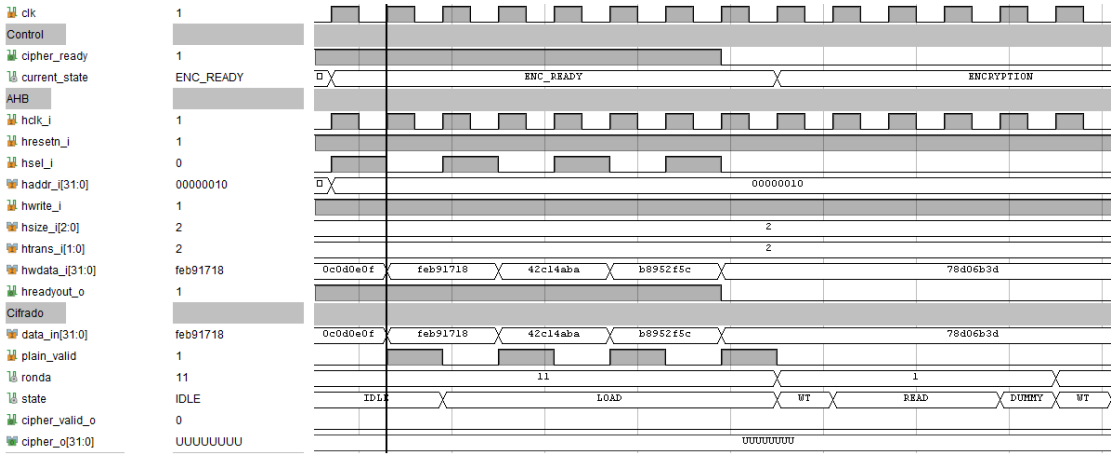


FIGURA 7.12: Carga de bloque de texto plano para arquitectura compacta mediante interfaz AHB.

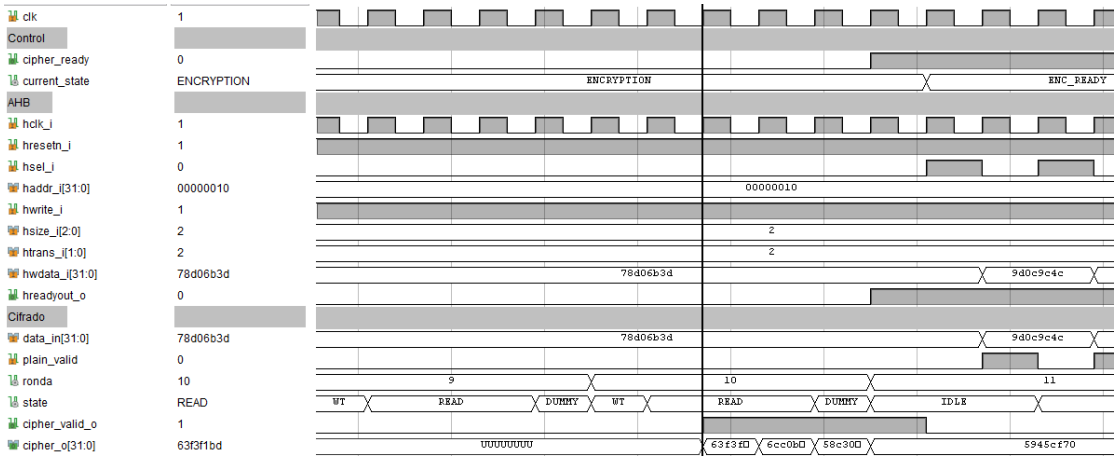


FIGURA 7.13: Cifrado para arquitectura compacta mediante interfaz AHB.

Una vez finalizado el proceso de cifrado, el bloque de texto cifrado se almacena en cuatro registros (*reg_ciph_3*, *reg_ciph_2*, *reg_ciph_1* y *reg_ciph_0*), tal como se observa en la Figura 7.13. Estos registros quedan a disposición para ser accedidos por el maestro mediante interfaz AMBA.

Assertions de la interfaz AMBA AHB

Las *assertions* incluidas en la arquitectura compacta chequean las siguientes características del envoltorio AHB:

- Ante una transferencia de tipo IDLE debe ignorar las señales de control y proveer una respuesta de tipo exitosa y sin ciclos de espera.
- Ante una transferencia de tipo BUSY debe ignorar las señales de control y proveer una respuesta de tipo exitosa y sin ciclos de espera.
- Tanto la operación de cifrado como la expansión de llave insertan ciclos de espera en las transferencias. Esto implica que HREADYOUT debe irse a 0 en la fase de datos al escribir el registro en el que se almacena el *word* menos significativo de llave y texto plano.
- El tiempo en el que la señal HREADYOUT puede estar en 0 tiene una cota máxima, dada por la latencia de la arquitectura compacta.
- Es necesario que HSIZE esté configurado en forma acorde. Dado que el tamaño de los registros es de 32 bits esto implica que HSIZE debe ser 010.
- Los registros en los que se almacena el valor de texto cifrado son de solo lectura. En caso que el master intente escribirlos, el esclavo indica una condición de error en su salida HRESP.

7.4. Latencia

La siguiente tabla resume los valores de latencia para los procesos de expansión de llave y cifrado en la arquitectura compacta.

Tabla 7.3: Valores de latencia para la arquitectura compacta.

Proceso	Ciclos de reloj
Expansión de llave	42
Cifrado	47
Cifrado con MixColumns compacta	200

Capítulo 8

Resultados de síntesis

La síntesis lógica es el proceso a partir del cual un diseño digital descrito en lenguaje HDL es transformado a una red de compuertas lógicas y elementos secuenciales (AND, OR, NOT, DFF), así como bloques ya especificados y optimizados para la tecnología objetivo (como podría ser el caso de bloques aritméticos). Para el caso específico de FPGA, cada fabricante ofrece su propio software de síntesis que está optimizado para ser utilizado con sus dispositivos reconfigurables. Estas herramientas generan reportes durante el proceso de síntesis, a partir de los cuales es posible conocer qué tipo de compuertas o bloques se utilizan para implementar cada sub-bloque del diseño. Es importante analizar en detalle estos reportes, ya que las advertencias que aparecen en ellos pueden ayudar a encontrar problemas en el código HDL desarrollado, como por ejemplo inferencia de latches (que en muchas FPGAs no pueden ser implementados), omisión de alguna de las posibilidades en una estructura de tipo *case*, omisión de alguna señal en una lista de sensibilidad, entre otros.

El resultado del proceso de síntesis depende en gran medida de la forma en la que se realiza la descripción HDL. Los fabricantes de FPGA emiten recomendaciones orientados a sus distintos modelos, en los que se detalla la manera adecuada en la que se debe realizar la codificación HDL para que la herramienta de síntesis infiera la utilización de ciertos recursos, como por ejemplo los bloques de memoria RAM. Es importante destacar que solo un subconjunto de los lenguajes HDL es sintetizable, ya que múltiples construcciones de estos lenguajes (principalmente aquellas relacionadas al manejo de

tiempos) están orientadas a la creación de bancos de pruebas o *testbenches* durante el proceso de simulación.

Por otro lado, los resultados de la síntesis también dependen de las restricciones (denominadas *constraints*) que el diseñador impone sobre el sistema. Estas restricciones permiten por ejemplo indicar que se desea optimizar el diseño para obtener su mayor desempeño, solicitando a la herramienta de síntesis que realice un esfuerzo extra al minimizar el camino crítico. Mediante estas restricciones también es posible determinar qué tipo de codificación se desea para una FSM para la cual la codificación de sus estados no fue realizada en forma explícita, se concede o no a la herramienta de síntesis ciertos permisos como mover registros o romper jerarquías dentro del diseño para mejorar sus características de temporizado.

Las herramientas de síntesis también tienen en cuenta las restricciones o *constraints* temporales. Estas restricciones permiten al diseñador especificar la frecuencia de reloj a la que tiene que funcionar el bloque, sus retardos de entrada y salida, la generación de relojes internos dentro del bloque diseñado y la interacción entre distintos dominios de reloj (Xilinx, 2018). Estas restricciones son sumamente importantes ya que son las que permiten posteriormente realizar un análisis de *timing* y determinar si el hardware resultante cumple con sus especificaciones, principalmente si va a poder trabajar en forma adecuada a la frecuencia requerida.

8.1. Características de la FPGA utilizada

La síntesis de las distintas arquitecturas propuestas para el bloque de cifrado AES se realizó utilizando la herramienta Vivado, perteneciente a Xilinx. Se estableció como dispositivo objetivo la placa de desarrollo ARTY, que cuenta con una FPGA Xilinx Artix-35T (xc7a35ticsg324-1L). En rasgos generales, las características de esta FPGA son:

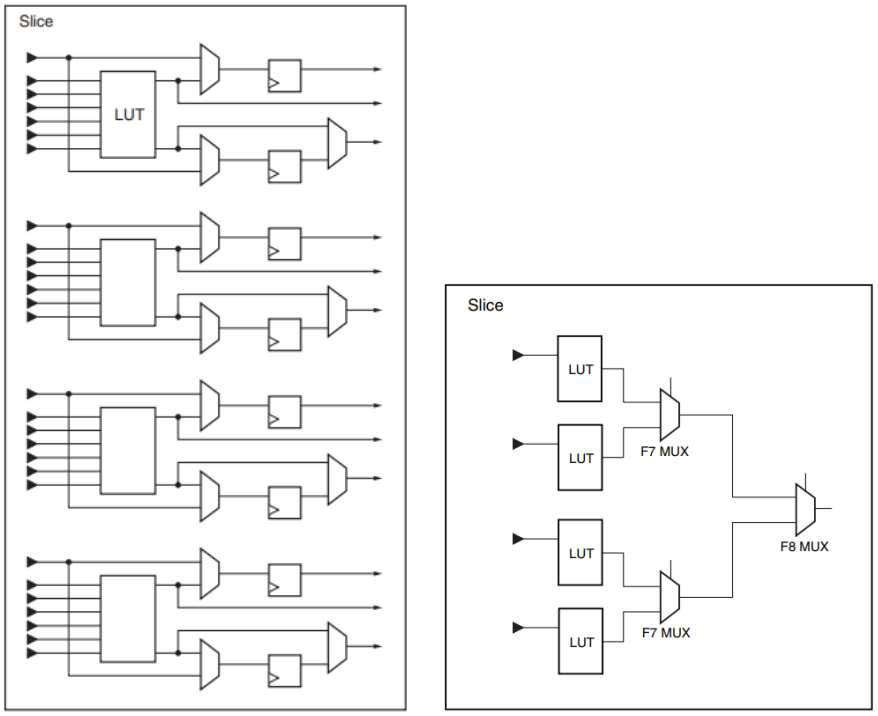
- Contiene 5000 slices. Cada uno de ellos compuesto por 8 flip-flops y 4 LUTs de 6 entradas.
- 1800 Kbits de bloques RAM.
- 90 slices especializados para DSP

- Capacidad de generar relojes internos de hasta 450 MHz mediante la utilización de PLLs.
- Conversor analógico a digital on-chip.
- Programable a través de JTAG y memoria flash.

Además, la placa de desarrollo ARTY cuenta con:

- Alimentación a partir de USB o fuente de 7V-15V.
- Oscilador de 100 MHz.
- Ethernet 10100 Mbps.
- Bridge USB-UART.
- 4 switches.
- 4 botones.
- LEDs RGB.
- Conectores pmod
- Conector ArduinoChipKit.

La estructura de las FPGAs de Xilinx se basa en el uso de Bloques Lógicos Configurables (CLBs). En la FPGA utilizada, cada CLB está compuesto por dos *slices* que pueden ser configurados para llevar a cabo funciones lógicas o aritméticas, así como almacenar información o implementar un registro de desplazamiento. En modelos anteriores, como Virtex o Spartan II, los *slices* están compuestos por dos LUTs de cuatro entradas cada una y dos registros, mientas que para los modelos de la serie 7 cada uno de los *slices* está compuesto por cuatros LUTs de seis entradas cada una y ocho registros, tal como se ilustra en la Figura 8.1.



(a) Slice (b) Muxs F7 y F8

FIGURA 8.1: Slice del FPGA Artix-35T.

Cada LUT puede ser configurada de distintas maneras (Xilinx, 2012)

- Puede implementar cualquier función lógica que sea producto de seis entradas.
- Debido a que posee dos salidas, una LUT también puede implementar dos funciones lógicas que comparten las mismas 5 entradas, o dos funciones con entradas independientes, siempre y cuando la suma de la cantidad de entradas de ambas sea menor o igual a 5.
- Puede ser configurada como 64 bits de memoria RAM distribuida.
- Puede ser configurada como un registro de desplazamiento de 32 bits.

Los multiplexores que forman parte del *slice* admiten la utilización de LUTs y registros en forma conjunta o individual: una de las entradas a *slice* puede ser conectada en forma directa a los registros, evitando el paso por una LUT, así como las salidas de las LUTs pueden conectarse a las salidas del *slice* evitando los registros.

Los multiplexores F7 y F8 se utilizan para seleccionar la salida de una LUT, admitiendo la implementación de funciones con más de 6 entradas en un mismo CLB, como se ilustra en la Figura 8.1.

8.2. Síntesis de las arquitecturas propuestas para el bloque de cifrado AES

Cada una de las arquitecturas propuestas se sintetizó especificando en los *constraints* de temporizado una frecuencia de reloj objetivo de 100 MHz, que es la frecuencia del oscilador con el que cuenta la placa de desarrollo utilizada. Los resultados obtenidos se muestran en la siguiente tabla:

Tabla 8.1: Resultados de síntesis - cantidad de recursos utilizados @100 MHz

Recurso	Disponible	Básica	Básica OTF	Pipeline	Pipeline int-ext	Compacta	Compacta OTF
LUTs	20800	1978	1532	7529	7520	1269	780
FFs	41600	2563	1276	3393	4554	1867	529
F7/F8 Muxs	24450	384	384	3600	3600	139	168
RAMB18	100	0	0	0	0	4	4
DSP	90	-	-	-	-	-	-
IOs	210	106	106	169	169	106	106

A partir de los reportes de temporizado es posible determinar la frecuencia máxima de operación de cada uno de las arquitecturas. Estos valores se muestran en la Tabla 8.2. Esta tabla también incluye el desempeño de cada una de las arquitecturas para una frecuencia de trabajo de 100MHz.

Para la arquitecturas básica y básica_otf se considera que en el escenario de mayor desempeño, tanto la escritura como la lectura de los registros AHB se realiza utilizando pipeline entre las transacciones, es decir, la fase de dirección de una transacción AHB se realiza en simultáneo con la fase de dato de la transacción siguiente. En este caso, se requieren 4 ciclos de reloj para almacenar un bloque de texto plano en los cuatro registros, más 11 ciclos de reloj que demora el bloque de cifrado en obtener el bloque de texto cifrado correspondiente y otros cuatro ciclos que utiliza el maestro AHB para leerlo. Esto arroja como resultado un desempeño de $\frac{128}{19} \times 100 = 673{,}68 Mbps$. Este desempeño es el máximo admisible para esta arquitectura cuando funciona a 100 MHz. En caso de que se utilicen transacciones AHB simples este valor se reduce a $474{,}07 Mbps$.

Para las arquitecturas pipeline y pipeline-int-ext, se considera que el escenario de mayor velocidad corresponde a una interfaz AHB con bus de datos de 128 bits en la cual el maestro tenga a capacidad de leer una salida de 128 bits por cada ciclo de reloj. En este caso ideal, el desempeño es de $128 * 100 = 12800 Mbps$.

Para la arquitecturas compacta se consideró que en el escenario de mayor desempeño, tanto la escritura como la lectura de los registros AHB se realiza utilizando pipeline entre las transacciones, es decir, la fase de dirección de una transacción AHB se realiza en simultáneo con la fase de dato de la transacción siguiente. En este caso, se requieren 4 ciclos de reloj para almacenar un bloque de texto plano en los cuatro registros, más 47 ciclos de reloj que demora el bloque de cifrado en obtener el bloque de texto cifrado correspondiente y otros cuatro ciclos que utiliza el maestro AHB para leerlo. Esto arroja como resultado un desempeño de $\frac{128}{55} \times 100 = 232{,}73 Mbps$. Este desempeño es el máximo admisible para esta arquitectura. En caso de que se utilicen transacciones AHB simples este valor se reduce a $203{,}17 Mbps$. Para la implemantación de esta arquitectura en la que se implementó la versión compacta de la transformación MixColumns, el desempeño máximo es $\frac{128}{200} \times 100 = 64 Mbps$.

Tabla 8.2: Resultados de síntesis - Frecuencia máxima de operación y desempeño a 100 MHz.

	Básica	Básica OTF	Pipeline	Pipeline int-ext	Compacta	Compacta OTF	Compacta OTF MC2
Frecuencia Max [MHz]	163.32	168.92	167.84	183.15	143.67	131.5	147
Desempeño Max @100MHz[Mbps]	673.68	673.68	12800	12800	232.73	232.73	64

En cuanto a los valores máximos de frecuencia, cabe notar que las arquitecturas básica y básica_OTF poseen diferentes valores de frecuencia máxima. Este resultado se debe a que el camino de mayor retardo en estas arquitecturas forma parte de la interacción entre el bloque de cifrado y el de expansión de llave.

Tal como se esperaba, la arquitectura básica tiene valores medios de frecuencia de operación y área, mientras que pipeline posee la mayor frecuencia de operación y área, y compacta presenta los valores mínimos para estas dos figuras de mérito.

Para la frecuencia de trabajo de 100 MHz, la herramienta de síntesis implementó a la matriz de estados como bloques RAM en la arquitectura compacta, tal como se esperaba. Sin embargo, cuando se sintetiza esta arquitectura a su máxima frecuencia de operación la herramienta no utiliza este tipo de estructura debido a que posee un mayor retardo al compararse con la memoria distribuida.

En cuanto a las diferencias entre las dos implementaciones de la arquitectura básica, puede observarse que la expansión de llave *on-the-fly* requiere menos área, debido a que

en este caso se prescinde del arreglo de llaves de ronda. En cuanto a las diferencias entre las dos implementaciones de la arquitectura pipeline, puede observarse que la diferencia radica en la cantidad de registros utilizados. Estos registros son los agregados entre las transformaciones de ronda. Por otro lado, la frecuencia máxima es mayor en la implementación que cuenta con pipeline interno, tal como se esperaba.

Puede observarse que para las implementaciones en las cuales se hace un preálculo de las llaves de ronda el área requerida por la arquitectura compacta dista de ser cuatro veces menor al área de la arquitectura básica. Esto se debe a que, a pesar de que se implementó una cuarta parte de la lógica correspondiente a una ronda, la lógica requerida para control, expansión y almacenamiento de llave está presente en ambas. Por otro lado, esta diferencia es más notoria para las implementaciones en las cuales se realiza un manejo de llavaes *on-the-fly* ya que no cuentan con el arreglo de llaves de ronda.

A continuación se muestra la cantidad de recursos que requiere cada sub-bloque del módulo de cifrado al sintetizar todas las arquitecturas propuestas, considerando una frecuencia de operación de 100MHz. Se observa que para ciertos recursos existe una diferencia entre el valor total y la suma de los recursos utilizados en cada sub-bloques, esto se debe la presencia de lógica en el bloque de mayor jerarquía en el que que instancian los bloques de control, expansión de llave y cifrado.

Tabla 8.3: Recursos por sub-bloque para la arquitectura básica a 100MHz

Bloque	LUTs	SRLs	FFs	RAMB18	DSP
Wrapper AHB	188	0	609	0	0
Control	5	0	2	0	0
Cifrado	1359	0	1683	0	0
Expansión	428	0	269	0	0
Total	1978	0	2563	0	0

Tabla 8.4: Recursos por sub-bloque para la arquitectura básica OTF a 100MHz

Bloque	LUTs	SRLs	FFs	RAMB18	DSP
Wrapper AHB	205	0	609	0	0
Control	4	0	130	0	0
Cifrado	903	0	269	0	0
Expansión	519	0	269	0	0
Total	1532	0	1276	0	0

A partir de los resultados mostrados también se observa que el envoltorio AHB de la arquitectura básica requiere mayor cantidad de recursos al compararse con las demás. Esto se debe a que para esta arquitectura el envoltorio cuenta con tres registros de 32 bits

Tabla 8.5: Recursos por sub-bloque para la arquitectura compacta a 100MHz

Bloque	LUTs	SRLs	FFs	RAMB18	DSP
Wrapper AHB	68	0	167	0	0
Control	6	0	5	0	0
Cifrado	877	0	1520	4	0
Expansión	273	0	175	0	0
Total	1257	0	1867	4	0

Tabla 8.6: Recursos por sub-bloque para la arquitectura compacta OTF a 100MHz

Bloque	LUTs	SRLs	FFs	RAMB18	DSP
Wrapper AHB	68	0	167	0	0
Control	49	0	5	0	0
Cifrado	426	0	182	4	0
Expansión	219	0	175	0	0
Total	780	0	529	4	0

Tabla 8.7: Recursos por sub-bloque para la arquitectura compacta OTF a 100MHz - Arquitectura compacta para MixColumns

Bloque	LUTs	SRLs	FFs	RAMB18	DSP
Wrapper AHB	86	0	167	0	0
Control	49	0	5	0	0
Cifrado	296	0	246	4	0
Expansión	200	0	175	0	0
Total	651	0	593	4	0

Tabla 8.8: Recursos por sub-bloque para la arquitectura pipeline a 100MHz

Bloque	LUTs	SRLs	FFs	RAMB18	DSP
Wrapper AHB	76	0	165	0	0
Control	15	0	3	0	0
Cifrado	6778	0	2956	0	0
Expansión	405	0	269	0	0
Total	7529	0	3393	0	0

en los que se almacenan los *words* del texto plano a medida que ingresan por la interfaz AHB. Para las arquitecturas pipeline y compacta estos registros no son necesarios ya que, debido a que el ancho del bus del módulo de cifrado coincide con el ancho del bus de datos del protocolo AHB, la conexión es directa.

La diferencia en cuanto al tamaño del bloque de control entre las implementaciones que consideran un manejo de llave con precálculo al compararse con aquellas que considera manejo de llaves *on-the-fly* se debe a que este último caso el módulo de control almacena la llave secreta cuando ingresa y la envía posteriormente al bloque de expansión, al ingresar un bloque de texto plano. Por otro lado, la diferencia en el tamaño del bloque de cifrado se debe al arreglo en el que se almacenan las once llaves de ronda.

8.3. Verificación funcional post-implementación

Una vez finalizado el proceso de Ubicación y Ruteo se llevaron a cabo simulaciones considerando los retardos totales de cada señal, incluyendo el tiempo de propagación de cada bloque lógico configurable así como el retardo aportado por la interconexión entre ellos. Estas simulaciones presentan gran importancia ya que son las que más se acercan al comportamiento real del sistema.

En esta instancia se corrieron los tests dirigidos. Los resultados obtenidos fueron favorables, pudiendo de esta manera asegurar el funcionamiento de las distintas arquitecturas propuestas en la FPGA objetivo, utilizando el oscilador de 100MHz integrado en la placa de desarrollo.

8.4. Validación

La validación de las arquitecturas se realizó mediante la programación de la FPGA con el archivo de configuración obtenido al sintetizar e implementar el sistema que se ilustra en la Figura 8.2, denominado *AES_top*. El mismo consta de un maestro, que al salir de su estado de reset envía en primer lugar una llave secreta, seguida de una secuencia de 32 bloques de texto plano. Al finalizar el envío del último, comienza nuevamente por el primero. El bloque AES recibe estos estímulos, y genera los bloques de texto cifrado correspondientes.

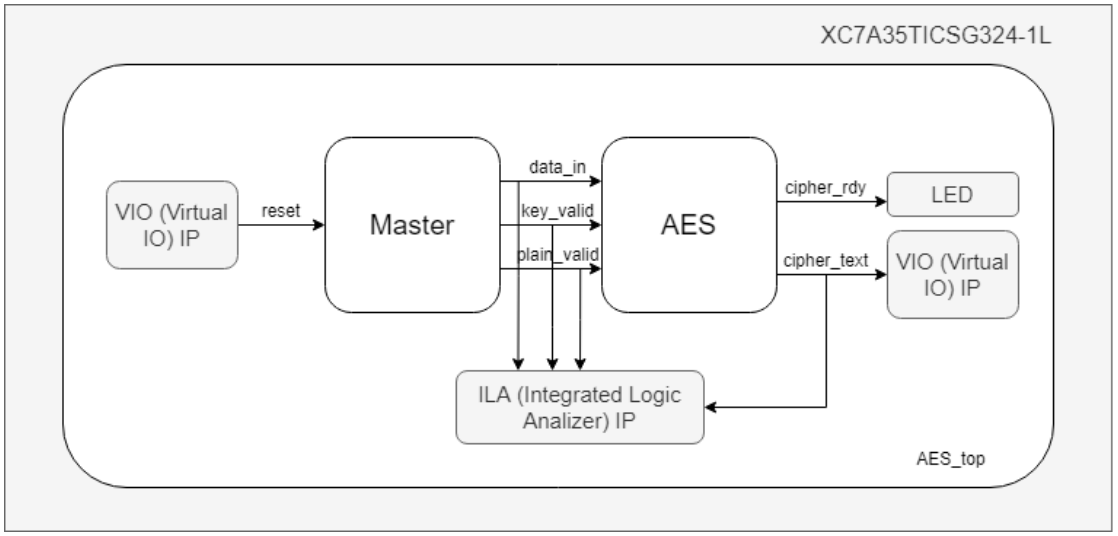


FIGURA 8.2: Sistema utilizado en la validación.

Se utilizaron IPs de Xilinx para poder controlar y observar los estímulos y resultados de la FPGA desde en entorno de Vivado: se utilizó una entrada virtual (VI) para controlar el valor de reset del maestro y así determinar el inicio de la secuencia de estímulos. Por otro lado, se conectaron las salidas del bloque AES (texto cifrado y su señal de habilitación) a una salida virtual (VO). La salida que indica que el bloque AES está listo para recibir un nuevo bloque de texto plano (*cipher_rdy)* se conectó a uno de los LEDs de la placa utilizada, de manera de poder observarla principalmente en los momentos en los que el maestro está en estado de reset y no envía estímulos. Por último, se utilizó un Analizador Lógico Integrado (ILA) para poder visualizar desde Vivado el valor de las señales internas de la FPGA y así verificar que el funcionamiento del bloque AES se corresponde con el esperado.

Capítulo 9

Conclusiones

En el presente trabajo de describió el desarrollo de tres arquitecturas para un módulo de cifrado AES. Estas arquitecturas se denominaron básica, compacta y pipeline, cada una de ellas orientada a un rango de aplicaciones.

El desarrollo de la arquitectura básica se basó en una implementación iterada, en la que se ejecuta una ronda del algoritmo de cifrado por ciclo de reloj. Se implementaron dos versiones de esta arquitectura utilizando dos enfoques diferentes para el manejo de las llaves de ronda: precálculo y *on-the-fly*.

El desarrollo de la arquitectura pipeline se basó en el procesamiento paralelo de múltiples bloques de texto plano, obteniendo de esta manera un desempeño óptimo. Se implementaron dos versiones de esta arquitectura: una en la que se realiza paralelismo entre rondas y una segunda en la que se paralelizan tanto las rondas como las transformaciones ejecutadas en cada una de estas rondas. El manejo de llaves en ambos casos sigue un esquema de precálculo ya que, al poseer un alto grado de paralelismo, en el caso en el que la estructura interna está completa se requieren las once llaves de ronda en un mismo ciclo de reloj.

El desarrollo de la arquitectura compacta se basó en una implementación iterada en la que se ejecuta la transformación de una columna de la matriz de estados por ciclo de reloj, razón por la cual se requieren cuatro ciclos de reloj para ejecutar una ronda completa. Se implementaron dos versiones de esta arquitectura utilizando dos enfoques diferentes para el manejo de las llaves de ronda: precálculo y *on-the-fly*. Por otro lado, para la arquitectura compacta se implementó una tercer versión, en la que se utilizó una

arquitectura distinta para implementar la transformación de ronda MixColumns. En esta tercer implementación la transformación MixColumns requiere una menor cantidad de recursos, pero su ejecución demora cuatro ciclos de reloj.

Para cada una de las arquitecturas propuestas se diseñó también un envoltorio que adapta las señales de entrada y salida del módulo de cifrado al estándar AHB, facilitando de esta manera su inserción en módulos de mayor complejidad.

La verificación funcional de las arquitecturas propuestas, utilizando tanto tests dirigidos como aleatorios, permite asegurar el correcto funcionamiento de las arquitecturas diseñadas y su cumplimiento con el estándar de NIST.

Una vez verificada su funcionalidad, las siete arquitecturas propuestas fueron sintetizadas utilizando el entorno Vivado de Xilinx. En todos los casos se estableció como dispositivo objetivo la placa de desarrollo ARTY, que cuenta con una FPGA Xilinx Artix-35T (xc7a35ticsg324-1L). La frecuencia de operación establecida en los *constraints* de temporizado es de 100MHz, correspondiente a la frecuencia de operación del oscilador con el que cuenta la placa de desarrollo ARTY. La validación de las arquitecturas propuestas se realizó programando la FPGA con un archivo de configuración obtenido al sintetizar e implementar un sistema que consta del módulo AES junto con un maestro que lo estimula y un analizador lógico integrado (ILA) que permitió la visualización de las señales de entrada y salida del módulo AES desde el entorno de Vivado.

A partir de los resultados se concluye que las tres arquitecturas cumplen holgadamente con los requisitos de desempeño impuestos por las aplicaciones objetivo planteadas al inicio del trabajo: VoIP para la arquitectura básica, Wi-Fi para pipeline e IoT para compacta, tal como se ilustra en la Tabla 9.1. De todas formas, es importante destacar que el desempeño que se notifica en la tabla es el máximo admisible para cada arquitectura considerando un protocolo de comunicación AHB entre el módulo de cifrado y su maestro. Estos valores pueden verse afectados si la comunicación se realiza utilizando otro protocolo o si la comunicación AHB no puede realizarse paralelizando las fases de datos y dirección de dos transacciones consecutivas.

Tabla 9.1: Comparación entre las arquitecturas propuestas.

	Básica	Compacta	Pipeline
Aplicación	VoIP	IoT	WiFi (IEEE 802.1)
Desempeño requerido	15-64 Kbps	0.3-50 Kbps	11-1300 Mbps
Desempeño máximo obtenido	673 Mbps	232 Mbps	12800 Mbps

Por otro lado, se concluye además que el esquema de manejo de llaves posee una importante influencia en cuanto a la cantidad de recursos que requiere cada implementación. Esto se debe a que el tamaño del arreglo necesario para almacenar las llaves de ronda es considerable al compararse con el tamaño del bloque de cifrado. De todas formas se decidió implementar ambas opciones ya que el manejo de llaves *on-the-fly* puede no ser óptimo en los casos en los que la llave secreta no es modificada con frecuencia.

Con respecto a las dos implementaciones de la transformada de ronda MixColumns para la arquitectura compacta se concluye que la segunda versión, en la que se intentó reducir el área, posee un impacto negativo en el desempeño mucho mayor al impacto positivo en área y frecuencia de operación. Por este motivo, su utilización solo es justificable en casos extremos en los que se necesita una frecuencia de operación mayor que la alcanzable por la implementación que paraleliza las transformaciones de las cuatro columnas.

El desarrollo de este trabajo significó un aprendizaje en cuanto al impacto que la estrategia de diseño tiene sobre la latencia y desempeño de un bloque, así como el efecto de las restricciones de síntesis sobre el resultado final y el correcto análisis de los reportes brindados por el software de Xilinx. Por otro lado, implicó un acercamiento a la criptografía, al álgebra de campos finitos y al protocolo de comunicación AMBA AHB. Las siete arquitecturas propuestas forman parte de la biblioteca de IPs (bloques de Propiedad Intelectual) del Centro de Micro y Nanoelectrónica perteneciente al Instituto Nacional de Tecnología Industrial.

Apéndice A

Introducción a la teoría de campos finitos

Los campos finitos, también llamados campos de Galois, son utilizados en aplicaciones como códigos de corrección de errores y criptografía. En este apéndice se presenta una descripción básica de los campos finitos, remarcando las características que son importantes desde el punto de vista de la criptografía. En el algoritmo AES, las transformaciones *SubBytes/InvSubBytes* y *MixColumns/InvMixColumns* se basan en operaciones aritméticas en el campo finito $GF\{2^8\}$.

A.1. Campos finitos de la forma $GF\{2^n\}$

Un campo finito F es un conjunto formado por una cantidad finita de elementos y dos operaciones, denominadas $\oplus$ y $\otimes$, que satisfacen las siguientes propiedades (Canright, 2005):

1. El conjunto es cerrado con respecto a ambas operaciones:

   a) $a \oplus b \in F$

   b) $a \otimes b \in F$.

2. Ambas operaciones son asociativas:

   a) $(a \oplus b) \oplus c = a \oplus (b \oplus c)$

   b) $(a \otimes b) \otimes c = a \otimes (b \otimes c)$.

3. Ambas operaciones son conmutativas:

   a) $a \oplus b = b \oplus a$

   b) $a \otimes b = b \otimes a$.

4. Las operaciones obedecen la ley distributiva:

   $(a \oplus b) \otimes c = (a \otimes c) \oplus (b \otimes c)$.

5. Cada operación tiene una identidad:

   a) $a \oplus 0 = a$

   b) $a \otimes 1 = a$.

6. Cada elemento a posee una inversa aditiva q tal que $a \oplus q = 0$. La notación estándar para la inversa aditiva de un elemento a es $-a$.

7. Cada elemento $a \neq 0$ posee una inversa multiplicativa r tal que $a \otimes r = 1$. La notación estándar para la inversa multiplicativa de un elemento a es a^{-1}.

Dado un conjunto finito de elementos, la definición de dos operaciones con las características recién mencionadas sólo es posible si la cantidad de elementos es de la forma p^n, donde p es un número primo y n un entero positivo. En este caso, p^n se denomina el orden y p la característica del campo. Si un subconjunto de un campo también es un campo en sí mismo, considerando las mismas operaciones, entonces se lo denomina subcampo.

Al considerar un conjunto de p^n elementos, existe más de una manera de definir las operaciones que producen un campo. Estas diferentes expresiones dan como resultado campos que son isomorfos. La estructura de estos campos se mantiene igual, la diferencia radica en el nombre utilizado para las operaciones y elementos que lo componen.

El campo finito más simple es $GF\{2\} = \{0, 1\}$. En la Tabla A.1 se muestra el resultado obtenido al operar entre los dos elementos de este conjunto. A partir de la misma se observa que, en este campo finito, cada elemento es su propia inversa aditiva. Esta propiedad se mantiene para todos los campos de la forma $GF\{2^k\}$.

Existen diferentes formas de representar a los elementos pertenecientes a un campo finito:

$\otimes$	0	1
0	0	0
1	0	1

$\oplus$	0	1
0	0	1
1	1	0

TABLA A.1: Operaciones en el campo finito $GF\{2\}$

- **Potencias de un elemento primitivo:** Un elemento a perteneciente al campo finito $GF\{p^n\}$ se denomina primitivo si todas sus potencias son diferentes, es decir, $a^0 \neq a^1 \neq a^2 \neq ... \neq a^{p^n-2}$. Por lo tanto, las potencias de un elemento primitivo dan como resultado todos los elementos ($\neq 0$) del campo finito. Todo campo finito tiene al menos un elemento primitivo.

  Esta manera de representar a los elementos presenta una ventaja al realizar el producto entre dos elementos pertenecientes al campo, ya que se limita a la suma de los exponentes de cada uno, módulo $p^n - 1$.

- **Polinomios sobre $GF\{p\}$:** Cada elemento perteneciente a un campo finito de la forma $GF\{p^n\}$ se representa como un polinomio de grado menor a n, cuyos coeficientes pertenecen a $GF\{p\}$. Al usar esta representación, la suma entre dos elementos consiste en la suma de sus coeficientes (módulo p), mientras que la multiplicación debe ser realizada módulo un polinomio irreducible de grado n.

  Por lo general, un polinomio de la forma $b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x + b_0$ se representa mediante sus coeficientes, en la forma $b_7b_6b_5b_4b_3b_2b_1b_0$.

  Si se considera un campo finito F de orden p^n que contiene un subcampo S de orden $r = p^j =$, siendo $n = jk$, los elementos de F pueden ser representados como polinomios de grado menor a k, con coeficientes en S (polinomios sobre S). Esta representación se denota F/S. La suma entre dos elementos consiste en la suma de los coeficientes, en el campo finito S, y la multiplicación se lleva a cabo módulo un polinomio irreducible de grado k, cuyos coeficientes también pertenecen al campo S. Por ejemplo, $GF\{5^6\} = GF\{5^2\}^3$. Por lo tanto, los elementos de $GF\{5^6\}$ pueden ser representados como polinomios de grado menor a 3, cuyos coeficientes pertenecen a $GF\{5^2\}$, módulo el polinomio $q(x) = x^3 + x^2 + x + 3$ (irreducible sobre $GF\{5^2\}$).

- **Espacio vectorial de dimensión n:** Se considera un campo $GF\{p^n\}$ como un espacio vectorial, junto con la suma entre vectores y el producto por un escalar perteneciente a $GF\{p\}$. La base de este espacio está conformada por n elementos

linealmente independientes $\{b_1, b_2, ..., b_n\}$ pertenecientes a $GF\{p^n\}$. Cada elemento del campo se expresa como una combinación lineal de estos elementos: $a = c_1 \otimes b_1 \oplus c_2 \otimes b_2 \oplus ... \oplus c_n \otimes b_n$ $(c_i \in GF\{p\})$.

A.2. Aritmética en $GF\{2^8\}$

La aritmética en campos finitos en el algoritmo de cifrado AES se realiza utilizando la representación polinomial. En esta sección se describe la forma en la que se llevan a cabo la suma, multiplicación e inversión de elementos (Kak, 2014).

Suma y resta

En aritmética en el campo $GF\{2\}$, la suma de dos valores corresponde a la XOR entre ellos. Por lo tanto, al sumar dos polinomios de grado 7, el resultado corresponde a la suma de sus respectivos coeficientes, es decir, a la XOR entre ellos. Como cada elemento es su propia inversa aditiva, en el campo $GF\{2\}$ la resta equivale a la suma.

A continuación se muestra un ejemplo que ilustra este tipo de operación

$$(x^7 + x^4 + x^2 + 1) + (x^5 + x^3 + x^2) = x^7 + x^5 + x^4 + x^3 + 1$$
$$(x^7 + x^4 + x^2 + 1) - (x^5 + x^3 + x^2) = x^7 + x^5 + x^4 + x^3 + 1$$
$$(10010101 \oplus 00101100 = 10111001).$$

Multiplicación

La multiplicación entre dos elementos pertenecientes a $GF\{2^8\}$ debe realizarse módulo un polinomio irreducible de grado 8, para asegurar que el resultado obtenido también pertenezca al campo mencionado.

Existe un método para determinar el resultado de la multiplicación entre un elemento arbitrario f y el polinomio x (elemento 00000010) de una manera sencilla, sin necesidad de calcular explícitamente el producto entre polinomios seguido de la división por el polinomio irreducible. Este método depende estrictamente del polinomio irreducible utilizado. En la Ecuación (A.1) se lo detalla considerando el polinomio irreducible

$q(x) = x^8 + x^4 + x^3 + x + 1$, utilizado en el algoritmo de cifrado AES.

$$f(x) \times x = (b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x + b_0) \times x \tag{A.1}$$
$$= \begin{cases} b_6x^7 + b_5x^6 + b_4x^5 + b_3x^4 + b_2x^3 + b_1x^2 + b_0x \text{ si } b_7 = 0 \\ (b_6x^7 + b_5x^6 + b_4x^5 + b_3x^4 + b_2x^3 + b_1x^2 + b_0x) \oplus (x^4 + x^3 + x + 1) \text{ si } b_7 = 1. \end{cases}$$

Para computar la multiplicación entre f y un elemento aleatorio b a partir del método descripto, se descompone a b en una serie de patrones de ocho bits conformados por un uno y siete ceros. Por ejemplo, si $b = x^7 + x + 1 = 10000011$ entonces:

$$\begin{aligned} f \times b &= f \times 10000011 \\ &= (f \times 00000001) \oplus (f \times 00000010) \oplus (f \times 10000000). \end{aligned}$$

La multiplicación individual entre f y un patrón que contiene un solo uno en la posición *j-ésima*, usando el índice $j = 0$ para identificar al bit menos significativo, se logra realizando j aplicaciones de la Ecuación A.1.

Inversa multiplicativa

Existen distintos métodos para computar la inversa multiplicativa de un elemento perteneciente al campo finito $GF\{2^8\}$, la que se analiza a continuación está basada en el método de Euclides para hallar el máximo divisor común (gcd) entre dos valores, y la Identidad de Bezout, que se enuncia a continuación.

Identidad de Bezout. Si a y n forman cualquier par de enteros positivos, existen enteros x e y tal que $gcd(a, n) = x \times a + y \times n$.

Se observa que x e y pueden no ser únicos. Si se considera que a y n son primos entre sí, entonces $gcd(a, n) = 1$. Por lo tanto,

$$x \times a + y \times n = 1. \tag{A.2}$$

Debido a que y es un valor entero, si se considera la Ecuación (A.2) módulo n se obtiene

$$x \times a = 1. \tag{A.3}$$

A partir de la Ecuación (A.3) se desprende que x es la inversa multiplicativa de a, en aritmética módulo n. Para hallar el valor de a suele usarse el Método Extendido de Euclides, que halla el máximo divisor común entre dos valores, en la forma dada por la Identidad de Bezout.

Algoritmo de Euclides

El algoritmo de Euclides para hallar el máximo divisor común entre dos enteros se basa en las siguientes observaciones:

- $gcd(a, a) = a$.
- *Si* $a = k \times b$ *entonces* $gcd(a,b) = b$.
- $gcd(a, 0) = a$.
- $gcd(a, b) = gcd(b, a \bmod b)$.

A continuación se muestran los pasos a seguir en el algoritmo de Euclides.

$$\begin{aligned} gcd(b_1, b_2) &= gcd(b_2, b_1 \bmod b_2) = gcd(b_2, b_3) \\ &= gcd(b_3, b_2 \bmod b_3) = gcd(b_3, b_4) \\ &\vdots \end{aligned}$$

Hasta que $b_{m-1} \bmod b_m = 0 \rightarrow gcd(b_1, b_2) = b_m$.

En los pasos mostrados se asume que $b_2 < b_1$, de todas formas, el algoritmo funciona para cualquier par de enteros positivos. En caso de que $b_1 < b_2$, la primer iteración intercambia la posición entre ellos.

Cuando el argumento más pequeño en una llamada a $gcd()$ es 1, no hay necesidad de iterar una vez más hasta que uno de los dos argumentos sea 0. En este caso los dos enteros b_1 y b_2 son primos entre sí y $gcd(b_1, b_2) = 1$.

A continuación se muestran dos ejemplos en los que se aplica el algoritmo de Euclides:

$$\begin{aligned} gcd(70,38) &= gcd(38,32) \\ &= gcd(32,6) \\ &= gcd(6,2) \\ &= gcd(2,0) \rightarrow gcd(70,38) = 2gcd(8,17) \qquad = gcd(17,8) \\ &= gcd(8,1) \\ &= gcd(1,0) \rightarrow gcd(70,38) = 1. \end{aligned}$$

Como se vio en la sección anterior, es necesario obtener el máximo divisor común entre dos elementos, en la forma dada por la Identidad de Bezout, con el fin de hallar la inversa multiplicativa de uno de ellos. Por lo tanto, a continuación se vuelve a analizar el algoritmo de Euclides, expresando el resto en cada iteración en la forma $a \times x + n \times y$.

$$\begin{aligned} gcd(b_1, b_2) = & \qquad\qquad\qquad\qquad\qquad\qquad \big| \text{Se asume que } b_2 < b_1 \\ &= gcd(b_2, b_1 \bmod b_2) = gcd(b_2, b_3) \qquad \big| b_3 = b_1 - q_1 \times b_2 \\ &= gcd(b_3, b_2 \bmod b_3) = gcd(b_3, b_4) \qquad \big| b_4 = b_2 - q_2 \times b_3 \\ &= gcd(b_4, b_3 \bmod b_4) = gcd(b_4, b_5) \qquad \big| b_5 = b_3 - q_3 \times b_4 \\ &\vdots \\ &= gcd(b_{m-1}, b_m) \qquad \big| b_m = b_{m-2} - q_{m-2} \times b_{m-1} \end{aligned}$$

Hasta que b_m es 0 o 1.

Si $b_m = 0$ y $b_{m-1} > 1$, entonces b_1 y b_2 no son primos entre sí, por lo tanto no existe inversa multiplicativa de ninguno de los dos. En cambio, si $b_m = 1$ entonces b_1 posee inversa multiplicativa en aritmética módulo b_2.

Cada uno de los restos hallados en las iteraciones del algoritmo pueden ser expresados en función de b_1 y b_2:

$$\begin{aligned}
b_3 &= b_1 - q_1 \cdot b_2 \\
b_4 &= b_2 - q_2 \cdot b_3 \\
&= b_2 - q_2 \cdot (b_1 - q_1 \cdot b_2) \\
&= b_2 - q_2 \cdot b_1 + q_1 \cdot q_2 \cdot b_2 \\
&= -q_2 \cdot b_1 + (1 + q_1 \cdot q_2) \cdot b_2 \\
b_5 &= b_3 - q_3 \cdot b_4 \\
&= (b_1 - q_1 \cdot b_2) - q_3 \cdot (-q_2 \cdot b_1 + b_2(1 + q_1 \cdot q_2)) \\
&= b_1 + q_2 \cdot q_3 \cdot b_1 - q_1 \cdot b_2 - q_3 \cdot (1 + q_1 \cdot q_2)b_2) \\
&= (1 + q_2 \cdot q_3) \cdot b_1 - (q_1 - q_1 \cdot q_2 \cdot q_3 - q_3) \cdot b_2 \\
&\vdots \\
b_m &= (\ldots\ldots) \cdot b_1 + (\ldots\ldots) \cdot b_2.
\end{aligned}$$

Si el proceso de iteración se detiene cuando $b_m = 1$, entonces el término que acompaña a b_1 en la expresión de b_m corresponde a la inversa multiplicativa de b_1 en aritmética módulo b_2. Este proceso se denomina como Algoritmo de Euclides Extendido. A continuación se muestra un ejemplo:

$$\begin{aligned}
gcd(32, 17) &= gcd(17, 15) && \big|_{\text{residuo: } 15 = 1 \times 32 - 1 \times 17} \\
&= gcd(15, 2) && \big|_{\text{residuo: } 2 = (-1) \times 32 + 2 \times 17} \\
&= gcd(2, 1). && \big|_{\text{residuo: } 1 = 8 \times 32 - 15 \times 17}
\end{aligned}$$

A partir de la expresión del último residuo se observa que la inversa multiplicativa de 32 en aritmética módulo 17 es 8.

Apéndice B

Protocolo AMBA AHB-Lite 3

Los componentes básicos en un sistema AHB-Lite son:

- Un maestro, que provee señales de control para realizar transacciones.
- Múltiples esclavos, que responden a las acciones iniciadas por el maestro.
- Un decodificador de direcciones, que habilita al esclavo solicitado por el maestro.
- Un multiplexor, que selecciona la salida del esclavo solicitado por el maestro.

La interconexión entre estos elementos y las señales más importantes se muestra en la Figura B.1. En la misma se observa que el bus de direcciones está conectado a un decodificador que genera las señales de selección de cada uno de los esclavos, *HSEL_x*, mientras que el bus de datos está conectado directamente a todos los esclavos. Con la finalidad de evitar conexiones de tres estados, cada esclavo tiene su propio bus de datos de salida, que se conecta a la entrada del multiplexor. Éste último selecciona al esclavo que corresponde, dependiendo del bus de direcciones.

B.1. Señales de entrada y salida de un esclavo AHB-Lite

Las señales de entrada y salida desde el punto de vista de un esclavo se detallan en la Tabla B.1.

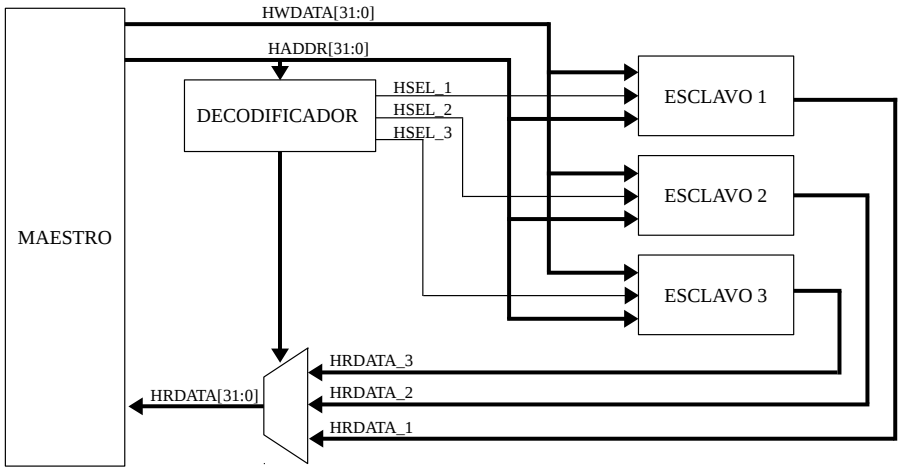


FIGURA B.1: Esquema de un sistema AHB-Lite.

TABLA B.1: Entradas y salidas de un esclavo AHB-Lite

Nombre	Fuente	Descripción
HCLK	Fuente de reloj	Reloj global del sistema
HRESETn	Controlador de reset	Reset gloal del sistema. Única señal activa con nivel lógico bajo.
HADDR[31:0]	Maestro	Bus de direcciones de 32 bits.
HBURST[2:0]	Maestro	Indica si una transferencia es simple o forma parte de una ráfaga.
HMASTLOCK	Maestro	Indica con nivel alto que la transferencia actual forma parte de una secuencia bloqueada.
HPROT[3:0]	Maestro	Indica si una transferencia forma parte de búsqueda de *opcode* o acceso de datos, y si está en modo de acceso privilegiado o de usuario.
HSIZE[2:0]	Maestro	Indica el tamaño de la transferencia.
HTRANS[1:0]	Maestro	Indica el tipo de transferencia (IDLE, BUSY, NONSEQUENTIAL o SEQUENTIAL).
HWDATA[31:0][1]	Maestro	Bus que transfiere datos desde el maestro hacia los esclavos durante transferencias de escritura.
HWRITE	Maestro	Indica la dirección de una transferencia. Es decir, si se trata de una lectura o escritura.
HSEL	Decodificador	Selección de los esclavos.
HREADY	Multiplexor	Indica mediante un nivel alto que la transferencia anterior está completa.
HRDATA[31:0] [2]	Esclavo	Bus que transfiere datos desde los esclavos hacia el maestro durante transferencias de lectura.
HREADYOUT	Esclavo	Indica con un nivel lógico alto la finalización de una transferencia. Esta señal es usada por el esclavo para extender una transferencia, mediante un nivel bajo.
HRESP	Esclavo	Provee información adicional al maestro sobre el estado de una transferencia.

B.2. Transferencias básicas

Las transferencias en AHB-Lite están conformadas por dos etapas: una de dirección y una de datos. Durante la primera, el maestro envía señales de control como HWRITE y HADDR indicando el tipo de transferencia y el esclavo con el que solicita interactuar. Durante la fase de datos, el esclavo responde a dicha transferencia. Esta forma de dividir las transferencias permite paralelizarlas, ya que mientras una transferencia se encuentra en la fase de datos, la próxima puede estar en fase de dirección.

Generalmente, las dos fases tienen una duración de un ciclo de reloj. En caso de que un esclavo requiera un tiempo mayor para completar sus tareas, debe proveer un nivel lógico bajo en la señal HREADYOUT durante la fase de datos. Cuando esto ocurre, la transferencia es extendida y el maestro mantiene el valor de sus señales de salida hasta que la transferencia finaliza. Como consecuencia del paralelismo, la fase de dirección de la transferencia siguiente también es extendida. Los esclavos sólo deben muestrear su señal HSEL si HREADY tiene un nivel lógico alto. Esto impide la ejecución de una transacción si la anterior aún no finalizó.

Escritura

Las Figuras B.2 y B.3 muestran diagramas temporales correspondientes a la escritura de un esclavo. En el segundo caso, el esclavo inserta un estado de espera, extendiendo la fase de datos de la transferencia con el esclavo A y, como consecuencia, la fase de dirección de la transferencia con el esclavo B. En ambos casos, en el primer flanco de reloj el maestro asigna los valores adecuados a las señales de control, identificando la transferencia que solicita. Estas señales son muestreadas en los esclavos en el segundo flanco de reloj. El esclavo que es seleccionado mediante el bus de direcciones y actúa según el caso:

- Para la Figura B.2, no requiere una extensión del ciclo de datos y, por lo tanto, en el tercer flanco del reloj escribe el valor presente en el bus HWDATA.
- Para la Figura B.3, requiere un tiempo extra para completar la transferencia. Por lo tanto, al reconocer la solicitud (segundo flanco de reloj) impone un nivel bajo en la señal HREADY y comienza a procesar la transferencia. Una vez que está listo

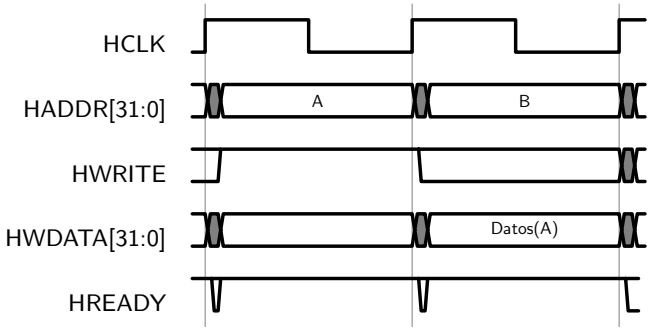


FIGURA B.2: Escritura sin espera.

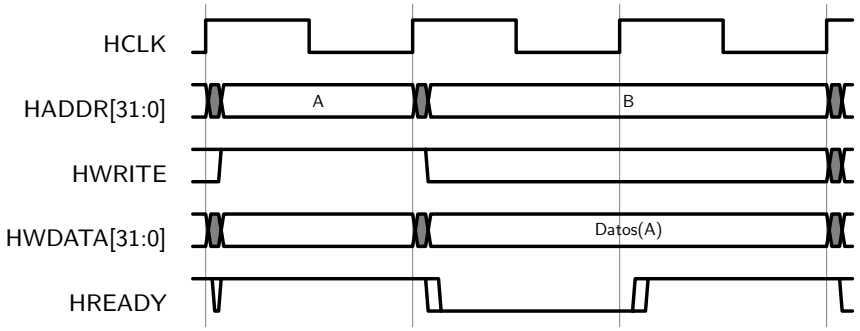


FIGURA B.3: Escritura con un ciclo de espera.

para finalizarla, vuelve HREADY a nivel alto indicando al maestro que puede continuar con la transferencia siguiente. En este caso, la escritura se produce en el cuarto flanco del reloj.

Para transferencias en las que el tamaño del dato a transferir es menor al ancho del bus, el maestro sólo tiene que asignar valores a los bytes correspondientes.

Lectura

Las Figuras B.4 y B.5 muestran diagramas temporales correspondientes a la lectura de un esclavo. En el segundo caso, el esclavo inserta un estado de espera, extendiendo la fase de datos de la transferencia con el esclavo A y, como consecuencia, la fase de dirección de la transferencia con el esclavo B. En ambos casos, en el primer flanco de reloj el maestro asigna los valores adecuados a las señales de control, identificando la transferencia que solicita. Estas señales son muestreadas en los esclavos en el segundo flanco de reloj. El esclavo que es seleccionado mediante el bus de direcciones y actúa según el caso:

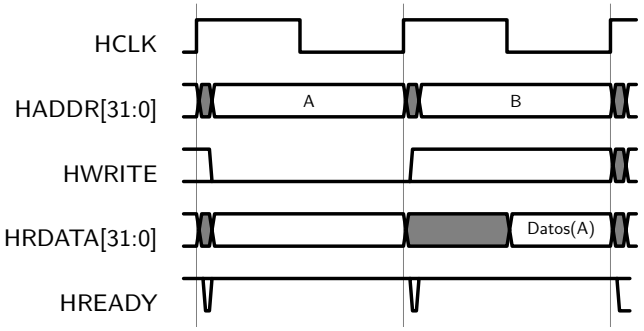


FIGURA B.4: Lectura sin espera.

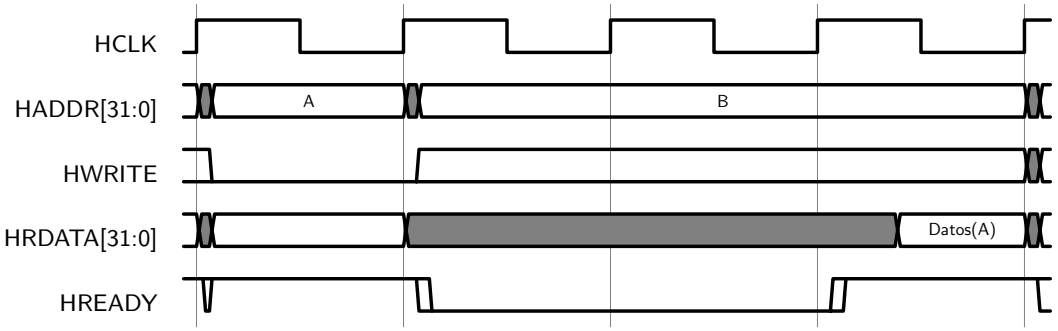


FIGURA B.5: Lectura con dos ciclos de espera.

- Para la Figura B.4, no requiere una extensión del ciclo de datos y, por lo tanto, en el tercer flanco del reloj el maestro muestrea el bus HRDATA y obtiene el valor que corresponde.
- Para la Figura B.5, requiere un tiempo extra para completar la transferencia. Por lo tanto, al reconocer la solicitud (segundo flanco de reloj) impone un nivel bajo en la señal HREADY y comienza a procesar la transferencia. Una vez que está listo para finalizarla, vuelve HREADY a nivel alto indicando al maestro que puede continuar con la transferencia siguiente. En este caso, el maestro lee el bus HRDATA en el quinto flanco del reloj.

Bibliografía

Accellera (2018). *Standard Universal Verification Methodologyl*. Disponible en: https://www.accellera.org/downloads/standards/uvm.

Alliance, LoRa (2018). *LoRaWAN 1.0.3 Specification*. Disponible en: https://lora-alliance.org/sites/default/files/2018-07/lorawan1.0.3.pdfl.

Anderson, Ross, Biham, Eli y Knudsen, Lars (1998). "Serpent: A Flexible Block Cipher With Maximum Assurance". En: *The First AES Candidate Conference*, págs. 589-606.

ANSII (2014). *Référentiel Général de Sécurité - Annexe B1*. Disponible en: http://www.ssi.gouv.fr/uploads/2015/01/RGS_v-2-0_B1.pdf.

ARM (2014). *AMBA 5 CHI Architecture Specification*.

ARM (2015). *ARM AMBA 5 AHB Protocol Specification*.

ARM (2017). *AMBA AXI and ACE Protocol Specification*.

ARM (2018a). *AMBA Generic Flash Bus Protocol Specification*.

ARM (2018b). *ARM AMBA Distributed Translation Interface (DTI) Protocol Specification*.

Battista Belasso, Giovan (1553). *La cifra del Sig. Giovan Battista Belasso*.

Boesgaard, Martin, Vesterager, Mette, Pedersen, Thomas, Christiansen, Jesper y Scavenius, Ove (2003). "Rabbit: A New High-Performance Stream Cipher". En: *10th Fast Software Encyption International Workshop*, págs. 307-329. ISBN: 978-3-540-20449-7.

Bogdanov, Andrey, Khovratovich, Dmitry y Rechberger, Christian (2011). "Biclique Cryptanalysis of the Full AES". En: *Advances in Cryptography - ASIACRYPT 2011*, págs. 344-371.

Burwick Carolynn, et al (1998). "MARS, a candidate cipher for AES". En: *First Advanced Encryption Standard (AES) Conference*.

Canniere, Christophe De y Preneel, Bart (2005). "TRIVIUM Specifications". En: *eSTREAM, ECRYPT Stream Cipher Project*.

Canright, David (2005). *A Very Compact Rijndael S-box*. Inf. téc. California: Naval Postgraduate School Monterey, págs. 3-6.

Chodowiec, Pawel (2002). "Comparison of the Hardware Performance of the AES Candidates Using Reconfigurable Hardware". Tesis de lic. George Mason University.

Chodowiec, Pawel y Gaj, Kris (2003). "Very Compact FPGA Implementation of the AES Algorithm". En: *Cryptographic Hardware and Embedded Systems-CHES 2003*, págs. 319-333. ISBN: 978-3-540-40833-8.

Daemen, Joan y Rijmen, Vincent (1998). "AES Proposal: Rijndael". En:

Diffie, Whitfield y Hellman, Martin E. (1976). "New Directions in Cryptography". En: *IEEE Transactions on Information Theory* IT-22.6. ISSN: 1557-9654.

ECRYPT (2012). *Yearly Report on Algorithms and Keysizes (2011-2012)*. Disponible en: http://www.ecrypt.eu.org/ecrypt2/documents/D.SPA.20.pdf.

Elbirt, Adam J, Yip, Wei y Paar, Christof (2001). "An FPGA Implementation and Performance Evaluation of the AES Block Cipher Candidate Algorithm Finalists". En: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 9.4, págs. 545-557. DOI: 10.1109/92.931230.

Elgamal, Taher (1985). "A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms". En: *IEEE Transactions on Information Theory* IT-31.4. DOI: 10.1109/TIT.1985.1057074.

Feistel, Horst (1973). "Cryptography and Computer Privacy". En: *Scientific American* 228, págs. 15-23.

Ichikawa, Tetsuka, Kasuya, Tomomi y Matsui, Mitsuru (2000). "Hardware Evaluation of the AES Finalists ". En: *The Third Advanced Encryption Standard Candidate Conference*, págs. 279-285.

Kak, Avi (2014). *Lecture Notes on "Computer and Network Security"*. Purdue University.

Kasiski Wilhelm, Friedrich (1863). *Die Geheimschriften und die Dechiffrierkunst*.

Kerckhoffs, Auguste (1883). "La Cryptographie Militaire". En: *Journal des Sciences Militaires* IX, págs. 5-38.

Koblitz, Neal (1987). "Elliptic Curve Cryptosystems". En: *Mathematics of Computation* 48.177, págs. 203-209. DOI: 10.1090/S0025-5718-1987-0866109-5.

Lai, Xuejia (1992). "On the Design and Security of Block Ciphers". Tesis doct. China: Xidian University.

Lu, Chih-Chung y Tseng, Shau-Yin (2002). "Integrated Design of AES (Advanced Encryption Standard) Encrypter and Decrypter". En: *Proceedings of the IEEE International Conference on Application-Specific Systems, Architectures and Processors*. DOI: 10.1109/ASAP.2002.1030726.

Menezes, Alfred (1996). *Handbook of Applied Cryptography*. CRC Press. ISBN: 0-8493-8523-7.

Miller, Victor S. (1985). "Use of Elliptic Curves in Cryptography". En: vol. V. Springer, págs. 417-426. DOI: 10.1007/3-540-39799-X_31.

NIST (1999). *Federal Information Processing Standards (FIPS) Publication 46-3: Data Encryption Standard (DES)*. Disponible en: http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf.

NIST (2001a). *Federal Information Processing Standards (FIPS) Publication 197: Advanced Encryption Standard (AES)*. Disponible en: http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf.

NIST (2001b). *Special Publication 800-38A: Recommendation for Block Cipher Modes of Operation*. Disponible en: http://csrc.nist.gov/publications/nistpubs/800-38a/sp800-38a.pdf.

NIST (2002). *The Advanced Encryption Standard Algorithm Validation Suite (AESAVS)l*. Disponible en: https://csrc.nist.gov/CSRC/media/Projects/Cryptographic-Algorithm-Validation-Program/documents/aes/AESAVS.pdf.

NIST (2007). *Special Publication 800-38D: Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC*. Disponible en: http://csrc.nist.gov/publications/nistpubs/800-38D/SP-800-38D.pdf.

NIST (2012a). *Special Publication 800-57: Recommendation for Key Management - Part 1: General*. Disponible en: http://csrc.nist.gov/publications/nistpubs/800-57/sp800-57_part1_rev3_general.pdf.

NIST (2012b). *Special Publication 800-67: Recommendation for the Triple Data Encryption Algorithm (TDEA) Block Cipher*. Disponible en: http://csrc.nist.gov/publications/nistpubs/800-67-Rev1/SP-800-67-Rev1.pdf.

Noo-Intara, P (2004). "Architectures for MixColumn Transform for the AES". En: *Icep2004, ICEP*.

NSA (2009). *National Security Agency Suit B Cryptography*. Disponible en: https://www.nsa.gov/ia/programs/suiteb_cryptography/index.shtml.

Paar, Christof y Pelzl, Jan (2010). *Understanding Cryptography*. Springer. ISBN: 978-3-642-44649-8.

Rijmen, Vincent (2000). *Efficient Implementation of the Rijndael S-box*. Katholieke Universiteit Leuven, Dept. ESAT. Belgium.

Rivest, R.L., Shamir, A. y Adleman, L. (1978). "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems". En: *Communications of the ACM* 21, págs. 120-126. DOI: 10.1145/359340.359342.

Rivest, Ronald L, Robshaw, MJB, Sidney, Ray y Yin, Yiqun Lisa (1998). "The RC6TM block cipher". En: *First Advanced Encryption Standard (AES) Conference*.

Satoh, Akashi, Kohji, Morioka, Takano, Kohji y Munetoh, Seiji (2001). "A Compact Rijndael Hardware Architecture with S-Box Optimization". En: *In Advances in Cryptology - ASIACRYPT 2001*, págs. 239-254.

Schneier, Bruce (1994). "Description of a New Variable-Length Key, 64-Bit Block Cipher (Blowfish)". En: *Fast Software Encryption, Cambridge Security Workshop Proceedings*, págs. 191-204. ISBN: 978-3-540-48456-1.

Schneier, Bruce, Kelsey, John, Whiting, Doug, Wagner, David, Hall, Chris y Ferguson, Niels (1998). En: *First Advanced Encryption Standard (AES) Conference*.

Shannon, Claude E. (1949). "Communication Theory of Secrecy Systems". En: *Bell Systems Technical Journal* 28, págs. 656-715.

Standaert, Francois-Xavier, Rouvroy, Gael, Quisquater, Jean-Jaques y Legat, Jean-Didier (2003). "Efficient Implementation of Rijndael Encryption in Reconfigurable Hardware: Improvements and Design Tradeoffs". En: *Cryptographic Hardware and Embedded Systems-CHES 2003*, págs. 334-350.

Voip-info.org (2018). *VoIP Bandwith consumption*. Disponible en: https://www.voip-info.org/bandwidth-consumption/l.

Wu, Hongjun (2004). "A New Stream Cipher HC-256". En: *11th Fast Software Encryption International Workshop*, págs. 226-244. ISBN: 978-3-540-22171-5.

Xilinx (2012). *Xilinx 7 Series FPGAs: The Logical Advantage*.

Xilinx (2018). *Vivado Design Suite User Guide: Using Constraints*.