Tecnológico de Monterrey

ADVANCED ENCRYPTION STANDARD
ENCRIPTACIÓN Y DESENCRIPTACIÓN

TE2002B.401
DISEÑO CON LÓGICA PROGRAMABLE

Profesores:

Rick Swenson Durie

Pedro Nájera García

3 de mayo de 2025

ADVANCED ENCRYPTION STANDARD

1. Introducción

Hoy en día la información se ha convertido en unos de los activos más valiosos a nivel mundial. Es de ese modo que, la protección de datos no es una opción, al contrario, se ha vuelto una necesidad. Desde tiempos muy remotos, el humano siempre ha buscado la manera de comunicarse manteniendo en todo momento la confidencialidad todo esto por medio de sistemas matemáticos complejos. Estos sistemas, denominados criptográficos, han evolucionado con el paso de los años. Pues, en un principio cada sistema de cifrado de texto, consistía en métodos rudimentarios, como el intercambio de algún carácter en particular por otro. No obstante, su seguridad se volvía vulnerable con los nuevos avances tecnológicos de la época. Es en base a lo mencionado anteriormente, se crean los sistemas criptográficos mucho más avanzados manteniendo la idea de origen, proteger la información a toda costa.

De ese modo es que surge uno de los sistemas más relevantes y ampliamente utilizados en la actualidad como es el algoritmo de seguridad AES, denominado por sus siglas en inglés *Advanced Encryption Standard.* Actualmente, el AES es el algoritmo de cifrado adoptado a nivel mundial, pues su presencia se ve reflejado en cada acción que el ser humano realiza. Desde pagar con un clic mediante su teléfono móvil, hasta enviar mensajes a otras partes del mundo, su relevancia ha hecho que toda información personal se encuentre altamente protegida. De ese modo, el propósito de este reporte es describir el proceso que se siguió para implementar dicho algoritmo de cifrado en el lenguaje descriptor de hardware como lo es VHDL.

2. Antecedentes (Data Encryption Standard)

Para poder comprender el algoritmo de cifrado AES, es pertinente mencionar a su predecesor. *Data Encryption Standard* fue de los primeros algoritmos ampliamente adoptados en los años 80s y 90s. No obstante, con los avances tecnológicos, su seguridad se volvió vulnerable ante posibles ataques. DES, por su siglas en inglés, consistía en un algoritmo simétrico, es decir, poseía la misma clave tanto para cifrar y descifrar el mensaje. Esta importante característica, hacía que tanto el receptor como el emisor conozcan y usarán la misma llave para ambos procesos. A parte de la característica mencionada anteriormente, a continuación se listan otras características importantes que describen el funcionamiento de lo que fue este algoritmo de cifrado:

- **Cifrado por bloques:** Este algoritmo se conoce por ser un algoritmo de cifrado por bloques, lo que implica que tanto la llave como el algoritmo, se aplican a un bloque de datos de manera simultánea. De esa forma, para cifrar o descifrar el mensaje, DES, agrupa el mensaje en bloques de 64 bits.
- **Varias rondas de cifrado:** El proceso de cifrado consiste en 16 rondas. Cada ronda consta de 4 modos diferentes, haciendo que cada bloque depende del bloque anterior. De la misma manera, el descifrado, consiste en el proceso inverso del cifrado. siguiendo los mismos pasos, pero invirtiendo el orden de aplicación de las claves.

- **Llave de 64 bits:** Este algoritmo utiliza una clave de 64 bits, no obstante el algoritmo genera otras 16 subclaves diferentes de 48 bits. Cada una siendo usada para las 16 rondas de cifrado. Es pertinente mencionar que, las claves se generan realizando permutaciones a la clave original.

3. Algoritmo de encriptación AES

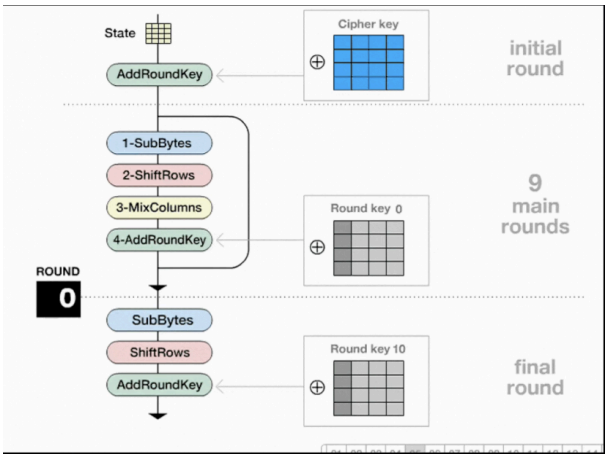
La evolución del cifrado AES, comenzó en los años 90s, cuando el NIST, *National Institute of Standard and Technology,* lanzó una convocatoria a nivel mundial con el fin de mejorar el algoritmo de cifrado. Se recibieron en total 15 propuestas, siendo la ganadora el algoritmo de Rijndael, lo que actualmente se conoce como el AES. La principales características de este fabuloso algoritmos son las siguientes:

- AES utiliza bloques de 128 bits, y permite el uso de claves de tres tipos en particular: 128, 192 y 256 bits. Esto permite una mayor flexibilidad, así también una mejor seguridad.
- Se caracteriza por ser un algoritmo con dualidad criptográfica. En otras palabras, tanto para cifrar como para descifrar el texto, es necesario usar la misma llave.
- Dicho algoritmo puede ser implementado tanto en software como en hardware. Este último punto es vital, debido a que, este proyecto consta en implementar este algoritmo en un lenguaje descriptor de hardware como lo es VHDL usando un FPGA.

4. Proceso de encriptación

Para poder iniciar este proceso este diagrama que se presenta a continuación, nos será de gran ayuda para una explicación mucho más clara.

Como primer paso es necesario tener un plaintext que está representado por “State” y nuestra llave que está representada por “Cipher Key”. Con esto es que empieza nuestro proceso, pero este se dividirá en tres principales parte:



Proceso de encriptación.

- **Initial round**

En este apartado o ciclo, recibiremos tanto el plain Text como la Cipher Key, dando así inicio al módulo de Key Schedule que nos ayudará a generar las subclaves que ocuparemos en los rounds siguientes. y por último entra al módulo de AddroundKey.

- **Main round**
  Esta parte tendrá en su interior todos los módulos del encriptador que son Subbytes,ShiftRows,Mixcolumns y Addroundkey. Esta secuencia de módulos previamente ordenada como se mencionó anteriormente se repetirá 9 veces para nuestro cifrado, cambiando en cada ronda la la Cipher Key que se generó con ayuda del módulo de Key Schedule.
- **Final round**
  Para la parte final del encriptador, solo hará un secuencia únicamente con los módulos de Subbytes, ShiftRows y AddroundKey, excluyendo el módulo de mixcolumns, una vez que termina esta secuencia tendremos la salida de nuestro plaintext completamente cifrado.

Como podemos observar la encriptación pasará por un total de 11 rounds para poder tener finalmente cifrado nuestro plainText. Gracias a la máquina de estados que se presentará más adelante, todo este proceso es posible debido a que esta nos ayuda a llevar el proceso de manera ordenada.

5. **Key Schedule**

Como primer módulo tenemos Key Schedule, el cual consiste en la expansión de la clave original de 128 bits en un conjunto de subclaves que serán utilizadas en cada ronda del algoritmo AES. Esta expansión permite que cada ronda del cifrado use una clave distinta, incrementando así la seguridad del sistema.

La implementación consta de dos bloques principales:

Generación de Palabras: La clave original se dividió en 4 palabras de 32 bits. A partir de la quinta palabra en adelante, cada nueva palabra se generó utilizando las anteriores, aplicando operaciones definidas por AES.

Transformaciones por Ronda: Para cada nueva palabra en la secuencia se aplicaron los siguientes pasos:

- **RotWord:**
  - Rotación de los bytes de una palabra 1 posición a la izquierda.
- **SubBytes:**
  - Sustitución no lineal de cada byte utilizando una tabla S-box.
- **XOR con Rcon:**
  - Se realizó una operación XOR con una constante de ronda específica para asegurar variabilidad.
- **XOR con palabras anteriores**:

    - Finalmente, se aplicó un XOR con una palabra generada previamente, completando el subclave.

También se añadió un sistema de control mediante señales Enable, Clk y Reset, que regula la generación secuencial de subclaves a través de un contador que indica la ronda actual. Se incluyó una señal Finish para marcar el final del proceso.

Uno de los principales desafíos fue asegurar la correcta dependencia entre las subclaves, ya que cada una se genera a partir de las anteriores. Inicialmente se intentó almacenar todas las subclaves en una sola estructura de 176 bytes (11 claves × 16 bytes), pero esto dificulta el acceso por ronda. Para resolverlo, optamos por usar una estructura de memoria SRAM con direccionamiento por ronda, lo que permitió acceder fácilmente al subclave correspondiente a cada iteración.

Para validar el funcionamiento del módulo, se implementaron pruebas en el testbench que verificaron la correcta generación de cada subclave. Las pruebas incluyeron:

- Comparación de las subclaves generadas contra vectores estándar definidos en AES.
- Verificación del efecto de cambios mínimos en la clave original, confirmando que las subclaves resultantes también cambian significativamente.
- Pruebas de sincronización para asegurar que las subclaves se generarán en los tiempos esperados sin errores de secuencia.

6. AddRoundKey

Como segundo módulo tenemos AddRoundKey el cual consiste en la implementación de un xor entre el texto que se recibe y la cipher key que tenemos en un inicio, después cuando haya más iteraciones se usará la llave brindada por el módulo Key Schedule.

La implementación consta de dos bloques principales:

- Separación de Bytes: Se dividieron tanto el texto como la clave en 16 bloques de 8 bits usando estructuras tipo SRAM. Esto facilita la manipulación individual de cada byte.
- Operación XOR: Cada byte del texto de entrada se combinó con el byte correspondiente de la clave mediante una operación XOR. El resultado se almacenó directamente en la salida TxtOut.

También se añadió un sistema de control mediante Enable, Clk y Rst, que permite controlar la ejecución del módulo. Se utilizó un contador de 2 bits y una señal Finish para indicar cuándo se ha completado la operación.

Uno de los principales desafíos fue organizar y manipular correctamente los 128 bits de entrada para que pudieran ser operados byte por byte. Inicialmente, intentamos hacer la

operación XOR directamente sobre los vectores de 128 bits, pero eso complicaba la interpretación de los resultados y su validación individual.

Para resolverlo, optamos por dividir los vectores en arreglos de 16 elementos de 8 bits cada uno, lo que permitió realizar la operación XOR de forma más clara y manejable.

Para garantizar el correcto funcionamiento del módulo, se implementaron pruebas en el testbench que verificó tanto la operación XOR a nivel de bits como la sincronización de las señales de control. Las pruebas incluyeron:

- Validación de la operación XOR mediante vectores de prueba estándar (como los definidos en el AES), confirmando que la salida fuera el resultado exacto de la operación bit a bit entre el texto y la clave.

- Pruebas de reversibilidad, aprovechando que el XOR es su propia inversa: al aplicar nuevamente la misma clave al resultado, se recuperó el texto original, validando así la consistencia del módulo.

7. SubBytes

El módulo de SubBytes es una etapa realiza una sustitución no lineal de cada byte en un bloque de 128 bits mediante una tabla conocida como S-Box. El objetivo es introducir confusión en el mensaje cifrado para fortalecer la seguridad del sistema.

El módulo SubBytes fue desarrollado utilizando el lenguaje VHDL, implementando una máquina de estados finita que controla el flujo de la operación en tres fases: inactividad (*IDLE*), procesamiento (*PROCESSING*) y finalización (*DONE*). Con la incorporación de la S-Box, va codificando una constante en forma de arreglo, que almacena los 256 valores estandarizados utilizados por el algoritmo AES. Durante la fase de procesamiento, el sistema recorre secuencialmente los 16 bytes del texto de entrada y, por medio de un contador, aplica la transformación correspondiente a cada byte utilizando la S-Box. Los resultados son almacenados en un registro temporal y enviados como salida solo al completarse la sustitución de todos los bytes. Todo esto es controlado por la señal de un reloj (*CLK*).

Para las pruebas que realizamos tenemos que validar el funcionamiento de nuestro módulo por lo que desarrollamos un testbench para simular diversas pruebas. En las pruebas principales, se aplicó un vector de prueba representativo tomado de un ejemplo típico de AES (x"193DE3BEA0F4E22B9AC68D2AE9F84808") y se monitoreó la salida hasta que la señal *Finish* indicará la finalización del proceso. También lo que se planteó fue el comportamiento cuando surgen cambios en la entrada cuando el Enable no está activo. Con estas pruebas confirmamos el buen comportamiento de la lógica secuencial y del uso de la S-Box, ante distintos escenarios.

Durante el desarrollo de este módulo nos enfrentamos con varios desafíos en el diseño y en la implementación uno de los principales problemas fue que inicialmente no se implementó una máquina de estados, por lo que la transformación de los 16 bytes del texto de entrada se intentaba realizar en una sola señal combinacional. Esto generaba un uso excesivo de recursos del FPGA, este llegó a ocupar un 7% del dispositivo, lo cual era ineficiente y poco escalable. Para resolver este problema, se decidió rediseñar el módulo utilizando una máquina de estados finita que dividiera el proceso en 16 ciclos de reloj, procesando un byte por ciclo. Esta modificación redujo la carga que tenía el FPGA, si no que también teníamos una mayor claridad en el control de flujo de los datos.

8. ShiftRows

El módulo de ShiftRows es una etapa del proceso de cifrado de datos dentro del algoritmo AES, este proceso trabaja con una matriz de 4x4 bytes,en donde cada celda de la matriz corresponde a un byte. Los bytes de cada fila se van desplazando cíclicamente hacia la izquierda una cantidad específica de posiciones.

La cantidad de posiciones que se desplazan varía en cada fila, en la primera fila permanece sin cambios, la segunda fila se desplaza una posición, la tercera dos y la cuarta fila tres posiciones.

El propósito de ShiftRows es aumentar la difusión de los datos cifrados, Busca asegurar que los bits del mensaje original se distribuyan de forma desordenada en el texto, para que no sea posible identificar patrones ni recuperar el mensaje original fácilmente.

El módulo fue desarrollado en VHDL, donde dentro de una arquitectura secuencial incluimos un máquina de estados finita (FSM) con tres estados: St0 (inicio) , St1 (procesamiento) y St2 (finalización). En el proceso principal separamos los 128 bits de entrada (TxtIn) en 16 bytes individuales siguiendo la misma estructura en todos los módulos, y luego realizamos el desplazamiento de filas como lo mencionamos anteriormente.

Cada una de las filas se manipula individualmente, donde asignamos los índices de los bytes, para después reconstruirlos en un vector de 128 bits (TxtOut). Para lograr un mayor control utilizamos una señal de Finish indicando cuando el módulo término el procesamiento y una señal Enable que activa la operación.

Para poder realizar pruebas y comprobar su funcionamiento, creamos un testbench en VHDL que simula el comportamiento del módulo bajo condiciones controladas. Las pruebas incluyeron:

- Un vector de prueba fijo: X"D42711AEE0BF98F1B8B45DE51E415230" qué comparamos con la salida esperada.
- Control de señales Clk, Rst y Enable para simular en ciclo de reloj y controlar el inicio de procesamiento.
- Observación de la señal Finish para verificar que el proceso terminó correctamente.
- Verificación visual del valor de TxtOut en simulación usando ModelSim.

Además, utilizamos la instrucción de reporte para confirmar en la consola que el testbench se había completado con éxito.

Durante la simulación del testbench, nos encontramos con un comportamiento donde la señal de TxtOut mostraba el resultado correcto brevemente y luego regresaba ceros. Este problema se debía a que el valor de resultado (result) era combinacional y no estaba registrado, por lo que al no mantenerse activo el Enable, el valor desapareció.

Nuestra solución fue almacenar el resultado en un registro (result_reg) durante el estado St1, y asignarlo como salida en TxtOut. Esto aseguró que el valor correcto se mantuviera visible en la simulación y mandar la salida correcta para el siguiente módulo.

9. MixColumns

El módulo de Mixcolumns es una transformación lineal que se aplica a cada columna de la matriz de estado, con el objetivo de difundir la información. Esta transformación consiste en una multiplicación matricial entre cada columna de la entrada (compuesta por 4 bytes) y una matriz fija, todo dentro del campo finito de Galois.

En el desarrollo del módulo, utilizamos VHDL como lenguaje de descripción de hardware. Considerando que las multiplicaciones en el campo finito implica tomar cada valor como un polinomio, aplicando el operador XOR de forma repetida, llegan a ser significativa costosas en términos de tiempo de procesamiento, por ello optamos por una implementación eficiente basada en memorias de solo lectura (*ROM*). En estas memorias se almacenan previamente los resultados de las multiplicaciones por dos y por tres, utilizadas en la transformación MixColumns, estas look-up tables reciben el nombre de G-Box en el código.

El diseño se estructuró en una arquitectura secuencial, controlada mediante una máquina de estados con tres fases principales: *idle*, *Processing* y *Finished*. el módulo permanece en estado *idle* (reposo) hasta que recibe la señal *Enable* = ‘1’ y *Rst* = ‘0’. Al activarse la señal *Enable*, el sistema transita al siguiente estado *Processing*, donde se realizan las operaciones de multiplicación y suma requeridas por Mixcolumns. Finalmente, al concluir el proceso, el sistema pasa al estado *Finished*, activa la señal *Finish* para indicar el término de la operación y entrega el resultado.

Para las pruebas de funcionamiento del módulo, realizamos una simulación en Questa. Para el código, empleamos una estructura que cuenta con un reloj ya que MixColumns funciona con lógica secuencial, el reset para reiniciar el módulo, el enable en ‘0’ mientras esté en reset y en ‘1’ para procesar datos. Para verificar si funcionaba de manera esperada usamos un caso de prueba, empleando un TxtIn (texto de entrada) con un vector de prueba que proporciona datos y espera para ver si coincide con el TxtOut (texto de salida). El vector de entrada que usamos como prueba fue “d4bf5d30e0b452aeb84111f11e2798e5”, y el texto de salida esperado y recibido fue “046681e5e0cb199a48f8d37a2806264c”; en caso contrario hubiera mostrado un mensaje de error. Para finalizar el proceso, incluimos una señal de finish, que funciona como una bandera que se encuentra en ‘1’ cuando termina el proceso, está listo el TxtOut, y se puede avanzar a la siguiente etapa del proceso.

10. Unidad de control

La unidad de control, también conocida como máquina de estados, es un módulo fundamental en este proyecto ya que es la encargada de indicarle a los módulos en qué momento entrar y cuando parar para dar paso al siguiente, esto lo que hace es que logremos tener un orden y se siga el proceso del AES y así encriptar correctamente, el mismo caso en la desencriptación.

La forma en la que está implementada la máquina de estados en vhdl es gracias a que se reciben señales de finish en cada módulo (Add Round Key, SubBytes, Shiftrows, Mixcolumns y Key Schedule), esto nos permite seguir el órden de AES y poder regresar un enable a los módulos para que puedan seguir los módulos que escojamos siguiendo el órden.

En la arquitectura definimos como señales a los Estados siguientes y al Estado actual, teniendo como valores a los estados posibles los cuales son los módulos del proyecto. También definimos los vectores de Select Key, y del Finish y Enable.

Teniendo esto definido, iniciamos con el proceso de definir qué módulo sigue, en este lo que se hace es si el Reset está en 1, el estado actual pasa a escoger una nueva llave pero si no está en 1 el Reset, el estado actual pasa al siguiente estado, es decir, cambia de módulo.

Finalmente se realiza el proceso fundamental de la máquina de estados ya que vamos iterando entre cada módulo siguiendo el proceso del AES, empezamos con un case, el cual funciona guardando en el Siguiente estado el valor que le sigue dependiendo el estado actual. Esto lo logramos explorando todas las opciones que puede haber en el estado actual y en el caso de que sea Key Select, escogemos como estado siguiente Add Round Key, después si es Add Round Key escogemos SubBytes como estado siguiente, después a Shiftrows, después Mix Columns y ahora repetimos los módulos para poder lograr todas las iteraciones que le corresponden al AES.

Para validar el correcto funcionamiento de la Unidad de Control, se realizaron pruebas de simulación utilizando testbenches en VHDL. En estas pruebas se observaron las señales de control (Enable, Finish, Select Key , Estado Actual y Estado Siguiente) para verificar que se activaran en el orden correcto según el flujo del algoritmo AES.

Se evaluó que la máquina de estados respondería correctamente al pulso de Reset, regresando al estado inicial, y que avanzará al siguiente módulo sólo cuando se recibía la señal de Finish del módulo anterior, además se verificó que en función del valor de Select Key, se eligieron correctamente los módulos que debían ejecutarse en cada iteración (AddRoundKey, SubBytes, ShiftRows, MixColumns, etc).

Estas pruebas nos permitieron asegurar que el orden de ejecución de los módulos fuera el esperado y que no se saltaran etapas del proceso AES, garantizando así la correcta coordinación del sistema de cifrado.

Durante la realización de este módulo nos enfrentamos a distintas complicaciones que afectan el correcto funcionamiento de la máquina de estados. Una de las principales dificultades fue en la programación de una prueba de máquina de estados donde el diseño llegaba a tener errores en los nombres de estados y señales, lo cual causaba que los módulos no se llevarán a cabo en el orden en el que se tenía que hacer. De igual manera no se controlaba la selección de claves y no se indicaba de manera correcta cuando finaliza el proceso. Estos problemas se pudieron resolver con una versión donde se utilizaban vectores para gestionar el Finish y Enable, así como una mejora en la lógica clara para el avance de estados. Ya con esto se nos permitió tener un control preciso del flujo del algoritmo AES.

11. Algoritmo de desencriptación AES

El algoritmo de desencriptación AES presenta varias similitudes con respecto a su contraparte de encriptación, sin embargo, se presentan algunos contrastes importantes.

Mientras que la encriptación aplica en cada ronda:

1. SubBytes → ShiftRows → MixColumns → AddRoundKey
2. Ronda final sin MixColumns

la desencriptación ejecuta exactamente los mismos pasos, pero en orden inverso:

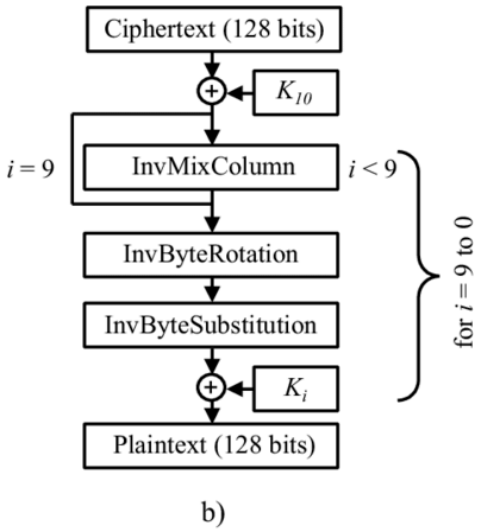
3. AddRoundKey → InvMixColumns → InvShiftRows → InvSubBytes
4. Ronda final sin InvMixColumns

Uso compartido de la expansión de clave

La generación de las subclaves es idéntica en ambos casos, pero la máquina de estados de descifrado las consume en orden descendente (desde la última hasta la primera), mientras que en cifrado van de la primera a la última.

Módulos similares

En hardware muchas de las estructuras (S-box, circuitos de mixcolumns, lógica de AddRoundKey) se reutilizan, solo que en descifrado se sustituyen por sus versiones “inversas”.



b)

Proceso de desencriptación

12. Sub Bytes Inverso

El módulo de inverse sub bytes funciona utilizando una tabla que contiene todos los valores de una tabla inversa a la utilizada en sub bytes, esta tabla se llama Inverse S Box. La manera en la que está destinada a funcionar el módulo es la siguiente:

- Se toma uno de los bytes de la matriz de 4x4 recibida
- Como consta de dos "dígitos", estos se separan y se obtienen dos valores por separado
- Estos dos valores se utilizan como coordenadas (x,y) para buscar dentro de la s-box inversa
- El valor que se encuentra en la coordenada encontrada se sustituye por el valor del byte original de la matriz recibida
- El proceso se repite con cada uno de los valores de la matriz de 16 bytes recibida.

El proceso anteriormente descrito es la manera en la que se tiene prevista que funcione este módulo de Sub Bytes Inverso. Sin embargo, nos encontramos con el problema de que esta implementación podría llevar más tiempo del necesario, debido a que se tenía que desarrollar un programa que fuese capaz de separar el contenido de un valor en dos y posteriormente hacer que el programa busque las coordenadas en una matriz. Todo lo anterior llevaría un proceso largo e inclusive complicado, refiriéndonos a la parte de planificación e implementación.

Sin embargo, pudimos darnos cuenta de una propiedad interesante de las S-box. Cada valor dentro de la s-box se encontraba en la i-ésima posición de la s-box, siendo "i" el valor original que se buscaba sustituir. Poniéndolo en un ejemplo con una de las pruebas que realizamos; si nosotros queríamos sustituir un byte de la matriz con valor de D4 (En hexadecimal), Si buscábamos la posición D4 (212 en decimal) dentro de la s-box, podíamos encontrar exactamente el mismo valor que hubiésemos encontrado si seguíamos el listado de pasos mencionado anteriormente. Esto nos ayudó a reducir la carga de trabajo debido a que ahora la s-box en lugar de ser una matriz, pasaría a ser un arreglo en el que se recorrería posición por posición hasta llegar al dato que se busca sustituir.

La implementación del código siguiendo la metodología mencionada funciona cuando el módulo recibe una señal de activación a través de el enable, empieza a convertir 2 bytes a la vez cada pulso de reloj. Cada pulso de reloj incrementa el contador que hace que los siguientes dos bytes tengan que ser sustituidos y hace esto hasta que todos los valores han cambiado. Una vez acabe manda una señal de finish al controlador y recibe otra señal para regresar a modo standby.

13. Shiftrows Inverso

El módulo de ShiftRows Inverso forma parte del proceso de descifrado de datos dentro del algoritmo AES, consiguiente al SubBytes Inverso donde buscamos revertir las alteraciones realizadas por la secuencia de ShiftRows realizados durante la etapa de cifrado. Al igual que en ella, este proceso trabaja con una matriz de 4x4 bytes, donde cada celda representa un

byte. Sin embargo, para este caso, los bytes de cada fila se desplazan cíclicamente hacia la derecha una cantidad específica de posiciones, realizando el procedimiento contrario al ShiftRows.

La cantidad de posiciones que se desplazan varía en cada fila: la primera fila permanece sin cambios, la segunda fila se desplaza una posición, la tercera dos y la cuarta fila tres posiciones. Recorriéndose por así decirlo el número de posiciones correspondiente a su índice (si empezamos la cuenta desde cero). Este proceso es esencial para restaurar el orden original de los datos antes del cifrado, y por tanto para permitir la recuperación del mensaje original.

El módulo fue desarrollado en VHDL, utilizando una arquitectura secuencial que incluye una máquina de estados finita (FSM) con tres estados: St0 (inicio), St1 (procesamiento) y St2 (finalización). Durante el proceso principal, se separan los 128 bits de entrada (*TxtIn*) en 16 bytes individuales, manteniendo la estructura aplicada en los demás módulos. Luego se realiza el desplazamiento inverso de las filas, tal como se describe en la lógica del algoritmo.

Cada fila es manipulada de forma individual, reasignando directamente los índices de los bytes para formar nuevamente la matriz en el orden correcto, y reconstruyendo el vector de 128 bits (*TxtOut*). Esta implementación fue diseñada específicamente para reducir al máximo el tiempo de procesamiento, evitando estructuras complejas y optimizando la lógica de reasignación de bytes mediante conexiones directas, lo cual facilita la síntesis y mejora el rendimiento general del proceso de descifrado. Para un control adecuado, se integraron las señales de *Finish*, que indica cuando el módulo ha finalizado el procesamiento, y Enable, que activa la operación.

Para comprobar el funcionamiento del módulo, se desarrolló un Testbench en VHDL que simula su comportamiento bajo condiciones controladas. Las pruebas incluyeron: Un vector de prueba fijo (obtenido con *Rijndael Inspector*): X"D42711AEE0BF98F1B8B45DE51E415230". Cuya salida esperada era X"7A89F2202F2B4358F23850467E9B105A". Control de señales *Clk*, *Rst* y *Enable* para simular el ciclo de reloj y controlar el inicio del procesamiento. Observación de la señal *Finish* para verificar que el proceso concluyó correctamente. Verificación visual del valor de *TxtOut* en simulación usando el simulador de Intel *Questa*. Uso de instrucciones de reporte para confirmar que el testbench finalizó de forma exitosa.

Durante la simulación del Testbench, se presentaron inicialmente algunos errores relacionados con la interpretación del vector de salida, debido a una confusión sobre si los bytes resultantes debían acomodarse por filas o por columnas al verificar el resultado en el simulador. Este detalle generó discrepancias al comparar contra los vectores esperados, ya que en versiones anteriores del diseño la reestructuración del estado no seguía el mismo orden que el utilizado por herramientas como *Rijndael Inspector*. Para resolverlo, se revisó cuidadosamente el mapeo de bytes asegurando que el orden aplicado al reconstruir el vector de salida *TxtOut* fuera estrictamente por filas, respetando el estándar del algoritmo AES. Una vez corregido este detalle, se logró una coincidencia precisa entre la salida del módulo y los

valores de referencia, validando la correcta implementación del desplazamiento inverso en cada fila de la matriz.

14. Mix Columns Inverso

El Módulo se basa en un multiplicación de matrices en un campo de Galois, los valores de las columnas se multiplican por 9, 11, 13 o 14 según sea el caso. Para lograr la multiplicación por estos números, se hace uso de polinomios de operaciones binarias. Siendo las únicas operaciones realidad individualmente la multiplicación por dos (por medio del corrimiento del byte a la izquierda una posición) y la suma (haciendo uso de XORs). Todos los valores por los que buscamos multiplicar son alcanzados aplicando estas dos operaciones en diversos órdenes.

Al entrar el valor a descifrar (TxtIn) se divide en las 4 columnas de 4 bytes de largo cada una, posterior a esto cada columna de forma paralela por la multiplicación por la matriz, esto haciendo las operaciones adecuadas por los valores de entrada y guardandolos en una columna de salida. Una vez las 4 columnas están listas se guardan en el vector de 16 bytes de salida.

Durante el desarrollo hubo errores al realizar las multiplicaciones, dando resultados muy distintos a los esperados en el test bench. Lo que hizo un poco más complicado el arreglar los errores fue como sólo conocemos los valores de entrada y de salida, tanto en los resultados de las pruebas como en nuestra base (es decir, no sabíamos qué valores producían las funciones individuales, solo la implementación de todas juntas). Afortunadamente, luego de rehacer una parte considerable del algoritmo de multiplicación se llegó a un resultado correcto y, como extra, más rápido de lo que esperamos.

15. Unidad de control

El módulo implementa una máquina de control que coordina de forma automática y secuencial todas las etapas necesarias para completar el proceso de descifrado AES-128. Su lógica se basa en tres ejes principales:

1. Orquestación de submódulos
    - Arranca activando la generación de claves derivadas y, una vez lista, pasa a aplicar la clave inicial al bloque de datos.
    - A continuación, entra en un bucle de rondas intermedias donde, en cada iteración, desencadena primero la operación de bytes inversos, luego el corrimiento de filas y, si no es la última ronda, la mezcla de columnas, antes de volver a aplicar la clave correspondiente.
    - Al llegar a la última ronda, omite el módulo mixcolumns y finaliza con la aplicación de la última subclave, tras lo cual permanece en un estado de “completado” hasta recibir un nuevo reinicio.

2. Gestión del conteo de rondas
    - Mantiene un contador que decrece en cada paso de “subBytes” para llevar la cuenta de cuántas rondas faltan por ejecutar.

- Cuando este contador llega a cero, la máquina sabe que debe saltarse el paso de mezcla de columnas y preparar la ronda final.

3. Handshake y sincronización
    - En cada transición, la máquina espera explícitamente la señal de “listo” de cada submódulo antes de avanzar. Esto garantiza que no se solapen operaciones y que cada bloque de procesamiento termine correctamente su tarea.
    - La lógica de reinicio asíncrono permite volver al inicio en cualquier momento, asegurando que todos los contadores y los indicadores de estado vuelvan a su configuración inicial.

El proceso entero partió de un **diagrama de flujo** general del AES inverso, identificando cada paso criptográfico como un estado.

Uno de los problemas que se encontró fue la incorrecta activación de los módulos, de esta manera, usaban una de las llaves que no estaba destinada para una de las rondas y la desencriptación no se realizaba correctamente, esto se solucionó condicionando la transición únicamente al bit de “fin” correspondiente.

16. Conclusiones

La implementación del Advanced Encryption Standard en hardware ha servido no solo como un proyecto para determinar los conocimientos y dominio del lenguaje en VHDL. Sino también la ejecución de este proyecto ha implicado un gran impacto en las denominadas habilidades blandas, especialmente en las de trabajo en equipo y colaboración. Pues, estas últimas han sido pilar fundamental para el desarrollo de este gran proyecto y sin su implementación el resultado no sería el mismo. Desde ponerse de acuerdo con la organización de cada sistema, tanto en encriptación como en desencriptación, así también establecer un estándar en las entradas y salidas de cada algoritmo. Todo esto ha implicado una organización detallada de todo el equipo, que con mucho esfuerzo, puntualidad y sobre todo compañerismo han hecho de este proyecto uno de los mejores.

17. Referencias bibliográficas

Digital, I. L. (2025, 12 marzo). ¿Cómo funciona el cifrado AES? Funcionamiento y características. Whitestack. https://whitestack.com/es/blog/cifrado-aes/#elementor-toc__heading-anchor-0

Gálvez, R. A. P. J. A. S. (s. f.). .:: Algoritmo AES (Advanced Encryption Standard) ::. https://repositorio-uapa.cuaed.unam.mx/repositorio/moodle/pluginfile.php/2184/mod_resource/content/2/contenido-uapa/index.html

Loshin, P., & Cobb, M. (2024, 2 mayo). What is Data Encryption Standard (DES)? Search Security. https://www.techtarget.com/searchsecurity/definition/Data-Encryption-Standard#:~:text=The%20Data%20Encryption%20Standard%20is,it%20into%2064%2Dbit%20blocks.

EITCA. (2023) ¿Cuál es el papel del polinomio irreducible en la operación de multiplicación en los campos de Galois? https://eitca.org/cybersecurity/eitc-is-ccf-classical-cryptography-fundamentals/aes-block-cipher-cryptosystem/introduction-to-galois-fields-for-the-aes/examination-review-introduction-to-galois-fields-for-the-aes/what-is-the-role-of-the-irreducible-polynomial-in-the-multiplication-operation-in-galois-fields/.

EITCA. (2023). ¿Cómo se realiza la multiplicación en Galois Fields en el contexto del algoritmo AES? https://es.eitca.org/cybersecurity/eitc-is-ccf-classical-cryptography-fundamentals/aes-block-cipher-cryptosystem/introduction-to-galois-fields-for-the-aes/examination-review-introduction-to-galois-fields-for-the-aes/how-is-multiplication-performed-in-galois-fields-in-the-context-of-the-aes-algorithm/.