



CCADET

CENTRO DE CIENCIAS APLICADAS Y DESARROLLO TECNOLÓGICO
UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

Informe Técnico

Comunicación Serial Cifrada entre dos Tarjetas SPARTAN 3E mediante el Algoritmo de Cifrado AES con CFB en FPGA

Autores

**Alejandro Padrón Godínez
Rafael Prieto Meléndez
Alberto Arturo Herrera Becerra
Venancio Gerardo Calva Olmos**

Fecha:

Octubre – 2013

Tipo de Proyecto:

Desarrollo Tecnológico

Financiamiento interno

Titulo: Comunicación Serial Cifrada entre dos Tarjetas SPARTAN 3E mediante el Algoritmo de Cifrado AES con CFB en FPGA

Autores:

Alejandro Padrón Godínez
Rafael Prieto Meléndez
Alberto Arturo Herrera Becerra
Venancio Gerardo Calva Olmos

Grupo Académico de Modelado y Simulación de Procesos, Centro de Ciencias Aplicadas y Desarrollo Tecnológico, Universidad Nacional Autónoma de México, Circuito Exterior S/N, Ciudad Universitaria AP.70-186, C.P. 04510, México D.F. México

Resumen

Debido a la naturaleza inherente a las comunicaciones móviles, diversos factores influyen en la calidad de los servicios que se desarrollan para los dispositivos de enlace cuando son fabricados, en los cuales no se encuentran técnicas y procedimientos de seguridad implementados en comparación con dispositivos fijos o de escritorio actuales. Con estos dispositivos, se encuentra la imperiosa necesidad de acceder a la información desde prácticamente cualquier lugar, pero habrá que hacerlo de manera segura. Por lo mismo hay que tomar conciencia de la seguridad de la información que manejan estos dispositivos, que para muchos usuarios los mecanismos de seguridad son invisibles.

En este reporte se presenta la implementación del algoritmo criptográfico por bloques “*Advanced Encryption Standard*” (AES) de 128 bits, para una comunicación serial entre dos tarjetas Spartan 3E fabricadas por Xilinx. Este algoritmo público según la FIPS-197 es el estándar para muchas aplicaciones en Seguridad Informática tanto en Software como Hardware. Este método criptográfico sigue siendo un mecanismo para implementar los servicios de seguridad recomendados en la ISO 7498-2. En particular la implementación del algoritmo se lleva a cabo en Lenguaje de Descripción de Hardware (VHDL) sobre los sistemas de lógica programable (FPGA). Desarrollando también el Modo de Operación de Cifrado por Retroalimentación de Bloques (CFB) de la NIST SP800-38A para convertir el desarrollo en un sistema cifrador por flujo tipo Vernam.

PALABRAS CLAVE: Comunicación Serial, Criptografía, VHDL, AES, CFB.

Índice

Resumen	2
Índice	3
Nomenclatura	4
Introducción	5
1. Plataforma de Comunicación Serial para el Cifrado	6
2. Comunicación Serial Segura	7
3. Algoritmo de Cifrado AES con modo de Operación CFB.....	8
3.1 Funcionamiento del Algoritmo AES	9
3.2 Modo de Operación CFB para Cifrado y Descifrado de datos	18
3.3 Diseño para la Implementación de AES con CFB en VHDL	20
3.3.1 AES – 128	21
4. Diseño de la Plataforma Criptográfica.....	21
5. Principales Características del UART	23
5.1 Operación del UART	24
5.2 Condición para romper con la operación UART	25
5.3 Tasa Temporal en BAUDs	25
5.3.1 El Macro UART_Tx	25
5.3.2 Operación del BUFFER de Transmisión.....	26
5.3.3 El Macro UART_Rx.....	27
5.3.4 Operación del BUFFER de Recepción	28
5.4 Configuración de las Interfaces Seriales en la Spartan 3E	29
6. Resultados	30
Conclusiones	34
Referencias.....	35
ANEXO I	36
ANEXO II	40

Nomenclatura

Nombre	Significado
AES:	Advanced Encryption Standard
ASIC:	Applications-Specific Integrated Circuit
CAD:	Computer Aided Design
CBC:	Cipher Block Chaining
CCM:	Cipher Counter Mode
CCMP:	Combinación de CBC y MAC
CFB:	Cifrado FeedBack
DES:	Data Encryption
DoD:	Department of Defense
FIPS:	Federal Information Processing Standart
FPGA:	Field programmable Gates Arrays
GSM:	Global System for Mobile communications
IDEA:	Internacional Data Encryption Algorithm
IEC:	Intlernational Electrotechnical Comission
IEEE:	Institute of Electrical and Electronic Engineers
ISO:	International Standard Organization
ITU:	International Telecommunication Union
IV:	Initial Vector
LCD:	Liquid Crystal Display
LSB	Least Significant Bit
MAC:	Message Autentication Code
MSB:	Mayor Significant Bit
NIST:	National Institute of Standart Technology
NSA:	National Security Agency
OSI:	Open Systems Interconnection
PGP:	Pretty Good Privacy
RSA:	Rivest Shamir Adleman
Soc:	System on Chip
SSL:	Secure Socket Layer
UART:	Universal Asynchronous Receiver/Transmitter
USB:	Universal Serial Bus
VHDL:	Very High Description Lenguage
VHSIC:	Very High Sppeed Integrated Circuit

Introducción

Los servicios de seguridad que puede cubrir un procedimiento de comunicación serial cifrada son la integridad de los datos y la confidencialidad del enlace. Ya que el sistema es infalible si se tiene control absoluto de la comunicación serial entre las dos tarjetas Spartan 3E. Las características que se buscan al desarrollar e implementar el modo de operación Cipher FeedBack (CFB) es evitar el ataque por repetición de bloque, enmascarar el mensaje de acuerdo al cifrado de flujo, que el espacio de claves sea de 128 bits y que la propagación de un error se limite a un bloque. Los resultados que se muestran están en función de la comparación de las simulaciones en la suite ISim versión 13.3 de Xilinx del “*Advanced Encryption Standard*” (AES), AES con clave de 128 bits y 8 bits de retroalimentación-AES con clave de 128 y modo de operación CFB con sus normas respectivas. Una vez comprobados los resultados de las simulaciones del algoritmo criptográfico y su modo de operación por flujo, se implanta en ambas tarjetas la programación para la comunicación serial. Para esto se hace uso de los manejadores antes desarrollados de las interfaces de comunicación serial, de un teclado y de una terminal. Así se tendrá el despliegue de los datos durante un enlace de la información cifrada enviada y recibida, cuando se escribe un texto plano como mensaje en el teclado. La plataforma para cifradores de flujo se presentó en un trabajo previo en el “*International Congress of Instrumentation and Applied Sciences*”-2011. Los intereses del Grupo Académico de Modelado y Simulación de Procesos son las aplicaciones directas que tienen este tipo de desarrollos criptográficos, en hardware, sobre todo en aquellas enfocadas a enlaces y comunicaciones seguras. Para evitar que la información que viaja en el medio no tenga alteraciones, modificaciones, interferencias y hasta robo de identidades. (Daltabuit E. et al. 2007).

Desde la década de los 70s, los dispositivos de lógica estándar eran todo un avance tecnológico, y los tableros con circuitos impresos eran lo mejor de su momento. Pero algunos se preguntaron “¿qué sucede si le damos a los diseñadores la capacidad de poner diversas interconexiones en ejecución en un dispositivo más grande?” Esto permite que el diseñador integre muchos dispositivos de lógica estándar en una partición. Para ofrecer lo último en flexibilidad del diseño, Ron Cline Signetics (que fue comprado más adelante por Philips y convertido eventualmente a Xilinx), vino con la idea de dos planos de programación. Estos planos proporcionaron cualquier combinación de compuertas lógicas “AND” y “OR”, así como sustituir las compuertas AND en términos de múltiples ORs.

En este nuevo siglo, los “*Field Programmable Gates Arrays*” (FPGA) son una de las tecnologías disponibles para el desarrollo de sistemas electrónicos. Representan una atractiva opción para la producción de pequeñas cantidades ya que sus costos fijos (costo del prototipo de la primera unidad) son bastante más bajos que los correspondientes en “*Applications-Specific Integrated Circuit*” (ASIC), circuitos integrados de aplicación específica. Además, en pos de reducir el tamaño y por tanto el costo unitario, una posibilidad interesante es reconfigurarlas mientras operan de modo que el mismo dispositivo pueda ejecutar diferentes funciones predefinidas. (Baena A., 2010).

Actualmente los sistemas electrónicos están presentes en el quehacer cotidiano, desde la electrónica de consumo a los sistemas de control industrial, pasando por las aplicaciones para automotriz, seguridad y una gama extensa en diferentes ámbitos. La tendencia de estos dispositivos con respecto a su diseño electrónico radica en la complejidad de muchos de los elementos que lo conforman, donde se exige, por ejemplo, que sean de fácil utilización y adaptables a varias aplicaciones que consuman pocos recursos y que puedan estar rápidamente a la venta. La tecnología actual de circuitos integrados permite realizar e integrar estos complejos sistemas en muy poco espacio; son los llamados sistemas embebidos (o empotrados) o “*System on Chip*” (SoC) por sus siglas en inglés. Estos sistemas embebidos usados para resolver tareas específicas incluyen hardware y software diseñado expresamente para ello. En el caso de las FPGA, al ser dispositivos reconfigurables son altamente útiles para desarrollo de prototipos o pequeñas cantidades a un precio razonablemente bajo.

La evolución de la microelectrónica desde su nacimiento ha sido impresionante, en poco más de diez años se pasó del primer dispositivo integrado (*flip-flop* con unos cuantos transistores) a las primeras memorias y procesadores. A los pocos años, el diseño de circuitos integrados salió de las fábricas y se hizo accesible a los ingenieros de aplicación, dando lugar a los ASIC. Luego a mediados de los 80’s, se puso de manifiesto la necesidad de disponer de un lenguaje estándar capaz de dar el soporte necesario

al proceso completo de diseño de *chips* y sistemas electrónicos (desde la idea hasta la implementación y explotación de un desarrollo) en sus distintas etapas y niveles de abstracción y cuya complejidad, ya de por sí elevada, se iba a incrementar drásticamente hacia los años noventa, (Terés L. et al, 1998).

Estas inquietudes, canalizadas a través del “*Department Of Defense*” (DoD) de los E.U.A., llevan al nacimiento del VHDL (1987), que tras ser adoptado como estándar del IEEE (Std.-1076, 1987), se presenta como el lenguaje estándar por excelencia del diseño electrónico. Desde el primer momento se convirtió en la herramienta y el motor de base que ha facilitado e impulsado los enormes avances de las metodologías, técnicas y herramientas “*Computer Aided Design*” (CAD) de diseño electrónico de los últimos diez años. Esta evolución arrastra al propio Verilog, ahora también convertido en estándar del (IEEE, Std.-1364, 1995), que es con quien compite y lleva de la mano al “*Very High Design Language*” (VHDL).

En definitiva, aunque sólo se considere al VHDL como herramienta de base, la aparición de este lenguaje ha supuesto tal cantidad de cambios en el diseño electrónico (nuevas metodologías de diseño de alto nivel y flujos descendentes, herramientas CAD de simulación/síntesis de circuitos y sistemas, nuevos de los equipos de diseño, cambios en la organización y desarrollo de proyectos, etc.), que sin lugar a dudas, se puede hablar de un antes y un después del VHDL.

Una línea de desarrollo de estos sistemas con las herramientas mencionadas, está dirigida a la protección de datos en dispositivos de almacenamiento masivo, para ello habría que crear los elementos y módulos electrónicos necesarios para el manejo de los datos en forma bidireccional (transceivers, memorias dinámicas, controladores de lectura y escritura), (Padrón et al, 2012). Ya sea escribiendo o leyendo los datos de forma segura, habría que detallar en donde sería el mejor sitio para el módulo criptográfico que proporcionará la seguridad de los datos. Para ello el desarrollo de una plataforma criptográfica de comunicación puede ayudar al concepto del manejo de información en una transmisión y recepción de datos cifrados, es decir proteger datos en canales de transmisión antes de almacenarlos. Como consecuencia de los desarrollos de protección de datos mediante sistemas en hardware, surgen desarrollos dedicados a optimizar las operaciones midiendo la eficiencia y el desempeño de las implementaciones para mejorar la velocidad de procesamiento.

La idea de esta introducción es percibir que el desarrollo de este reporte está siendo implementado con las nuevas técnicas y herramientas para el diseño electrónico. Donde se ha tenido que aprender nuevas técnicas de programación, existen muchas ayudas en los manuales y en la red para desarrollar e implementar los programas ante cualquier obstáculo encontrado, sin embargo tienen que ser modificadas e integradas a las tareas que se quieren resolver.

Así el desarrollo del algoritmo criptográfico para su implementación en hardware, el modo de operación y las interfaces de la plataforma criptográfica para comunicación de datos aquí mostrados están en el lenguaje de programación VHDL. Se utiliza además la ventaja de poder realizar una simulación de los módulos construidos, en algunos casos antes de llevar las implantaciones (programación de los FPGA) en hardware. (Trejo E., 2005).

1. Plataforma de Comunicación Serial para el Cifrado

Una preocupación muy importante en las comunicaciones es el garantizar la seguridad en la información en lo relativo a la confidencialidad. Una medida que se aplica en este sentido es la aplicación de algún algoritmo criptográfico para evitar que la información pueda ser accedida por alguien que no sea el destinatario. Uno de los objetivos del trabajo es contar con una plataforma de diseño para implantar codificadores de flujo, para garantizar la transmisión segura de información entre dos puntos que se comunican por medio de algún protocolo de comunicación serial.

Así, en este trabajo se presenta el desarrollo de una plataforma base que contiene los elementos que permiten recibir y enviar la información, hacer el proceso de codificación y decodificación de los mensajes, y establecer el intercambio de información entre los dos puntos, dejando el proceso de codificación definido de manera genérica, de tal manera que para implantar una comunicación cifrada solamente se tenga que definir el algoritmo criptográfico que se desea utilizar. Un aspecto importante para el funcionamiento de un codificador de flujo es la velocidad de funcionamiento, para que el uso de este elemento no impacte en la velocidad de la comunicación. (Prieto R., Padrón A., et al, 2011b).

Para ello esta plataforma está embebida en un *FPGA*, lo que permite que este proceso de cifrado se pueda implantar y opere en tiempo real en un sistema de comunicación. Se presenta la prueba y validación de la operación de la plataforma al comparar su funcionamiento utilizando diferentes algoritmos criptográficos.

2. Comunicación Serial Segura

Como parte del reporte, se desarrolla un sistema de comunicación seguro, así que se inicia por estudiar las características de los sistemas de comunicación digital y la manera de poder garantizar que la información que se envía con ellas es alcanzada y entendida solamente por el destinatario deseado. Durante este proceso del análisis se identifican varios aspectos que determinan las características deseables para un algoritmo de cifrado, que se verán adelante en este capítulo.

En el proceso para fijar los parámetros de una comunicación sea segura, se desea alcanzar esto, afectando los sistemas de comunicación lo menos posible, hay que tomar cuidado particularmente de no tener que hacer modificaciones al equipo de comunicación usado. El uso de cualquier mecanismo para hacer una comunicación segura debe ser transparente al usuario y debe funcionar en tiempo real, sin alterar las características de la comunicación en términos de calidad y de la velocidad de la transmisión. (Prieto R., Padrón A., et al, 2011a).

Actualmente, la mayoría de los sistemas de comunicación realizan el intercambio de la información usando una transmisión de datos serial, que es generalmente "*full-duplex*" (*doble dirección*). En la búsqueda de los sistemas de comunicación seguros funcionales y eficientes se necesita analizar y evaluar en ejecución un algoritmo del cifrado estándar. Eventualmente se podrán implementar y evaluar otros algoritmos.

De estas ideas vienen las características para desarrollar la plataforma de comunicación criptográfica, que son las siguientes.

- a. El cifrado debe poder ser integrado en la línea de comunicación sin tener que modificar el equipo de comunicaciones.
- b. Debe manejar un flujo de información *full-duplex* asincrónica serial, soportando las diferentes velocidades de transmisión que son de uso general.
- c. Debe permitir cambiar del método del cifrado de una manera simple.
- d. El proceso de cifrado/descifrado se debe hacer continuamente y en tiempo real, pues está fluyendo la información.

Para alcanzar la primera meta, los dispositivos fueron diseñados para ser integrados en cada lado del sistema de comunicación, se pueden colocar en los extremos de la línea de la transmisión, o se colocan como una interfaz entre el usuario y el equipo de comunicación. Dependiendo del uso en donde el cifrado será utilizado y de las características del equipo y del medio de transmisión será el lugar más

conveniente para implementar el proceso de cifrado. La Figura (1) muestra un diagrama de bloques de cómo la integración del sistema de seguridad se puede emplear en un sistema de comunicación.

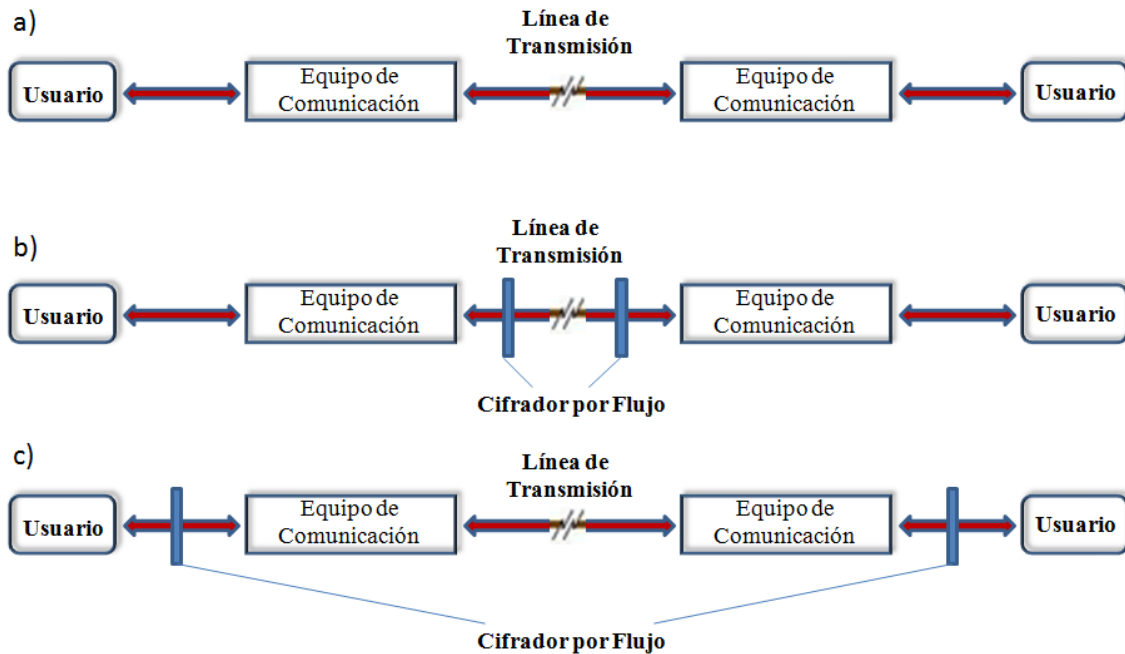


Figura 1. Diagrama de bloques de un sistema de comunicación simplificado. a) Sistema Inseguro. b) Sistema con el cifrado en los extremos de la línea de la transmisión. c) Sistema con el cifrado como interfaz entre el usuario y el equipo de comunicación.

El mecanismo que cifra debe contener generalmente un elemento central de control para realizar el proceso del cifrado/descifrado, manejando y obteniendo la información a partir de dos interfaces en serie, según las indicaciones de la Figura (2).

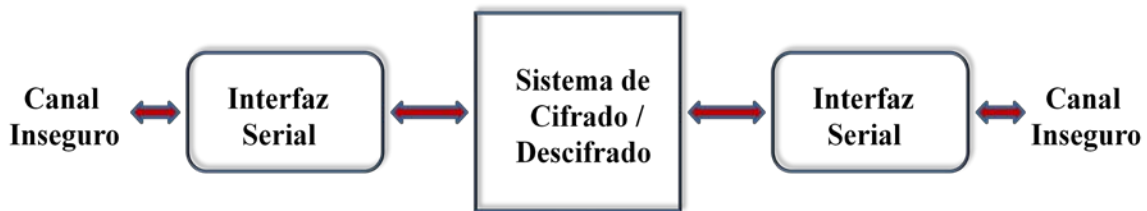


Figura 2. Diagrama de bloques del dispositivo de cifrado y Descifrado.

Para asegurarse de que el proceso de cifrado/descifrado se realice continuamente sin afectar el flujo de información original, se decide trabajar con algoritmos de cifrado por flujo.

3. Algoritmo de Cifrado AES con modo de Operación CFB

En este capítulo se presenta el funcionamiento de las operaciones llevadas a cabo para el algoritmo de cifrado AES por bloques, empleando diferentes tamaños de claves y calculando el número de rondas para cada caso. Se mostrará cómo trabaja el proceso para la expansión de claves necesarias para cada caso. Se toma la norma (FIPS-197, 2001), como referencia para consultar y cotejar los

resultados de algunos detalles en específico. Para el caso del modo de operación *CFB* se toma como referencia la norma (NIST SP800-38A, 2001), siguiendo el procedimiento que se debe integrar para cumplir con el propósito de este trabajo. (Padrón A., Velázquez J., et al, 2011)

3.1 Funcionamiento del Algoritmo AES

Para entender cómo funciona el algoritmo *AES* será necesario mencionar algunos antecedentes en el campo de la Aritmética de Números Finitos y Matemáticas Discretas. Antecedentes que habrá que recordar o revisar consultando bibliografía específica en los temas de representaciones de números en diferentes bases (binaria, decimal, hexadecimal y polinomial), operaciones sus básicas en cada base (suma, resta, multiplicación y división).

Estrictamente hablando, *AES* no es precisamente Rijndael (aunque en la práctica se los llama de manera indistinta) ya que Rijndael permite un mayor rango de tamaño de bloques y longitud de claves; *AES* tiene un tamaño de bloque fijo de 128 bits y tamaños de llave de 128, 192 ó 256 bits, mientras que Rijndael puede ser especificado por una clave que sea múltiplo de 32 bits, con un mínimo de 128 bits y un máximo de 256 bits. La mayoría de los cálculos del algoritmo *AES* se hacen en un campo finito determinado. *AES* opera en una matriz de 4x4 bytes, llamada *state* (algunas versiones de Rijndael con un tamaño de bloque mayor tienen columnas adicionales en el *state*). (Ramió J., 2006). Estas diferencias se mencionan debido a que la implementación del *AES* solamente se hará de acuerdo al la norma FIPS-197. La estructura del algoritmo Rijndael está formada por un conjunto de “rondas”, entendiendo por rondas un conjunto de iteraciones de 4 funciones matemáticas diferentes e invertibles. El algoritmo representa el “estado” como una matriz rectangular de bytes, que posee 4 renglones y N_b columnas. Siendo el número de columnas N_b en función del tamaño del bloque y se puede calcular de acuerdo a la ecuación (1).

$$N_b = \text{tamaño del bloque utilizado en bits} / 32 \quad (1)$$

Por ejemplo la representación de una matriz de estado para un tamaño de 160 bits ($N_b = 5$), se muestra en la Tabla (1).

Tabla 1. Matriz del texto plano de 4X5 bytes.

a_{00}	a_{01}	a_{02}	a_{03}	a_{04}
a_{10}	a_{11}	a_{12}	a_{13}	a_{14}
a_{20}	a_{21}	a_{22}	a_{23}	a_{24}
a_{30}	a_{31}	a_{32}	a_{33}	a_{34}

La clave del sistema se representa con una estructura análoga a la del “estado”, es decir, se representa mediante una matriz rectangular de bytes de 4 renglones y N_k columnas. Siendo el número de columnas N_k en función del tamaño de la clave, como se muestra en la ecuación (2).

$$N_k = \text{tamaño de la clave en bits} / 32 \quad (2)$$

Por ejemplo la representación de una clave de 128 bits ($N_k = 4$), en forma de matriz rectangular se muestra en la Tabla (2).

Tabla 2. Representación de la matriz clave de 4X4 byte.

k_{00}	k_{01}	k_{02}	k_{03}	k_{04}
k_{10}	k_{11}	k_{12}	k_{13}	k_{14}
k_{20}	k_{21}	k_{22}	k_{23}	k_{24}
k_{30}	k_{31}	k_{32}	k_{33}	k_{34}

Una vez establecidos estos parámetros iniciales el bloque que se pretende cifrar y descifrar se traslada byte a byte sobre la matriz de estado, siguiendo la secuencia $a_{00}, a_{01}, a_{20}, a_{30}, a_{01}, \dots, a_{34}$; y análogamente los bytes de la clave se copian en la matriz de la clave siguiendo el mismo criterio, $k_{00}, k_{10}, k_{20}, k_{30}, k_{01}, \dots, k_{33}$.

A partir de este momento la matriz de estado sufre 4 transformaciones por ronda (iteración), utilizándose en el proceso subclaves para cada ronda que se generan de la clave original elegida. Las 4 transformaciones que aplica el algoritmo a la matriz de estado por ronda son:

- Función ByteSub: sustitución con propiedades óptimas de no linealidad.
- Función ShiftRow: Permiten un alto nivel confusión de la información a lo largo de las diferentes rondas.
- Función MixColumn: Permiten un alto nivel difusión de la información a lo largo de las diferentes rondas.
- Función AddRoundKey: Permite aplicar a la matriz de estado una operación “or” exclusiva con la subclave correspondiente a cada ronda.

El número de iteraciones o rondas de las 4 transformaciones sobre la información, o mejor dicho sobre la matriz de estado intermedio depende de la versión del algoritmo que se utilice. Los autores definen que para tamaños de bloques y claves entre 128 y 256 bits (con incrementos de 32 bits) el número de rondas N_r es determinado por la expresión de la ecuación (3).

$$N_r = \max(N_k, N_b) + 6 \quad (3)$$

Por ejemplo, para un algoritmo Rijndael de tamaño de clave y de bloque 128 bits, el número de rondas es 10. Se observa claramente que el número de vueltas o iteraciones del algoritmo dependen del tamaño de bloque y clave elegidos, Tabla (3.3).

Tabla 3. Número de rondas para Rijndael en función de tamaños de clave y bloque:

Clave/Bloque	$N_b = 4$ (128 - bits)	$N_b = 6$ (192 - bits)	$N_b = 8$ (256 - bits)
$N_k = 4$ (128 - bits)	10	12	14
$N_k = 6$ (192 - bits)	12	12	14
$N_k = 8$ (256 - bits)	14	14	14

En la Figura (3) se muestra el proceso de cifrado para un texto inicial, manejado también como texto plano o texto en claro.

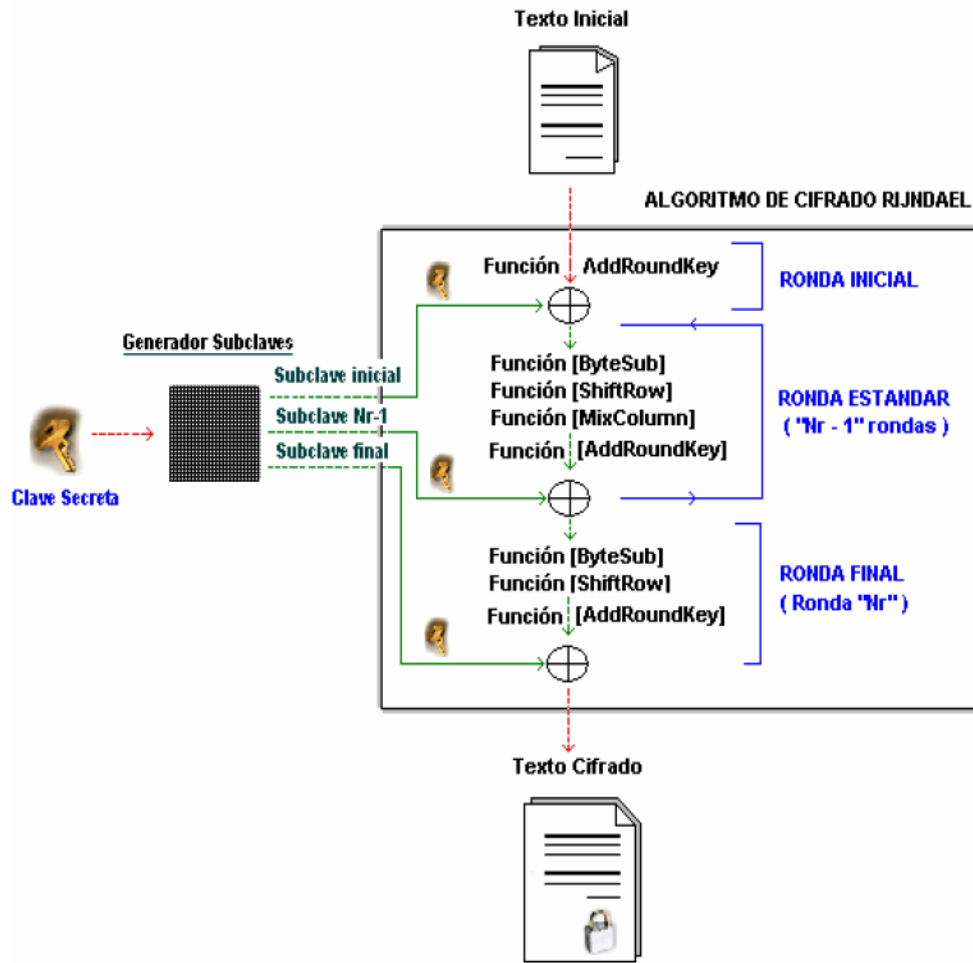


Figura 3. Esquema descriptivo del algoritmo por rondas.

A continuación se describirán cada una de las funciones que se emplean en el algoritmo.

Función AddRoundKey

En el paso *AddRoundKey*, la subclave se combina con el state (estado-actual). En cada ronda se obtiene una subclave de la clave principal, usando la iteración de la clave; cada subclave es del mismo tamaño del state. La subclave se agrega combinando cada byte del state con el correspondiente byte de la subclave usando la operación or-exclusiva. Para esto se necesita hacer la expansión de la clave dependiendo si es un AES – 128, 192 o 256. Figura (4).

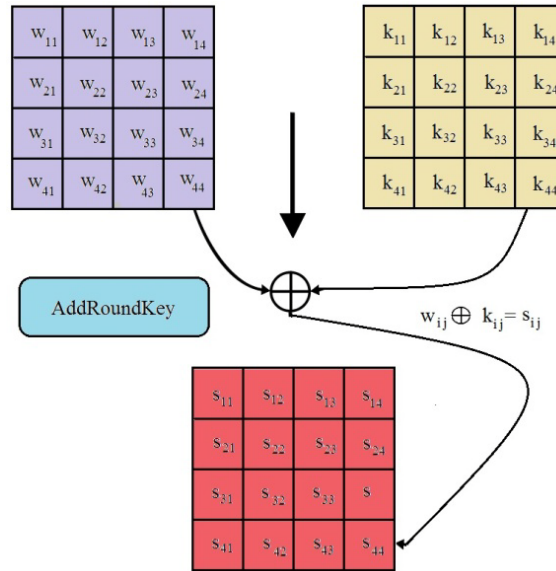


Figura 4. Esquema que describe la función *AddRoundKey* usando la operación or-exclusiva.

En el paso *AddRoundKey*, cada byte del “state” se combina con un byte de la subclave usando la operación or-exclusiva ($\oplus$).

Cálculo de las Subclaves usando el esquema de claves de Rijndael

La función de selección simplemente toma consecutivamente de la secuencia obtenida por la función de expansión de clave bytes que va asignando a cada subclave K_i , para formar bloques del mismo tamaño que la matriz de estado. Es decir, toma $N_b \cdot 4$ bytes para cada vuelta. La generación de las claves (expansión de la clave) para el proceso de descifrado se hace de forma idéntica al procesos de cifrado. La diferencia reside en la función de selección de la clave. En el proceso de descifrado se toman bytes de la lista de claves desde los valores finales hasta llegar a los iniciales, que es la propia clave del usuario. Es decir, la última clave que se utilizó para cifrar, será la primera que se utilizará para descifrar. La función de expansión de clave permite generar bytes útiles como subclaves a partir de la clave original K . Esta función de expansión se puede describir como un arreglo lineal, denominado W , de palabras de 4 bytes y con una longitud de $N_b \cdot (N_r + 1)$. Las primeras N_k palabras de este arreglo contienen la clave de cifrado, ya que la clave del usuario se mapea tal cual al arreglo W , mientras que el resto de palabras se van generando a partir de estas primeras N_k palabras.

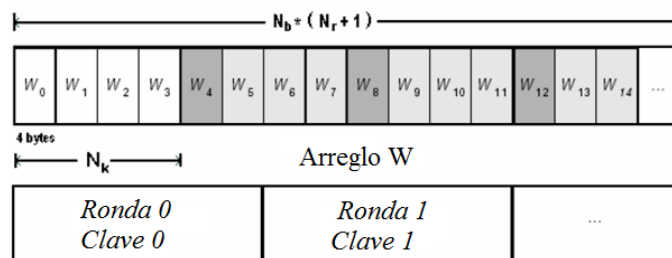


Figura 5. Ejemplo de subclaves de expansión para $N_b=6$ y $N_k=4$.

Se observa cómo la función de expansión de clave depende del valor N_k , ante este hecho los autores definieron dos versiones para esta función, una para $N_k \leq 6$ y otra para $N_k > 6$, por motivos de seguridad. Se mapea directamente la clave del usuario al arreglo W entonces se mapean N_k palabras. El

resto del funcionamiento permite generar bytes para subclaves, en este proceso entran en juego la función ByteSub que devuelve el resultado de aplicar la S-BBOX de Rijndale a cada uno de los bytes de los 32 bits de la palabra que se le pasa como parámetro. La función “Rot” que rota una posición a la izquierda los bytes de la palabra, de tal forma que si se le pasa como parámetros la palabra de 4 bytes (a, b, c, d) devuelve (b, c, d, a) . Y la función “Rcon” que genera una constante teniendo en cuenta que: $Rcon(j) = (R(j), 0, 0, 0)$. Cada $R(j)$ es el elemento $\in GF(2^8)$ correspondiente al valor x^{j-1} .

Cuando $N_k \leq 6$, para todo valor de “i” que no sea múltiplo de N_k , las palabras de subclaves se calculan como: $W(i) = W(i - N_k) \oplus W(i - 1)$. Para todo valor de “i” que sea múltiplo de N_k se calculan como:

$$W(i) = W(i - N_k) \oplus [ByteSub(RotByte[W(i - 1)]) \oplus Rcon(i / N_k)].$$

La variable “i” denota la posición dentro del arreglo W (i) que contiene los bytes de subclaves del sistema. Si por ejemplo se utiliza una clave de 128 bits ($N_k = 4$) y un bloque de 128 bits ($N_b = 4$) la longitud del arreglo W sería 44, $(4 * [10 + 1])$. En las cuatro primeras posiciones del arreglo (0 a la 3) se copia la clave de usuario, el resto de posiciones, de la posición 4 a la 43, se rellenaría con valores calculados. En este caso la variable “i” tomaría los valores de 4 a 43. Para las posiciones del arreglo W(i) con valor de “i” 4, 8, 12, 16, 20, 24, 28, 32, 36, 40 (i múltiplo de N_k), la palabra de 4 bytes se calcularía tomando una palabra que se encuentra 4 posiciones antes que ella y realizando una operación or-exclusiva con una transformación de la palabra que se encuentra una posición antes. Para el resto de los valores de “i” la palabra correspondiente a W (i) se calcula realizando una operación or-exclusiva entre la palabra que se encuentra una posición antes que la posición en la que se encuentra actualmente. Este ejemplo permite observar el procedimiento que sigue el algoritmo para generar nuevos bytes de subclaves, apoyándose en palabras ya existentes en el arreglo W, ya sea palabras de la clave de usuario directamente o palabras generadas de esta.

Para $N_k > 6$, el funcionamiento es igual que para $N_k \leq 6$ salvo cuando el valor de la variable “i” satisface que “i mod $N_k = 4$ ”, ver figura (6), en este caso las palabras de subclaves se calculan de acuerdo a la ecuación (4).

$$W(i) = W(i - N_k) \oplus ByteSub(W[i - 1]) \quad (4)$$

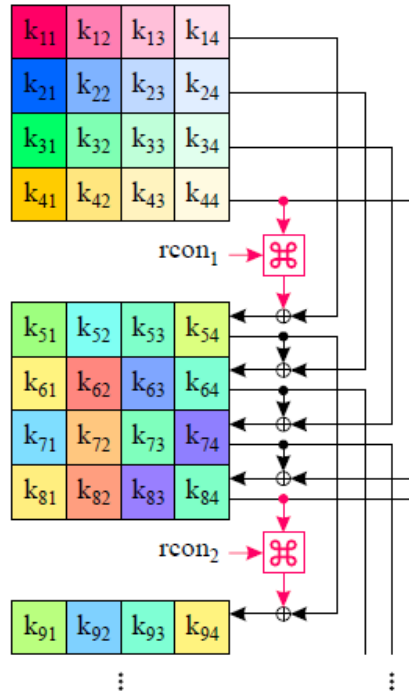


Figura 6. Esquema que describe la expansión de las subclave.

Una vez conocido el funcionamiento de la función de expansión de clave, para finalizar con la explicación de la función un ejemplo real de generación de bytes de subclaves para un algoritmo Rijndael con clave de 128 bits.

Se selecciona clave de usuario:

Clave = 2b 7e 15 16 28 ae d2 a6 ab f7 15 88 09 cf 4f 3c ($N_k=4$)

Se mapea a las 4 primeras posiciones del arreglo la clave del usuario,

$W[0]=2b\ 7e\ 15\ 16$

$W[1]=28\ ae\ d2\ a6$

$W[2]=ab\ f7\ 15\ 88$

$W[3]=09\ cf\ 4f\ 3c$

A partir de este momento se empieza a generar bytes de subclaves. Calcular $W[4]$:

$i=4$ (i múltiplo de N_k)

$temp = W[3] = 09\ cf\ 4f\ 3c$

Se aplica la función Rot $\rightarrow temp = cf\ 4f\ 3c\ 09$

Se aplica la función Byte Sub $\rightarrow (temp) = 8a\ 84\ eb\ 01$

$Rcon[4/4]=Rcon[1] = [x0, \{00\}, \{00\}, \{00\}] = [\{01\}, \{00\}, \{00\}, \{00\}]$

$Temp = Rcon[1] \oplus temp = 8a\ 84\ eb\ 01 \oplus 01\ 00\ 00\ 00 = 8b\ 84\ eb\ 01$

$W[4] = W[0] \oplus temp = 2b\ 7e\ 1516 \oplus 8b\ 84\ eb\ 01 = a0\ fa\ fe\ 17$

Siguiendo todos los criterios descritos en esta función se podrían generar todos los bytes de subclaves necesarios para la versión del algoritmo con la que se trabajará. (Buchholz J., 2001).

Función SubBytes

En la fase de *ByteSub*, cada byte en el state es reemplazado con su entrada en una tabla de búsqueda fija de 8 bits, S ; $b_{ij} = S(a_{ij})$. Figura (7).

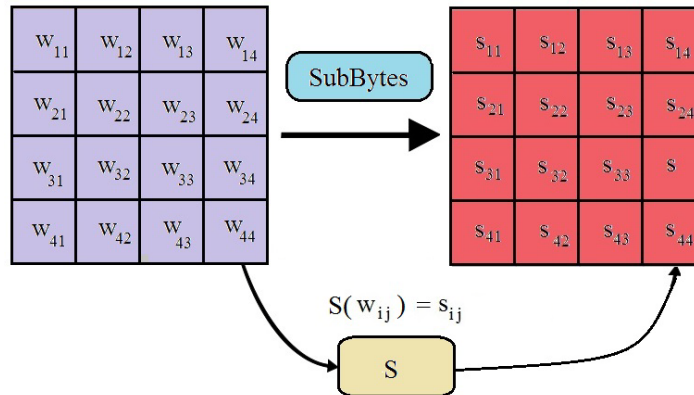


Figura 7. Esquema de la Función ByteSub para el intercambio de bits.

En la etapa ByteSub, cada byte en la matriz es actualizado usando la caja-S de Rijndael de 8 bits. Esta operación provee la no linealidad en el cifrado. La caja-S utilizada proviene de la función inversa alrededor del $GF(2^8)$, conocido por tener grandes propiedades de no linealidad.

Para evitar ataques basados en simples propiedades algebraicas, la caja-S se construye por la combinación de la función inversa con una transformación afín invertible. La caja-S también se elige para evitar puntos estables (y es por lo tanto una permutación), y también cualesquiera puntos estables opuestos.

La transformación *ByteSub* consiste en una sustitución no lineal que se aplica a cada byte de la matriz de estado (estado intermedio 1) de forma independiente, generando un nuevo byte. Esta

transformación consiste en la sustitución de cada byte por el resultado de aplicarle la tabla de sustitución S-BOX. Esta tabla lógicamente es invertible y se construye mediante dos transformaciones.

1ª Transformación.- Cada byte es considerado como un elemento en $GF(2^8)$ que genera el polinomio irreducible $m(x) = x^8 + x^4 + x^3 + x + 1$, siendo sustituido por su inversa multiplicativa. El valor cero queda inalterado, ya que no tiene inversa.

2ª Transformación.- Al resultado de la 1ª transformación se le aplica la siguiente transformación afín en $GF(2^8)$, siendo $x_0, x_1, x_2, x_3, x_4, x_5, x_6$, y x_7 los bits del byte resultante de la 1ª transformación, e $y_0, y_1, y_2, y_3, y_4, y_5, y_6$ y y_7 los bits del resultado final de la transformación ByteSub.

Tabla 4. Transformación afín en $GF(2^8)$.

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

Por ejemplo, si el byte al cual se le aplica la función ByteSub es "A=11001011", se debería calcular su inversa multiplicativa. Cada byte en Rijndael se representa como un polinomio $a(x)$, calcular la inversa multiplicativa consiste en buscar un polinomio $b(x)$ (que es único) que multiplicado por $a(x)$ modulo $m(x)$ es igual a 1, es decir:

$$a(x) \bullet b(x) \bmod x^8 + x^4 + x^3 + x + 1 = 1 \quad (5)$$

El polinomio buscado en este caso es $b(x) = x^2$ que tiene una representación binaria de "B=00000100". Se dice entonces que B es la inversa en $GF(2^8)$ de A. Una vez que se tiene el resultado de la primera transformación, se debe aplicarle la transformación afín definida:

Tabla 5. Ejemplo de la transformación afín en $GF(2^8)$.

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

Ejemplo de un valor concreto.

Se obtiene el valor final de "y=00011111". Luego el byte "A=11001011" se convierte en y al aplicar la función ByteSub.

Utilizando estas dos transformaciones para todos los valores posibles de entrada (256 valores ya que se trabaja con un byte) se calcula una tabla de sustitución denominada S-Box útil para el proceso de cifrado. Gracias a esta tabla aplicar la función ByteSub resulta trivial, consiste en dividir el byte de la

matriz de estado en dos partes de 4 bits. Los 4 bits más significativos, denominados por x (toma valores de 0 a 15) actúan del renglón en la tabla y los 4 bits menos significativos de columna, denominados por “y” (toma valores de 0 a 15). El valor para ese renglón y columna en la tabla es resultado de aplicar S-Box a un byte.

Siguiendo con el ejemplo anterior, si se tiene el byte “A=00011111” y se aplica la función *ByteSub* el resultado será (x=1100 [renglón c] y=1011 [columna b]) “y=0x1F” que en binario equivale a “y=00011111”, valor que es idéntico al calculado previamente.

Tabla 6. S-Box para un byte genérico xy (hexadecimal).

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

Para el proceso de descifrado es necesario calcular la función inversa de ByteSub. Esta función inversa consiste en calcular una tabla inversa a la utilizada en el proceso de cifrado.

Tabla 7. Tabla inversa S-Box (hexadecimal).

Hex		y															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	F
x	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	d	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	d0	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

Función ShiftRow

En el paso ShiftRows, los bytes en cada fila del “state” son rotados de manera cíclica hacia la izquierda. El número de lugares que cada byte es rotado difiere para cada fila. Figura (8).

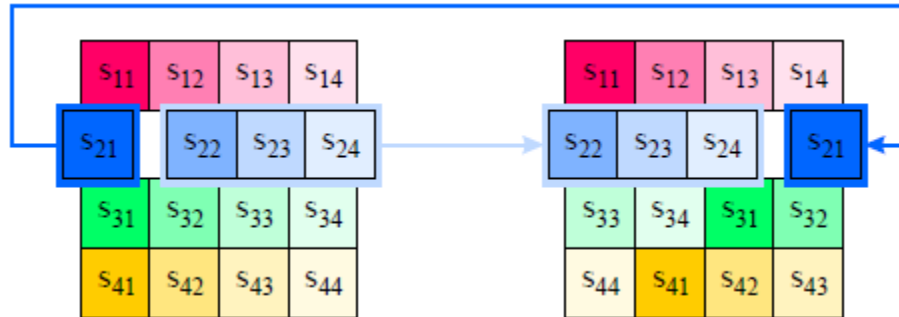


Figura 8. Esquema que describe la función ShiftRow para el desplazamiento de cada renglón.

El paso ShiftRows opera en las filas del “state”; rota de manera cíclica los bytes en cada fila por un determinado “offset”. En AES, la primera fila queda en la misma posición. Cada byte de la segunda fila es rotado una posición a la izquierda. De manera similar, la tercera y cuarta filas son rotadas por los “offsets” de dos y tres respectivamente. De esta manera, cada columna del state resultante del paso ShiftRows está compuesta por bytes de cada columna del “state” inicial. (variantes de Rijndael con mayor tamaño de bloque tienen “offsets” distintos).

Función MixColumns

En el paso MixColumns, cada columna del “state” es multiplicada por un polinomio constante $c(x)$. Figura (9).

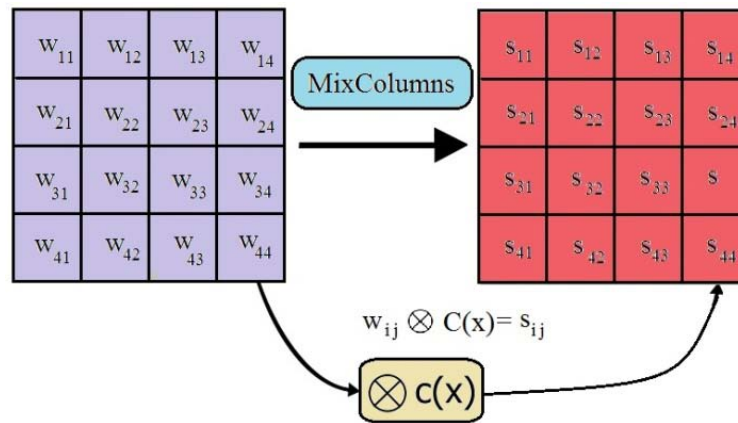


Figura 9. Esquema que describe la función MixColumns con multiplicación polinomial $GF(2^8)$.

En el paso MixColumns, los cuatro bytes de cada columna del state se combinan usando una transformación lineal invertible. La función MixColumns toma cuatro bytes como entrada y devuelve cuatro bytes, donde cada byte de entrada influye todas las salidas de cuatro bytes. Junto con ShiftRows, MixColumns implica difusión en el cifrado. Cada columna se trata como un polinomio $GF(2^8)$ y luego se multiplica el módulo $x^4 + 1$ con un polinomio fijo $c(x)$. El paso MixColumns puede verse como una multiplicación matricial en el campo finito de Rijndael.

3.2 Modo de Operación CFB para Cifrado y Descifrado de datos

La norma NIST SP800-38A, (2001) es una recomendación de confidencialidad del modo de operación de cinco algoritmos para criptografía simétrica por bloque, como el algoritmo que se empleará en este trabajo, el estándar avanzado de cifrado (AES). Los modos se pueden utilizar conjuntamente con cualquier algoritmo simétrico que cifra por bloques que ha sido aprobado por la NIST. Los cinco modos son: Código de Libro Electrónico (ECB), cifrado por encadenamiento de bloques (CBC), cifrado por retroalimentación (CFB), retroalimentación de la salida (OFB), y modo contador (CTR), pueden proporcionar confidencialidad de los datos.

Esta recomendación asume que un algoritmo simétrico por bloques se ha elegido como el algoritmo estándar y que una clave secreta aleatoria K , se ha establecido entre todas las partes involucradas en la comunicación. La clave criptográfica regula el funcionamiento del algoritmo de cifrado por bloques y por consiguiente, regula el funcionamiento del modo. Las especificaciones de los algoritmos de cifrado por bloques y los modos operación son públicas, así que la seguridad del modo depende del secreto de la clave.

Un modo de operación sobre un algoritmo de cifrado por bloques consiste en dos procesos que son contrarios, uno cifra y el otro descifra. El cifrado es la transformación de un mensaje, llamado texto-plano, en una forma ilegible, llamado el texto-cifrado; el desciframiento es la transformación que recupera el texto-plano a partir del texto-cifrado. Estos procesos son dos funciones que como parte de la opción del algoritmo de cifrado por bloques, una de las dos funciones se señala como la función de cifrado, $CIPH_K$; y la otra función entonces se llama función inversa de cifrado, $CIPH_K^{-1}$. Las entradas y las salidas de ambas funciones se llaman los bloques de la entrada y los bloques de la salida. Los bloques de la entrada y de la salida del algoritmo de cifrado por bloques tienen la misma longitud en bits, llamado el tamaño de bloque, denotado por " b ". El objetivo de la implementación de este modo de operación CFB es convertir el algoritmo de cifrado y descifrado por bloques en un algoritmo que cifre y descifre por flujo de datos.

Vectores de Inicialización

La entrada a los procesos del cifrado del modo CFB, incluye además del texto-plano, un bloque de los datos llamado el Vector de Inicialización (IV, por sus siglas en inglés). El IV utiliza en un paso inicial en el cifrado de un mensaje y en el descifrado correspondiente del mensaje.

El IV no necesita ser secreto, sin embargo, para los modos de CBC y de CFB, el IV para cualquier ejecución particular del proceso de cifrado debe ser imprevisible. Para el modo de OFB, IV es único y se debe utilizar para cada ejecución del proceso del cifrado.

Modo de Operación de Cifrado por Retroalimentación

El modo de cifrado por retroalimentación (CFB) es un modo que ofrece confidencialidad por medio de la retroalimentación de los segmentos sucesivos del texto cifrado en los bloques de la entrada. Esto sucede al inicio de cada proceso de cifrado para generar los bloques de la salida que son calculados bajo la operación lógica or-exclusiva con el texto plano, produciendo el texto cifrado, y viceversa. El modo CFB requiere un vector de inicio como el bloque inicial de la entrada. El IV no necesita ser secreto, sino que debe ser imprevisible. El formalismo de este modo de operación aparece en la sección 6.3 de la norma NIST SP800-38A pp11-13, para este reporte sólo se toman las expresiones que definen el cifrado y el descifrado. Además de la figura (3) de la misma norma, donde se muestra el esquema descriptivo del modo de operación, esta se muestra más adelante explicando brevemente su funcionamiento.

El modo CFB requiere un número entero " s " como parámetro, tal que $1 \leq s \leq b$. En las expresiones del modo de CFB, mostradas en seguida, cada segmento del texto plano ($P_i^{\#}$) y el segmento del texto cifrado ($C_i^{\#}$) está formado por s bits. El valor de s se incorpora a veces en el nombre del modo usado, por ejemplo, si $s = 1, 8, 64$ o 128 bits el nombre será el modo 1CFB, 8CFB, 64CFB o 128CFB. En la plataforma desarrollada se implementa el caso particular de 8CFB para el cifrado y descifrado.

El modo de operación *CFB* está definido por las siguientes expresiones:

$$\begin{aligned}
 \text{Cifrado CFB: } & I_1 = IV; \\
 & I_j = LSB_{b-s}(I_{j-1}) \parallel C_{j-1}^\# \quad \text{for } j = 2 \dots n; \\
 & O_j = CIPH_K(I_j) \quad \text{for } j = 1, 2 \dots n; \\
 & C_j^\# = P_j^\# \oplus MSB_s(O_j) \quad \text{for } j = 1, 2 \dots n.
 \end{aligned}$$

$$\begin{aligned}
 \text{Descifrado CFB: } & I_1 = IV; \\
 & I_j = LSB_{b-s}(I_{j-1}) \parallel C_{j-1}^\# \quad \text{for } j = 2 \dots n; \\
 & O_j = CIPH_K(I_j) \quad \text{for } j = 1, 2 \dots n; \\
 & P_j^\# = C_j^\# \oplus MSB_s(O_j) \quad \text{for } j = 1, 2 \dots n.
 \end{aligned}$$

En el cifrado de *CFB*, el primer bloque de la entrada es el *IV*, y la operación inicial del cifrado se aplica al *IV* para producir el primer bloque de la salida. El primer segmento del texto cifrado es producido por una or-exclusiva con el primer segmento del texto plano de tamaño *s*, el resultado de la operación es el primer bloque de salida que será el primer bloque retroalimentado. (Los bits restantes de los *b-s* del primer bloque de la salida se desechan.) Los *b-s* bits menos significativos del *IV* entonces se concatenan con los bits de *s* del primer segmento del texto cifrado para formar el segundo bloque de la entrada. Una descripción alternativa de la formación del segundo bloque de la entrada es que los bits cifrados van desplazando al *IV* hacia la izquierda, y entonces el segmento del texto cifrado substituye los bits *s* menos significativos del *IV*. Esto sucede cuando *s* = 1, 8, 64; y cuando el bloque cifrado es de *s*=128 bits simplemente sustituye las siguientes entradas al algoritmo.

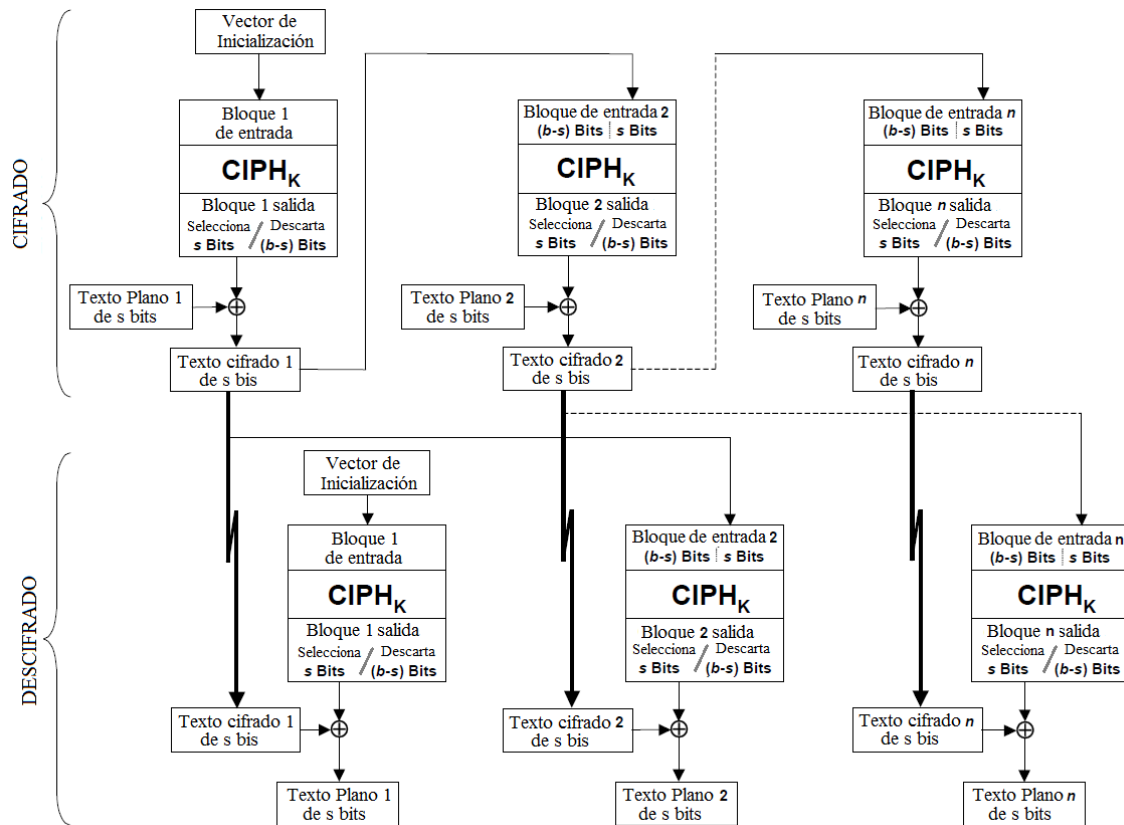


Figura 10. Esquema descriptivo del modo de operación *CFB*.

El proceso se repite con los bloques sucesivos de la entrada hasta que cada segmento del texto cifrado sea producido por cada segmento del texto plano. Cada bloque sucesivo de la entrada se codifica generalmente para producir un bloque de la salida.

En el descifrado se inicializa con el vector IV a la entrada del algoritmo y el resultado cifrado producido se opera bajo la or-exclusiva con el primer texto cifrado del proceso de cifrado. Las siguientes entradas al algoritmo son los textos cifrados obtenidos del proceso de cifrado, haciendo los mismos desplazamientos si $s=1, 8, 64$; recordando que para $s=128$ bits simplemente se sustituye el bloque de 128 bits cifrado. De igual forma el proceso se repite hasta recuperar todos los segmentos del texto plano.

3.3.1 Diseño para la Implementación de AES con CFB en VHDL

En esta sección del capítulo se mostrarán los archivos con extensión “*vhd*” desarrollados para cada operación que se lleva a cabo en el algoritmo de cifrado y descifrado. Primero se enlistarán los archivos “*vhd*” comunes a las implementaciones y en cada subsección se expondrán los archivos “*vhd*” que se desarrollaron específicamente para cada caso, presentando como están conformadas sus señales de entradas y salidas. (Elbirt A. J., 2009).

Si se toma como base el esquema mostrado en la Figura (3), se programaron cada operación que aparece en el esquema. A continuación se enlistan estas operaciones de acuerdo al orden aparecido en la Sección (3.1) y se escribirán los archivos generados para cada operación cuando se dan los procesos de cifrado y descifrado.

Para el cifrado:

- Función ByteSub: el archivo generado se llama “SBox_transformacion.vhd”
- Función ShiftRow: el archivo generado se llama “ShiftRowTransformacion.vhd”
- Función MixColumn: el archivo generado se llama “MixColumnsTransformacion.vhd”
- Función AddRoundKey: la función se llama “Add_RoundKey”

Para el descifrado:

- Función ByteSub: el archivo generado se llama “Inverse_SBox.vhd”
- Función ShiftRow: el archivo generado se llama “Inverse_ShiftRowTransf.vhd”
- Función MixColumn: el archivo generado se llama “Inverse_MixColumnsTransf.vhd”
- Función AddRoundKey: la función se llama igual “Add_RoundKey.vhd” (operación OR- exclusiva).

Dentro de otro archivo generado hay otras funciones que son comunes a los algoritmos de cifrado, este archivo generado se llama “rijndael_pack.vhd”. Además en este archivo están los valores para la SBox y la SBox_inversa, también tiene la formación de las claves para los diferentes tamaños. Entre las funciones que se encuentran en el archivo anterior están:

- función SBOX_LOOKUP;
- función INV_SBOX_LOOKUP;
- función BYTE_SUB_FUNCT;
- función INV_BYTE_SUB_FUNCT;
- función SHIFT_ROW_FUNCT;
- función INV_SHIFT_ROW_FUNCT;
- función MIX_COLUMN_FUNCT;
- función POLY_MULTFUNCT;
- función POLY_MU_LTD_FUNCT;
- función INV_MIX_COLUMN_FUNCT;
- función ADD_ROUNDKEY_FUNCT;

- función ROUNDKEY_GEN192;
- función ROUNDKEY_GEN256;

También en forma común a todas las implementaciones está el programa desarrollado para la generación de llaves y dependiendo del caso, es como se mandan llamar de acuerdo a sus tamaños y sus rondas. El archivo generado tiene el nombre de “*KeyGeneracion.vhd*”.

3.3.1.1 AES-128

En este caso se tienen los siguientes parámetros de entrada y salida para el cifrado:

texto_plano: entrada de 128 bits;
 key128: entrada de 128 bits;
 reloj_1: entrada;
 data_out: salida de 128 bits;
 Archivo generado “CifradoAES128.vhd”.

Para la generación de llaves se manda llamar por medio de la función:

- function ROUNDKEY_GEN128;
 -Para el descifrado:
 texto_cifrado: entrada de 128 bits;
 key128: entrada de 128 bits;
 reloj_1: entrada;
 data_out: salida de 128 bits;
 Archivo generado “DescifradoAES128.vhd”.

4. Diseño de la Plataforma Criptográfica

Para diseñar la plataforma criptográfica se inicia a partir de las especificaciones establecidas en la sección anterior. El método generalmente usado para intercambiar la información es la transmisión de datos serial o en serie, así se implementan dos puertos “Universal Asynchronous Receiver/Transmitter” (*UART*) para conectar la plataforma con el medio de comunicación, uno para recibir y enviar la información al usuario en el formato original, y el otro para recibir y enviar la información cifrada por el medio de transmisión. Esto permite integrar el proceso de cifrado en cualquier sistema de comunicación digital que utilice una transmisión de datos serial full-duplex asincrónica, sólo teniendo que desarrollar una interfaz para conectar el cifrador en el sistema de comunicación, dependiendo del lugar en donde será utilizado.

Por otro lado para poner la plataforma criptográfica en ejecución se decide utilizar un *FPGA*, aprovechando sus características, incluyendo su operación de alta velocidad, bajo costo, facilidad de empleo y reconfiguración. En trabajo se decidió utilizar una tarjeta de desarrollo con un integrado XC3S500E, de la familia Spartan 3E de Xilinx. Este *FPGA* es de tamaño medio, con el equivalente a 500 mil compuertas lógicas. El integrado tiene gran capacidad de poder integrar la plataforma criptográfica junto con cualquier bloque que se quiera cifrar y evaluar. Esta plataforma de desarrollo funciona a 50 [MHz], que permite configurar el *UART* para funcionar a una velocidad de 3.125 [Mbps], que es bastante para funcionar en tiempo real como la mayor parte de los sistemas de comunicación.

Ahora bien para probar la operación total del cifrador, que podría ser integrado en una línea de comunicación, primero se desarrolló el esquema de cifrado que se muestra en la Figura (11). Esta implementación consiste de un módulo con una máquina de estados que controle al dispositivo, maneje la operación de los *UART* y que lleve a cabo los procesos de cifrado y descifrado.

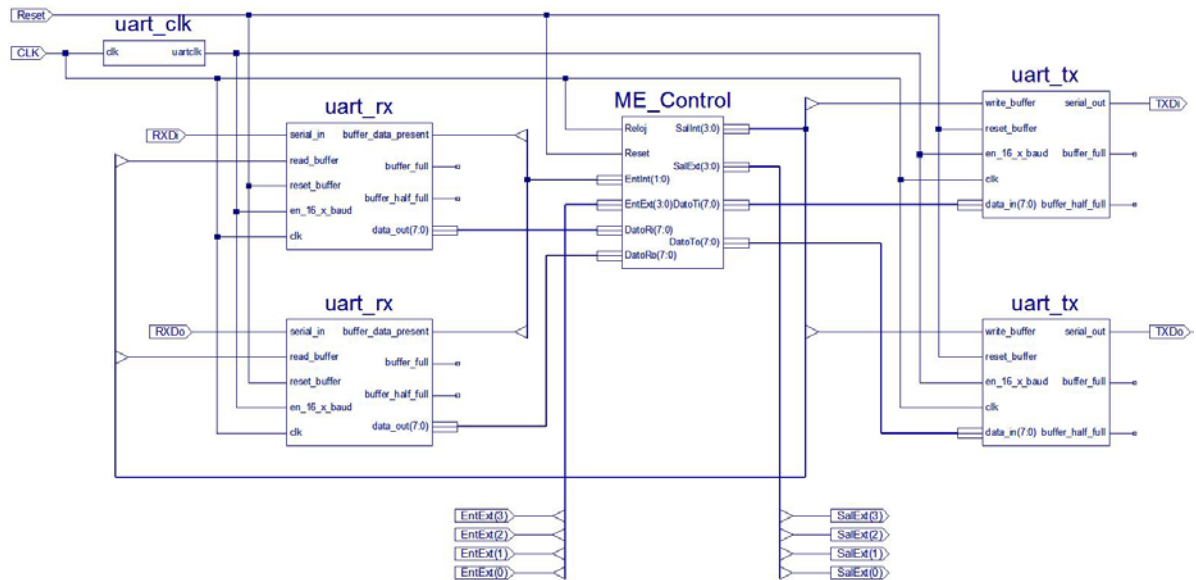


Figura 11. Diseño de la Plataforma integrando el proceso de cifrado y el bloque de control.

Con este dispositivo, se prueba la plataforma criptográfica y se utiliza para poner un proceso de cifrado en ejecución en un sistema de comunicación. En las pruebas de funcionamiento se colocaron dos de estos dispositivos entre dos PC que se comunican a través de un puerto serial RS-232. Los resultados de estas pruebas se presentan en el capítulo cinco de resultados.

Con este diseño se puede verificar que la idea general funciona, pero esto no resuelve las características que se precisaron para la plataforma criptográfica, aquello de que debe ser fácil cambiar el algoritmo criptográfico usado. Esto es debido a que como se encaja dentro de la máquina de estados del control es necesario modificar el sistema entero para realizar un cambio simple. Por lo que se modifica este sistema para llevar al elemento de seguridad fuera de este control, colocando los procesos de cifrado/descifrado en un bloque separado, de modo que el bloque de control sea solamente responsable de manejar la comunicación a través de los *UARTs*, una vez que los datos sean procesados.

Se toma la precaución para proveer al bloque de cifrado/descifrado los medios para que pueda trabajar con el bloque de control sin importar el algoritmo usado, de tal manera que la interfaz entre estos dos bloques sea tan general como sea posible y que cuente con los mecanismos necesarios para alcanzar esto. Se define un protocolo simple de apretón de manos para informar al bloque de cifrado/descifrado cuando hay ciertos datos que se procesarán y posteriormente que de informe al bloque de control cuando han terminado los procesos, tomando cuidado de no perder información en el proceso evitando mandar más datos al bloque cifrador hasta que no haya terminado el proceso de los datos actuales.

Al final, aunque el alcance de este reporte está limitado al uso del AES con CFB en la plataforma criptográfica de comunicación para datos, el sistema puede tener bloques de cifrado/descifrado intercambiables del mismo tipo con algunos otros algoritmos criptográficos integrados, de modo que cuando se tiene que implementar un nuevo dispositivo que cifre/descifre, se desarrolle el bloque con el algoritmo criptográfico deseado y se conecte con la plataforma criptográfica para tener así un sistema versátil para la protección de datos en una comunicación. El dispositivo que resulta se puede observar en la Figura (12), donde la parte intercambiable sería el bloque con el nombre de cifrador con sus dos operaciones cifrado y descifrado.

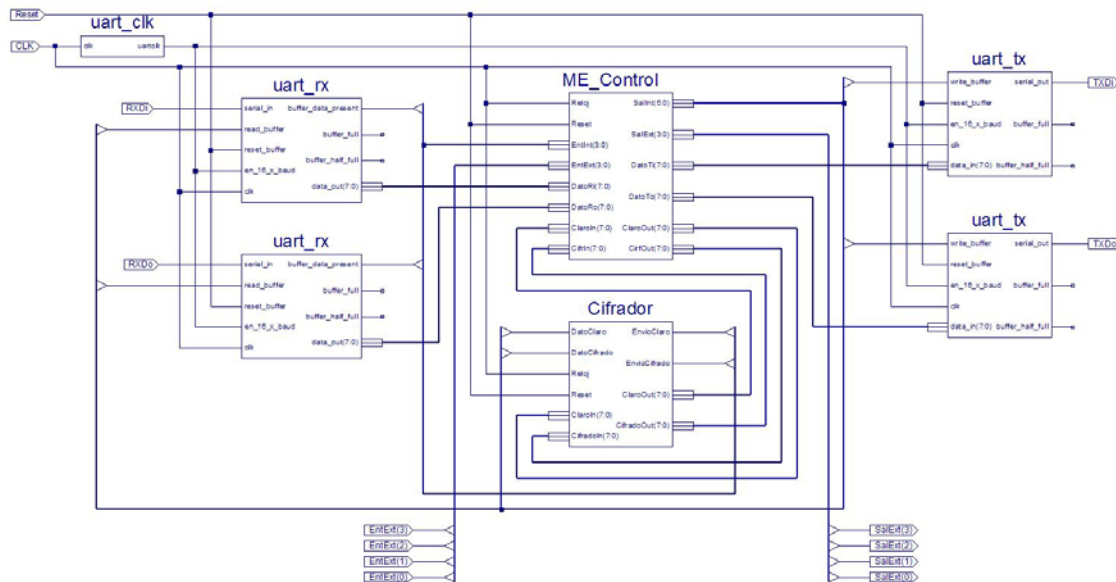


Figura 12. Diseño de la Plataforma Criptográfica incorporando el proceso de cifrado/descifrado como un bloque independiente.

5. Principales características del UART

El *UART* controla los puertos y dispositivos serie para comunicaciones. La tarjeta de desarrollo Starter kit Spartan 3E *FPGA* tiene conectados dos puertos RS-232, uno con conector hembra *DB9 DCE* y otro con conector macho *DTE*. Ya están desarrollados los bloques para generar el módulo que se encarga de implementar la *UART* en *VHDL* para cada puerto serie de la tarjeta. La mayoría de los ordenadores modernos utilizan el chip *UART 16550*, que soporta velocidades de transmisión de hasta 921,6 [Kbps]. Las funciones principales de una *UART* son de manejar las interrupciones de los dispositivos conectados al puerto serie y de convertir los datos en formato paralelo, transmitidos al bus de sistema, a datos en formato serie, para que puedan ser transmitidos a través de los puertos y viceversa, (<http://es.wikipedia.org/uart>, 2013)

El paquete de los bloques contiene un par de macros, los cuales han sido optimizados para usarse en varias tarjetas de desarrollo fabricadas por Xilinx. Los macros proporcionan una funcionalidad sencilla tanto para transmitir como para recibir, cada uno con las siguientes características:

- 1 bit de inicio
- 8 bits de datos (transmitidos serialmente y recibidos primero por el bit menos significativo)
- Sin Paridad
- 1 bit de parada

Estos dispositivos implementados son capaces de comunicarse con cualquier otra interface *UART* que sea compatible, como por ejemplo con el protocolo de comunicación con las PC (se proporcionan variaciones de niveles para generar señalamientos RS-232).

Cada macro contiene un buffer *FIFO* embebido de 16 bytes que es justamente su tamaño, su tamaño pequeño lo hace ideal para ser usados junto con macros de procesadores *KCPSM* y *KCPSM-II*, ver Figura (13).

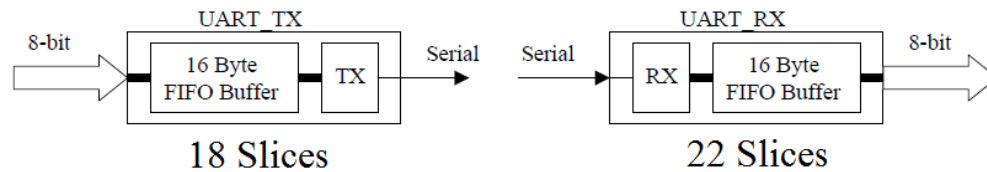


Figura 13. Esquemas de las estructuras para los UART, izquierda transmisión derecha recepción.

En cuanto a velocidad, a pesar de que soporta tasas de 9600 bauds, estos macros son capaces de exceder los 10 [Mbps] ofreciendo una manera fácil de comunicación entre dos dispositivos fabricados por Xilinx, (Chapman K., 2003).

Como su tamaño y desempeño se mide de acuerdo a los reportes que se dan sobre las diferentes plataformas de desarrollo, los resultados se mostrarán conforme a la Spartan 3E.

5.1 Operación de la UART

Un UART por su forma de construcción del transmisor y receptor no están sincronizados, sin embargo ambos utilizan un tiempo de referencia, el cual tiene una tolerancia que permite la transferencia serial de cada byte de datos. El dato es transmitido serialmente primero el *LSB* a una razón de bits dada (tasa en *BAUD*), la cual es conocida por el transmisor y el receptor. Así el transmisor puede empezar a enviar datos en cualquier momento, el receptor necesita un método de identificación cuando el primer bit (*LSB*) ha sido enviado. Esto es producido por el trasmisor enviando una señal de inicio activada en bajo mientras dura la transmisión de un bit, como lo muestra la Figura (14).

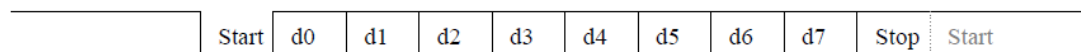


Figura 14. Esquema de tiempo para la transmisión de un byte.

El receptor usa el flanco de bajada del bit de inicio para iniciar un circuito temporal interno, este tiempo es usado para muestrear el valor de la entrada serial en un punto, el cual es aproximadamente a la mitad de la posición de cada bit de dato. Aquí es donde el dato debería ser estable. Después de que el último bit del dato (*MSB*) se ha leído, el receptor checa si el bit de parada (en alto) es el valor esperado, el cual ayuda a confirmar la operación correcta.

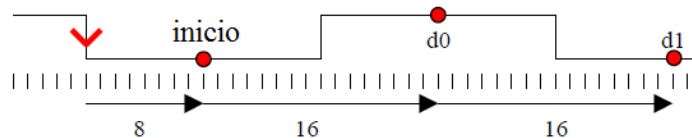


Figura 15. Esquema temporal para la recepción de datos.

Esta es la forma en que el receptor vuelve a sincronizar (con el circuito temporal interno) en el flanco de caída cada bit de inicio, el tiempo de la transmisión y recepción solamente necesitan la misma tolerancia de la mitad del periodo del bit en cada 10 periodos de bit. Este 5% de precisión no es realmente alcanzado por cualquier sistema digital.

En común con muchas soluciones *UARTs*, estos macros esperan a que un tiempo de referencia sea proporcionado en forma de una señal habilitadora ("en_16_x_baud"), la cual se aplica a razón de una vez por 16 bits.

5.2 Condición para romper con la operación UART

El estatus normal una línea serial es activa o en alto, de esta forma un bit nuevo de inicio se identifica por un flanco de caída. Bajo una condición de ruptura, un transmisor continuamente forzará un nivel bajo en la línea (posiblemente debido a la no alimentación). A pesar de esto el receptor detectará esto como un bit de inicio seguido de puros cero en los datos, el bit de paro no será válido y este dato incorrecto será descartado, Figura (16).

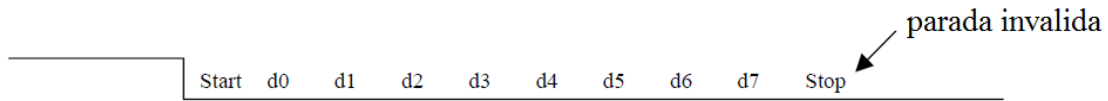


Figura 16. Esquema temporal de una parada inválida.

El receptor esperará hasta la línea regrese a alto y solamente se volverá a sincronizar al siguiente flanco de caída asociado con el bit de inicio.



Figura 17. Esquema temporal de una operación de ruptura.

El macro del transmisor naturalmente no transmitirá bajo una condición de ruptura. El macro del receptor sin embargo entenderá esta situación y operará como se describió.

5.3 Tasa temporal en BAUDs

A partir de los macros de transmisión y recepción se derivan los tiempos de la señal de referencia "en_16_x_baud". Como el nombre sugiere, esta señal debería ser aplicada al macro de la tasa el cual es 16x la tasa de bits deseada, Figura (18).

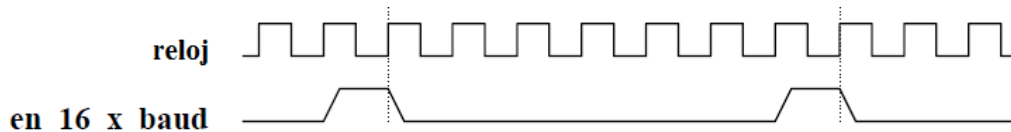


Figura 18. Esquema temporal en BAUD's.

Así la señal es usada como un reloj habilitador dentro de los macros, esto debería ser proporcionado por un reloj sincronizado y tener una duración de pulso de un ciclo de reloj solamente (se requiere al menos una tasa máxima de reloj/16).

5.3.1 El Macro UART_Tx

El UART transmisor proporcionado está formado por un conjunto de tres archivos VHDL, el nivel más alto es el archivo "uart_tx.vhd" es usado junto con el buffer FIFO "bbfifo_16x8.vhd" y la constante (K) compactada en el módulo UART transmisor "kuart_tx.vhd". La Figura (19) muestra la unión de los archivos descritos en un bloque.

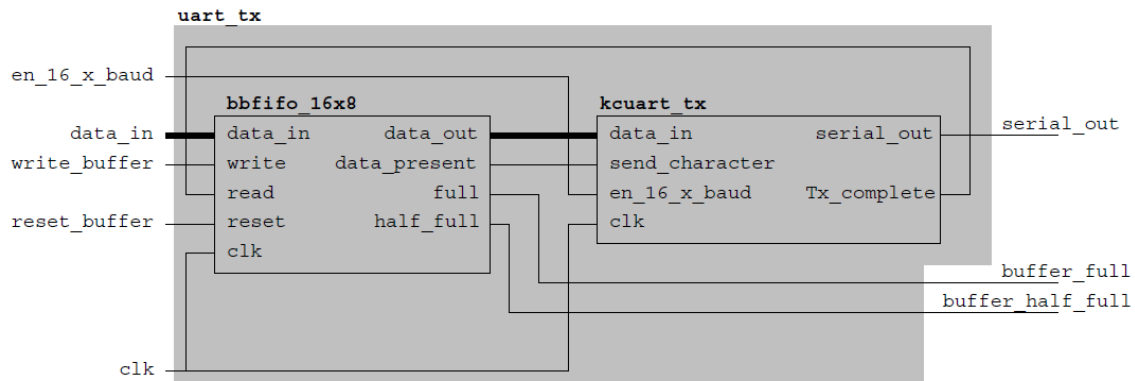


Figura 19. Esquema del módulo UART de transmisión.

En la Figura (19) se muestra el módulo UART de transmisión, en la misma ilustración se pueden ver cuáles son las señales tanto de entrada como de salida. Estas señales se enlistan a continuación:

- *en_16_x_baud*: proporciona el tiempo de referencia para la transmisión serial.
- *data_in*: el byte de datos en paralelo (8-bit) de entrada que serán transmitidos serialmente.
- *write_buffer*: una señal alta indica que los datos corrientes están sobre la entrada *data_in*.
- *reset_buffer*: una señal de entrada activa en alto causa que el buffer interno de 16-byte sea reiniciado y que todos los datos corrientes en el buffer se pierdan.
- *reloj*: usada por todos los elementos del macro síncronos.
- *serial_out*: son los datos de salida conformados por 1 bit de inicio, 8 bit de datos (primero el LSB) y 1 bit de parada.
- *buffer_full*: cuando el FIFO de 16-byte está lleno, esta salida se activa en alto.
- *buffer_half_full*: cuando el FIFO de 16-byte toma 8 o más byte de datos esperando a ser transmitidos.

5.3.2 Operación del buffer de Transmisión

Si se requiere, el archivo "bbfifo_16x8.vhd" puede ser usada para proporcionar un *buffer FIFO* de 16-byte síncrono usando sólo 8 "*slices*" (número de capas o niveles, que se puede ver como medida de ocupación dentro del *FPGA* que se emplea). Como parte del macro "*UART_TX*", la salida del lado del *FIFO* está bajo el control del transmisor "*kcuart_tx*". El "*reset_buffer*" puede ser usado para descartar el contenido corriente del buffer y necesita ser usado con cuidado para no destruir información valiosa. Cabe la posibilidad de que cualquier dato sobre la transmisión serial en progreso sea corrompido. En la mayoría de los casos esta señal no será usada, cuando el FIFO esté lleno se reiniciará a la siguiente configuración.

El *buffer FIFO* es usado para aceptar bytes de datos para transmitir cuando escribe en el macro. El buffer es leído automáticamente por el circuito "*kcuart_rx*" para pasar el dato a la línea serial. El dato se escribe en el buffer en el flanco de subida del reloj cuando la señal "*write_buffer*" está activa. Pueden escribirse datos asilados o un conjunto de varios bytes, ver Figura (20).

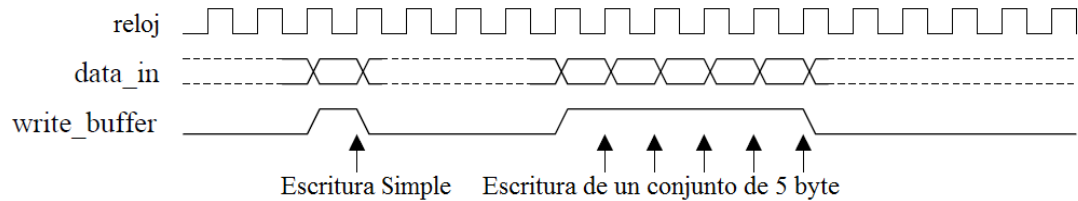


Figura 20. Esquema temporal de la escritura de bytes en el buffer.

Cuando el buffer *FIFO* se llena, la señal “*buffer_full*” se activará. Esta señal permanecerá activa hasta que el dato *MSB* del dato corriente se haya completado (es decir, cuando el próximo bit de parada haya sido transmitido). Ningún dato puede escribirse al buffer cuando está lleno, Figura (21).

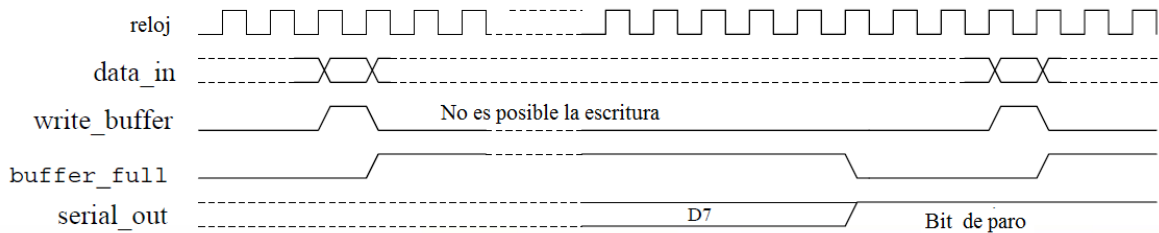


Figura 21. Esquema temporal para cuando no es posible la escritura.

5.3.3 El Macro UART_Rx

El *UART* receptor está formado por un conjunto de tres archivos en *VHDL*. El nivel principal es el archivo “*uart_rx.vhd*” que es usado en combinación con los módulos de la constante (*k*), el archivo de la *UART* receptor “*kcuart_rx*” y el archivo del buffer *FIFO* “*bbfifo_16x8.vhd*”.

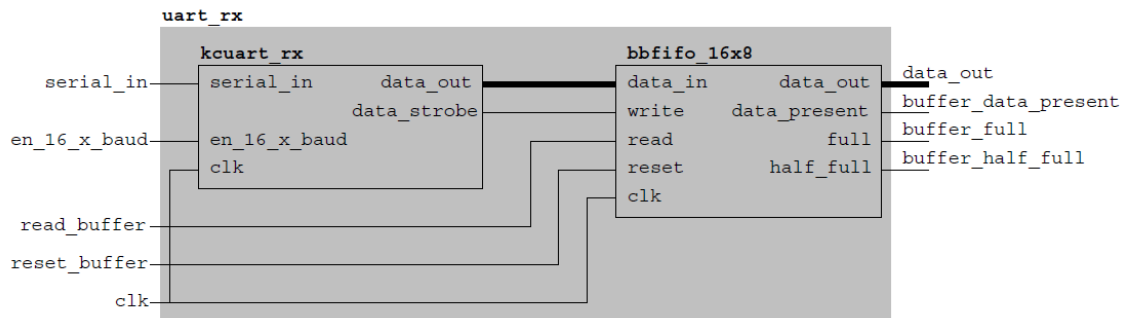


Figura 22. Esquema del módulo *UART* de recepción.

En la Figura (22) se muestra el módulo *UART* de recepción, en la misma ilustración se pueden ver cuáles son las señales tanto de entrada como de salida. Estas señales se enlistan a continuación:

- *serial_in*: este es el dato serial conformada por 1 bit de inicio, 8 bit de datos (primero el *LSB*) y un bit de paro.
- *data_out*: El byte dato en paralelo (8-bit) que ha sido recibido, este dato *serial_out*: es válido cuando la señal “*buffer_data_present*” esta activa.
- *read_buffer*: una señal alta indica que los datos proporcionados por *data_out* han sido leídos.

- *reset_buffer*: una señal de entrada activa en alto causa que el buffer interno de 16-bytes sea reiniciado y que todos los datos corrientes en el buffer se pierdan.
- *en_16_x_baud*: proporciona el tiempo de referencia para la transmisión serial.
- *buffer_data_present*: cuando el buffer interno contiene uno o más bytes de los datos recibidos, esta señal se activará y el dato válido estará disponible en el puerto “data_out”.
- *buffer_full*: cuando el FIFO de 16-byte está lleno, esta salida se activa en alto.
- *buffer_half_full*: cuando el FIFO de 16-byte toma 8 o más byte de datos esperando a ser transmitidos.
- *reloj*: usada por todos los elementos del macro síncronos.

5.3.4 Operación del buffer de Recepción

Si se requiere, el archivo “*bbfifo_16x8.vhd*” puede ser usado para proporcionar un buffer FIFO de 16-byte síncrono usando sólo 8 “*slices*”. Como parte del macro “*uart_rx*”, la entrada del lado del FIFO está bajo el control del receptor “*kcuart_rx*”. El “*reset_buffer*” puede ser usado para descartar el contenido corriente del buffer y necesita ser usado con cuidado para no destruir información valiosa. La mayoría de los casos esta señal no será usada, cuando el FIFO esté lleno se reiniciará a la siguiente configuración. Cuando el dato se presente (el *buffer_data_present*=1), la lectura puede realizarse individual o en conjunto desde el buffer FIFO, Figura (23).

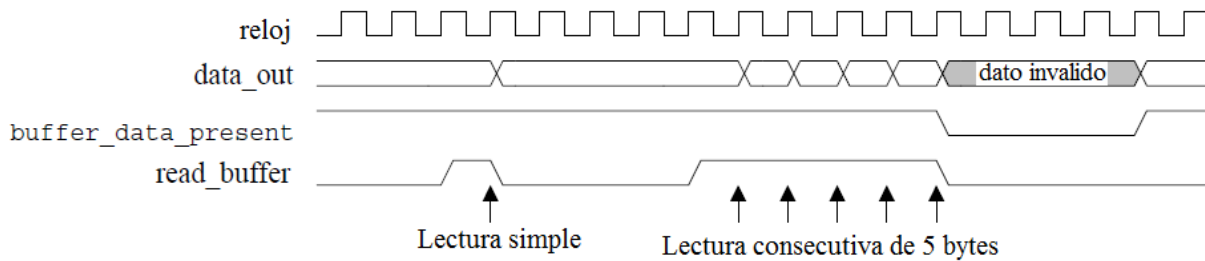


Figura 23. Lectura del buffer individual o en conjunto.

El buffer FIFO es escrito automáticamente por el circuito “*kcuart_rx*” para cada transmisión serial valida del “*carácter*” capturado por el receptor. El dato es escrito en el buffer tan pronto como el bit de paro se ha confirmado en alto, pero solamente si el buffer no está lleno, Figura (24).

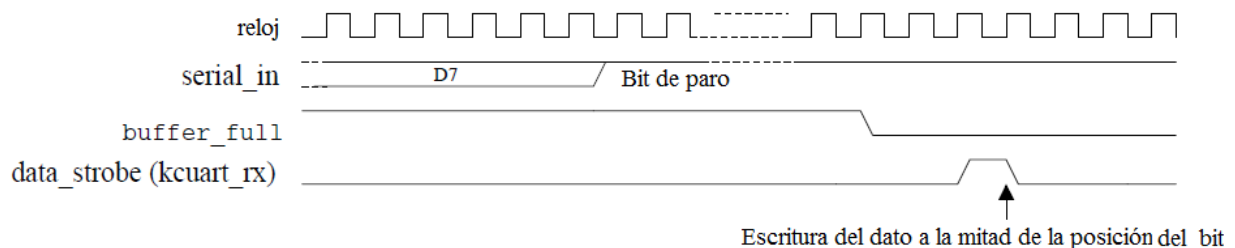


Figura 24. Esquema para la indicación del buffer lleno.

Lo anterior puede ser visto como cuando el buffer FIFO se llena, no se previene el próximo carácter que se va a recibir. Por lo tanto el sistema anfitrión tiene el tiempo asociado con 10 bits seriales de lectura con al menos un byte de dato recibido desde FIFO extraviado.

5.4 Configuración de las Interfaces Seriales en la Spartan 3E

En esta sección se presenta la configuración que tienen las interfaces seriales en la tarjeta de desarrollo Starter Kit Spartan 3E de Xilinx, además de mostrar un esquema de conexión según la plataforma diseñada para realizar la comunicación serial segura.

La Figura (25) muestra los dos puertos serie que tiene la tarjeta Spartan 3E en la parte superior derecha, en realidad son dos conectores físicos DB9 denotados por DCE (hembra) y DTE (macho).

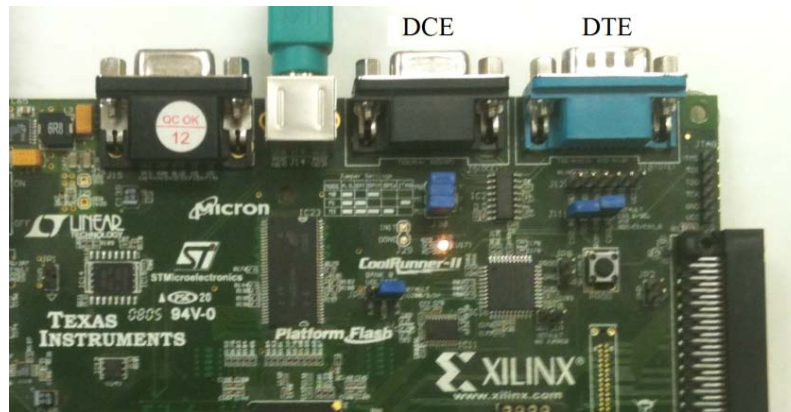


Figura 25. Spartan 3E con dos puertos seriales.

El puerto estilo *DCE* se conecta directamente con el conector del puerto serial disponible en la mayoría de las computadoras personales vía un cable serial “*straight-through*” estándar. Un adaptador de géneros o cables cruzados no se requieren. Se usa el conector estilo *DTE* para controlar los otros periféricos *RS-232*, tales como módems o impresoras, o realice la prueba de “*loopback*” simple con el conector del *DCE*. La Figura (26) muestra la configuración de los pines para ambos estilos de conectores.

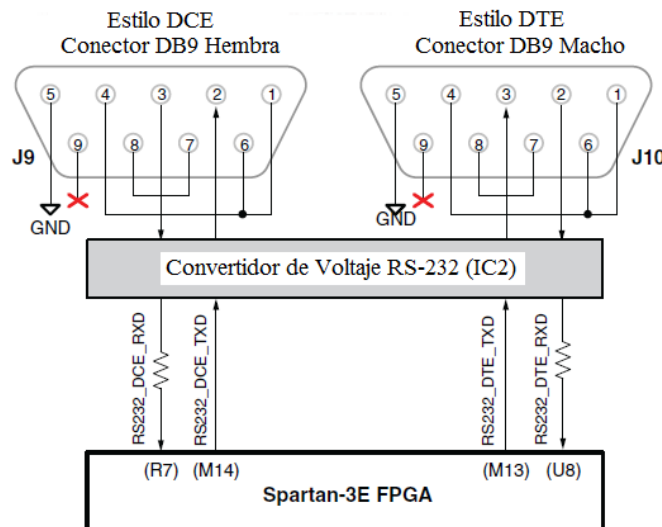


Figura 26. Puertos Seriales RS-232.

El control de flujo del hardware no se apoya en el conector. Las señales *DCD*, *DTR*, y *DSR* del puerto se conectan juntas, según las indicaciones de la Figura (26). Similarmente, las señales *RTS* y *CTS* del puerto conectan juntas. La misma figura también muestra los pin de conexión a la tarjeta Spartan 3E de acuerdo a las siguientes señales:

- *RS-232_DCE_RXD* que corresponde al pin R7
- *RS-232_DCE_TXD* que corresponde al pin M14
- *RS-232_DTE_RXD* que corresponde al pin U8
- *RS-232_DTE_TXD* que corresponde al pin M13

A continuación se presenta el esquema para las conexiones usado de acuerdo a la plataforma criptográfica para la comunicación serial entre dos tarjetas Spartan 3E desarrollada, Figura (27). (Spartan-3E FPGA Starter Kit Board User Guide, 2008).

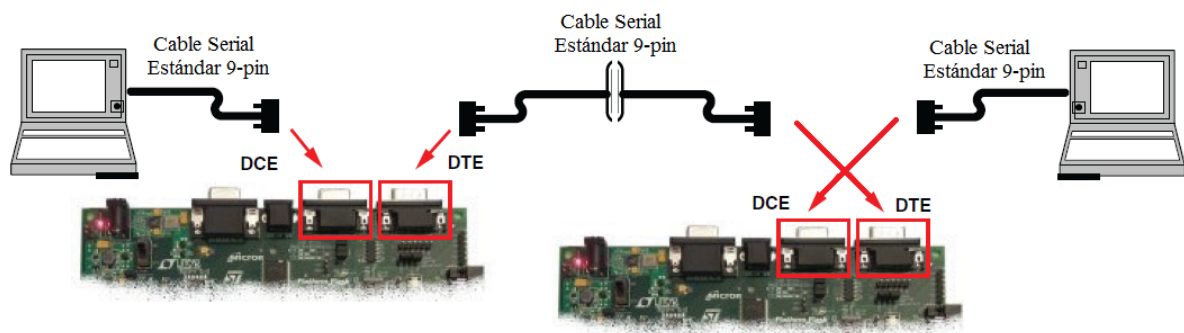


Figura 27. Esquema de conexiones para la plataforma criptográfica.

6. Resultados

Para probar la operación de la plataforma, primero se desarrollan bloques cifrados con algoritmos criptográficos simples, para medir el funcionamiento de la plataforma sin el efecto de la respuesta en tiempo de un bloque cifrado que resulta de un algoritmo criptográfico complejo. Esta es la razón del porqué las primeras pruebas fueron realizadas usando Cesar, Lucifer y algoritmos polialfabéticos. (Menezes A. J., et. al., 1997).

Estos primeros cifradores fueron desarrollados originalmente dentro de la máquina de estados en el módulo de control de la Figura (2). Con esta clase de operación se encontró que el retardo total, incluyendo la lectura de un carácter desde la entrada del *UART*, el cifrado/descifrado del carácter y de escritura de ese carácter en la salida *UART* fue hecho de un máximo de 5 ciclos de reloj, como la tarjeta de desarrollo Spartan 3E trabaja a 50 [MHz], se tiene un tiempo total del proceso de 100 [nanosegundo]. Para establecer si este tiempo puede afectar al sistema de comunicación se debe comparar con el tiempo que toma para enviar un carácter como resultado de la velocidad de transmisión. Los puertos del *UART* que se utilizan en la plataforma necesitaron dividir la frecuencia de reloj por 16 para generar la frecuencia de transmisión de datos que garantiza una comunicación exacta. Consecuentemente se tiene 3.125 [Mbps] como velocidad máxima de transmisión.

De esto se concluye que la plataforma puede funcionar correctamente sin afectar la velocidad de transmisión del sistema original mientras el tiempo de proceso de datos total no exceda 16 ciclos de reloj. Desde este punto de vista, la plataforma resuelve completamente este requisito.

Para probar la operación del cifrado se toma un sistema de comunicación que consiste en dos computadoras *PC* que se comunican a través de un puerto serial *RS-232*, poniendo un cifrador en cada extremo del cable de la conexión, según las indicaciones de la Figura (25). Fue encontrado que la comunicación no fue alterada al enviar la información a cualquiera de las velocidades soportadas por el puerto serial *RS-232* de la *PC*, que incluye velocidades de hasta 115200 [bps].

Una vez que se validó la operación apropiada de la plataforma criptográfica, se hicieron las modificaciones apropiadas para extraer el algoritmo de cifrado del módulo de control y dejarlo como módulo externo que puede ser intercambiado fácilmente. El sistema que resulta tiene la estructura mostrada en la Figura (26), y se probó su desempeño con los tres algoritmos básicos de cifrado usados previamente. El resultado de esta modificación se podría verificar que la plataforma criptográfica mantiene un desempeño similar que la versión anterior, solamente aumento en un ciclo de reloj el tiempo de reacción total, tomando 6 ciclos de reloj, que todavía está por debajo del sistema de límite de 16 ciclos de reloj dado por el *UART*.

Entonces se evalúa el desempeño de la plataforma con un algoritmo más robusto, específicamente con el algoritmo estándar *AES-128* con modo de operación *8CFB* (cifrador por flujo), que es más conveniente para la clase de aplicaciones que se desean implementar. La respuesta en tiempo que necesita el módulo que cifra y descifra depende del algoritmo de cifrado que en particular se utiliza. Se encontró que se tuvo que establecer un protocolo de apretón de manos para evitar soltar la información, esto es porque las señales de control fueron agregadas para informar al módulo de cifrado/descifrado cuando los datos están disponibles para ser procesados, y también informar al módulo de control cuando se termina el proceso de cifrado/descifrado.

Esto dio lugar a un pequeño aumento en el tiempo de respuesta del sistema, dejando en 7 ciclos de reloj la respuesta total del tiempo, este aumento tiene que ser agregado al tiempo en que se cifra y descifra en el módulo cifrador. También deberá ser considerado cuando otras implementaciones de algoritmos tomen más tiempo para procesar los datos.

En la Figura (28) se presenta el resumen del diseño de la Plataforma Criptográfica que genera Xilinx de los recursos consumidos, en porcentajes.

Plataforma Project Status (07/19/2013 - 21:55:04)			
Project File:	PlatafCrip1.xise	Parser Errors:	No Errors
Module Name:	Plataforma	Implementation State:	Programming File Generated
Target Device:	xc3s500e-5fg320	•Errors:	No Errors
Product Version:	ISE 13.3	•Warnings:	48 Warnings (15 new, 0 filtered)
Design Goal:	Balanced	•Routing Results:	All Signals Completely Routed
Design Strategy:	Xilinx Default (unlocked)	•Timing Constraints:	All Constraints Met
Environment:	System Settings	•Final Timing Score:	0 (Timing Report)

Device Utilization Summary				
Logic Utilization	Used	Available	Utilization	Note(s)
Total Number Slice Registers	189	9,312	2%	
Number used as Flip Flops	132			
Number used as Latches	57			
Number of 4 input LUTs	216	9,312	2%	
Number of occupied Slices	157	4,656	3%	
Number of Slices containing only related logic	157	157	100%	
Number of Slices containing unrelated logic	0	157	0%	
Total Number of 4 input LUTs	216	9,312	2%	
Number used as logic	144			
Number used as Shift registers	72			
Number of bonded IOBs	14	232	6%	
IOB Flip Flops	4			
Number of BUFMUXs	1	24	4%	
Average Fanout of Non-Clock Nets	3.40			

Figura 28. Resumen del diseño sobre la Plataforma Criptográfica.

El reporte de Mapeo y el reporte de traza que genera la Plataforma de Xilinx sobre el proyecto generado, se presentan en el Apéndice IV después de este capítulo.

Algunos datos obtenidos por el reporte de traza, en cuanto a las frecuencias de los módulos de transmisión y recepción de datos son los siguientes:

Tabla 8. Frecuencias de los bloques de comunicación serial.

UART_TX Di 5.617 [ns] → 178.0 [MHz]
UART_TX Do 5.617 [ns] → 178.0 [MHz]
UART_RX Di 3.967 [ns] → 252.2 [MHz]
UART_RX Do 3.949 [ns] → 253.2 [MHz]

Para calcular los parámetros en cuanto al rendimiento de la plataforma criptográfica se tomaron los que se mencionan en (López E. 2006), pág.87, estos son el área ocupada del *FPGA xc3s500e-5fg320* de la Spartan 3E en unidades de [Slices] y la velocidad de procesamiento calculada por:

$$Throughput = \frac{\text{frecuencia permitida} \times \text{Número de Bits}}{\text{Número de Rondas}} \left[\frac{\text{bits}}{\text{s}} \right] \quad (6)$$

En la figura (27) en la parte del resumen de utilización del dispositivo marca 157 de 4656 [Slices] y esto corresponde a un 3% de ocupación del chip *xc3s500e-5fg320*.

Para calcular la velocidad de procesamiento se van a tomar dos periodos convertidos a frecuencias, uno de ellos es obtenido del reporte de síntesis del Xilinx vs 3.13 mostrados en el Apéndice IV de este reporte, está clasificado como el periodo mínimo estimado 5.259 [ns] con el cual se obtiene la frecuencia máxima de 190.150 [MHz]. El otro periodo es obtenido por el reporte de traza en la parte del “*place-and-route*” que marca 5.621 [ns] con una frecuencia de 177.904 [MHz], el cual en la nota del reporte dice que es más preciso.

Realizando el cálculo para la velocidad de proceso con estas dos frecuencias, tomando 8 bits de salida del algoritmo *CFB8-AES128* y que la operación de cifrado la realiza en 1 ciclo, se obtiene:

$$\text{Throughput}_{\tau_1} = 190.150 \text{ [MHz]} * 8[\text{bits}] / 1 = 1521.2[\text{Mbps}]$$

$$\text{Throughput}_{\tau_2} = 177.904 \text{ [MHz]} * 8[\text{bits}] / 1 = 1423.23[\text{Mbps}]$$

Estos resultados son con base a los periodos citados y los 8 bits de salida del modo de operación *CFB8*, sin embargo los datos de inicio para el algoritmo *AES-128* se tienen que tomar en cuenta. Así con el vector de inicio “IV” de 128-bits, la clave de 128-bits para el *AES-128* y donde el cifrado del *AES-128* se obtiene después de 5 ciclos de trabajo, que se procesan en el sistema desarrollado se pueden calcular otras velocidades de proceso de la siguiente forma:

$$\text{Throughput}_{\tau_3} = 190.150 \text{ [MHz]} * 128[\text{bits}] / 5 = 4.867[\text{Gbps}]$$

$$\text{Throughput}_{\tau_4} = 177.904 \text{ [MHz]} * 128[\text{bits}] / 5 = 4.554[\text{Gbps}]$$

$$\text{Throughput}_{\tau_5} = 190.150 \text{ [MHz]} * 256[\text{bits}] / 5 = 9.735[\text{Gbps}]$$

$$\text{Throughput}_{\tau_6} = 177.904 \text{ [MHz]} * 256[\text{bits}] / 5 = 9.108[\text{Gbps}]$$

Estos cálculos son sólo del sistema de cifrado, que no están muy alejados si se toman los datos de la Tabla (8) de los periodos y frecuencias de los bloques de recepción y transmisión para la comunicación serial. En las conclusiones se darán otros resultados tomando en cuenta la compilación de la plataforma de comunicación serial.

Dentro de la siguiente Figura (29) se muestra un video del funcionamiento de la Plataforma Criptográfica, empleando para ello una terminal sobre la Laptop conectada a un puerto *USB* mediante un convertidor *RS-232*. La terminal se configura para la comunicación serial con el *FPGA* de la *Spartan 3E* a 38400 [BAUDs], mediante el *COM8* que es como lo reconoce la Laptop y se da de alto el “eco” para ver al mismo tiempo la transmisión y la recepción mediante un pequeño “loop” conectado al puerto serial *DTE*.



Figura 29. Video del funcionamiento de la Plataforma Criptográfica. Nota hay que hacer doble clic sobre la imagen.

Conclusiones y Perspectivas

La aportación de este trabajo se da cuando todas las implementaciones desarrolladas se integran en la Plataforma Criptográfica para Comunicación Serial Segura entre dos partes, explotando por ahora las capacidades que tiene una comunicación Serial *UART*. Para la adecuación a la plataforma de comunicación se implementó el modo de operación *CFB8* al *AES-128* que convierte al cifrado por bloques a un cifrado por flujo. Una nota importante es que cuando se realiza la compilación del sistema en Xilinx, tarda alrededor de 36 minutos en una Laptop con un procesador Intel(R) Core(TM)2 Duo, 2.10 GHz/2094 [MHz]. Luego se programaron los *FPGA* de dos tarjetas Spartan 3E y se realizaron las pruebas mostrando los resultados obtenidos. Este trabajo abre una gran posibilidad de aplicaciones sobre cualquier tipo de comunicación alambrada o inalámbrica (interfaces), donde se requiera implementar al mismo tiempo servicios y mecanismos de seguridad. Al colocar este tipo de dispositivos dentro del canal de comunicación, se puede darse cuenta que para la comunicación juega un papel importante los pasos a seguir para que la información sea enviada y recibida íntegramente sin pérdidas. Por lo cual, para este tipo de tareas, hay que tomar en cuenta el diseño de los protocolos de comunicación.

ANEXO I

Anexo I.- Códigos en VHDL de los algoritmos: CFB8_AES-128

En este apéndice se presentan los códigos que se programaron para cada uno de los Algoritmos de Cifrado y Descifrado.

Cifrado CFB8 AES-128 *****

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
use work.rijndael_package.all;
use IEEE.NUMERIC_STD.ALL;
--use UNISIM.VComponents.all;
entity CifradoCFB128 is
    Port ( vec_inicio : in STD_LOGIC_VECTOR (127 downto 0);
          texto_plano : in STD_LOGIC_VECTOR (7 downto 0);
          key128      : in STD_LOGIC_VECTOR (127 downto 0);
          data_out    : out STD_LOGIC_VECTOR (127 downto 0);
          cipher      : out STD_LOGIC_VECTOR (7 downto 0);
          reloj_1     : in STD_LOGIC;
          reset       : in STD_LOGIC;
          --empieza   : in STD_LOGIC;
          --termina   : out STD_LOGIC);
end CifradoCFB128;
architecture Behavioral of CifradoCFB128 is
    constant call_f: INTEGER := 10;
    constant Nr : INTEGER := 10;
    type ram_type is array (0 to call_f) of std_logic_vector (127 downto 0);
    signal clave: ram_type;
    --signal itera_temp: std_logic_vector(127 downto 0):= vec_inicio; --IV es el vector de inicio a
desplazar
    begin
    process(key128)
        variable call: INTEGER:=call_f;
        variable cla_temp, tempor: std_logic_vector (127 downto 0);
        begin
        clave(0)<=key128;
        cla_temp :=key128;
        for i in 1 to call loop
            tempor := ROUNDKEY_GEN128(cla_temp,i);
            clave(i) <= tempor;
            cla_temp:=tempor;
        end loop;
        end process;
        process(reloj_1,reset)
            variable Nr: INTEGER:=10;
            variable state,b_sub,s_row,m_col,a_rou,b_sub_f,s_row_f,a_rou_f: STD_LOGIC_VECTOR (127
downto 0);
            variable ciph_temp: STD_LOGIC_VECTOR (7 downto 0);
            variable itera_temp: std_logic_vector(127 downto 0):= vec_inicio;
            begin
            -- if <condition> then
            --     <statement>
```

```

-- elsif <condition> then
--   <statement>
-- else
--   <statement>
-- end if;
if (reloj_1'event and reloj_1 = '1') then
    if reset = '1' then
        itera_temp := vec_inicio;
    end if;
    state:= ADD_ROUNDKEY_FUNCT(itera_temp,clave(0));
    for i in 1 to Nr-1 loop
        b_sub := BYTE_SUB_FUNCT(state);
        s_row := SHIFT_ROW_FUNCT(b_sub);
        m_col := MIX_COLUMN_FUNCT(s_row);
        a_rou := ADD_ROUNDKEY_FUNCT(m_col,clave(i));
        state := a_rou;
    end loop;
    b_sub_f := BYTE_SUB_FUNCT(a_rou);
    s_row_f := SHIFT_ROW_FUNCT(b_sub_f);
    a_rou_f := ADD_ROUNDKEY_FUNCT(s_row_f,clave(Nr));
    data_out <= a_rou_f;
    ciph_temp := a_rou_f(127 downto 120) xor texto_plano ; --XOR de la salida de 8-bits con
texto_plano
    ciphер <= ciph_temp;
    --Se obtiene el cifrado del primer bloque del texto plano
    for i in 119 downto 0 loop
        itera_temp(i+8) := itera_temp(i);
    end loop;
    --itera_temp(7 downto 0) := ciph_temp; --cifrado
    itera_temp(7 downto 0) := ciph_temp; --descifrar

end if;
end process;
end Behavioral;

```

Descifrado CFB8 AES-128 *****

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
use work.rjndael_package.all;
use IEEE.NUMERIC_STD.ALL;
--use UNISIM.VComponents.all;
entity DescifradoCFB128 is
    Port ( vec_inicio  : in STD_LOGIC_VECTOR (127 downto 0);
          texto_cifrado: in  STD_LOGIC_VECTOR (7 downto 0);
          key128       : in  STD_LOGIC_VECTOR (127 downto 0);
          data_out      : out STD_LOGIC_VECTOR (127 downto 0);
          tex_plano     : out STD_LOGIC_VECTOR (7 downto 0);
          reloj_2       : in STD_LOGIC;
          reset         : in STD_LOGIC;
          --empieza     : in STD_LOGIC;
          --termina      : out STD_LOGIC);
end DescifradoCFB128;

```



```

architecture Behavioral of DescifradoCFB128 is
    constant call_f: INTEGER := 10;
    constant Nr : INTEGER := 10;
    type ram_type is array (0 to call_f) of std_logic_vector (127 downto 0);
    signal clave: ram_type;
    --signal itera_temp: std_logic_vector(127 downto 0):= vec_inicio; --IV es el vector de inicio a
desplazar
begin
    process(key128)
        variable call: INTEGER:=call_f;
        variable cla_temp, tempor: std_logic_vector (127 downto 0);
        begin
            clave(0)<=key128;
            cla_temp :=key128;
            for i in 1 to call loop
                tempor := ROUNDKEY_GEN128(cla_temp,i);
                clave(i) <= tempor;
                cla_temp:=tempor;
            end loop;
        end process;
    process(reloj_2,reset)
        variable Nr: INTEGER:=10;
        variable state,b_sub,s_row,m_col,a_rou,b_sub_f,s_row_f,a_rou_f: STD_LOGIC_VECTOR (127
downto 0);
        variable ciph_temp: STD_LOGIC_VECTOR (7 downto 0);
        variable itera_temp: std_logic_vector(127 downto 0):= vec_inicio;
        begin
            if (reloj_2'event and reloj_2 = '1') then
                if reset = '1' then
                    itera_temp := vec_inicio;
                end if;
                state:= ADD_ROUNDKEY_FUNCT(itera_temp,clave(0));
                for i in 1 to Nr-1 loop
                    b_sub := BYTE_SUB_FUNCT(state);
                    s_row := SHIFT_ROW_FUNCT(b_sub);
                    m_col := MIX_COLUMN_FUNCT(s_row);
                    a_rou := ADD_ROUNDKEY_FUNCT(m_col,clave(i));
                    state := a_rou;
                end loop;
                b_sub_f := BYTE_SUB_FUNCT(a_rou);
                s_row_f := SHIFT_ROW_FUNCT(b_sub_f);
                a_rou_f := ADD_ROUNDKEY_FUNCT(s_row_f,clave(Nr));
                data_out <= a_rou_f;
                ciph_temp := a_rou_f(127 downto 120) xor texto_cifrado ; --XOR de la salida de 8-bits con
texto_plano
                tex_plano <= ciph_temp;
                --Se obtiene el cifrado del primer bloque del texto plano
                for i in 119 downto 0 loop
                    itera_temp(i+8) := itera_temp(i);
                end loop;
                --itera_temp(7 downto 0) := ciph_temp; --cifrado
                    itera_temp(7 downto 0) := texto_cifrado; --descifrar
            end if;
        end process;
    end Behavioral;

```

AQUÍ TERMINA EL CODIGO DE LA IMPLEMENTACION

A continuación se presentan los resultados de la implementación en hardware del diseño CFB8_AES-128 y sus recursos consumidos:

cfb8_aes Project Status (08/22/2013 - 16:15:06)			
Project File:	CFB8_AES.xise	Parser Errors:	No Errors
Module Name:	cfb8_aes	Implementation State:	Programming File Generated
Target Device:	xc3s500e-4fg320	• Errors:	No Errors
Product Version:	ISE 13.3	• Warnings:	15 Warnings (7 new)
Design Goal:	Balanced	• Routing Results:	All Signals Completely Routed
Design Strategy:	Xilinx Default (unlocked)	• Timing Constraints:	All Constraints Met
Environment:	System Settings	• Final Timing Score:	0 (Timing Report)

Device Utilization Summary					[+]
Logic Utilization	Used	Available	Utilization	Note(s)	
Number of Slice Flip Flops	21	9,312	1%		
Number of 4 input LUTs	1	9,312	1%		
Number of occupied Slices	11	4,656	1%		
Number of Slices containing only related logic	11	11	100%		
Number of Slices containing unrelated logic	0	11	0%		
Total Number of 4 input LUTs	21	9,312	1%		
Number used as logic	1				
Number used as a route-thru	20				
Number of bonded IOBs	19	232	8%		
IOB Flip Flops	8				
Number of BUFGMUXs	1	24	4%		
Average Fanout of Non-Clock Nets	0.92				

ANEXO II

ANEXO II.-Reportes de Mapeo y Traza sobre el proyecto de la Plataforma

Antes de mostrar los reportes se presenta la Figura (AIV.1) que muestra las características del equipo donde fue desarrollado el proyecto.

Operating System Information				
Operating System Information	xst	ngdbuild	map	par
CPU Architecture/Speed	< data not available >	Intel(R) Core(TM)2 Duo CPU T6500 @ 2.10GHz/2094 MHz	Intel(R) Core(TM)2 Duo CPU T6500 @ 2.10GHz/2094 MHz	Intel(R) Core(TM)2 Duo CPU T6500 @ 2.10GHz/2094 MHz
Host	< data not available >	Pelafuztán	Pelafuztán	Pelafuztán
OS Name	< data not available >	Microsoft Windows 7 , 64-bit	Microsoft Windows 7 , 64-bit	Microsoft Windows 7 , 64-bit
OS Release	< data not available >	Service Pack 1 (build 7601)	Service Pack 1 (build 7601)	Service Pack 1 (build 7601)

Figura AIV.1 Características de computo para programar las tarjetas Spartan 3E.

REPORTE DE MAPEO

Release 13.3 Map O.76xd (nt64)
Xilinx Mapping Report File for Design 'Plataforma'

Design Information

Command Line : map -intstyle ise -p xc3s500e-fg320-5 -cm area -ir off -pr b -c 100 -o Plataforma_map.ncd Plataforma.ngd Plataforma.pcf
Target Device : xc3s500e
Target Package : fg320
Target Speed : -5
Mapper Version : spartan3e -- \$Revision: 1.55 \$
Mapped Date : Fri Jun 28 15:25:35 2013

Design Summary

Number of errors: 0
Number of warnings: 6

Logic Utilization:

Total Number Slice Registers: 189 out of 9,312 2%
Number used as Flip Flops: 132
Number used as Latches: 57
Number of 4 input LUTs: 216 out of 9,312 2%

Logic Distribution:

Number of occupied Slices: 157 out of 4,656 3%
Number of Slices containing only related logic: 157 out of 157 100%
Number of Slices containing unrelated logic: 0 out of 157 0%

*See NOTES below for an explanation of the effects of unrelated logic.

Total Number of 4 input LUTs: 216 out of 9,312 2%
Number used as logic: 144
Number used as Shift registers: 72

The Slice Logic Distribution report is not meaningful if the design is over-mapped for a non-slice resource or if Placement fails.

Number of bonded IOBs: 14 out of 232 6%

IOB Flip Flops: 4

Number of BUFGMUXs: 1 out of 24 4%

Average Fanout of Non-Clock Nets: 3.40

Peak Memory Usage: 247 MB

Total REAL time to MAP completion: 3 secs

Total CPU time to MAP completion: 2 secs

NOTES:

Related logic is defined as being logic that shares connectivity - e.g. two LUTs are "related" if they share common inputs. When assembling slices, Map gives priority to combine logic that is related. Doing so results in the best timing performance.

Unrelated logic shares no connectivity. Map will only begin packing unrelated logic into a slice once 99% of the slices are occupied through related logic packing.

Note that once logic distribution reaches the 99% level through related logic packing, this does not mean the device is completely utilized.

Unrelated logic packing will then begin, continuing until all usable LUTs and FFs are occupied. Depending on your timing budget, increased levels of unrelated logic packing may adversely affect the overall timing performance of your design.

Table of Contents

Section 1 - Errors

Section 2 - Warnings

Section 3 - Informational

Section 4 - Removed Logic Summary

Section 5 - Removed Logic

Section 6 - IOB Properties

Section 7 - RPMs

Section 8 - Guide Report

Section 9 - Area Group and Partition Summary

Section 10 - Timing Report

Section 11 - Configuration String Information

Section 12 - Control Set Information

Section 13 - Utilization by Hierarchy

Section 1 - Errors

Section 2 - Warnings

WARNING:Security:42 - Your software subscription period has lapsed. Your current version of Xilinx tools will continue to function, but you no longer qualify for Xilinx software updates or new releases.

WARNING:PhysDesignRules:372 - Gated clock. Clock net XLXI_8/DatoTi_or0000 is sourced by a combinatorial pin. This is not good design practice. Use the CE pin to control the loading of data into the flip-flop.

WARNING:PhysDesignRules:372 - Gated clock. Clock net XLXI_8/DatoTo_or0000 is sourced by a combinatorial pin. This is not good design practice. Use the CE pin to control the loading of data into the flip-flop.

WARNING:PhysDesignRules:372 - Gated clock. Clock net XLXI_19/DEC/S_dplano_or0000 is sourced by a combinatorial pin. This is not good design practice. Use the CE pin to control the loading of data into the flip-flop.

WARNING:PhysDesignRules:372 - Gated clock. Clock net XLXI_19/COD/S_dcifrado_or0000 is sourced by a combinatorial pin. This is not good design practice. Use the CE pin to control the loading of data into the flip-flop.

WARNING:PhysDesignRules:372 - Gated clock. Clock net XLXI_8/SaInt_or0000 is sourced by a combinatorial pin. This is not good design practice. Use the CE pin to control the loading of data into the flip-flop.

WARNING:PhysDesignRules:372 - Gated clock. Clock net XLXI_19/EnvioCifrado_mux0000 is sourced by a combinatorial pin. This is not good design practice. Use the CE pin to control the loading of data into the flip-flop.

Section 3 - Informational

INFO:Security:54 - 'xc3s500e' is a WebPack part.

INFO:MapLib:562 - No environment variables are currently set.

INFO:LIT:244 - All of the single ended outputs in this design are using slew rate limited output drivers. The delay on speed critical single ended outputs can be dramatically reduced by designating them as fast outputs.

Section 4 - Removed Logic Summary

2 block(s) optimized away

Section 5 - Removed Logic

Optimized Block(s):

TYPE	BLOCK
GND	XST_GND
VCC	XST_VCC

To enable printing of redundant blocks removed and signals merged, set the detailed map report option and rerun map.

Section 6 - IOB Properties

+-----+									
-----+									
Reg (s)	IOB Name	Resistor	IOB	Type	Direction	IO Standard	Diff	Drive	Slew
Delay						Term	Strength	Rate	
+-----+									
-----+									
0 / 0	CLK			IBUF	INPUT	LVCMOS25			
	EntExt<0>			IBUF	INPUT	LVCMOS25			
	0 / 0								
	EntExt<1>			IBUF	INPUT	LVCMOS25			
	0 / 0								
	EntExt<2>			IBUF	INPUT	LVCMOS25			
	0 / 0								
	EntExt<3>			IBUF	INPUT	LVCMOS25			
	0 / 0								
	RXDi			IBUF	INPUT	LVCMOS25			IFF1
	0 / 3								

	RXDo		IBUF	INPUT	LVCMOS25				IFF1
	0 / 3								
	Reset		IBUF	INPUT	LVCMOS25				
PULLDOWN	0 / 0								
	SalExt<0>		IOB	OUTPUT	LVCMOS25		12	SLOW	
	0 / 0								
	SalExt<1>		IOB	OUTPUT	LVCMOS25		12	SLOW	
	0 / 0								
	SalExt<2>		IOB	OUTPUT	LVCMOS25		12	SLOW	
	0 / 0								
	SalExt<3>		IOB	OUTPUT	LVCMOS25		12	SLOW	
	0 / 0								
	TXDi		IOB	OUTPUT	LVCMOS25		12	SLOW	
OFF1		0 / 0							
	TXDo		IOB	OUTPUT	LVCMOS25		12	SLOW	
OFF1		0 / 0							

-----+-----
-----+-----

Section 7 - RPMs

Section 8 - Guide Report

Guide not run on this design.

Section 9 - Area Group and Partition Summary

Partition Implementation Status

No Partitions were found in this design.

Area Group Information

No area groups were found in this design.

Section 10 - Timing Report

This design was not run using timing mode.

Section 11 - Configuration String Details

Use the "-detail" map option to print out Configuration Strings

Section 12 - Control Set Information

No control set information for this architecture.

Section 13 - Utilization by Hierarchy

Use the "-detail" map option to print out the Utilization by Hierarchy section.

REPORTE DE TRAZA

Release 13.3 Trace (nt64)

Copyright (c) 1995-2011 Xilinx, Inc. All rights reserved.

C:\Xilinx\13.3\ISE_DS\ISE\bin\nt64\unwrapped\trce.exe -intstyle ise -e 3 -s 5

-n 3 -xml Plataforma.twx Plataforma.ncd -o Plataforma.twr Plataforma.pcf -ucf Plataforma.ucf
 Design file: Plataforma.ncd
 Physical constraint file: Plataforma.pcf
 Device, package, speed: xc3s500e,fg320,-5 (PRODUCTION 1.27 2011-10-03)
 Report level: error report
 Environment Variable Effect

NONE No environment variables were set

INFO:Timing:2698 - No timing constraints found, doing default enumeration.
 INFO:Timing:2752 - To get complete path coverage, use the unconstrained paths option. All paths that are not constrained will be reported in the unconstrained paths section(s) of the report.
 INFO:Timing:3339 - The clock-to-out numbers in this timing report are based on a 50 Ohm transmission line loading model. For the details of this model, and for more information on accounting for different loading conditions, please see the device datasheet.
 INFO:Timing:3390 - This architecture does not support a default System Jitter value, please add SYSTEM_JITTER constraint to the UCF to modify the Clock Uncertainty calculation.
 INFO:Timing:3389 - This architecture does not support 'Discrete Jitter' and 'Phase Error' calculations, these terms will be zero in the Clock Uncertainty calculation. Please make appropriate modification to SYSTEM_JITTER to account for the unsupported Discrete Jitter and Phase Error.

Data Sheet report:

All values displayed in nanoseconds (ns)

Setup/Hold to clock CLK

	Max Setup to	Max Hold to	Clock	
Source	clk (edge)	clk (edge)	Internal Clock(s)	Phase
EntExt<0>	2.140(R)	0.050(R)	CLK_BUFGP	0.000
EntExt<1>	2.014(R)	0.052(R)	CLK_BUFGP	0.000
EntExt<2>	1.810(R)	0.104(R)	CLK_BUFGP	0.000
EntExt<3>	3.144(R)	-0.235(R)	CLK_BUFGP	0.000
RXDi	3.967(R)	-0.817(R)	CLK_BUFGP	0.000
RXDo	3.949(R)	-0.795(R)	CLK_BUFGP	0.000
Reset	3.602(R)	-0.107(R)	CLK_BUFGP	0.000

Clock CLK to Pad

	clk (edge)	Clock	
Destination	to PAD	Internal Clock(s)	Phase
SalExt<0>	10.226(R)	CLK_BUFGP	0.000
SalExt<1>	9.759(R)	CLK_BUFGP	0.000
TXDi	5.617(R)	CLK_BUFGP	0.000
TXDo	5.617(R)	CLK_BUFGP	0.000

Clock to Setup on destination clock CLK

| Src:Rise| Src:Fall| Src:Rise| Src:Fall|

Source Clock |Dest:Rise|Dest:Rise|Dest:Fall|Dest:Fall|

-----+-----+-----+-----+-----+

CLK | 5.621 | | | |

-----+-----+-----+-----+-----+

*******Tiempo usado para los cálculos THROUGHPUT**

Analysis completed Fri Jun 28 15:26:09 2013

Trace Settings:

Trace Settings

Peak Memory Usage: 169 MB

=====

Frecuencia:

UART_TX Di 5.617 [ns] --> 178.0 [MHz]

UART_TX Do 5.617 [ns] --> 178.0 [MHz]

UART_RX Di 3.967 [ns] --> 252.2 [MHZ]

UART_RX Do 3.949 [ns] --> 253.2 [MHZ]

=====

TIMING REPORT

NOTE: THESE TIMING NUMBERS ARE ONLY A SYNTHESIS ESTIMATE.

**FOR ACCURATE TIMING INFORMATION PLEASE REFER TO THE TRACE REPORT
GENERATED AFTER PLACE-and-ROUTE.**

Clock Information:

-----+-----+-----+-----+-----+

Clock Signal | Clock buffer(FF name) | Load |

-----+-----+-----+-----+-----+

CLK | BUFGP | 208 |

XLXI_19/EnvioCifrado_mux0000(XLXI_19/EnvioCifrado_mux00001:O) |

NONE(*) (XLXI_19/EnvioClaro) | 1 |

XLXI_19/epresente_FSM_FFd1 | NONE(XLXI_19/EnvioCifrado) | 1 |

XLXI_8/epresente_FSM_FFd12 | NONE(XLXI_8/ClaroOut_7) | 8 |

XLXI_8/epresente_FSM_FFd11 | NONE(XLXI_8/CifrOut_7) | 8 |

XLXI_8/DatoTo_or0000(XLXI_8/DatoTo_or00001:O) | NONE(*) (XLXI_8/DatoTo_7) |

8 | XLXI_8/DatoTi_or0000(XLXI_8/DatoTi_or00001_f5:O) | NONE(*) (XLXI_8/DatoTi_7) | 8

| XLXI_8/SalInt_or0000(XLXI_8/SalInt_or00001:O) | NONE(*) (XLXI_8/SalInt_5) | 5 |

XLXI_19/COD/edo_presente_FSM_FFd3 | NONE(XLXI_19/COD/cipher_7) | 8

| XLXI_19/COD/S_dcifrado_or0000(XLXI_19/COD/S_dcifrado_or00001:O) |

NONE(*) (XLXI_19/COD/S_dcifrado) | 1 |

XLXI_19/DEC/edo_presente_FSM_FFd3 | NONE(XLXI_19/DEC/tex_plano_7) |

8 | XLXI_19/DEC/S_dplano_or0000(XLXI_19/DEC/S_dplano_or00001:O) |

NONE(*) (XLXI_19/DEC/S_dplano) | 1 |

-----+-----+-----+-----+-----+

(*) These 6 clock signal(s) are generated by combinatorial logic,
and XST is not able to identify which are the primary clock signals.

Please use the CLOCK_SIGNAL constraint to specify the clock signal(s) generated by
combinatorial logic.

INFO:Xst:2169 - HDL ADVISOR - Some clock signals were not automatically buffered by XST with
BUFG/BUFR resources. Please use the buffer_type constraint in order to insert these buffers to the clock
signals to help prevent skew problems.

Asynchronous Control Signals Information:

Control Signal	Buffer(FF name)	Load
Reset	IBUF	24

Timing Summary:

Speed Grade: -5

Minimum period: 5.259ns (Maximum Frequency: 190.150MHz)

Minimum input arrival time before clock: 6.475ns

Maximum output required time after clock: 5.683ns

Maximum combinational path delay: No path found

Timing Detail:

All values displayed in nanoseconds (ns)

Timing constraint: Default period analysis for Clock 'CLK'

Clock period: 5.259ns (frequency: 190.150MHz)

Total number of paths / destination ports: 1223 / 465

Delay: 5.259ns (Levels of Logic = 3)
Source: XLXI_1/buf/data_width_loop[0].data_srl (FF)
Destination: XLXI_1/kcuart/pipeline_serial (FF)
Source Clock: CLK rising
Destination Clock: CLK rising

Data Path: XLXI_1/buf/data_width_loop[0].data_srl to XLXI_1/kcuart/pipeline_serial

Cell:in->out	Gate	Net	fanout	Delay	Delay	Logical Name (Net Name)
SRL16E:CLK->Q (XLXI_1/fifo_data_out<0>)			1	3.224	0.426	XLXI_1/buf/data_width_loop[0].data_srl
LUT4:I1->O	1	0.612	0.000	XLXI_1/kcuart/mux1_lut	(XLXI_1/kcuart/data_01)	
MUXF5:I0->O	1	0.278	0.000	XLXI_1/kcuart/mux5_muxf5	(XLXI_1/kcuart/data_0123)	
MUXF6:I0->O (XLXI_1/kcuart/data_01234567)			1	0.451	0.000	XLXI_1/kcuart/mux7_muxf6
FDRS:D		0.268		XLXI_1/kcuart/pipeline_serial		
Total		5.259ns (4.833ns logic, 0.426ns route)			(91.9% logic, 8.1% route)	

Timing constraint: Default OFFSET IN BEFORE for Clock 'CLK'

Total number of paths / destination ports: 36 / 29

Offset: 4.086ns (Levels of Logic = 3)
Source: EntExt<3> (PAD)
Destination: XLXI_8/epresente_FSM_FFd1 (FF)
Destination Clock: CLK rising
Data Path: EntExt<3> to XLXI_8/epresente_FSM_FFd1

Cell:in->out	Gate	Net	fanout	Delay	Delay	Logical Name (Net Name)
IBUF:I->O			26	1.106	1.101	EntExt_3_IBUF (EntExt_3_IBUF)

```

LUT3:I2->O      1  0.612  0.387  XLXI_8/epresente_FSM_FFd1-In_SW1 (N33)
LUT4:I2->O      1      0.612    0.000  XLXI_8/epresente_FSM_FFd1-In
(XLXI_8/epresente_FSM_FFd1-In)
FDC:D           0.268    XLXI_8/epresente_FSM_FFd1
-----

```

```

Total           4.086ns (2.598ns logic, 1.488ns route)
                (63.6% logic, 36.4% route)
=====

```

Timing constraint: Default OFFSET IN BEFORE for Clock 'XLXI_8/epresente_FSM_FFd12'

Total number of paths / destination ports: 8 / 8

```

Offset:         2.660ns (Levels of Logic = 1)
Source:         EntExt<3> (PAD)
Destination:    XLXI_8/ClaroOut_7 (LATCH)
Destination Clock: XLXI_8/epresente_FSM_FFd12 falling
Data Path: EntExt<3> to XLXI_8/ClaroOut_7

```

```

      Gate  Net
Cell:in->out  fanout Delay Delay Logical Name (Net Name)
-----
IBUF:I->O     26  1.106  1.071 EntExt_3_IBUF (EntExt_3_IBUF)
LDE:GE        0.483    XLXI_8/ClaroOut_7
-----
Total         2.660ns (1.589ns logic, 1.071ns route)
                (59.7% logic, 40.3% route)
=====

```

Timing constraint: Default OFFSET IN BEFORE for Clock 'XLXI_8/epresente_FSM_FFd11'

Total number of paths / destination ports: 8 / 8

```

Offset:         2.660ns (Levels of Logic = 1)
Source:         EntExt<3> (PAD)
Destination:    XLXI_8/CifrOut_7 (LATCH)
Destination Clock: XLXI_8/epresente_FSM_FFd11 falling

```

Data Path: EntExt<3> to XLXI_8/CifrOut_7

```

      Gate  Net
Cell:in->out  fanout Delay Delay Logical Name (Net Name)
-----
IBUF:I->O     26  1.106  1.071 EntExt_3_IBUF (EntExt_3_IBUF)
LDE:GE        0.483    XLXI_8/CifrOut_7
-----
Total         2.660ns (1.589ns logic, 1.071ns route)
                (59.7% logic, 40.3% route)
=====

```

Timing constraint: Default OFFSET IN BEFORE for Clock 'XLXI_8/DatoTi_or0000'

Total number of paths / destination ports: 28 / 6

```

Offset:         6.475ns (Levels of Logic = 5)
Source:         EntExt<3> (PAD)
Destination:    XLXI_8/DatoTi_2 (LATCH)
Destination Clock: XLXI_8/DatoTi_or0000 falling

```

Data Path: EntExt<3> to XLXI_8/DatoTi_2

```

      Gate  Net
Cell:in->out  fanout Delay Delay Logical Name (Net Name)
-----
IBUF:I->O     26  1.106  1.140 EntExt_3_IBUF (EntExt_3_IBUF)

```

```

LUT2:I1->O      4  0.612  0.502  XLXI_8/DatoTi_mux0001<6>111 (XLXI_8/N10)
LUT4:I3->O      4  0.612  0.651  XLXI_8/DatoTi_mux0001<2>11 (XLXI_8/N111)
LUT3:I0->O      1  0.612  0.360  XLXI_8/DatoTi_mux0001<2>_SW0 (N25)
LUT4:I3->O      1  0.612  0.000  XLXI_8/DatoTi_mux0001<2>
(XLXI_8/DatoTi_mux0001<2>)
LD:D            0.268      XLXI_8/DatoTi_2
-----

```

```

Total          6.475ns (3.822ns logic, 2.653ns route)
              (59.0% logic, 41.0% route)
=====

```

Timing constraint: Default OFFSET IN BEFORE for Clock 'XLXI_8/SalInt_or0000'

Total number of paths / destination ports: 3 / 3

```

Offset:        3.126ns (Levels of Logic = 2)
Source:        EntExt<3> (PAD)
Destination:   XLXI_8/SalInt_5 (LATCH)
Destination Clock: XLXI_8/SalInt_or0000 rising

```

Data Path: EntExt<3> to XLXI_8/SalInt_5

```

      Gate Net
Cell:in->out  fanout Delay Delay Logical Name (Net Name)
-----
IBUF:I->O     26  1.106  1.140 EntExt_3_IBUF (EntExt_3_IBUF)
LUT2:I1->O      1  0.612  0.000  XLXI_8/SalInt_mux0000<5>1
(XLXI_8/SalInt_mux0000<5>)
LD_1:D        0.268      XLXI_8/SalInt_5
-----

```

```

Total          3.126ns (1.986ns logic, 1.140ns route)
              (63.5% logic, 36.5% route)
=====

```

Timing constraint: Default OFFSET OUT AFTER for Clock 'CLK'

Total number of paths / destination ports: 11 / 4

```

Offset:        5.683ns (Levels of Logic = 3)
Source:        XLXI_8/epresente_FSM_FFd2 (FF)
Destination:   SalExt<0> (PAD)
Source Clock:   CLK rising

```

Data Path: XLXI_8/epresente_FSM_FFd2 to SalExt<0>

```

      Gate Net
Cell:in->out  fanout Delay Delay Logical Name (Net Name)
-----
FDC:C->Q      10  0.514  0.753  XLXI_8/epresente_FSM_FFd2
(XLXI_8/epresente_FSM_FFd2)
LUT4:I3->O      1  0.612  0.000  XLXI_8/epresente_FSM_Out141
(XLXI_8/epresente_FSM_Out14)
MUXF5:I0->O    1  0.278  0.357  XLXI_8/epresente_FSM_Out14_f5 (SalExt_0_OBUF)
OBUF:I->O      3.169      SalExt_0_OBUF (SalExt<0>)
-----

```

```

Total          5.683ns (4.573ns logic, 1.110ns route)
              (80.5% logic, 19.5% route)
=====

```

Total REAL time to Xst completion: 2071.00 secs

Total CPU time to Xst completion: 2070.88 secs

-->

Total memory usage is 636472 kilobytes

Number of errors : 0 (0 filtered)
Number of warnings : 31 (0 filtered)
Number of infos : 29 (0 filtered)

Referencias Bibliográficas

- Baena A., (2010): "*Diseño sobre FPGA de una Unidad Aritmética Decimal*". Tesis de Titulación en Ingeniería Técnica Industrial en Electrónica Industrial. Universitat Rovira i Virgili, Cataluña España.
- Buchholz J., (2001): "Matlab Implementation of the Advanced Encryption Standard".
- Chapman K. (2003): "UART Transmitter and Receiver Macros". Xilinx Ltd.
- Daltabuit E., Mallén G. Hernández L. Vázquez J., (2007): La seguridad de la información; Limusa.
- Elbirt A. J., (2009): Understanding and applying cryptography and data security; CRC Press.
- FIPS-197, (2001): Specification for the Advanced Encryption Standard (AES).
- Menezes A. J., et. al. (1997): Handbook of applied cryptography; CRC Press.
- NIST Special Publication 800-38A, (2001): Recommendation for Block 2001 Edition_Cipher Modes of Operation.
- Padrón A., Velázquez J., Prieto R., Herrera A., Calva G., (2011): "*FPGA implementation of cipher block AES with operation mode CFB*", Second International Congress Instrumentation and Applied Sciences, Puebla-México.
- Prieto R., Padrón A., Herrera A., Calva G., (2011a): Desarrollo de Funciones para el Manejo de Periféricos por Medio de un FPGA, Clave de control: II-INME-2013-336.
- Prieto R., Padrón A., Herrera A., Calva G., (2011b): "*Generic Platform for implementation of stream cipher in FPGA'S*", Second International Congress Instrumentation and Applied Sciences, Puebla-México.
- Ramió J., (2006). "Libro Electrónico de Seguridad Informática y Criptografía", Universidad Politécnica de Madrid, Sexta Edición, Versión v 4.1.
- Spartan-3E FPGA Starter Kit Board User Guide, (2008): UG230 (v1.1) June 20.
- Terés L., Torroja Y., Olcoz S., Villar E. (1998): VHDL Lenguaje estándar de diseño electrónico, Ed. McGrawHill, España.
- Trejo E., (2005): "Implementación Eficiente en FPGA del modo CCM usando AES", Tesis de Maestría en Ciencias, CINVESTAV, Unidad Zacatenco del IPN.