Diseño e implementación de algoritmos criptográficos sobre FPGA

Autores:
Palomino Guzmán, Abelardo
Romero Zamora, Ángel Manuel
Solbes Bosch, Alfonso

Directores de proyecto:
Sánchez-Élez Martín, Marcos
Resano Ezcaray, Javier

Proyecto de Sistemas Informáticos, Facultad de Informática, Universidad
Complutense de Madrid.
Curso: 2005/2006

Resumen.

En este proyecto, se ha diseñado e implementado un sistema para gestionar un grupo de usuarios, capaz de permitir acceso a información privada mediante claves encriptadas. La agregación o desagregación al grupo de uno de sus usuarios provoca la redistribución de claves para alguno o todos ellos. La encriptación de estas claves se lleva a cabo mediante el algoritmo Advanced Encryption Standard (AES) y, para incrementar la eficiencia, éstas se organizan jerárquicamente. Además, para conseguir un mejor rendimiento, el sistema se implementó en FPGA usando el lenguaje VHDL.
Durante el desarrollo de este proyecto se han evaluado distintas alternativas de diseño tanto para el AES como para el sistema completo realizándose implementaciones optimizadas tanto para área como para rendimiento. La implementación final de nuestro sistema es capaz de gestionar la baja o el alta de usuarios en tan sólo 2microsegundos.

Abstract.

In this project we designed and implemented a system to manage a group of users, which allows the access to private information by means of encrypted keys. Adding or removing one of the users to or from the group redistributes the keys for some or all of them. The keys were encrypted with the Advanced Encryption Standard (AES) algorithm and were organised hierarchically to increase efficiency. Furthermore, the system was implemented in FPGA using VHDL language in order to achieve a better performance. During the project development we have evaluated different design alternatives for the AES and the overall system. We have implemented different version of the system optimized for area and performance. The final implementation manages users join and disjoin operations in just 2 microseconds.

Palabras Clave.

Rekeying, Encriptación, Gestión de Claves, Gestión de Usuarios, Seguridad, Arquitecturas Reconfigurables, Lenguajes de descripción hardware, Multicast

Índice

1. Introducción 7
1.1. La plataforma FPGA 7
1.2. Algoritmos Criptográficos 8
1.3. AES 9
1.4. El generador ANSI X9.17 12
1.5. Sistema de gestión de claves de usuarios. 12
2. Descripción del problema 13
3.1. Funcionamiento 16
3.2. Estructura 16
3.3. Decisiones de diseño 16
3.3.1. Organización de direcciones: Árbol de direcciones 16
3.3.2. Procesamiento de la dirección 17
3.4. Iteración de los elementos 18
3.5. Especificación generador y encriptador de claves (E/G) 18
3.6. Funcionalidades 19
3.7. Ejemplo de funcionamiento del sistema 23
4. Implementación 27
4.1. Ruta de datos 27
4.2. El Generador/Encriptador 27
4.2.1. El AES 27
4.2.2. El AES en VHDL. 36
4.2.3. Resultados y Conclusiones del Módulo AES. 38
4.2.4. Módulo Generador/Encriptador 48
4.2.5. Resultados del módulo Generador/Encriptador 56
4.2.6. Mejoras del Módulo Generador/Encriptador 57
4.3. Procesador de direcciones 63
4.4. El procesador de LR 63
4.5. Controlador 64
4.5.1. Funcionamiento 64
4.5.2. Señales de control 64
4.6. Resultados para el Sistema Completo 68
4.7. Mejoras respecto a versiones anteriores 78
Incluir dos AES 78
4.8. Resultados de la versión final 80
4.9. Mejoras para versiones futuras 89
4.10. Conclusiones 89
5.Bibliografía 91

1. Introducción

Se quiere realizar un sistema en el que haya varios usuarios registrados.
Cada usuario tendrá un grupo de claves asignadas para su acceso a dicho sistema.
Los usuarios pueden ir cambiando a lo largo del tiempo uniéndose o marchándose.
Como queremos que solamente los usuarios registrados en cada momento puedan acceder al sistema, necesitamos poder controlar estos accesos, lo que se conseguirá con la encriptación claves.
El proyecto que se ha realizado consiste en la implementación sobre FPGA del sistema gestor claves. El sistema puede dar de alta o baja a un usuario reasignado el grupo de claves correspondientes a cada usuario y debe garantizar que los nuevos usuarios no tengan acceso a contenidos antiguos, ni los antiguos a contenidos nuevos. Las claves se han encriptado con el algoritmo criptográfico AES y para la generación de nuevas claves se ha utilizado el algoritmo ANSI X9.17.

1.1. La plataforma FPGA

FPGA es el acrónimo de *Field-programmable gate array* (Matriz de puertas programable por un usuario en el 'campo' de una aplicación). Se trata de dispositivos electrónicos digitales programables de muy alta densidad.

Estructura

Internamente una FPGA es una serie de pequeños dispositivos lógicos, que algunos fabricantes llaman CLB, organizados por filas y columnas.

Entre los CLB hay un gran número de elementos de interconexión, líneas que pueden unir unos CLB con otros y con otras partes de la FPGA. Puede haber líneas de distintas velocidades.

También hay pequeños elementos en cada una de las patillas del chip para definir la forma en que ésta trabajará (entrada, salida, entrada-salida...). Se suelen llamar IOB.

Aparte de esta estructura, que es la básica, cada fabricante añade sus propias ideas, por ejemplo hay algunos que tienen varios planos con filas y columnas de CLB.

Los CLB contienen en su interior elementos hardware programable que permiten que su funcionalidad sea elevada. También es habitual que contengan dispositivos de memoria.

Programación

La tarea del programador es definir la función lógica que realizará cada uno de los CLB, seleccionar el modo de trabajo de cada IOB e interconectarlos todos.

El diseñador cuenta con la ayuda de herramientas de programación. Cada fabricante suele tener las suyas, aunque usan unos lenguajes de programación comunes. Estos lenguajes son los HDL o *Hardware Description Language* (lenguajes de descripción de hardware):

VHDL, Verilog o ABEL

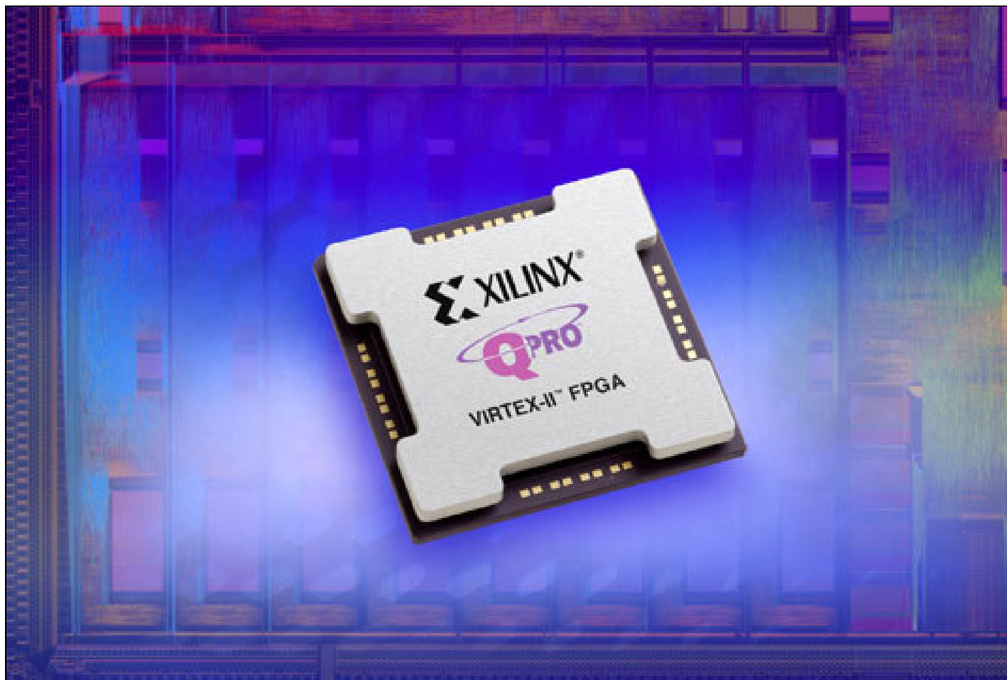


Figura 1.1 FPGA Virtex II PRO.

Aplicaciones típicas

Las características de las FPGA son su flexibilidad, capacidad de procesado en paralelo y velocidad. Esto les convierte en dispositivos idóneos para:

- Simulación y depuración en el diseño de microprocesadores.
- Simulación y depuración en el diseño de ASICs.
- Procesamiento de señal digital, por ejemplo vídeo.
- Sistemas aeronáuticos y militares.

1.2. Algoritmos Criptográficos

La **criptografía** (del griego *kryptos*, "ocultar", y *grafos*, "escribir", literalmente "escritura oculta") es el arte o ciencia de cifrar y descifrar información utilizando técnicas matemáticas que hagan posible el intercambio de mensajes de manera que sólo puedan ser leídos por las personas a quienes van dirigidos.

La finalidad de la criptografía es, en primer lugar, garantizar el secreto en la comunicación entre dos entidades (personas, organizaciones, etc.) y, en segundo lugar, asegurar que la información que se envía es auténtica en un doble sentido: que el remitente sea realmente quien dice ser y que el contenido del mensaje enviado, habitualmente denominado criptograma, no haya sido modificado en su tránsito.

En la actualidad, la criptografía no sólo se utiliza para comunicar información de forma segura ocultando su contenido a posibles fisgones. Una de las ramas de la criptografía que más ha revolucionado el panorama actual de las tecnologías informáticas es el de la

firma digital: tecnología que busca asociar al emisor de un mensaje con su contenido de forma que aquel no pueda posteriormente repudiarlo.

1.3. AES

En criptografía, Advanced Encryption Standard (AES), también conocido como Rijndael, es un esquema de cifrado por bloque adoptado como un estándar de encriptación por el gobierno de los Estados Unidos, y se espera que sea usado en el mundo entero, como también analizado exhaustivamente, como fue el caso de su predecesor, el Estándar de Encriptación de Datos (DES). Fue adoptado por el Instituto Nacional de Estándares y Tecnología (NIST) como un FIPS (PUB 197) en noviembre del 2001 después de 5 años del proceso de estandarización.

El cifrador fue desarrollado por dos criptólogos belgas, Joan Daemen y Vincent Rijmen, y enviado al proceso de selección AES bajo el nombre "Rijndael", un portmanteau empaquetado de los nombres de los inventores.

Desarrollo

Rijndael fue un refinamiento de un diseño anterior de Daemen y Rijmen, Square; Square fue a su vez un desarrollo de Shark.

AES es rápido tanto en software como en hardware, es relativamente fácil de implementar, y requiere poca memoria. Como nuevo estándar de cifrado, esta siendo desplegado actualmente a gran escala.

Descripción del cifrado

Estrictamente hablando, AES no es precisamente Rijndael ya que Rijndael permite un mayor rango de tamaño de bloque y clave; 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.

Para el cifrado, cada ronda de la aplicación del algoritmo AES (excepto la última) consiste en cuatro pasos:

`SubBytes`: En este paso se realiza una sustitución no lineal donde cada byte es reemplazado con otro de acuerdo a una tabla lookup table.

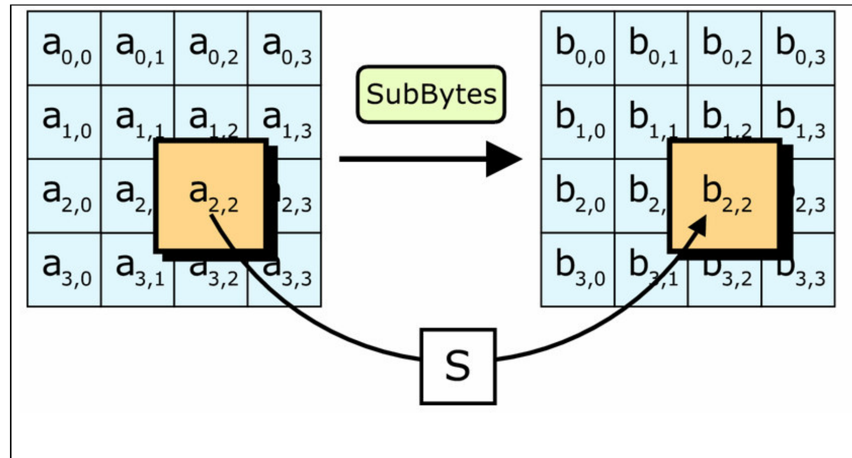


Figura 1.3.1. SubBytes.

`ShiftRows`: En este paso se realiza un transposición donde cada fila del state es rotado de manera cíclica un número determinado de veces.

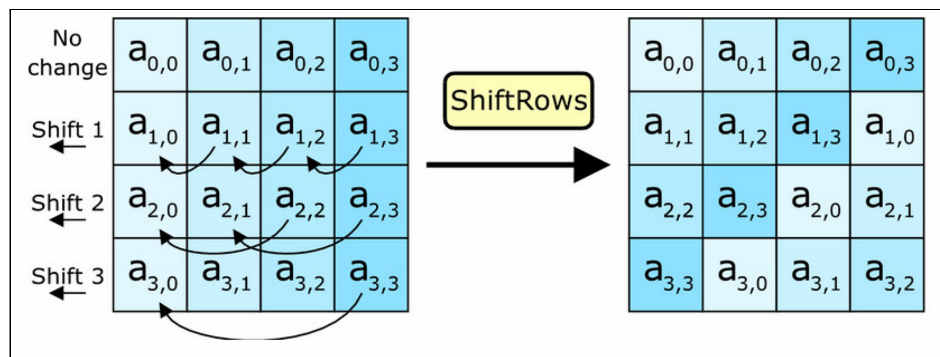


Figura 1.3.2 ShiftRows.

`MixColumns:` Operación de mezclado que opera en las columnas del «state», combinando los cuatro bytes en cada columna usando una transformación lineal.

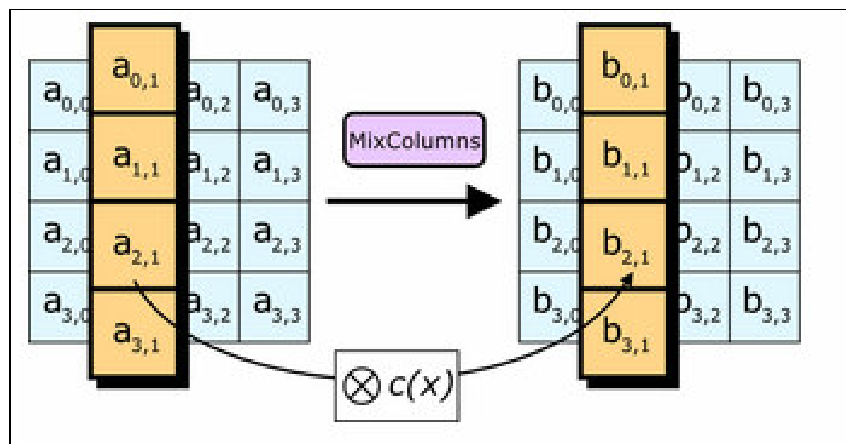


Figura 1.3.3 MixColumns.

`AddRoundKey`: Cada byte del «state» es combinado con la clave «round»; cada clave «round» se deriva de la clave de cifrado usando una key schedule.

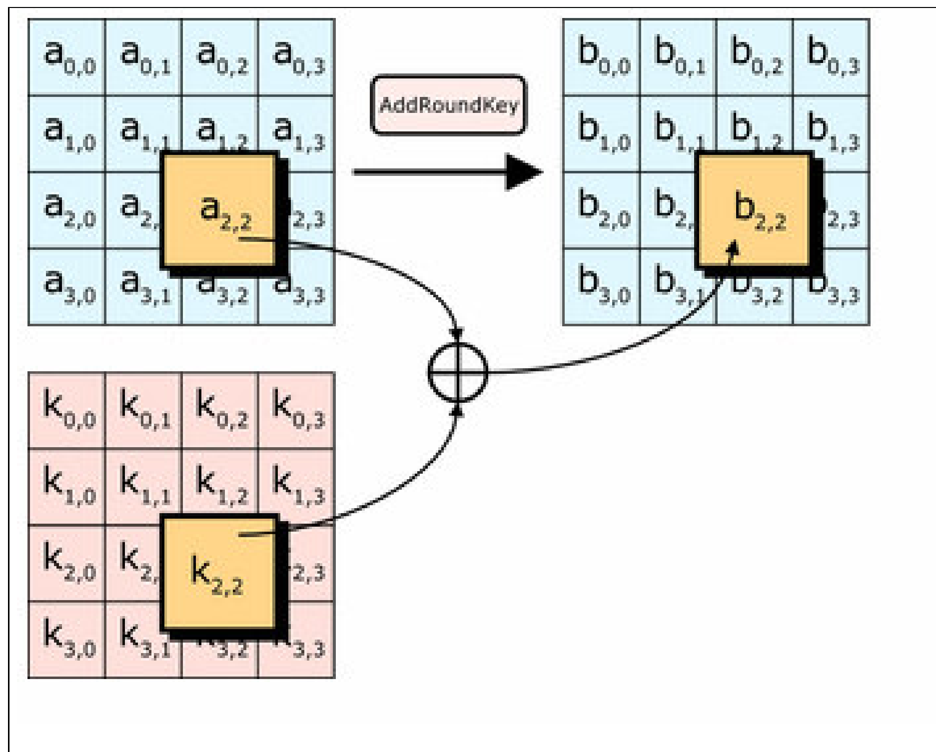


Figura 1.3.4 AddRoundKey.

La ronda final omite la fase `MixColumns`.

Seguridad

Hasta 2005, no se ha encontrado ningún ataque exitoso contra el AES. La Agencia de Seguridad Nacional de los Estados Unidos (NSA) revisó todos los finalistas candidatos al AES, incluyendo el Rijndael, y declaró que todos ellos eran suficientemente seguros para su empleo en información no clasificada del gobierno de los Estados Unidos.

Este hecho marca la primera vez que el público ha tenido acceso a un cifrador aprobado por la NSA para información super secreta (TOP SECRET). Es interesante notar que muchos productos públicos usan llaves de 128 bits por defecto; es posible que la NSA sospeche de una debilidad fundamental en llaves de este tamaño, o simplemente prefieren tener un margen de seguridad para documentos super secretos.

El método más común de ataque hacia un cifrador por bloques consiste en intentar varios ataques sobre versiones del cifrador con un número menor de rondas. El AES tiene 10 rondas para llaves de 128 bits, 12 rondas para llaves de 192 bits, y 14 rondas para llaves de 256 bits. Hasta 2005, los mejores ataques conocidos son sobre versiones reducidas a 7 rondas para llaves de 128 bits, 8 rondas para llaves de 192 bits, y 9 rondas para llaves de 256 bits.

Algunos criptógrafos muestran preocupación sobre la seguridad del AES. Ellos sienten que el margen entre el número de rondas especificado en el cifrador y los mejores ataques conocidos es muy pequeño. El riesgo es que se puede encontrar alguna manera de mejorar los ataques y de ser así, el cifrador podría ser roto. En el contexto criptográfico se considera "roto" un algoritmo si existe algún ataque más rápido que una búsqueda exhaustiva - ataque por fuerza bruta. De modo que un ataque contra el AES de llave de 128 bits que requiera 'sólo' 2^{120} operaciones sería considerado como un ataque que "rompe" el AES aún tomando en cuenta que por ahora sería un ataque irrealizable. Hasta el momento, tales preocupaciones pueden ser ignoradas.

Otra preocupación es la estructura matemática de AES. A diferencia de la mayoría de cifradores de bloques, AES tiene una descripción matemática muy ordenada. Esto no ha llevado todavía a ningún ataque, pero algunos investigadores están preocupados que futuros ataques quizá encuentren una manera de explotar esta estructura.

1.4. El generador ANSI X9.17

Este generador está pensado como mecanismo para generar claves DES y vectores de inicialización, usando triple- DES como primitiva (podría usarse otro algoritmo de cifrado de bloques). También es ampliamente usado en banca y otras aplicaciones.
K es una clave secreta para triple DES generada de algún modo en el momento de la inicialización. Debe ser aleatoria y usada sólo para este generador. Nunca varía con las entradas.
Cada vez que se desee una salida se hace lo siguiente
Ti = EK(timestamp)
salida(i)= EK(Ti xor semilla(i))
semilla(i+1)= EK(Ti xor salida(i))

El 'timestamp' se basa en el uso de la CPU del programa y su resolución depende de la plataforma utilizada, por ejemplo en Linux tiene una resolución de 0.01 segundos.

1.5. Sistema de gestión de claves de usuarios.

Un sistema de gestión de claves puede ser utilizado en un sistema de multidifusión segura, en el que sólo se permite el acceso al material difundido a los usuarios registrados y cuyas dos tareas principales son asegurar el control de acceso hacia adelante y hacia atrás, un ejemplo de un sistema de este tipo es la televisión de pago.

El control de acceso hacia adelante consiste en que un usuario dado de baja ya no pueda acceder a lo transmitido posteriormente, de forma análoga el control de acceso hacia atrás impide que un usuario acceda a material emitida antes del momento de su alta.

Para conseguir estos objetivos, se encripta el material con cierta clave, denominada clave de grupo, que poseen todos los usuarios dados de alta en el sistema, esto limita el acceso a dichos usuarios. Al producirse un alta se debe producir una nueva clave y suministrarla a todos los usuarios existentes y al que se da de alta, esto es necesario para asegurar el control de acceso hacia atrás. Al producirse una baja, también es necesario actualizar dicha clave para garantizar el control de acceso hacia adelante.

2. Descripción del problema

Nuestro objetivo es realizar un sistema gestor de claves de usuarios, con varias características: deseamos utilizar el mismo medio para transmitir las claves y el contenido en sí, también deseamos un sistema eficiente, escalable y con respuesta rápida

En primer lugar deseamos utilizar el mismo medio para transmitir las claves y el contenido y éste es un medio de multidifusión, por tanto necesitamos transmitir las claves encriptadas para que sólo las reciban los usuarios adecuados en cada momento.

Una primera aproximación para poder transmitir las claves a los usuarios es que éstos posean una clave única con la que se encriptarán las nuevas claves que se les necesite enviar. De este modo implementar una unión es simple y eficiente; se distribuye la nueva clave de grupo a los antiguos usuarios encriptándola con la clave de grupo antigua y al nuevo usuario con su clave personal. Sin embargo implementar la desunión es terriblemente ineficiente ya que no es posible distribuir la clave nueva a todos los usuarios a la vez porque el usuario a desunir también posee la clave de grupo antigua, por lo tanto no queda más remedio que distribuir la clave a los usuarios que permanecen en el sistema uno a uno, encriptando la clave nueva con cada una de sus claves personales. Este inconveniente genera un problema importante de escalabilidad, al aumentar el coste de realizar una desunión al aumentar el número de usuarios.

Para solucionar esto utilizamos una jerarquía de claves que explicaremos con detalle más adelante, que nos permitirá crear un sistema eficiente y escalable.

Por último, para conseguir que el sistema sea rápido lo implementaremos utilizando FPGA.

La jerarquía de claves

Para afrontar el problema de la escalabilidad dividimos el grupo de usuarios en subgrupos jerárquicos y proporcionamos a los miembros de cada subgrupo una clave compartida, llamada clave auxiliar. Consideremos por ejemplo un grupo de ocho usuarios:

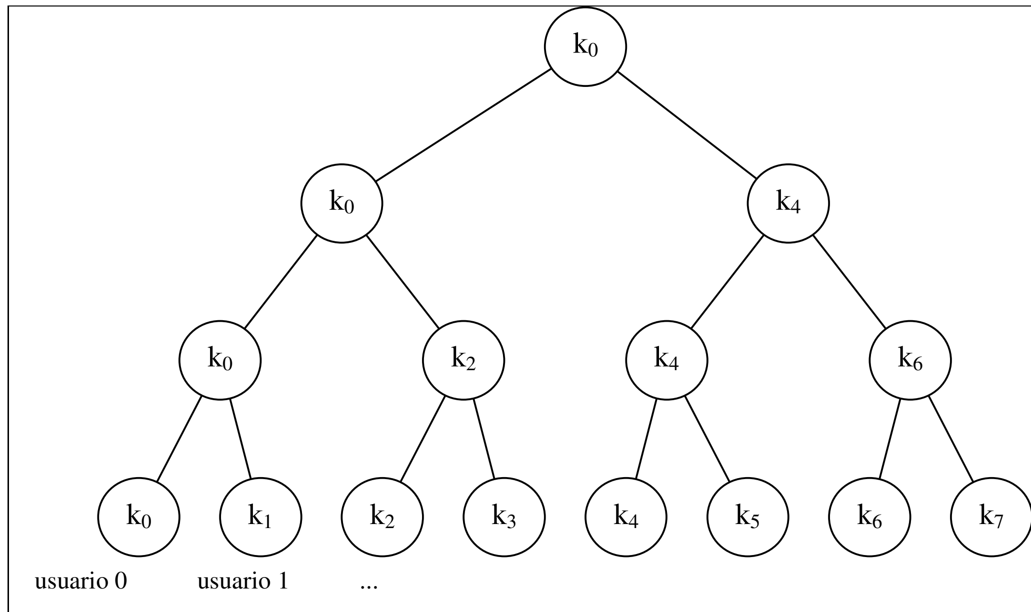


Figura 2.1 Árbol de Claves.

En la Figura 2.1 los usuarios 0 y 1 forman un subgrupo con la clave auxiliar k_{0-1}, los usuarios 0, 1,2 y3 forman otro cuya clave auxiliar es k_{0-3}. Todos los usuarios forman parte del subgrupo más grande con clave k_{0-7} que se corresponde con la clave de grupo del esquema simple, visto anteriormente, y es la utilizada para encriptar el material a distribuir.

De este modo cada usuario posee ahora varias claves en lugar de dos como antes. Estas son la clave personal k_d, conocida sólo por este usuario y por el servidor, la de grupo k_g, conocida por todos los usuarios, y varias claves auxiliares k_{x-y} correspondientes a los subgrupos a los que pertenece el usuario.

Supongamos que el usuario 2 quiere darse de **baja**. Excepto su clave personal, todas las claves que tenía deben cambiarse. Entonces tres claves nuevas, k_{2-3}' ,k_{0-3}' y k_{0-7}' se generan, encriptan y envían al los demás usuarios que las necesiten.

Usando la notación $E_{ka}(k_b)$ que simboliza encriptar k_b con k_a, entonces se deben hacer las siguientes encriptaciones:

$E_{k3}(k_{2-3}')$, $E_{k2-3'}(k_{0-3}')$, $E_{k0-1}(k_{0-3}')$, $E_{k0-3'}(k_{0-7}')$, $E_{k4-7}(k_{0-7}')$.

Esto son dos encriptaciones por cada nivel del árbol, excepto para el último nivel que sólo se hace una, entonces son $2*\log_2 n-1$.

Supongamos ahora que se da de **alta** el usuario 2.

También deben actualizarse k_{2-3} ,k_{0-3} y k_{0-7} y entonces se realizan las siguientes encriptaciones:

$E_{k2-3}(k_{2-3}')$, $E_{k0-3}(k_{0-3}')$, $E_{k0-7}(k_{0-7}')$, $E_{k2}(k_{2-3}')$, $E_{k2}(k_{0-3}')$, $E_{k2}(k_{0-7}')$.

Que se corresponden a encriptar dichas claves con las antiguas y con la del nuevo usuario. Obsérvese que encriptar una clave con la anterior no viola el control de acceso hacia atrás ya que el nuevo usuario no posee las antiguas.

Esto son dos encriptaciones por cada nivel del árbol, entonces son $2*\log_2 n$.

Comparación al esquema simple:

Número de encriptaciones	Esquema simple	Árbol de claves
Unión	2	$2*\log_2 n$
Desunión	n-1	$2*\log_2 n - 1$
Media	*O(n)*	$O(\log_2 n)$

Tabla 2.2 Costes de Operaciones

Podemos ver que el sistema de jerarquía de claves es claramente superior en eficiencia y escalabilidad.

3. Especificación

3.1. Funcionamiento

Existe un sistema externo que hace peticiones al sistema generador de claves.

El sistema que gestiona la producción de nuevas claves es el encargado de procesar las peticiones que recibe y, según estas, realizar distintas operaciones con el fin de lograr los resultados adecuados.

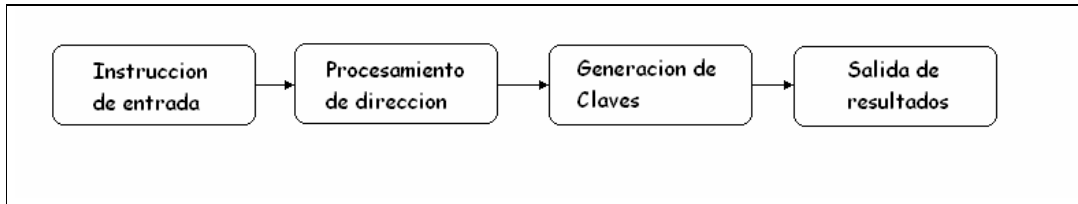


Figura 3.1.1 Flujo de Operaciones

Al sistema le llegan distintas peticiones del sistema externo y debe procesarlas para identificarlas y actuar en consecuencia, para ello debe obtener los datos implicados en dicha operación, procesar la direcciones de las claves implicadas, generar nuevas claves a partir de estas, modificar el sistema y entregar los resultados.

Este comportamiento es posible gracias a las distintas operaciones que puede realizar el sistema y que veremos más adelante.

3.2. Estructura

El sistema consta de distintos módulos para cada una de las partes de la ejecución de una instrucción.

- Tenemos una memoria FIFO que almacena las instrucciones de entrada para procesarlas según nos van llegando.
- Para el procesamiento de la dirección tenemos un módulo gestor de direcciones llamado DirProcess.
- Las Claves son generadas gracias al módulo Generador-Encriptador (EG)
- Los resultados se almacenan en una memoria FIFO para que sean tratados por el sistema externo.

3.3. Decisiones de diseño

3.3.1. Organización de direcciones: Árbol de direcciones

Cada usuario tiene un grupo de claves asignado. El grupo esta formado por la clave de usuario, y un grupo de claves compartidas con otros usuarios llamadas claves auxiliares.

La clave de usuario es la clave con la que se identifica un usuario y viene dada por le sistema externo.

Las claves auxiliares son claves compartidas elegidas por el sistema gestor de claves para garantizar una mayor seguridad.

Se ha elegido una representación en forma de árbol binario para el almacenamiento de claves de modo que las direcciones correspondientes a las posiciones menos significativas se dedican para las claves de usuario y las más significativas para las claves auxiliares. De este modo con la codificación de una dirección podemos determinar el nivel en el árbol y si es parte de un subárbol derecho o izquierdo. La representación es la siguiente:

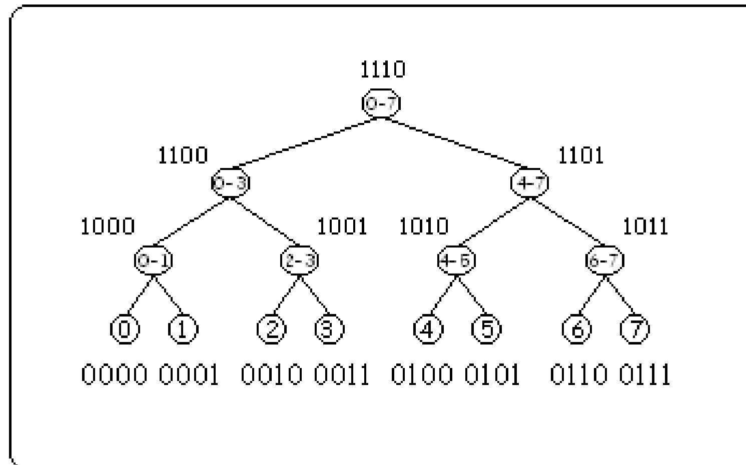


Figura 3.3.1 Ejemplo de un sistema de 8 usuarios completo.

3.3.2. Procesamiento de la dirección

Procesamiento de LR:

Por motivos de optimización asociamos cada clave auxiliar con una palabra de 2 bits llamada LR para indicar que clave esta en uso. En la tabla 3.3.2 se muestra la codificación de LR.

LR	Significado
00	Clave no usada
01	Clave usada de la derecha
10	Clave usada de la izquierda
11	Clave usada de derecha e izquierda

Tabla 3.3.2 Codificación LR.

Debido a esta modificación el sistema consta ahora de un módulo más correspondiente al tratamiento de este añadido.

Más adelante se verá un ejemplo del uso de estos valores LR.

3.4. Iteración de los elementos

Todos los elementos están conectados entre si, formando una ruta de datos, y controlados por un sistema controlador.

El controlador se encarga de decidir que sector esta operativo y de que forma se debe comportar en caso de estarlo mediante señales de control.

3.5. Especificación generador y encriptador de claves (E/G)

Este módulo se encarga de dos funciones esenciales en el sistema: encriptación y generación de nuevas claves. Para la primera de las funciones utiliza el algoritmo de encriptación AES y para la segunda utiliza una combinación del estándar ANSI X9.17 y del propio AES.

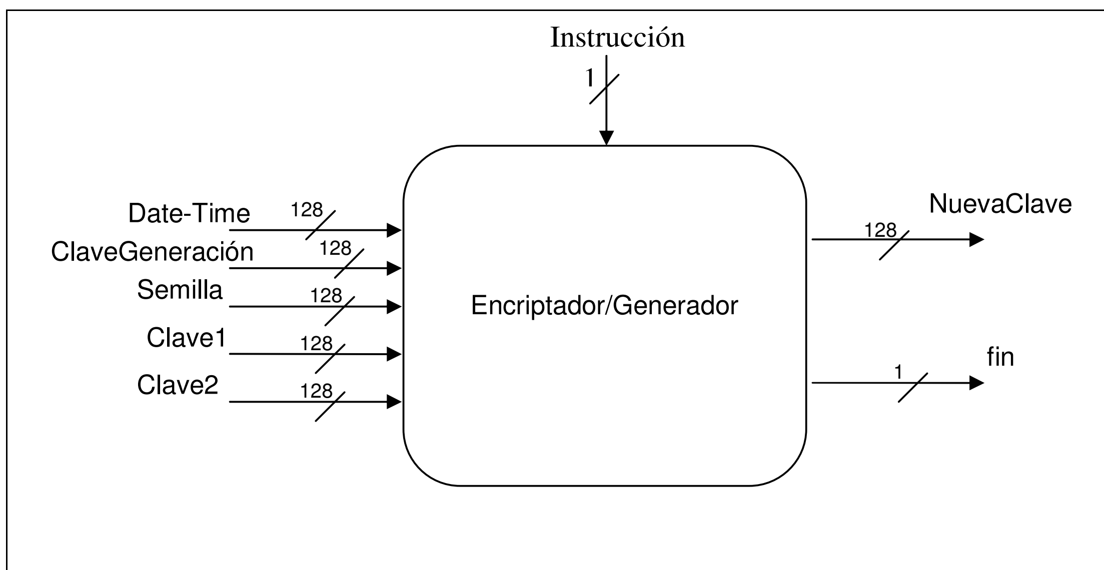


Figura 3.5.1 Módulo Generador/Encriptador.

Inicialización
La función de generación precisa una fase inicial de inicialización, para ello requiere tres datos de 128 bits:
-. Información de la fecha: consiste en la fecha que se inicia el sistema.
-. Clave de generación.
-. Semilla.

Generación
Una vez realizada la inicialización, este módulo no necesita la ayuda externa para la generación de nuevas claves, es decir, funciona de forma autónoma. Para que el sistema sea más eficiente en la generación, las nuevas claves se almacenan en una FIFO, de forma que se aprovechen los ciclos en que el sistema está ocupado realizando otras operaciones. Cuando se necesite una nueva clave se consulta primero la FIFO y en el caso de que haya alguna clave disponible, no será necesario esperar la generación de una nueva.

Encriptación
Esta función encripta una clave con otra clave de 128 bits que le sean proporcionada, mediante le algoritmo AES.

Lectura
Como se ha dicho anteriormente las nuevas claves se guardan en la FIFO, en el momento en que se precisa una nueva clave sólo es necesaria la lectura de la FIFO.

3.6. Funcionalidades

Modularidad

Descripción de elementos que componen el sistema:

El sistema esta compuesto por:

1 - Una FIFO de entrada de operaciones de la cual se extraen periódicamente las instrucciones a tratar.

2 - Unidad de Tratamiento de la dirección y el acceso a memoria.

- Un módulo procesador de la dirección DirProcess el cual nos permite calcular el padre o los hijos de la dirección a tratar según la estructura de árbol de la memoria.
- Una memoria de direcciones organizada lógicamente como una estructura en forma de árbol.

3 – Unidad de Generación/encriptación de claves.

- Un módulo generador de claves nuevas.
- Un módulo encriptador de claves.

4- Unidad de tratamiento de los valores LR.

- Una memoria de valores LR.
- Un módulo procesador de valores LR el cual nos da el nuevo valor LR de la clave que se esta tratando.

5 - Una FIFO de salida donde se guardan los resultados para que el sistema los consuma.

Operaciones

El sistema gestor de claves debe ser capaz de realizar varias operaciones sobre dichas claves, como son las operaciones de unión de una nueva clave, eliminación (desunión) de una clave existente, resincronización (recalcular) alguna de las claves o Actualizar y obtener a clave de grupo.

Dado que la memoria se gestiona por eficiencia como un árbol de claves, cada vez q se opera sobre una dirección no se implica en esa operación toda al memoria si no solo el camino que va desde la clave (hoja del árbol) hasta al raíz de dicho árbol.

De este modo se pueden concebir las operaciones como bucles que iteran desde la hoja hasta la raíz realizando operaciones oportunas en cada nodo.

Las operaciones implementadas son : inicio, unión, desunión, resincronización, actualizar clave y entregar clave.

- Inicio:

Operación que pone a punto el sistema.

- Carga los valores necesarios en el generador de claves para su inicialización (palabra de fecha D, la semilla S0 y la clave generadora KG).

- Inicia el generador.

- Inicia la memoria principal y de LR.

-Unión:

Operación que introduce una nueva clave en el sistema.

- Lee la dirección de usuario y la clave con la que se quiere unir.

- Guarda la clave en la posición de memoria del usuario.

- Introduce el usuario al grupo realizando las encriptaciones pertinentes de clave. Para cada clave auxiliar vinculada en el proceso se realiza una encriptación con una nueva clave generada y con la clave del usuario que se une. Las claves vinculadas son las correspondientes al camino desde la clave de usuario (hoja) hasta la clave de grupo (raíz).

- Actualiza los valores LR

```
Union (clave k, direccion dir, orden n) {

DirRaiz = generaRaiz(n);            // calculamos la raíz del sistema (n-1 unos y un 0)

Almacenar(k,dir);                   // almacenamos la clave del usuario que se une

Mientras (dir != DirRaiz) {         // mientras no lleguemos a la raíz

        dir= padre (dir);           // calculamos la dirección del padre del nodo en el
                                            //que nos encontramos

        claveNueva = ObtenerClaveGenerada(); // generamos una clave nueva
```

```
        encriptar (claveNueva,ObtenerClave(dir)); // la encriptamos con    la actual

        encriptar (claveNueva, k);     // y la encriptamos con la del usuario

        Almacenar(claveNueva,dir);  //generamos la clave obtenida para
                                                //actualizar asi la clave del nodo

        Actualizar(LR);                         // Actualizamos el LR del nodo

        }

}
```

Tabla 3.6.1 .Algoritmo Operación de unión.

-Desunión

Operación que elimina una clave de usuario del sistema.

- Lee la dirección del usuario que deja el sistema.

- Elimina la clave de dicho usuario.

- Extrae el usuario al grupo realizando las encriptaciones pertinentes de clave. Para cada clave auxiliar vinculada en el proceso se realiza una encriptación con una nueva clave generada y con la clave de cada uno de sus hijos, si existen. Las claves vinculadas son las correspondientes al camino desde la clave de usuario (hoja) hasta la clave de grupo (raíz).

```
Desunion (direccion dir, orden n) {//similar a la unión pero con encriptaciones distintas

DirRaiz = generaRaiz(n);

Mientras (dir != DirRaiz) {

        dir= padre (dir);

        claveNueva = ObtenerClaveGenerada();

        Encriptar (claveNueva,Obtener(hijoder(dir)));

        Encriptar (claveNueva,Obtener(hijoizq(dir)));

        Almacenar(claveNueva,padre(dir));

        Actualizar(LR);

        }

}
```

Tabla 3.6.2 .Algoritmo Operación de desunión.

- Resincronización:

Operación que genera una clave nueva para una clave existente y actualiza lso valores de las claves implicadas (las del camino hasta la raíz).

- Lee la dirección del miembro que se resincroniza

- Genera una clave nueva y la almacena en la dirección del usuario

- Realiza la resincronización del grupo, encriptando para cada clave auxiliar vinculada en el proceso, hasta la raíz del árbol, una nueva clave generada con la antigua y con la clave del usuario que se resincroniza.

```
Resincroniza() { // como la unión pero la claves es generada en vez de recibirla

DirRaiz = generaRaiz(n);

k = ObtenerClaveGenerada();

Almacenar(k,dir);

Mientras (dir != DirRaiz) {

        dir= padre (dir);

        claveNueva = ObtenerClaveGenerada();

        Encriptar (claveNueva,ObtenerClave(dir));

        Encriptar (claveNueva, k);

        Almacenar(claveNueva,dir);

        Actualizar(LR);

        }

}
```

Tabla 3.6.3 .Algoritmo Operación de resincronización.

- *Entregar clave:*
    - Entrega la clave de grupo (la de la raíz)

- *Actualizar clave:*
    - Genera una nueva clave para la raíz.

```
ActualizaClave() { //da un nuevo valor a la clave de grupo

        claveNueva = ObtenerClaveGenerada();

        Encriptar (claveNueva,ObtenerClave(raiz));

        Almacena(claveNueva, raiz)

}
```

Tabla 3.6.4 .Algoritmo Operación de Actualizar Clave.

Éstas son las versiones sencillas de los algoritmos, al implementar los hemos mejorado consultando el árbol lr y evitando encriptaciones innecesarias cuando alguna clave no es utilizada por alguno de los dos hijos.

3.7. Ejemplo de funcionamiento del sistema

Suponiendo el árbol de claves vacío, veamos unos ejemplos de funcionamiento con la ejecución de varias instrucciones.

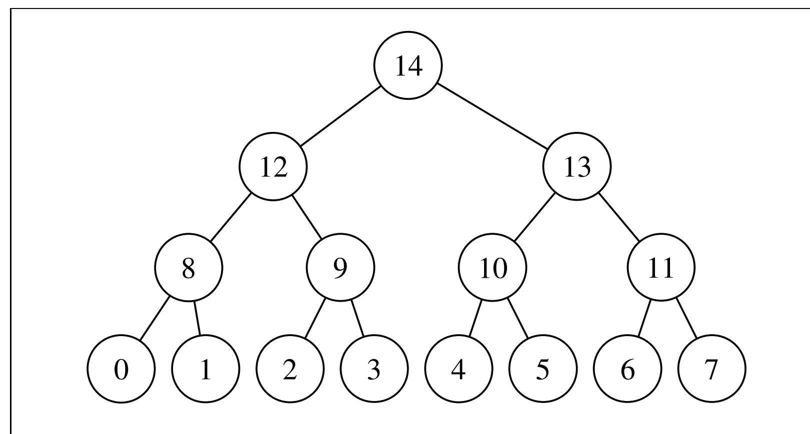


Figura 3.7.1 Árbol de Claves.

Unión del usuario 3:

Se escriben las nuevas claves en el árbol de claves:
En la posición 3 la clave que le pasamos en la instrucción.
En la posición 9 (padre de 3) la 1ª clave generada.
En la posición 12 (padre de 9) la 2ª clave generada.
En la posición 14 (padre de 12) la 3ª clave generada.

En la FIFO de salida se escriben estas tres claves encriptadas con la del usuario 3.

Unión del usuario 2:

Se producen además de los mensajes generados para el usuario 2 otros para 9, C y E, esto es porque el sistema sabe que estas claves estaban en uso (por el usuario 3)

entonces al ser generada una nueva clave 9 ésta se encripta con la antigua clave 9, de este modo el usuario 3 puede obtener la nueva clave.

Ahora en la FIFO se almacenan los siguientes mensajes:
La nueva clave 9 encriptada con la clave de 3:
La nueva clave de 9 encriptada con la antigua 9.
La nueva de 12 encriptada con la de 3.
La nueva de 12 encriptada con la antigua 12.
La nueva de 14 encriptada con la de 3.
La nueva de 14 encriptada con la antigua 14.

Es importante observar que éste es el caso peor, con 6 encriptaciones ($2*\log_2(N)$) y 3 generaciones $\log_2(N)$, dado este número N nunca serán necesarias más operaciones para realizar una unión gracias al esquema de árbol de claves.

El sistema lleva controlado perfectamente el uso de cada clave gracias a la tabla LR. Sabe que están usadas la clave 9 tanto por la derecha como por la izquierda, la 12 por la derecha y la 14 por la izquierda:

En forma de árbol:

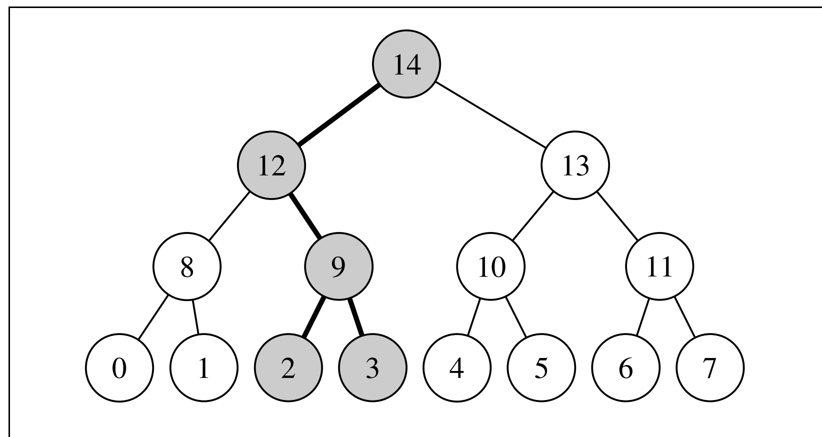


Figura 3.7.2. Árbol de Claves.

Unión del usuario 0:

Se produce un mensaje dirigido a 0 y consiste en la nueva clave 8 encriptada con la clave de 0, la clave 8 al no estar utilizada previamente no es necesario encriptarla de nuevo.
El siguiente es la nueva clave para 12 encriptada con la de 0.
Como esta sí que estaba utilizada se encripta con su antigua clave, de este modo tanto el usuario 2 como el 3 pueden desencriptarla. Este es el siguiente mensaje.
El siguiente mensaje es la nueva clave para 14 encriptada con la de 0.
Y el último es la nueva clave para 14 con la antigua clave 14.

Desunión del usuario 3:
Esta operación es complicada, veamos el árbol lr para comprender que mensajes van a ser necesarios:

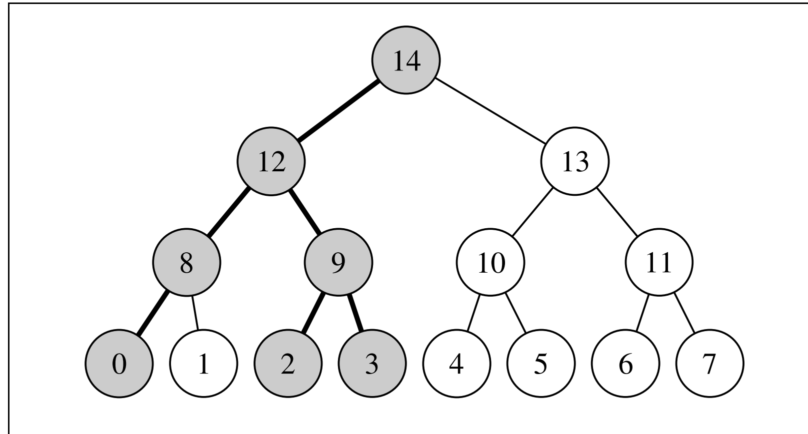


Figura 3.7.3. Árbol de Claves.

Es necesario actualizar las claves 9, 12 y 14 y distribuirlas de modo que el usuario 3 no pueda desencriptarlas. Entonces encriptamos la nueva clave 9 con la clave de 2, la nueva clave 12 con la clave 8 y la nueva 9 (que 2 posee pero no 3), la nueva 14 con la nueva 12.

Se puede comprobar que los cuatro mensajes resaltados en la captura se corresponden a los mencionados más arriba.

Una consideración acerca de la eficiencia de este mecanismo: vemos que han sido necesarias 4 encriptaciones para una operación desunión 5 en el caso peor en el que la clave superior también hubiera necesitado ser encriptada dos veces ($2*\log_2(N)-1$), mientras que utilizando el esquema “plano” en el que cada usuario sólo posee una clave sólo hubieran hecho falta 2 (una para 0 y otra para 2), pero imaginemos nuestro sistema lleno de usuarios, hubiéramos necesitado 5 encriptaciones y sin embargo el sistema “plano” 9, mejor aún imaginemos un sistema de un millón de usuarios, el mecanismo de árbol necesitaría ($2*\log_2(2^{10})-1$) = 19 encriptaciones mientras que uno “plano” 999.999 encriptaciones.

Desunión del usuario 2:
Esta operación es similar a la comentada antes, una única consideración: no es necesario generar una nueva clave para 9 ya que es una clave que se queda sin utilizar, por tanto es necesario: generar una clave nueva para 12 y encriptarla con 8, y generar una nueva para 14 y encriptarla con la nueva 12.

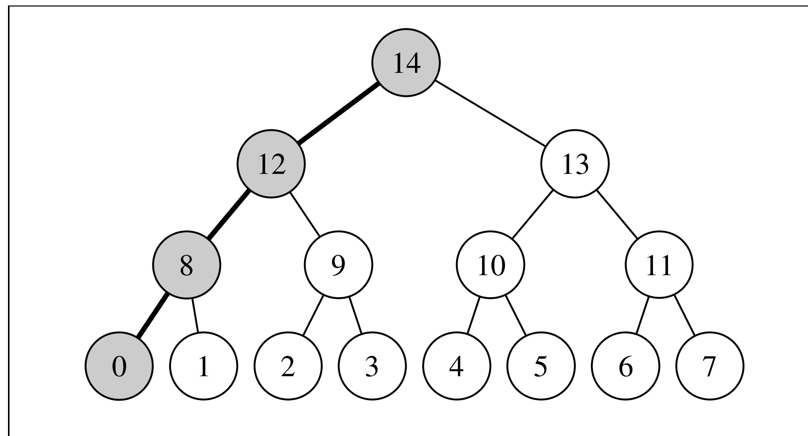


Figura 3.7.4. Árbol de Claves.

Observamos que se hacen las dos operaciones requeridas, además se ha cambiado la clave de 9, pero esto no es una clave válida, es simplemente basura, pero no importa porque la clave 9 está marcada como no utilizada.

4. Implementación

4.1. Ruta de datos

El sistema ha sido implementado siguiendo la siguiente ruta de datos, cuyos componentes se irán explicando a continuación.

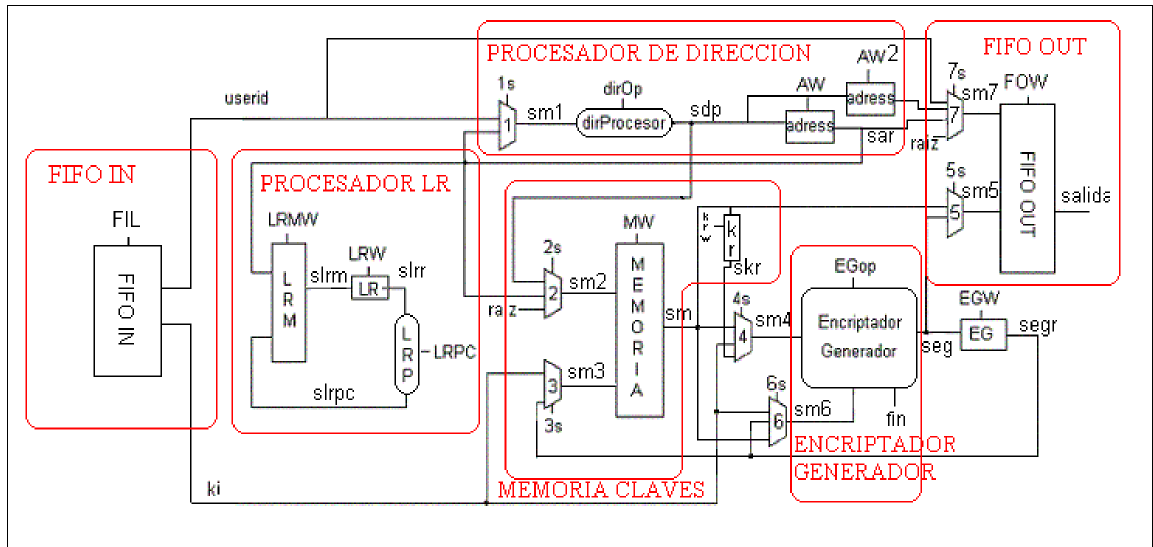


Figura 4.1. Ruta de Datos del Sistema.

4.2. El Generador/Encriptador

Para describir este módulo vamos a empezar desde su interior.

4.2.1. El AES

Como se dicho ya, el tamaño de bloque es de 128 bits, por tanto las entradas del AES serán dos claves de este tamaño, al igual que la salida que será una nueva clave resultado de encriptar la primera con la segunda.

La encriptación se lleva a cabo en 10 vueltas, cada un de los cuales cuenta con cuatro fases, excepto la última que omite una de las cuatro fases. En cada uno de los ciclos se añade la clave de ronda. La clave que se proporciona como entrada se expande en un vector de cuarenta y cuatro palabras de 32 bits, "w[i]". La clave de ronda estaría formada por cuatro palabras de estas cuarenta y cuatro.
El cálculo de "w" sería de la siguiente forma:

1) Las primeras cuatro palabras (*w[0]* hasta *w[3]*) de la clave son el inicio de la clave de cifrado.
2) Para calcular *w[i]* para *i* desde 4 hasta 43:
   2.1) Asignar *temp = w[i-1]*.

2.2) Si *i* = 4, 8, 12, 16,..., 40 (múltiplos de 4).
2.2.1) Desplazar un lugar los bytes, es decir, si tenemos una palabra de 32 bits w[i][0] = w[i][1], w[i][1] = w[i][2], w[i][2] = w[i][3], w[i][4] = w[i][0], suponiendo que la palabra está compuesta por 4 cadenas de bytes.
2.2.2) Reemplazar cada byte (usando función de sustitución Sustituir Bytes, Sbox, que luego explicaremos).
2.2.3) Realizar una X-Or con la constante de redondeo Rcon[i]. Donde Rcon[i]= (RC[k], 00, 00, 00), con RC[1] = 1, y RC[k] = 2*RC[k-1] , i = 4,8,12,...,44, es decir, múltiplos de 4.
2.2.4) Asignar w[i] = w[i-4] xor temp.

Tener un vector de cuarenta y cuatro palabras de 32 bits sería en total 1408 bits, que es bastante grande, por tanto lo que haremos será calcular para cada ciclo su correspondiente clave de ronda, lo que sería cuatro palabras de 32 bits, es decir, 128 bits. Con esto conseguimos no tener que guardar un dato tan grande y distribuir el tiempo de expansión de la clave de forma uniforme en los diez ciclos.
Para que sea un poco más fácil de entender mostramos el código en C de la función "extenderclave".
clave: palabra de 128 bits, dividida en 16 bytes (sería equivalente w[i]..w[i+3]).
ronda: indica porque ciclo vamos.

```
void extenderClave(byte* clave,int ronda){
//2.2)
      byte aux[4];
//2.2.1)
      aux[0]=clave[13];
      aux[1]=clave[14];
      aux[2]=clave[15];
      aux[3]=clave[12];
//2.2.2)
      for (int i=0;i<4;i++)
      {
            aux[i] = SBox[aux[i]];
      }
//2.2.3)
// byte Pot2[] = {0x01,0x02,0x04,0x08,0x10,0x20,0x40,0x80,0x1b,0x36};
      aux[0] ^= Pot2[ronda];
      //2.2.4)
      clave[0] ^= aux[0];
      clave[1] ^= aux[1];
      clave[2] ^= aux[2];
      clave[3] ^= aux[3];
//2.1)
      for (int i=4;i<16;i+=4)
      {
            clave[i] ^= clave[i-4];
            clave[i+1] ^= clave[i-3];
            clave[i+2] ^= clave[i-2];
            clave[i+3] ^= clave[i-1];
      }
}
```

Figura 4.2.1.1 Código C ExtenderClaves.

Fases:

1.- Añadir Clave.
Se efectúa en esta etapa una operación XOR binaria con una porción de la clave expandida, es decir, la clave de ronda correspondiente.

entrada y clave: palabra de 128 bits, dividida en 16 bytes.

```
void addKey(byte *entrada, byte* clave)
{
     for (int i=0;i<16;i++)
          entrada[i] = entrada[i] ^ clave[i];
}
```

Figura 4.2.1.2 Código C Añadir Claves.

Sólo la etapa de adición de clave de ronda hace uso de la clave. Por esta razón el cifrado comienza y termina con la etapa de adición de clave de ronda. Cualquier otra etapa aplicada a la principio o al final, es reversible sin necesidad de conocer la clave por eso no añadiría seguridad. Aún así la fase de adición de clave de ronda no sería excepcional en si misma. Las otras tres etapas juntas desordenan los bits, pero no proporcionan seguridad.

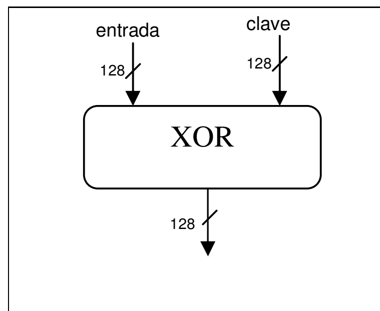


Figura 4.2.1.3 Entidad Añadir Claves.

2.- Sustituir bytes
En esta fase se utiliza una tabla, llamada caja-S (Sbox), caja de sustitución que se emplea comúnmente en la descripción de los cifradores simétricos para referirse a una tabla utilizada para un mecanismo de sustitución del tipo *consulta en tabla*, para efectuar una sustitución del bloque byte a byte.

```
void subBytes(byte *entrada)
{
     for (int i=0;i<16;i++)
          entrada[i] = SBox[entrada[i]];
}
```

Figura 4.2.1.4 Código C Sustituir Bytes.

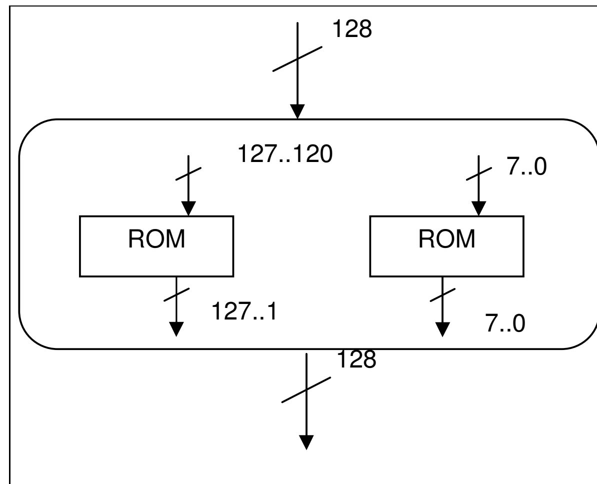


Figura 4.2.1.5.Entidad Sustituir Bytes.

Sbox

Para la implantación de las SBox utilizaremos memorias ROM. Esta parte va a constituir la parte más costosa del sistema, en cuanto a espacio, ya que para que el sistema sea eficiente se tendrá que replicar numerosas veces.

```
byte SBox[] =
{
 0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5,
 0x30, 0x01, 0x67, 0x2b, 0xfe, 0xd7, 0xab, 0x76,
 0xca, 0x82, 0xc9, 0x7d, 0xfa, 0x59, 0x47, 0xf0,
 0xad, 0xd4, 0xa2, 0xaf, 0x9c, 0xa4, 0x72, 0xc0,
 0xb7, 0xfd, 0x93, 0x26, 0x36, 0x3f, 0xf7, 0xcc,
 0x34, 0xa5, 0xe5, 0xf1, 0x71, 0xd8, 0x31, 0x15,
 0x04, 0xc7, 0x23, 0xc3, 0x18, 0x96, 0x05, 0x9a,
 0x07, 0x12, 0x80, 0xe2, 0xeb, 0x27, 0xb2, 0x75,
 0x09, 0x83, 0x2c, 0x1a, 0x1b, 0x6e, 0x5a, 0xa0,
 0x52, 0x3b, 0xd6, 0xb3, 0x29, 0xe3, 0x2f, 0x84,
 0x53, 0xd1, 0x00, 0xed, 0x20, 0xfc, 0xb1, 0x5b,
 0x6a, 0xcb, 0xbe, 0x39, 0x4a, 0x4c, 0x58, 0xcf,
 0xd0, 0xef, 0xaa, 0xfb, 0x43, 0x4d, 0x33, 0x85,
 0x45, 0xf9, 0x02, 0x7f, 0x50, 0x3c, 0x9f, 0xa8,
 0x51, 0xa3, 0x40, 0x8f, 0x92, 0x9d, 0x38, 0xf5,
 0xbc, 0xb6, 0xda, 0x21, 0x10, 0xff, 0xf3, 0xd2,
 0xcd, 0x0c, 0x13, 0xec, 0x5f, 0x97, 0x44, 0x17,
 0xc4, 0xa7, 0x7e, 0x3d, 0x64, 0x5d, 0x19, 0x73,
 0x60, 0x81, 0x4f, 0xdc, 0x22, 0x2a, 0x90, 0x88,
 0x46, 0xee, 0xb8, 0x14, 0xde, 0x5e, 0x0b, 0xdb,
 0xe0, 0x32, 0x3a, 0x0a, 0x49, 0x06, 0x24, 0x5c,
 0xc2, 0xd3, 0xac, 0x62, 0x91, 0x95, 0xe4, 0x79,
 0xe7, 0xc8, 0x37, 0x6d, 0x8d, 0xd5, 0x4e, 0xa9,
 0x6c, 0x56, 0xf4, 0xea, 0x65, 0x7a, 0xae, 0x08,
 0xba, 0x78, 0x25, 0x2e, 0x1c, 0xa6, 0xb4, 0xc6,
 0xe8, 0xdd, 0x74, 0x1f, 0x4b, 0xbd, 0x8b, 0x8a,
```

```
 0x70, 0x3e, 0xb5, 0x66, 0x48, 0x03, 0xf6, 0x0e,
 0x61, 0x35, 0x57, 0xb9, 0x86, 0xc1, 0x1d, 0x9e,
 0xe1, 0xf8, 0x98, 0x11, 0x69, 0xd9, 0x8e, 0x94,
 0x9b, 0x1e, 0x87, 0xe9, 0xce, 0x55, 0x28, 0xdf,
 0x8c, 0xa1, 0x89, 0x0d, 0xbf, 0xe6, 0x42, 0x68,
 0x41, 0x99, 0x2d, 0x0f, 0xb0, 0x54, 0xbb, 0x16
};
```

Figura 4.2.1.6 Código C SBox.

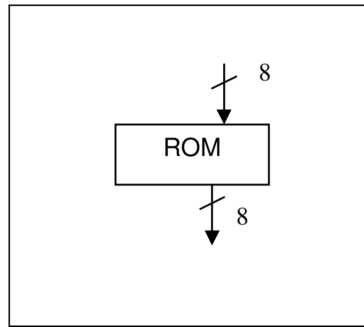


Figura 4.2.1.7 Entidad SBox.

3.- Desplazar filas.
En esta fase se efectúa una permutación simple fila a fila.
Para el texto:

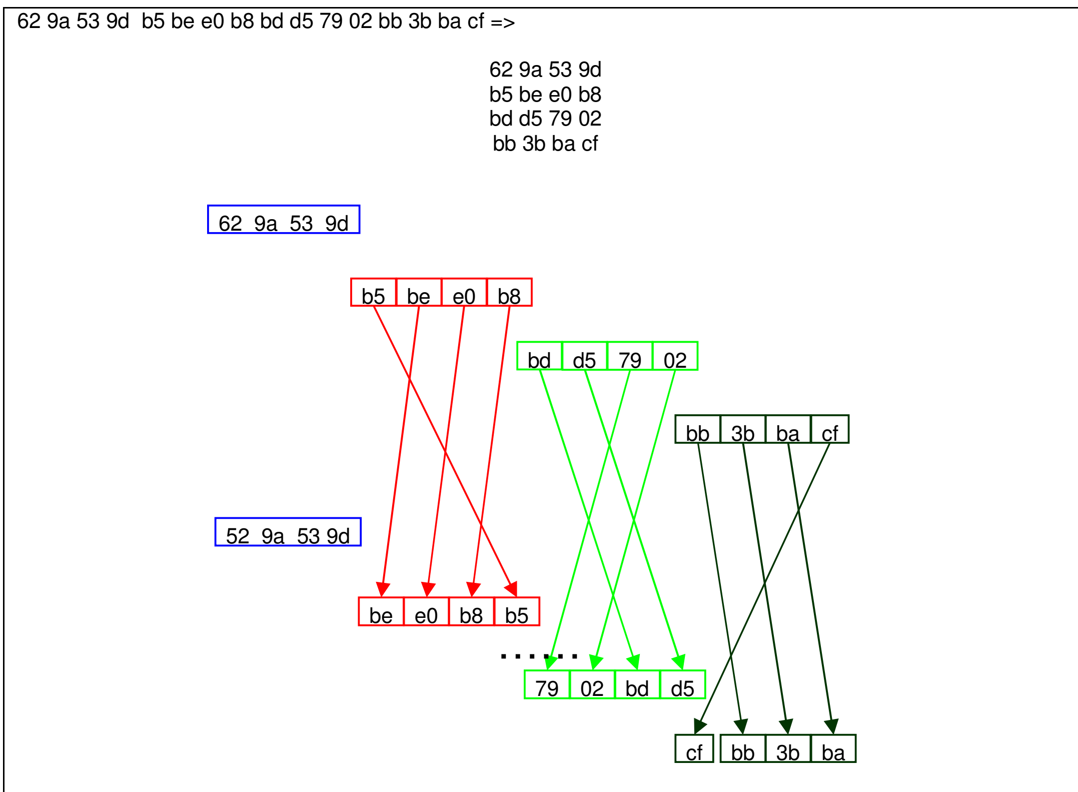


Figura 4.2.1.8 Algoritmo Desplazar Filas.

Se puede ver en la figura 4.2.1.8 que los desplazamientos siguen un esquema.

```
void shiftRows(byte *entrada)
{
     byte aux;

     aux = entrada[1];
     entrada[1] = entrada[5];
     entrada[5] = entrada[9];
     entrada[9] = entrada[13];
     entrada[13] = aux;

     aux = entrada[2];
     entrada[2] = entrada[10];
     entrada[10] = aux;
     aux = entrada[6];
     entrada[6] = entrada[14];
     entrada[14] = aux;

     aux = entrada[15];
     entrada[15] = entrada[11];
     entrada[11] = entrada[7];
     entrada[7] = entrada[3];
     entrada[3] = aux;
}
```

Tablas 4.2.1.9 Código C Desplazar Filas.

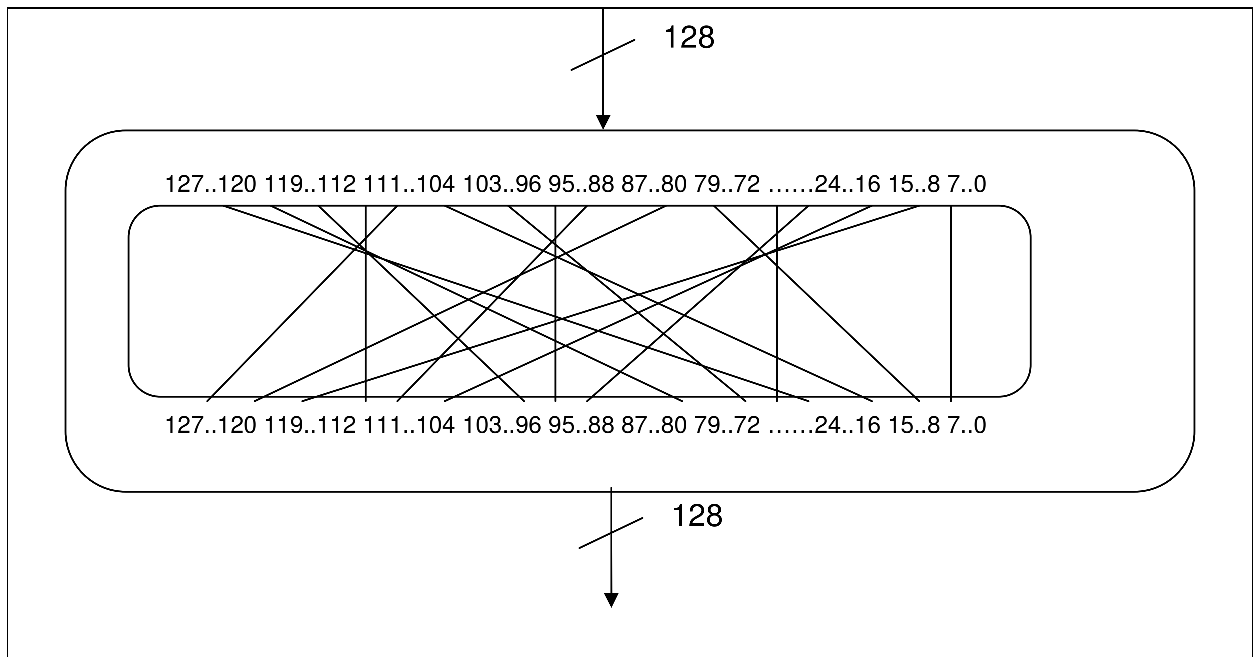


Figura 4.2.1.10 Entidad Desplazar Filas.

4.- Barajar Columnas.
Se lleva a cabo una sustitución que modifica cada byte de una columna por una función de todos los byte de la misma.

Esta función mezcla los datos de cada columna de acuerdo a las siguientes fórmulas:

```
  s(c*4) <= x2(c*4) xor x3(c*4+1) xor e(c*4+2) xor e(c*4+3);
 s(c*4+1) <= e(c*4) xor x2(c*4+1) xor x3(c*4+2) xor e(c*4+3);
 s(c*4+2) <= e(c*4) xor e(c*4+1) xor x2(c*4+2) xor x3(c*4+3);
 s(c*4+3) <= x3(c*4) xor e(c*4+1) xor e(c*4+2) xor x2(c*4+3);
```

Figura 4.2.1.11 Formulas Barajar Columnas

Donde: e es la entrada, s la salida, c = 0-3;
x2, x3 son el resultado de multiplicar por 2 y por 3 en GF(256) respectivamente, donde cada número binario representa un polinomio, la suma se corresponde a la operación xor y la multiplicación a la multiplicación de polinomios módulo un polinomio irreducible ($x^8+x^4+x^3+x+1$).

```
i = 1x8+0x7+0x6+0x5+1x4+1x3+0x2+1x+1 = x8+x4+x3+x+1

byte irreducible = 0x11b;

byte por2(byte e)
{
	unsigned short r;
	r = e << 1;		// r=e*2
	if ((r & 0x100) != 0) //r=r mod x^8 + x^4 + x^3 + x
+1
		r ^= irreducible;
	return r;
}

byte por3(byte e)
{
	unsigned short r;
	r = e << 1;
	r ^= e;			// r = (e*2)+e;
	if ((r & 0x100) != 0)
		r ^= irreducible;
	return r;
}
void mixColumns(byte *entrada)
{
	byte x[4];
	byte x2[4];
	byte x3[4];

	int pos=0;
	for (int c=0;c<4;c++)
	{
		for (int i=0;i<4;i++)
		{
			x[i]=entrada[pos+i];
			x2[i]=por2(entrada[pos+i]);
			x3[i]=por3(entrada[pos+i]);
```

```
            }

            entrada[pos] = x2[0] ^ x3[1] ^  x[2] ^  x[3];
            pos++;
            entrada[pos] =  x[0] ^ x2[1] ^ x3[2] ^  x[3];
            pos++;
            entrada[pos] =  x[0] ^  x[1] ^ x2[2] ^ x3[3];
            pos++;
            entrada[pos] = x3[0] ^  x[1] ^  x[2] ^ x2[3];
            pos++;
      }
}
```

Figura 4.2.1.12 Código C Barajar Columnas.

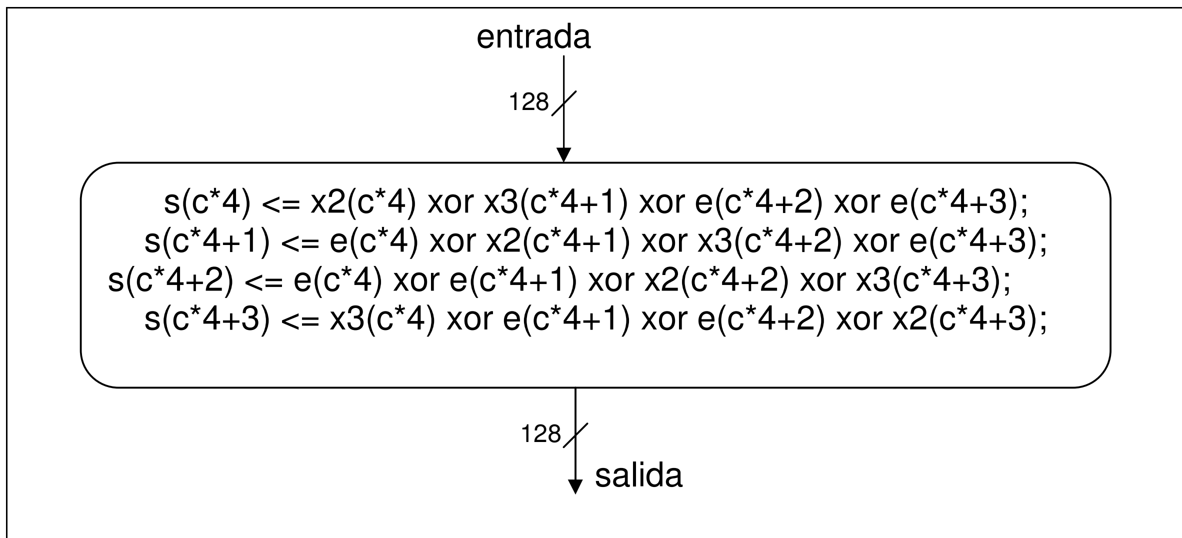


Figura 4.2.1.13 Entidad Barajar Columnas.

Por3 y Por2

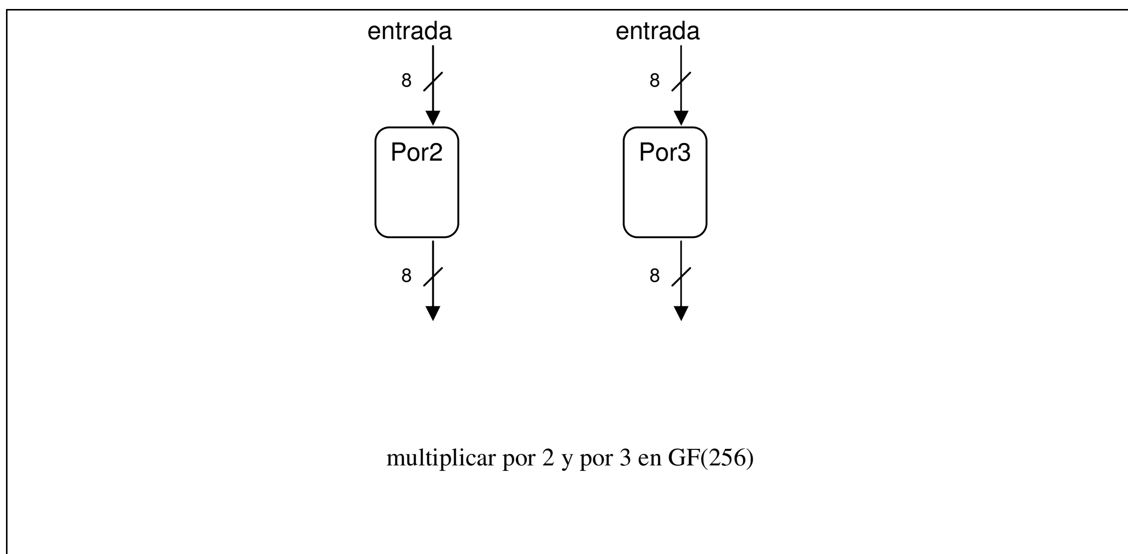


Figura 4.2.1.14 Entidad Por2 y Por3

Podemos ver el cifrado como una sucesión de operaciones alternas de cifrado XOR de un bloque (adición de la clave de ronda), seguido por el barajado de bloque (las otras tres etapas), seguido de cifrado XOR y así sucesivamente. Este esquema es eficiente y muy seguro.
Cada etapa es fácilmente reversible. Para la etapa de sustitución de bytes, desplazamiento de filas y adición de clave de ronda se utiliza una función inversa en el algoritmo de descifrado. Para la etapa de adición de clave de ronda, la inversión de obtiene mediante la operación XOR de la misma clave de ronda sobre el bloque, debido a que A xor A xor B = B.
Como la mayoría de los cifradores de bloque, el algoritmo de descifrado hace uso de la clave expandida en orden inverso. Sin embargo, el algoritmo de descifrado no es idéntico al de cifrado. Esto consecuencia de la particular estructura del AES.
La ronda final del cifrado y descifrado consta sólo de tres etapas. De nuevo, es consecuencia de la estructura particular del AES, siendo necesario para que el descifrado se posible.

```
void aes(byte* entrada, byte* clave){
	byte aux[16];
	//adicción de la clave de ronda
	addKey(entrada,clave);
	//los nueve ciclos iniciales son idénticos
	for (int i=0;i<9;i++)
	{
		//sustituir bytes
		subBytes(entrada);
		//desplazar filas
		shiftRows(entrada);
		//barajar columnas
		mixColumns(entrada);
		//calcular clave de ronda
		extenderClave(clave,i);
		//adicción de la clave de ronda
		addKey(entrada,clave);
	}
	//ciclo final sin fase de barajar columnas
	subBytes(entrada);
	shiftRows(entrada);
	extenderClave(clave,9);
	addKey(entrada,clave);}
```

Tabla 4.2.1.15 Código C AES.

Como se ha visto la adicción de la clave de ronda se hace 11 veces, como dijimos cada clave de ronda tiene 4 palabras de 32 bits (w[i]..w[i+3]), lo que hace las cuarenta y cuatro palabras.

4.2.2. El AES en VHDL.

Para cada una de las fases, así como, para la creación de la clave de ronda, vamos a crear una entidad, de forma que se siga el mismo esquema que la versión de C. Para seguir este esquema la representación de los datos será la siguiente:
Todos los datos de entrada y salida, es decir, las claves tendrán tipo “word". El tipo word será un array de 16 posiciones de tipo byte (bit_vector (7 downto 0)), con lo que contaríamos con un total de 128 bits. Este formato nos facilitará los accesos por bytes a los datos.

Una vez comentadas todas las entidades que son necesarias para la implementación del algoritmo sólo queda reunirlas para formar el AES.

- Formato combinacional.

Para esta primera implementación no vamos utilizar registros sino que haremos que se realice todo el proceso en un solo ciclo.Para esta implementación como puede verse en la figura 4.2.2.1 se han replicado todas las entidades, Barajar Columnas 9 veces, Sustituir Bytes, Desplazar Filas y Extender Clave 10 veces y Añadir Clave 11veces.

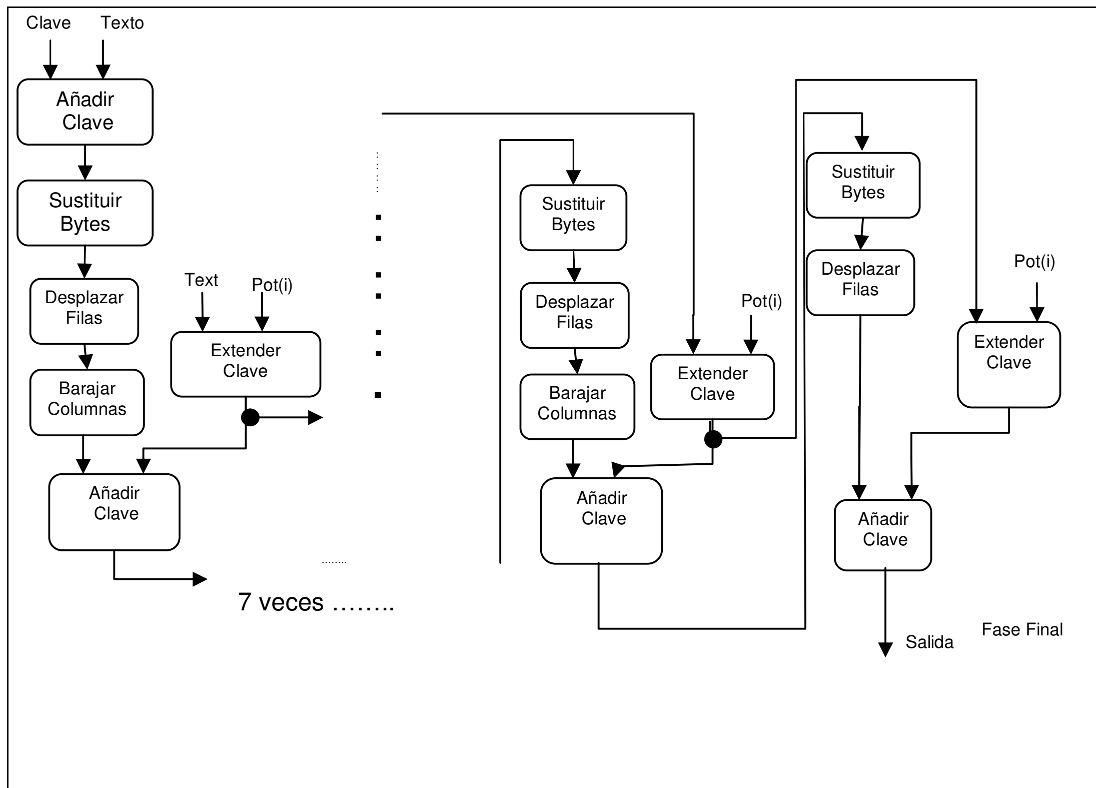


Figura 4.2.2.1 Versión Combinacional.

Expresado en lenguaje Vhdl sería:

```
c11 : addkey PORT MAP(texto,clave,sb(0));
c12: extenderclave PORT MAP (clave,pot2(0),akk(0));
c2: FOR i IN 0 TO 8 GENERATE
      c21: subbytes PORT MAP (sb(i),sr(i));
```

```
      c22: shiftrows PORT MAP (sr(i),mc(i));
      c23: mixcolumns PORT MAP (mc(i),ak(i));
      c24: extenderclave PORT MAP (akk(i),pot2(i+1),akk(i+1));
      c25: addkey PORT MAP (ak(i),akk(i),sb(i+1));
    END GENERATE;
c31 : subbytes PORT MAP (sb(9),sr(9));
c32 : shiftrows PORT MAP (sr(9),ak(9));
c33 : addkey PORT MAP(ak(9),akk(9),sl);
```

Figura 4.2.2.2 Código Vhdl Versión Combinacional.

- Formato Secuencial

Para pasarlo a forma secuencial hemos decidido introducir registro en cada ciclo del algoritmo por lo que dividiríamos el AES en 10 ciclos, con esto conseguimos un reloj más pequeño que en la implementación anterior y podemos reutilizar todas la entidades, de forma, que el coste en espacio es mucho menor.
Una primera versión podría ser:

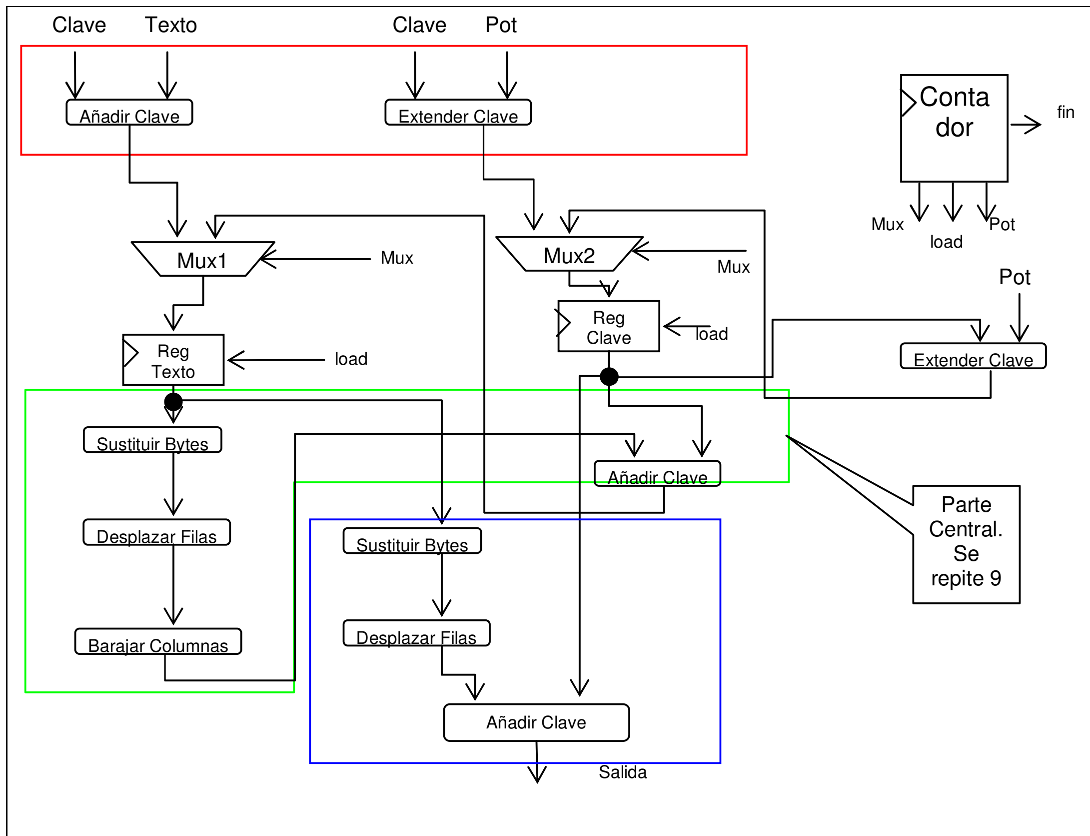


Figura 4.2.2.3 Versión Secuencial.

Como se puede ver en la figura 4.2.2.3 se han introducido dos registro. En uno de ellos guardaremos la clave y en otro el dato a encriptar. La lógica de control la llevaremos por medio de un contador que nos servirá para saber en que ciclo estamos, necesario para el cálculo de la clave de ronda. También se encargará de la activación de los multiplexores, así como de activar una señal de "fin" cuanto el algoritmo haya finalizado.
Se puede observar que esta primera versión seguimos replicando entidades, esto es debido a que se intentó hacer una versión lo más similar posible a la obtenida en C. Si

eliminamos las entidades replicadas el resultado final sería el que aparece en la figura 4.2.2.4.

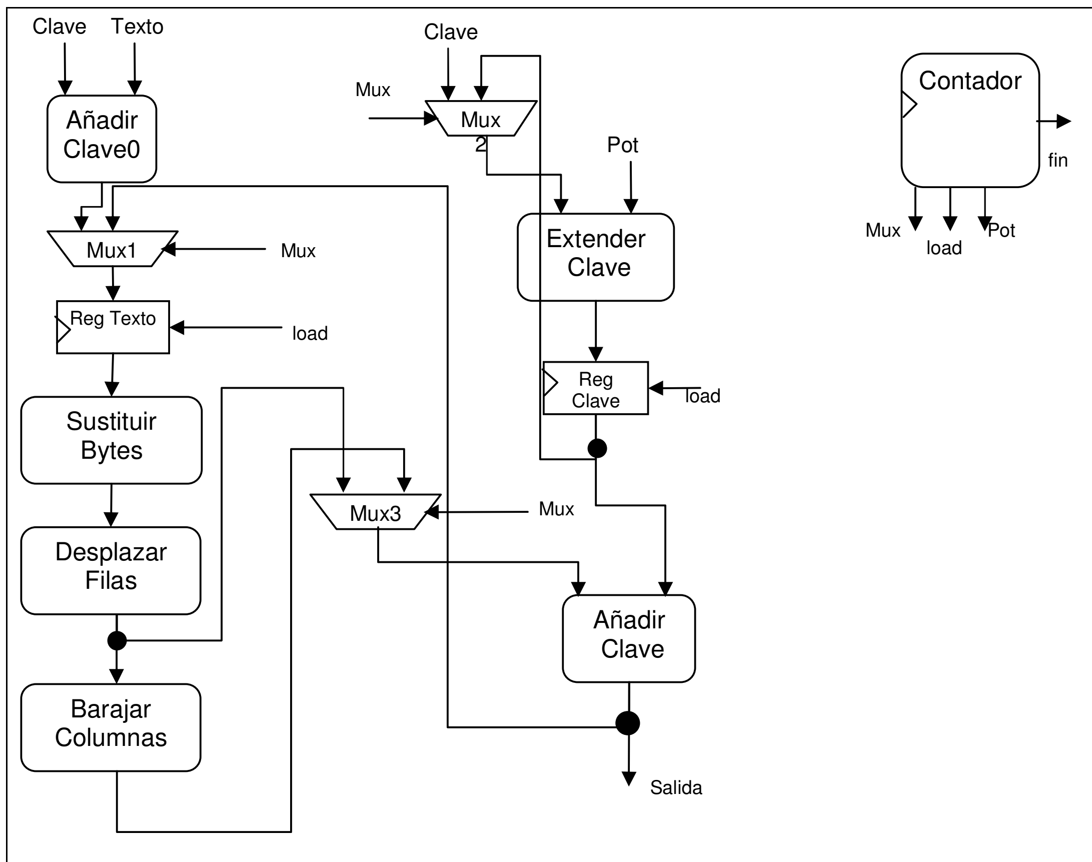


Figura 4.2.2.4 Versión Secuencial Mejorada.

Expresado en Vhdl:

```
contad: contador PORT MAP(clk,reset,pot,mux1,mux2,regload,fin);
añadirClave0 : addkey PORT MAP(entrada,clave,sb0);
mux1: mux PORT MAP(sb0,sb2,mux1,reg1);
registroTexto: registro PORT MAP(clk,regload,reg1,sb1);
mux2: mux PORT MAP(clave,akk1,mux1,akk2);
extenderclave: extenderclave port map (akk2,pot,reg2);
registroClave: registro PORT MAP(clk,regload,reg2,akk1);
sustituirBytes: subbytes port map (sb1,sr1);
desplazarFilas: shiftrows port map (sr1,mc);
mezclarColumnas: mixcolumns port map (mc,ak1);
mux3:mux PORT MAP(mc,ak1,mux2,ak3);
añadirClave: addkey port map (ak3,akk1,sb2);
salida<=sb2;
```

Figura 4.2.2.5 Código Vhdl Versión Secuencial.

4.2.3. Resultados y Conclusiones del Módulo AES.

Una vez implementado el AES pasamos a ver los resultados obtenidos. Para obtener estos resultados hemos utilizado Xilinx ISE y Model Sim.
1-.Versión Combinacional.
Debido al gran tamaño de la implementación hemos tenido que utilizar una FPGA relativamente grande, Virtex2P xc2vp40, para esta FPGA hemos obtenido:

El tiempo empleado en obtener el resultado correcto sería 132.7 ns. Un ejemplo de simulación sería:

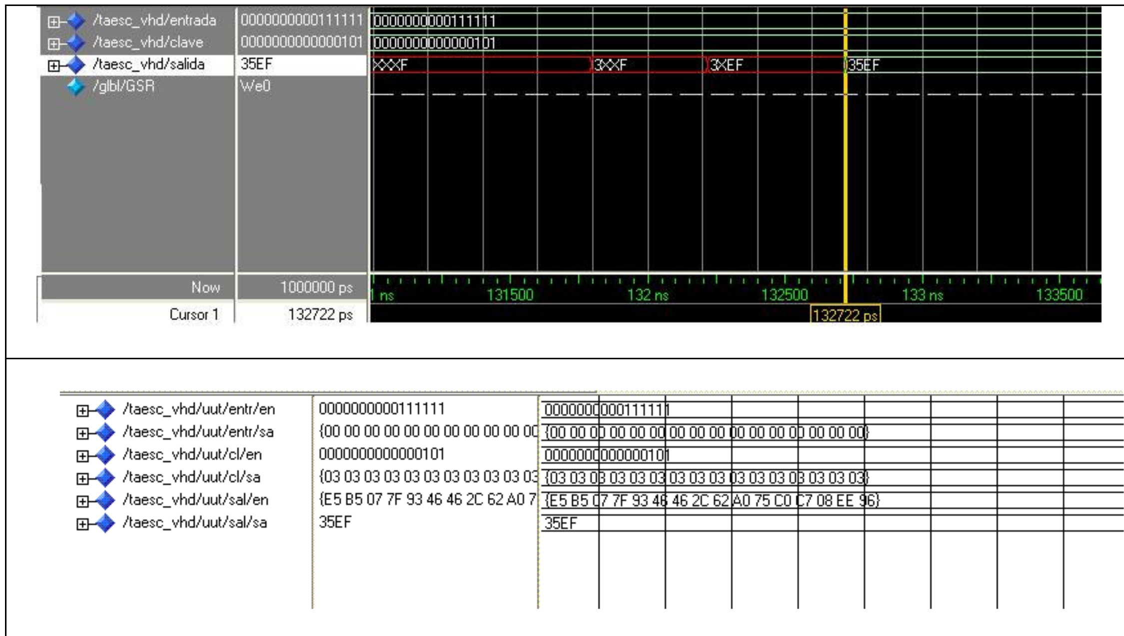


Figura 4.2.3.1 Ejemplo de Simulación Versión Combinacional.

Si nos fijamos en el archivo de informe generado por Xilinx tras la síntesis (tabla 4.2.3.2),(están subrayadas las partes más importantes), podemos ver:

-. La implementación carece de reloj, algo obvio al ser todo combinacional.
-. Para la generación de las Sbox, se utilizan memorias ROM.
-. El área ocupada es el 75%, el número de Slices es 13580.

```
=========================================================================
*                           HDL Synthesis
=========================================================================
Synthesizing Unit <por3>.
    Related source file is "C:/pol/c2/aesc.vhd".
    Found 1-bit xor2 for signal <$n0000> created at line 148.
    Found 1-bit xor2 for signal <$n0001> created at line 148.
    Found 1-bit xor2 for signal <$n0002> created at line 148.
    Found 1-bit xor2 for signal <$n0003> created at line 148.
    Found 1-bit xor2 for signal <$n0004> created at line 148.
    Found 1-bit xor2 for signal <$n0005> created at line 148.
    Found 1-bit xor2 for signal <$n0006> created at line 148.
    Found 1-bit xor2 for signal <$n0007> created at line 148.
Unit <por3> synthesized.
Synthesizing Unit <por2>.
    Related source file is "C:/pol/c2/aesc.vhd".
Unit <por2> synthesized.
Synthesizing Unit <sbox>.
    Related source file is "C:/pol/c2/aesc.vhd".
    Found 256x8-bit ROM for signal <$n0001> created at line 48.
    Summary:
      inferred   1 ROM(s).
Unit <sbox> synthesized.
Synthesizing Unit <casal>.
    Related source file is "C:/pol/c2/aesc.vhd".
    Found 16-bit xor8 for signal <sa>.
```

```
    Summary:
      inferred  16 Xor(s).
Unit <casal> synthesized.
Synthesizing Unit <mixcolumns>.
    Related source file is "C:/pol/c2/aesc.vhd".
    Found 128-bit xor4 for signal <s>.
    Summary:
      inferred 128 Xor(s).
Unit <mixcolumns> synthesized.
Synthesizing Unit <shiftrows>.
    Related source file is "C:/pol/c2/aesc.vhd".
Unit <shiftrows> synthesized.
Synthesizing Unit <subbytes>.
    Related source file is "C:/pol/c2/aesc.vhd".
Unit <subbytes> synthesized.
Synthesizing Unit <extenderclave>.
    Related source file is "C:/pol/c2/aesc.vhd".
    Found 8-bit xor3 for signal <tmp<0>>.
    Found 120-bit xor2 for signal <tmp<1:15>>.
    Summary:
      inferred   8 Xor(s).
Unit <extenderclave> synthesized.
Synthesizing Unit <addkey>.
    Related source file is "C:/pol/c2/aesc.vhd".
    Found 128-bit xor2 for signal <s>.
Unit <addkey> synthesized.
Synthesizing Unit <caentra>.
    Related source file is "C:/pol/c2/aesc.vhd".
Unit <caentra> synthesized.
Synthesizing Unit <aes>.
    Related source file is "C:/pol/c2/aesc.vhd".
Unit <aes> synthesized.

=======================================================================
*                        Advanced HDL Synthesis
=======================================================================
HDL Synthesis Report
Macro Statistics
# ROMs                                   : 200
 256x8-bit ROM                           : 200
# Xors                                   : 1664
 1-bit xor2                              : 1168
 1-bit xor8                              : 16
 8-bit xor2                              : 326
 8-bit xor3                              : 10
 8-bit xor4                              : 144
=======================================================================
*                          Low Level Synthesis
==========================================================================
Loading device for application Rf_Device from file '2vp40.nph' in
environment C:/Xilinx.
Mapping all equations...
Building and optimizing final netlist ...
Found area constraint ratio of 100 (+ 5) on block aes, actual ratio is
75.
=======================================================================
*                                Final Report
=======================================================================
Device utilization summary:
---------------------------
Selected Device : 2vp40ff1148-7
```

```
 Number of Slices:                     13580  out of  19392    70%
 Number of 4 input LUTs:               26622  out of  38784    68%
 Number of bonded IOBs:                   48  out of    804     5%
=========================================================================
TIMING REPORT
Clock Information:
------------------
No clock signals found in this design
```

Tabla 4.2.3.2 Informe Xilinx Simulación Combinacional Virtex2P xc2vp40.

Además de la FPGA mencionada anteriormente hemos probado el comportamiento del sistema en otras. Para una FPGA menor, por ejemplo Virtex2P xc2vp30, el diseño no cabe en la placa. Si por el contrario utilizamos una FPGA mayor, los resultados, en relación con el tiempo, mejoran. Esto es debido a que la disponibilidad de los recursos de la placa facilita el proceso de enrutado.

Los resultados obtenidos para la placa Virtex2P xc2vp50 so los siguientes: el área ocupada es el 61% (marcada en rojo) lo que supone 13580 Slices y el tiempo empleado en dar una respuesta correcta es 118.5 ns.

```
Found area constraint ratio of 100 (+ 5) on block aes, actual ratio is
61.
=========================================================================
*                            Final Report                               *
=========================================================================
Final Results
RTL Top Level Output File Name     : aes.ngr
Top Level Output File Name         : aes
Output Format                      : NGC
Optimization Goal                  : Speed
Keep Hierarchy                     : NO

Design Statistics
# IOs                              : 48

Macro Statistics :
# ROMs                             : 200
#      256x8-bit ROM               : 200
# Xors                             : 170
#      1-bit xor8                  : 16
#      8-bit xor3                  : 10
#      8-bit xor4                  : 144

Cell Usage :
# BELS                             : 48224
#      GND                         : 1
#      LUT2                        : 214
#      LUT3                        : 368
#      LUT4                        : 26040
#      MUXF5                       : 11520
#      MUXF6                       : 5760
#      MUXF7                       : 2880
#      MUXF8                       : 1440
#      VCC                         : 1
# IO Buffers                       : 48
#      IBUF                        : 32
#      OBUF                        : 16
=========================================================================
Device utilization summary:
---------------------------
Selected Device : 2vp50ff1148-6
```

```
Number of Slices:                     13580  out of  23616    57%
Number of 4 input LUTs:               26622  out of  47232    56%
Number of bonded IOBs:                   48  out of    812     5%
==================================================================
```

Tabla 4.2.3.2 Informe Xilinx Simulación Combinacional Virtex2P xc2vp50.

Figura 4.2.3.3 Placa Versión Combinacional Virtex2P xc2vp50.

Figura 4.2.3.4 Ejemplo Simulación Combinacional Virtex2P xc2vp50.

Como podemos ver a continuación en la tabla 4.2.3.5 la mayor parte del sistema lo ocupa las SBox (marcado en blanco).

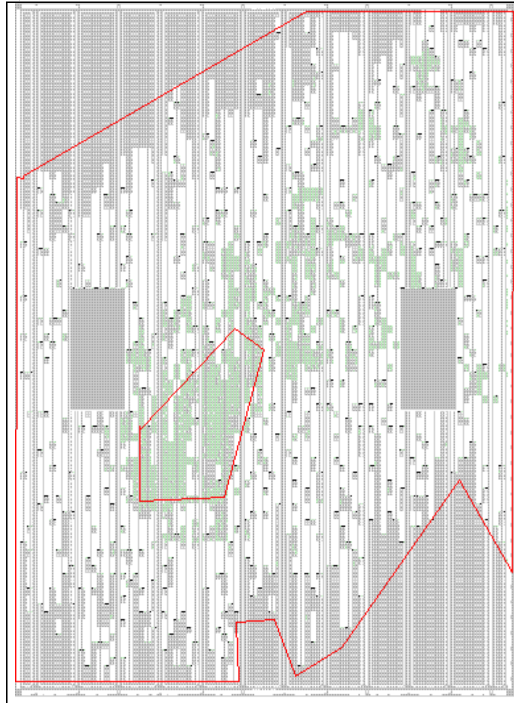


Tabla 4.2.3.5 Área ocupada por la SBox Versión Combinacional Virtex2Pxc2vp50

-. Versión Secuencial.

En este caso la FPGA utilizada, y la que usaremos para meter el sistema completo, es Virtex2P xc2vp20. Después de analizar el informe generado, vemos que esta implementación sólo ocupa el 20% de la placa, 2099 Slices.

```
======================================================================*
Advanced HDL Synthesis
======================================================================
HDL Synthesis Report
Macro Statistics
# ROMs                                : 24
 256x8-bit ROM                        : 24
# Counters                            : 1
 4-bit up counter                     : 1
# Registers                           : 32
 8-bit register                       : 32
# Multiplexers                        : 1
 8-bit 11-to-1 multiplexer            : 1
# Xors                                : 256
 1-bit xor2                           : 144
 1-bit xor8                           : 16
 8-bit xor2                           : 78
 8-bit xor3                           : 2
 8-bit xor4                           : 16

======================================================================
*                          Low Level Synthesis
=========================================================================
Loading device for application Rf_Device from file '2vp20.nph' in
environment H:/Xilinx.
Mapping all equations...
```

```
Building and optimizing final netlist ...
Found area constraint ratio of 100 (+ 5) on block aes, actual ratio is 20.
=======================================================================*
Final Report
=========================================================================
Final Results
RTL Top Level Output File Name     : aes.ngr
Top Level Output File Name         : aes
Output Format                      : NGC
Optimization Goal                  : Speed
Keep Hierarchy                     : NO

Design Statistics
# IOs                              : 50

Macro Statistics :
# ROMs                             : 24
#      256x8-bit ROM               : 24
# Registers                        : 33
#      4-bit register             : 1
#      8-bit register             : 32
# Multiplexers                     : 1
#      8-bit 11-to-1 multiplexer   : 1
# Xors                             : 34
#      1-bit xor8                  : 16
#      8-bit xor3                  : 2
#      8-bit xor4                  : 16

Cell Usage :
# BELS                             : 6129
#      INV                         : 1
#      LUT2                        : 92
#      LUT2_L                      : 13
#      LUT3                        : 83
#      LUT3_L                      : 6
#      LUT4                        : 431
#      LUT4_D                      : 157
#      LUT4_L                      : 2826
#      MUXF5                       : 1344
#      MUXF6                       : 672
#      MUXF7                       : 336
#      MUXF8                       : 168
# FlipFlops/Latches                : 1122
#      FDCE                        : 4
#      FDE                         : 1118
# Clock Buffers                    : 1
#      BUFGP                       : 1
# IO Buffers                       : 49
#      IBUF                        : 33
#      OBUF                        : 16
=====================================================================Device
utilization summary:
---------------------------
Selected Device : 2vp20fg676-7
 Number of Slices:                    2099  out of   9280    22%
 Number of Slice Flip Flops:          1122  out of  18560     6%
 Number of 4 input LUTs:              3608  out of  18560    19%
 Number of bonded IOBs:                 50  out of    404    12%
 Number of GCLKs:                        1  out of     16     6%
```

Tabla 4.2.3.6 Informe Xilinx para Versión Secuencial Virtex2P xc2vp20

Podemos ver continuación en la Figura 4.2.3.6 (marcado en rojo), el área de la placa ocupada por la implementación:

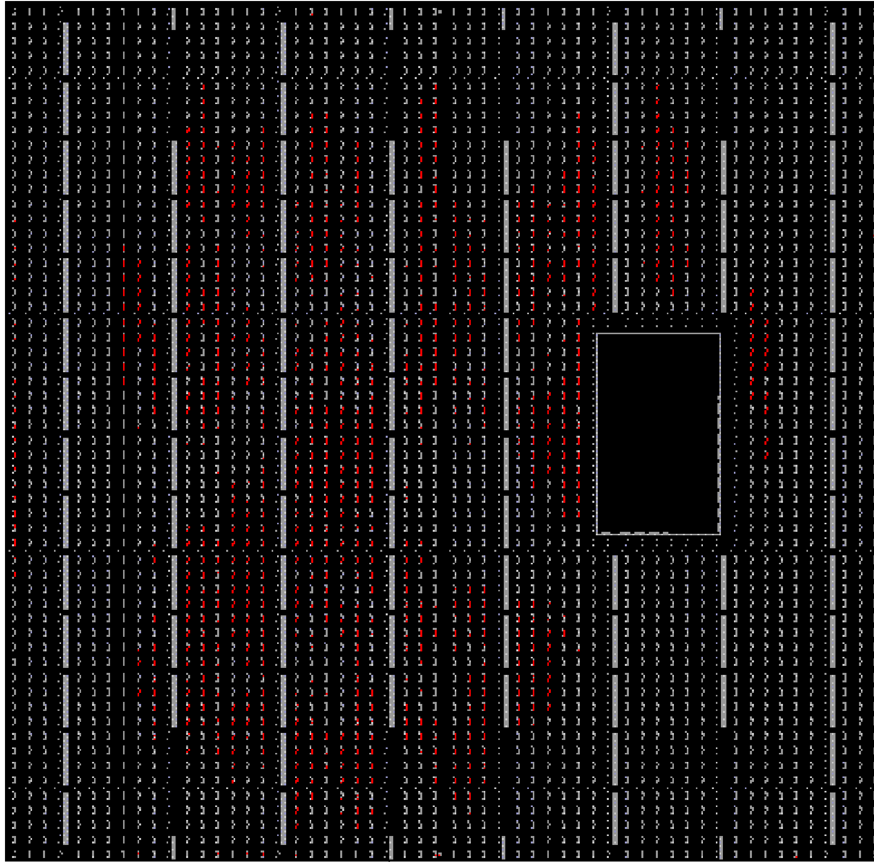
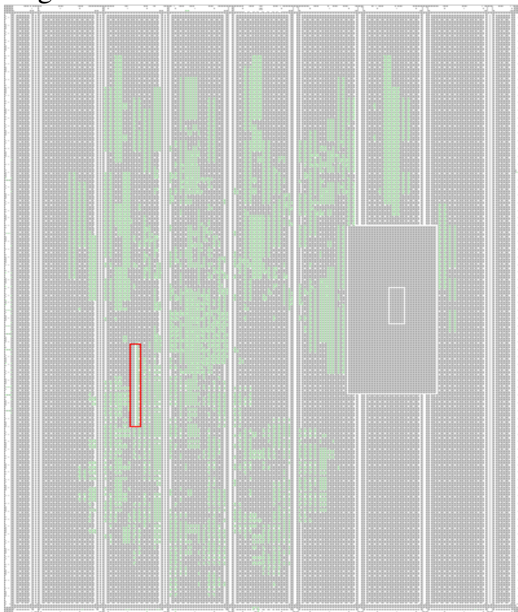


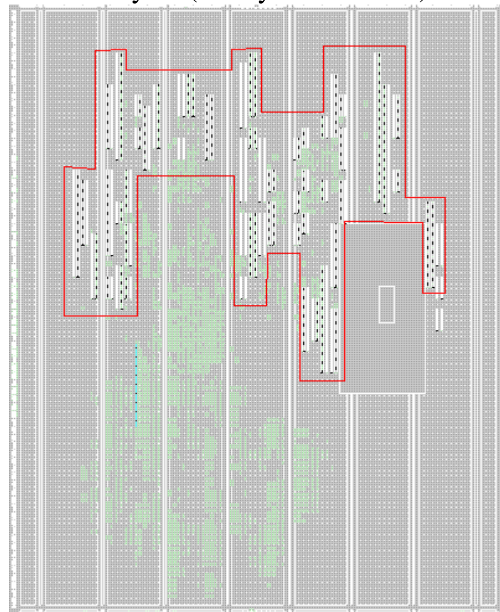
Figura 4.2.3.7 Área placa Versión Secuencial

Si analizamos la placa por partes podemos ver donde esta situada cada entidad:

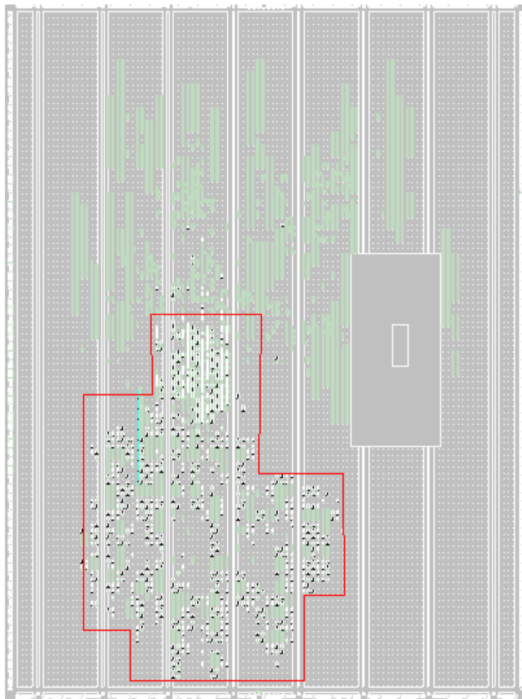
Lógica de control:



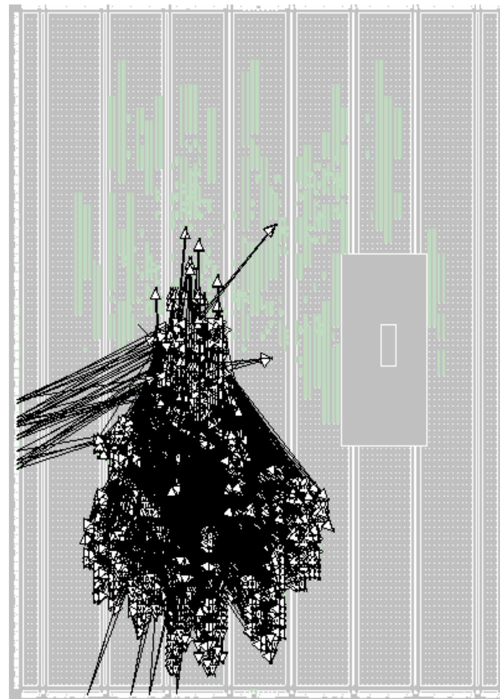
SustituirBytes (incluyendo SBOX):



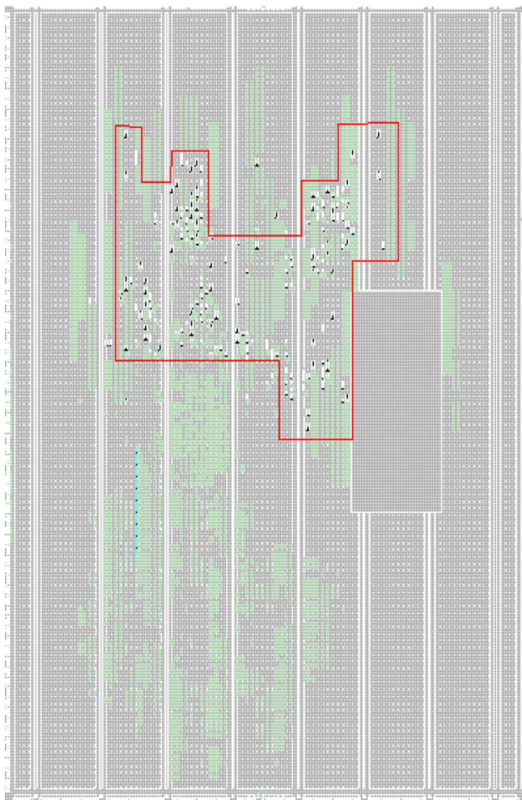
ExtenderClave:



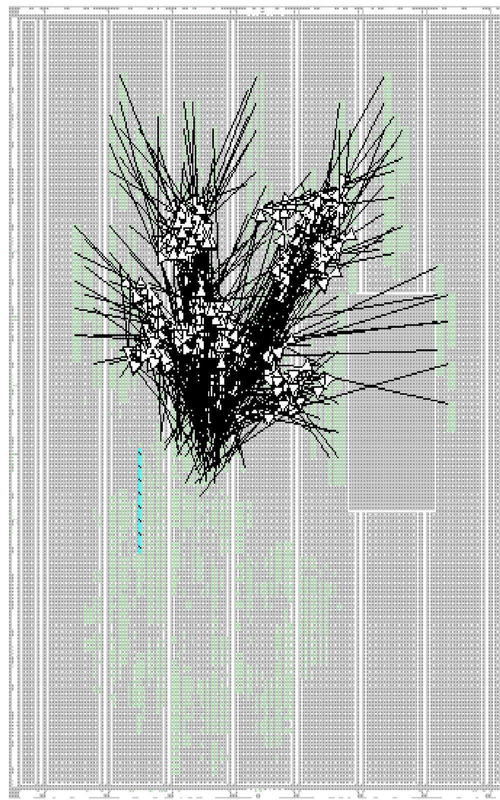
Conexiones del componente ExtenderClave



MezclarColumnas:



Conexiones



Añadir Claves:

Conexiones

Figura 4.2.3.8 Área de la placa ocupado por las entidades

Como se ha podido ver en las capturas (Figura 4.2.3.8), la herramienta sintetiza y sitúa las entidades en áreas cercanas y las conexiones se realizan con zonas próximas.
Las entidades que ocupan más área son ExtenderClave y SustituirByte, debido a que estas entidades utilizan las SBox. La entidad BarajarFilas no aparece localizada, ya que por su definición sólo es un entrelazado de señales.

El menor ciclo de reloj obtenido para esta versión ha sido 12 ns. Este ciclo de reloj es un dato importante que nos permitirá saber cuanto tiempo va invertir nuestra aplicación en proporcionar la salida deseada.

/tases_vhd/clk 1
/tases_vhd/reset 0
/tases_vhd/entrada 000000000 0000000000111111
/tases_vhd/clave 000000000 0000000000000101
/tases_vhd/salida 35EF 59D5 2516 E5E6 5... 15EF 35EF
/tases_vhd/fin 1
/glbl/GSR We0
Now 00000 ps 244 ns 248 ns 252 ns
Cursor 1 50265 ps 250265 ps

Figura 4.2.3.9 Ejemplo de Simulación Versión Secuencial

Como se puede ver en la Figura 4.2.3.9 el valor correcto se consigue después de 250 ns. El inicio del ciclo actual acaba a los 252ns, si le restamos los 120ns iniciales en los que está activo la señal de reset nos quedan 132 ns. Con esto queda claro que el número de ciclos necesarios para la ejecución del AES es 11.

Comparación Combinacional-Secuencial.

Una vez que hemos obtenido el tiempo de ciclo pasamos a comparar los resultados de ambas versiones. Para ello vamos a utilizar la misma FPGA, Virtex2P xc2vp50.
Esta FPGA ha sido la que nos ha proporcionado mejores resultados para la versión combinacional.

	Área ocupada	Tiempo necesario para obtener el resultado correcto	Number of Slices	Number of 4 input LUTs	Number of bonded IOBs
Combinacional	61	118.5 ns	13580	26622	48
Secuencial	8	150 ns	2099	3608	50

Tabla 4.2.3.10 Comparación Combinacional-Secuecial

El área ocupada por la versión combinacional es 7,5 veces mayor, lo que nos obliga a utilizar una FPGA bastante más grande, es tanta la diferencia, que como veremos posteriormente, en el área ocupada por esta versión podrían caber más de 3 sistemas completos como el que estamos desarrollando.
En cuanto al tiempo necesario la versión combinacional supone una mejora de 1,27. Es una mejora importante, pero no tanto como para asumir un tamaño tan grande.

4.2.4. Módulo Generador/Encriptador

4.2.4.1. Ruta de Datos

Una vez explicado el funcionamiento del AES pasamos a ver el resto de la ruta de datos. Este módulo estará formado por una ruta de datos y un controlador. La ruta de datos además de contener el componente principal que sería el módulo AES, está compuesta por una serie de registros, un componente XOR, multiplexores y una FIFO. Todos estás estructuras son necesarias para la generación de claves utilizando el algoritmo ANSI X09.17 y su posterior almacenamiento.

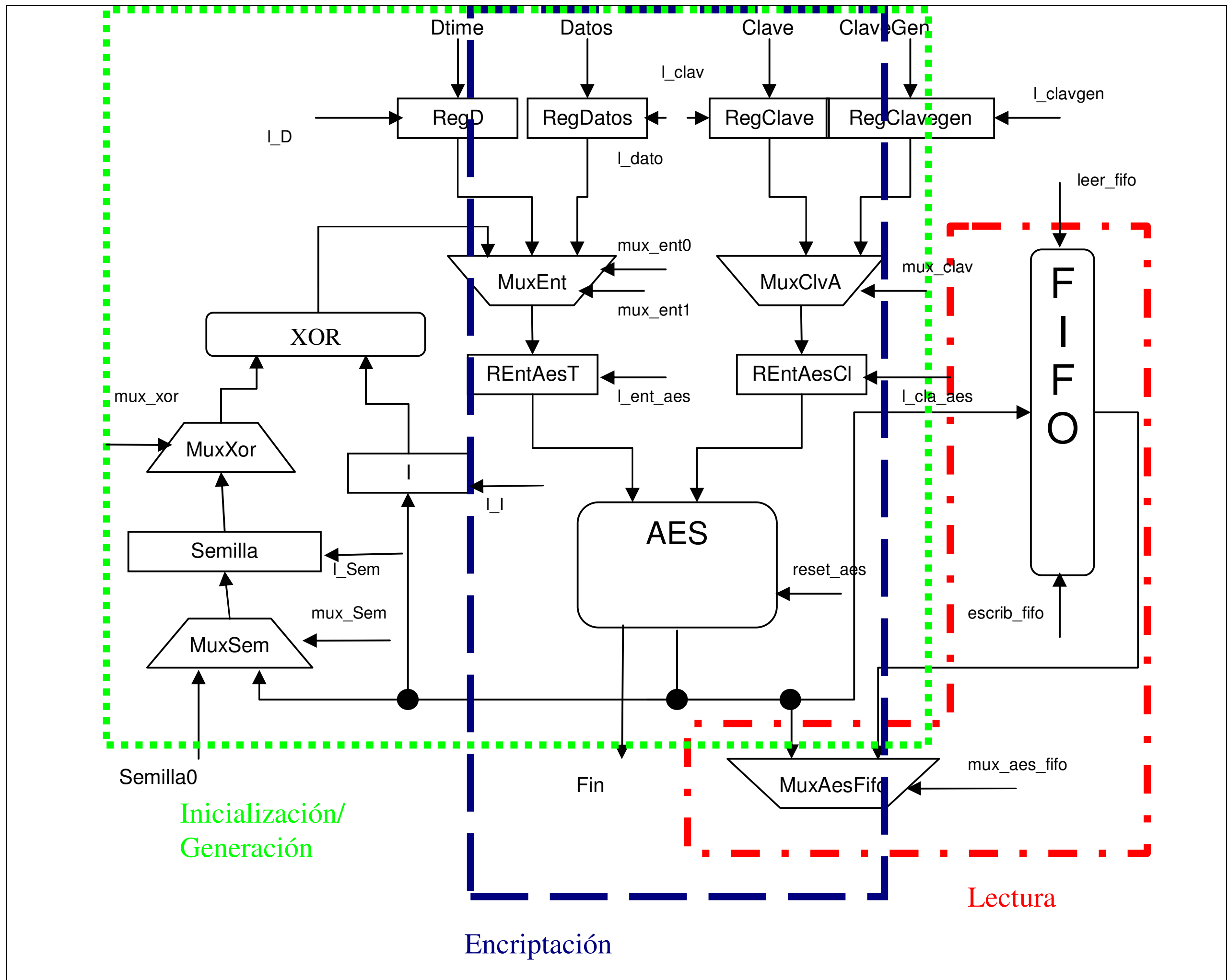


Figura 4.2.4.1 Ruta datos Generador/Encriptador

4.2.4.2. Operaciones

Si dividimos los elementos según en función del Encriptador/Generador en la actúan, la distribución sería la siguiente:

-. Inicialización:

Señales: Dtime, Semilla0, Clavegen, fin.
Registros: RegD, RegClavegen, REntAesTex, REntAesClv, I, Semilla.
Multiplexores: MuxEnt, MuxClv, MuxSem, MuxXor.
AES.

-. Generación:

Señales: fin.
Registros: RegClavegen, REntAesTex, REntAesClv, I, Semilla.
Multiplexores: MuxEnt, MuxClv, MuxSem, MuxXor.
AES.
FIFO.

-. Encriptación:

Señales: fin, Datos, Clave.
Registros: RegDatos, RegClv, REntAesTex, REntAesClv.
Multiplexores: MuxEnt, MuxClv, MuxAesFifo.

AES.

-. Lectura:

Multiplexores: MuxAesFifo.

FIFO.

Lectura

En la operación de lectura sólo sería necesaria la activación de la señal de lectura de la FIFO y del Multiplexor MuxAesFifo, suponiendo que haya claves guardadas y que han sido generadas anteriormente.

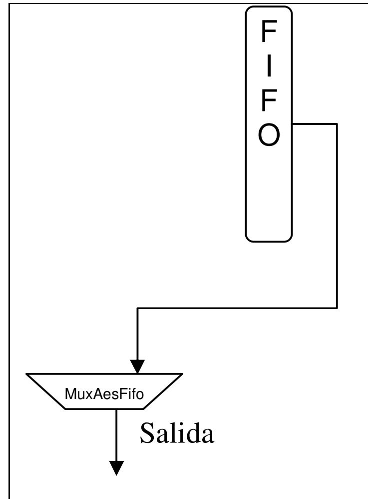


Figura 4.2.4.2 Instrucción Lectura

Encriptación

Para la encriptación tenemos que cargar el texto que queremos encriptar (Datos) y la clave (Clave) en RegDatos y RegClv, pasarle estos datos al AES, a través de MuxEntAes, REntAesText, MuxClvAes y REntAesClv, y una vez encriptado, activaremos el multiplexor de la salida (MuxAesFifo).

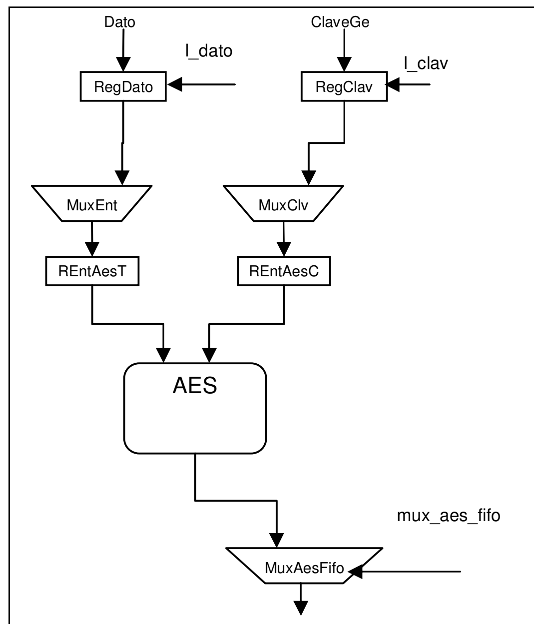


Figura 4.2.4.3 Instrucción Encriptación

Inicialización y Generación
La inicialización y generación de nuevas claves implementan el algoritmo X9.17.
En nuestro caso el algoritmo de cifrado de bloques que utilizamos es el AES, una versión aplicada a nuestro caso sería la siguiente:
K* = claves generadas.
C = Clave de generación.
AES(E,C) = encriptación de E con C.
S = Semilla.
DT = date-time (timestamp).

1. I = AES(DT, C). Inicialización
2. R = AES(I XOR S, C) = Nueva clave generada.
3. S = AES(R XOR I, C) = Nueva Semilla.
4. Ir al paso 2.

El paso 1 sería equivalente a la operación de inicialización, los pasos 2, 3 y 4 comprenden la generación.

La fase de inicialización cargaremos los valores de la semilla generada aleatoriamente, la clave de encriptación y el DateTime o información del la fecha. El DateTime lo encriptamos con la clave de encriptación y lo guardamos en el registro I.

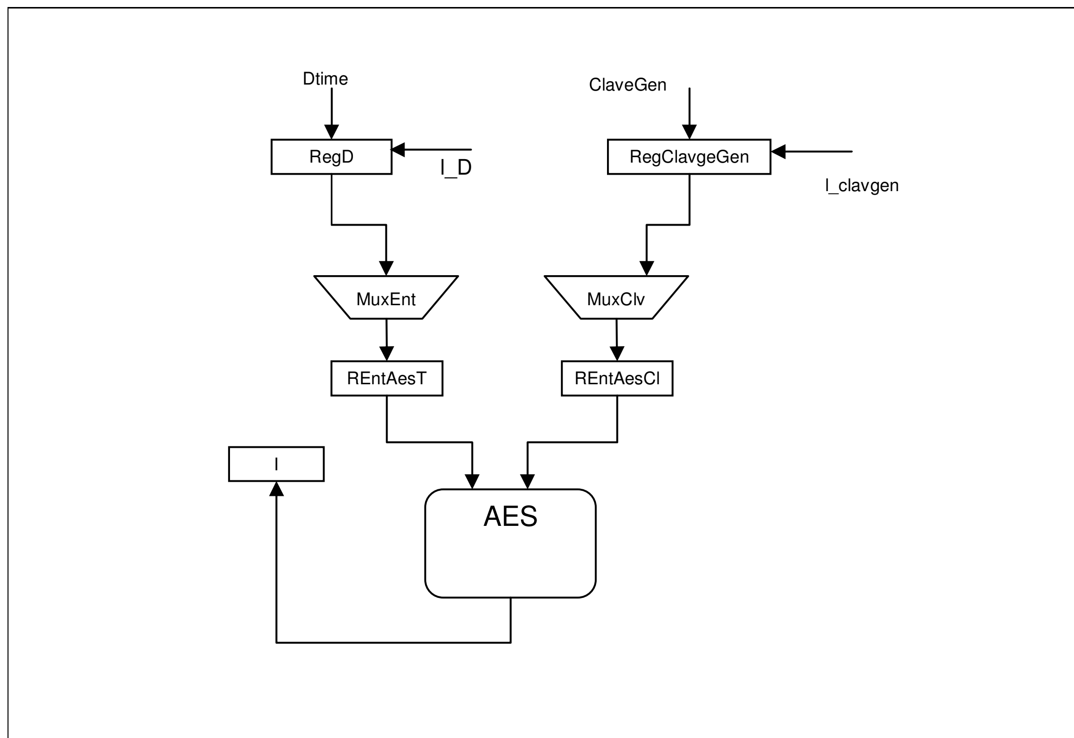


Figura 4.2.4.4 Instrucción Inicialización

En la fase de generación engloba dos fases a su vez:
1ª.- Generación de una nueva clave.

Para ello primero se realiza una XOR del contenido del registro I y del registro Semilla, posteriormente se realiza la encriptación del resultado con la clave de encriptación, para después guardar esa nueva clave en la FIFO.

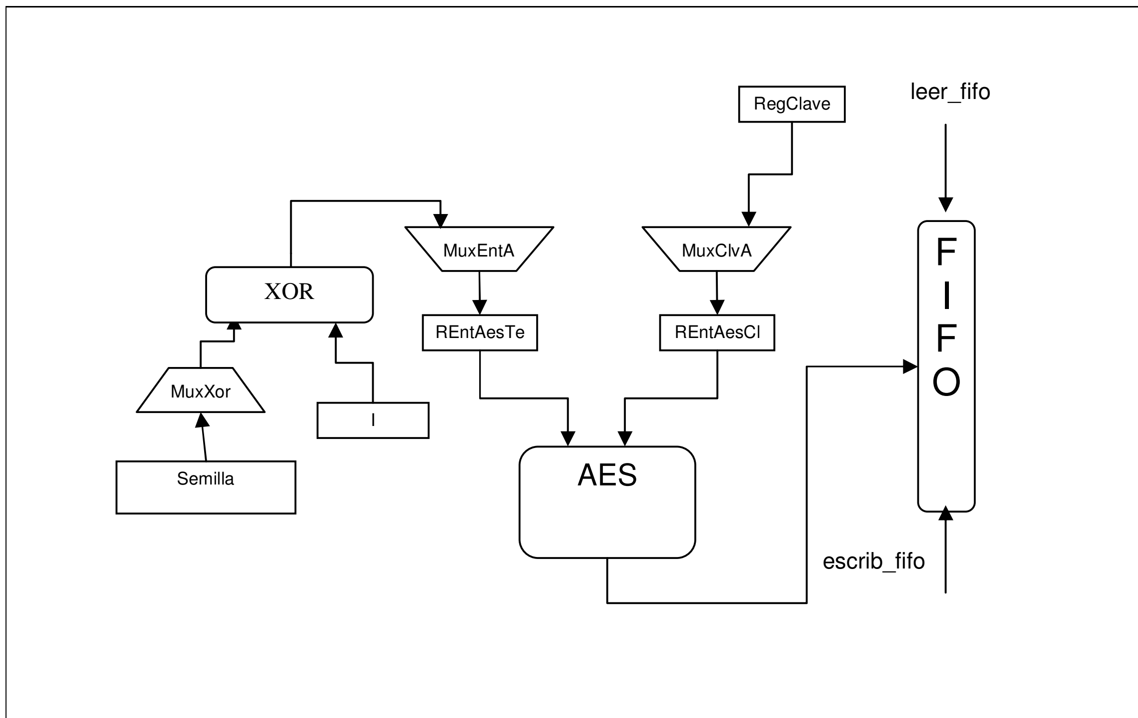


Figura 4.2.4.4 Instrucción Generación1

2ª.- Generación de una nueva semilla.

En este caso se hace la XOR de la nueva clave generada y del contenido del registro I, posteriormente se encripta con la clave de encriptación y se guarda en el registro Semilla.

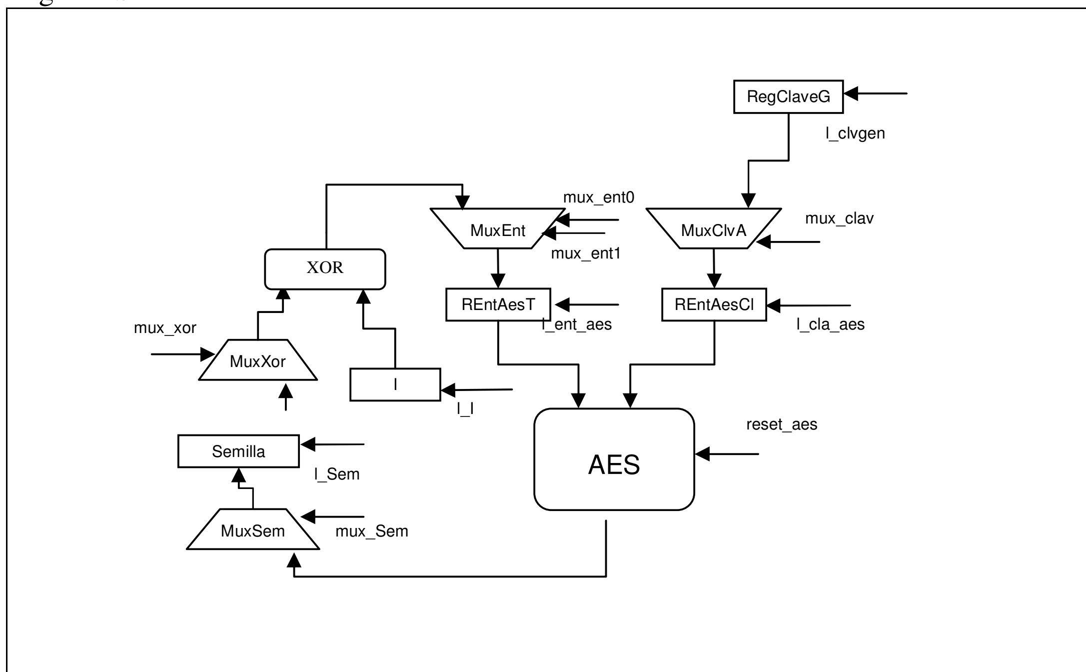


Figura 4.2.4.6 Instrucción Generación2

4.2.4.3. Controlador

El controlador se encarga de gestionar y activar las señales correspondientes de la ruta de datos dependiendo de la instrucción que se le pida que realice.
Para gestionarlo hemos creado un tipo de datos, un array de señales:

```
TYPE senales_controlAES IS (l_D,l_datos,l_clav,l_clavgen,l_I,l_Seem,l_ent_aes,
       l_cla_aes,mux_ent0,mux_ent1,mux_clav,mux_xor,mux_Sem,
       leer_fifo,escrib_fifo,reset_aes,mux_aesfifo);

TYPE vector_senales_controlAES IS ARRAY (natural RANGE <>) OF senales_controlAES;

TYPE bus_controlAES IS ARRAY (senales_controlAES) OF bit;
```

Tabla 4.2.4.7 Tipos Controlador.

Debido a la complejidad de las instrucciones ha habido que dividirlas en varios pasos:

Inicialización

paso	l_D	l_datos	l_clav	l_clavgen	l_I	l_Seem	l_ent_aes	l_cla_aes	mux_aesfifo
00	1	0	0	1	0	1	0	0	0
01	0	0	0	0	0	0	1	1	0
10	0	0	0	0	1	0	0	0	0

paso	mux_ent0	mux_ent1	mux_clav	mux_xor	mux_Sem	leer_fifo	escrib_fifo	reset_aes
00	0	0	0	0	0	0	0	0
01	0	1	1	0	0	0	0	1
10	0	0	0	0	0	0	0	0

Tabla 4.2.4.8 Señales Inicialización.

```
if paso="00" then
     control<=ctrl(l_d&l_seem&l_clavgen  );
     paso<="01";
elsif paso="01" then
     control<=ctrl(mux_ent1&mux_clav&l_ent_aes&l_cla_aes&reset_AES);
      paso<="10";
elsif fin='1' then
      control <= ctrl(l_I);
      paso<="00";
      final<='1';
```

Tabla 4.2.4.9 Código Vhdl Señales Inicialización.

La señal "fin" la activa el AES cuando termina de encriptar. Para indicar el final de la instrucción activamos "final".

Generación

paso	l_D	l_datos	l_clav	l_clavgen	l_I	l_Seem	l_ent_aes	l_cla_aes	reset_aes	mux_aesfifo
00	0	0	0	0	0	0	1	1	1	0
01	0	0	0	0	0	0	1	1	1	0
10	0	0	0	0	0	1	0	0	0	0

paso	mux_ent0	mux_ent1	mux_clav	mux_xor	mux_Sem	leer_fifo	escrib_fifo
00	0	0	1	0	0	0	0
01	0	0	1	1	0	0	1
10	0	0	0	0	1	0	0

Tabla 4.2.4.10 Señales Generación.

```
if (paso="00") then
    control <= ctrl(mux_clav&l_ent_aes&l_cla_aes&reset_AES  );
    paso<="01";
elsif paso="01" and fin='1' then
    control <=
    ctrl(mux_xor&mux_clav&l_ent_aes&l_cla_aes&escrib_fifo&reset_aes);
    paso<="10";
elsif fin='1' then
    control <= ctrl(mux_sem & l_seem);
    paso<="00";
    final<='1';
end if;
```

Tabla 4.2.4.11 Código Vhdl Señales Generación.

Encriptación

paso	l_D	l_datos	l_clav	l_clavgen	l_I	l_Seem	l_ent_aes	l_cla_aes	mux_ent0
00	0	1	1	0	0	0	0	0	0
01	0	0	0	0	0	0	1	1	1
10	0	0	0	0	0	0	0	0	0

paso	mux_ent1	mux_clav	mux_xor	mux_Sem	escrib_fifo	reset_aes	mux_aesfifo	leer_fifo
00	0	0	0	0	0	0	0	0
01	0	1	0	0	0	1	0	0
10	0	0	0	0	0	0	0	0

Tabla 4.2.4.12 Señales Encriptación.

```
if (paso="00") then
      control <= ctrl(l_datos & l_clav   );
      paso<="01";
elsif (paso="01") then
     control <= ctrl(mux_ent0 & l_ent_aes & l_cla_aes & reset_AES);
                              paso<="10";
elsif fin='1' then
     final<='1';
     paso<="00";
end if;
```

Tabla 4.2.4.13 Código Vhdl Señales Encriptación.

Lectura

Si la FIFO está vacía generamos una nueva clave de la misma forma que antes, si no está vacía haríamos lo siguiente:

paso	l_D	l_datos	l_clav	l_clavgen	l_I	l_Seem	l_ent_aes	l_cla_aes	mux_ent0	mux_aesfifo
00	0	1	1	0	0	0	0	0	0	0
01	0	0	0	0	0	0	0	0	0	1

paso	mux_ent1	mux_clav	mux_xor	mux_Sem	leer_fifo	escrib_fifo	reset_aes
00	0	0	0	0	1	0	0
01	0	0	0	0	0	0	0

Tabla 4.2.4.15 Señales Lectura.

```
 if fin='1' then
       control <= ctrl(mux_sem & l_seem);
       paso<="00";
 end if;
 if fifo_vacia='1' then
if (paso="00") then
             control <= ctrl(mux_clav&l_ent_aes&l_cla_aes&reset_AES);
             paso<="01";
      elsif paso="01" and fin='1' then
             control <=
             ctrl(mux_xor&mux_clav&l_ent_aes&l_cla_aes&escrib_fifo&reset_aes);
             paso<="10";
      end if;
else
      if (paso="00") then
             control <= ctrl(leer_fifo);
             final<='1';
             paso<="01";
     elsif (paso="01") then
             control <= ctrl(mux_aesfifo);
             paso<="00";
     end if;
 end if;
```

Tabla 4.2.4.16 Código Vhdl Señales Lectura.

Son necesarios por tanto:

Inicialización 13 ciclos (1 ciclo para la lectura de datos, 11 necesarios para la encriptación y 1 para la escritura del registro I).

Generación 24 ciclos (1 ciclo necesario para la carga de datos, 11 para la primera encriptación, 1 ciclo para cargar de nuevo los datos y 11 para la segunda encriptación).

Encriptación 13 ciclos (1 ciclo para la lectura de datos, 11 necesarios para la encriptación y 1 para la escritura del registro).

Lectura 2 ciclos, en caso de haber claves disponibles (1 ciclo de activación de la lectura de la FIFO y 1 para la carga del registro. Se realiza en dos ciclos, debido a que se comprueba primero que la FIFO no se encuentra vacía).

Una primera versión del generador/encriptador tendría el aspecto de la figura 4.2.4.14.

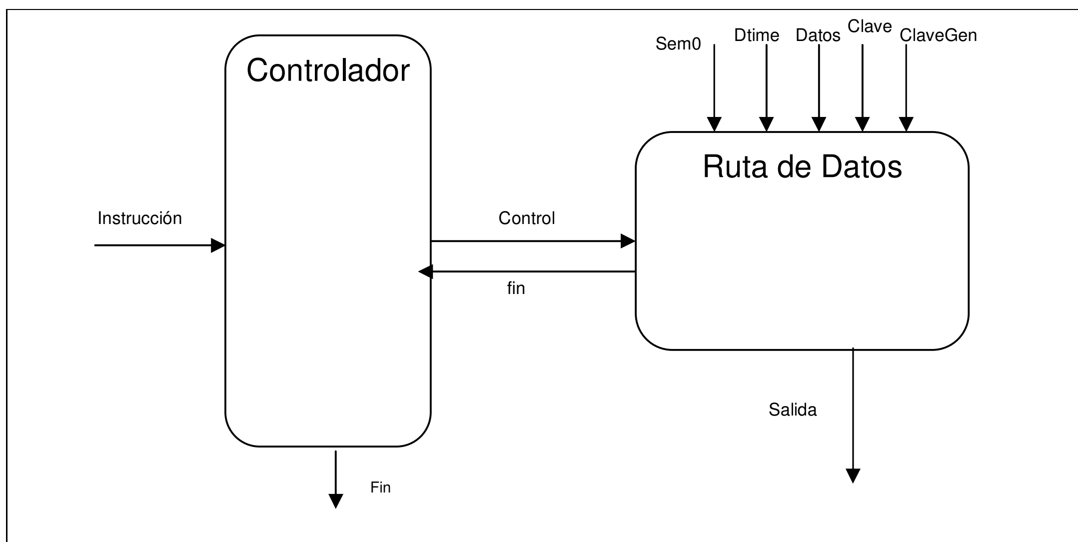


Figura 4.2.4.17 Generador/Encriptador

4.2.5. Resultados del módulo Generador/Encriptador

Utilizando la misma FPGA sobre la que sintetizamos el módulo AES para esta versión del módulo Generador/Encriptador se obtienen los siguientes resultados:
Espacio utilizado de la placa 37%. Como se puede ver en el report y recordando los datos de la síntesis obtenidos para el componente AES (2099 Slices), este módulo ocupa aproximadamente el doble que el AES (4289 Slices), es decir el, resto de la ruta de datos más el controlador ocupa el mismo área que el AES.

```
Found area constraint ratio of 100 (+ 5) on block genencryp, actualratio
is 37.
Final Results
RTL Top Level Output File Name     : genencryp.ngr
Top Level Output File Name         : genencryp
Output Format                      : NGC
Optimization Goal                  : Speed
Keep Hierarchy                     : NO

Design Statistics
# IOs                              : 100

Macro Statistics :
# ROMs                             : 20
#      256x8-bit ROM               : 20
# Registers                        : 453
#      1-bit register              : 129
#      17-bit register             : 1
#      4-bit register              : 3
#      8-bit register              : 320
# Multiplexers                     : 103
#      1-bit 4-to-1 multiplexer    : 4
#      17-bit 4-to-1 multiplexer   : 2
#      8-bit 12-to-1 multiplexer   : 1
#      8-bit 4-to-1 multiplexer    : 96
# Adders/Subtractors               : 2
#      5-bit subtractor            : 1
#      6-bit adder                 : 1
# Comparators                      : 1
#      4-bit comparator equal      : 1
# Xors                             : 33
#      1-bit xor8                  : 16
#      8-bit xor3                  : 1
#      8-bit xor4                  : 16

Cell Usage :
# BELS                             : 11373
#      GND                         : 1
#      INV                         : 3
#      LUT1                        : 32
#      LUT2                        : 171
#      LUT2_L                      : 133
#      LUT3                        : 264
#      LUT3_D                      : 30
#      LUT3_L                      : 1358
#      LUT4                        : 1731
#      LUT4_D                      : 225
#      LUT4_L                      : 4230
#      MUXCY                       : 35
#      MUXF5                       : 2035
```

```
#       MUXF6                         : 640
#       MUXF7                         : 320
#       MUXF8                         : 128
#       VCC                           : 1
#       XORCY                         : 36
# FlipFlops/Latches                   : 3080
#       FD                            : 7
#       FDCE                          : 32
#       FDE                           : 3027
#       FDR                           : 14
# Clock Buffers                       : 1
#       BUFGP                         : 1
# IO Buffers                          : 99
#       IBUF                          : 82
#       OBUF                          : 17
=========================================================================

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

Selected Device : 2vpx20ff896-6

 Number of Slices:                     4289  out of   9792    43%
 Number of Slice Flip Flops:           3080  out of  19584    15%
 Number of 4 input LUTs:               8174  out of  19584    41%
 Number of bonded IOBs:                 100  out of    556    17%
 Number of GCLKs:                         1  out of     16     6%


=========================================================================
```

Tabla 4.2.5.1 Informe Generador/Encriptador

En cuanto al reloj, ha sido necesario aumentarlo 8ns. Ya que no era suficiente para la carga de los registros, situándose en 20ns. Por tanto queda marcado el camino crítico por la carga de estos registros.

```
# ** Error: C:/DOCUME~1/ADMINI~1/CONFIG~1/Temp/xil_2212_6(6549):
$hold( posedge CLK:155100 ps, posedge I &&& (in_clk_enable ==
1):155126 ps, 192 ps );
#    Time: 155126 ps  Iteration: 0  Instance:
/tgen_vhd/uut/\rut/opaes/registroClave/tmp_3_1_16876\
/tgen_vhd/uut/\rut/opaes/registroTexto/tmp_0_2_11_17641\
```

Tabla 4.2.5.2 Informe Error Model Sim

4.2.6. Mejoras del Módulo Generador/Encriptador

Transformación de la FIFO.

En la versión anterior la FIFO donde se guardaban las claves generadas no era sintetizada como una memoria RAM. Además ocupaba una gran parte de la placa como se ve a continuación. Todo el espacio marcado en rojo lo ocupaba la FIFO. Esto suponía una carga bastante grande e innecesaria, ya que existe la posibilidad de utilizar los recursos de la placa concebidos para este uso.

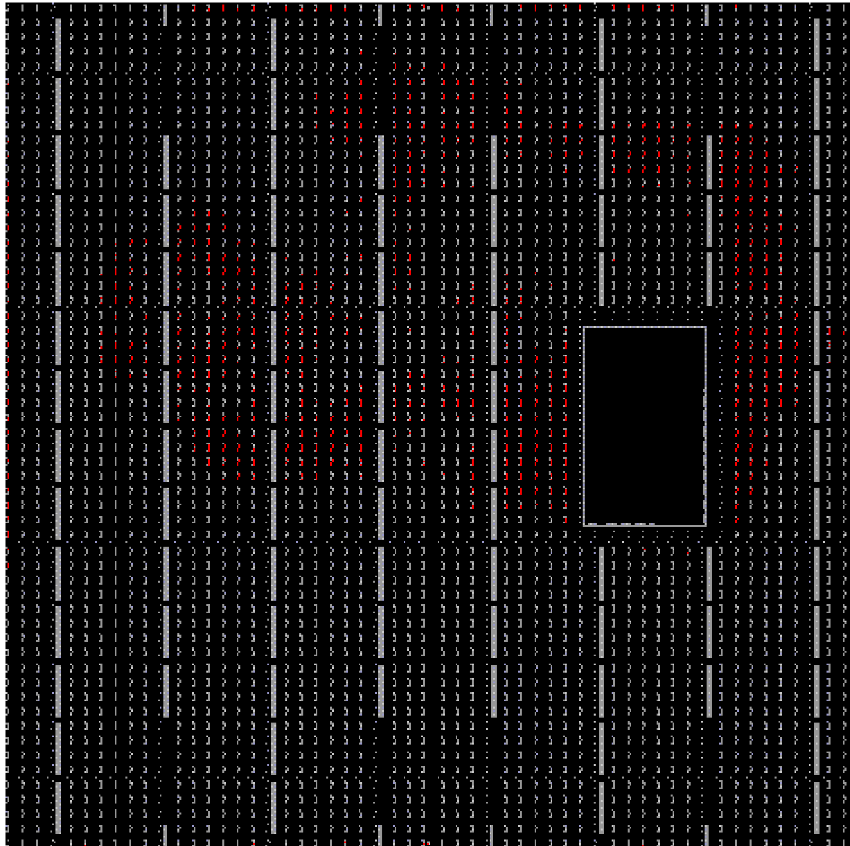


Tabla 4.2.6.1Área FIFO

Para esta implementación el número de conexiones era bastante elevado.

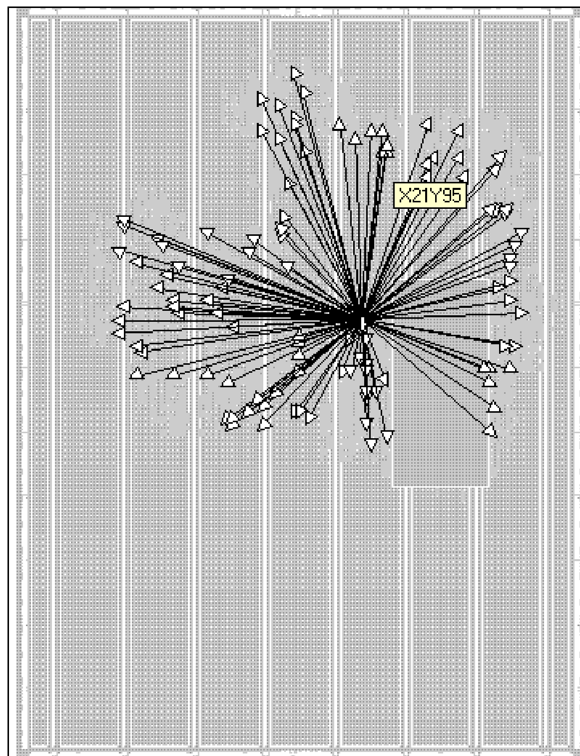


Figura 4.2.6.2 Conexiones FIFO

En una versión posterior hemos logrado este paso, es decir, hacer que se sinteticen en las memorias de la placa. El problema ocurría por dos motivos:
-. La lógica de control que manejaba los punteros de la memoria.
-. El tipo de datos guardados en la memoria.
Para resolverlo, en el primer caso lo solucionamos simplificando la lógica de control, utilizando una sola instrucción de control “if”. Para el segundo hemos tenido que transforma el tipo de datos que guardamos en la FIFO, si antes el tipo que teníamos era el tipo word, ahora será un vector de de 128 bits (std_logic_vector (0 to 128)). Para esto utilizamos dos funciones simples de transformación.

```
FUNCTION vector_word (v:std_logic_vector)RETURN word IS
VARIABLE w:word;VARIABLE b:byte;
BEGIN
            for i IN 0 TO 15 LOOP
                  b:=v(i*8+7 DOWNTO i*8);
                  w(15-i):=b;
            END LOOP;
            RETURN w;
END vector_word;
FUNCTION word_vector(v:word)RETURN std_logic_vector Is
VARIABLE w:std_logic_vector(127 downto 0);
BEGIN

      w:=v(0)&v(1)&v(2)&v(3)&v(4)&v(5)&v(6)&v(7)&v(8)&v(9)&
                  v(10)&v(11)&v(12)&v(13)&v(14)&v(15);
            RETURN w;
END word_vector;
```

Tabla 4.2.6.3 Funciones de Conversión.

Una vez transformada la FIFO, los resultados obtenidos son los siguientes:
-. La FIFO ocupaba aproximadamente un 14% - 15% de la placa en la versión anterior ahora se implementa utilizando los recursos específicos de memoria (BLOCKRAMS) disponibles dentro de la placa. Por lo tanto el área ocupada por el generador/encriptador es ahora 25% frente al 37% anterior. El 2% restante lo ocupa la lógica de control, es decir, los punteros de dirección y las nuevas conexiones.
-. Para la nueva versión no hemos tenido que modificar el reloj.
Dado que el sistema completo cuenta con tres FIFOs, una de ellas incluso más pesada que la analizada anteriormente, no tenemos elección, y es por tanto necesario optar por la implementación de las FIFOs como memoria RAM.

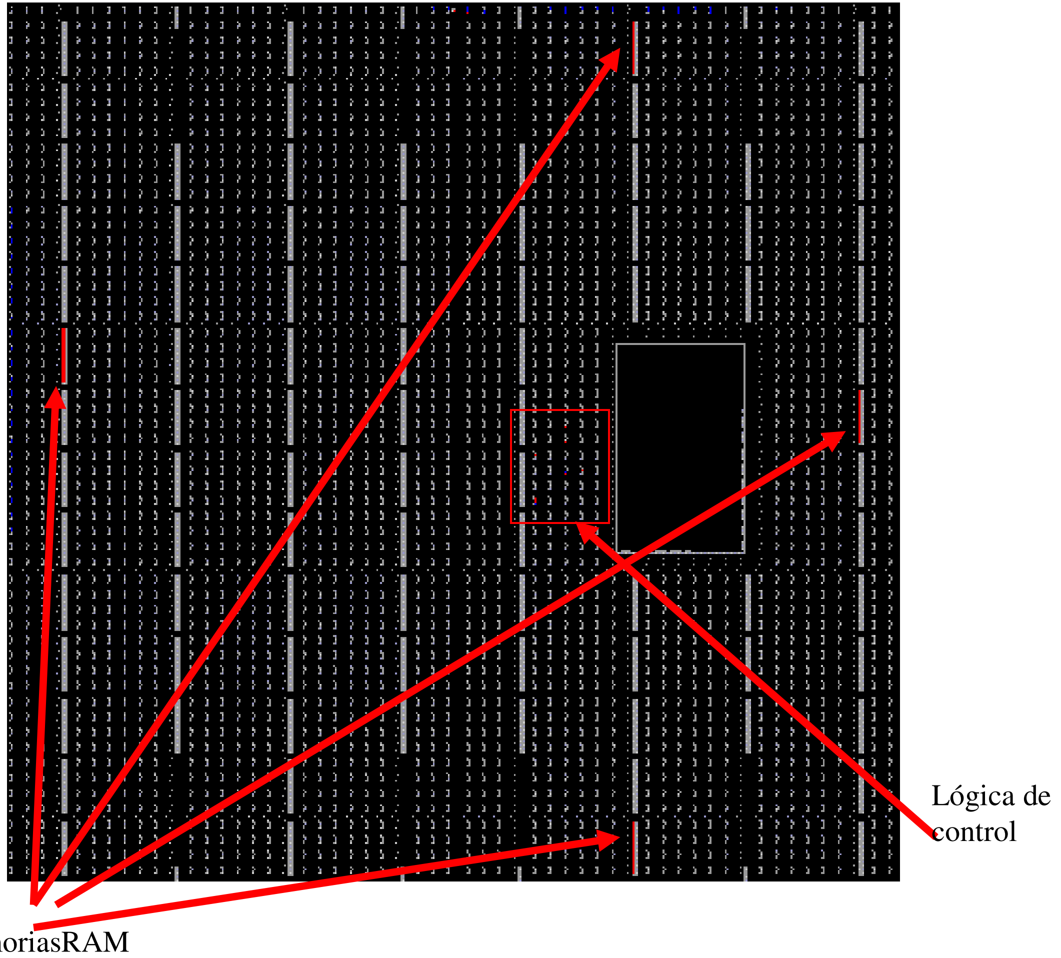


Tabla 4.2.6.4 Área FIFO Mejorada.

Las conexiones serían:

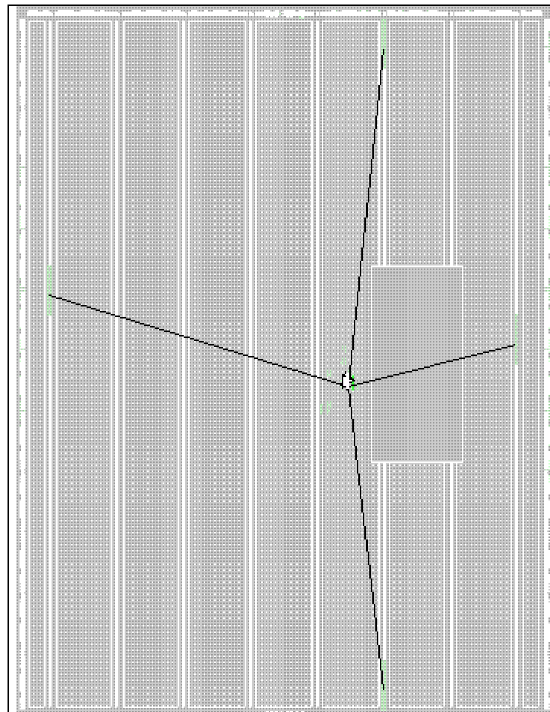


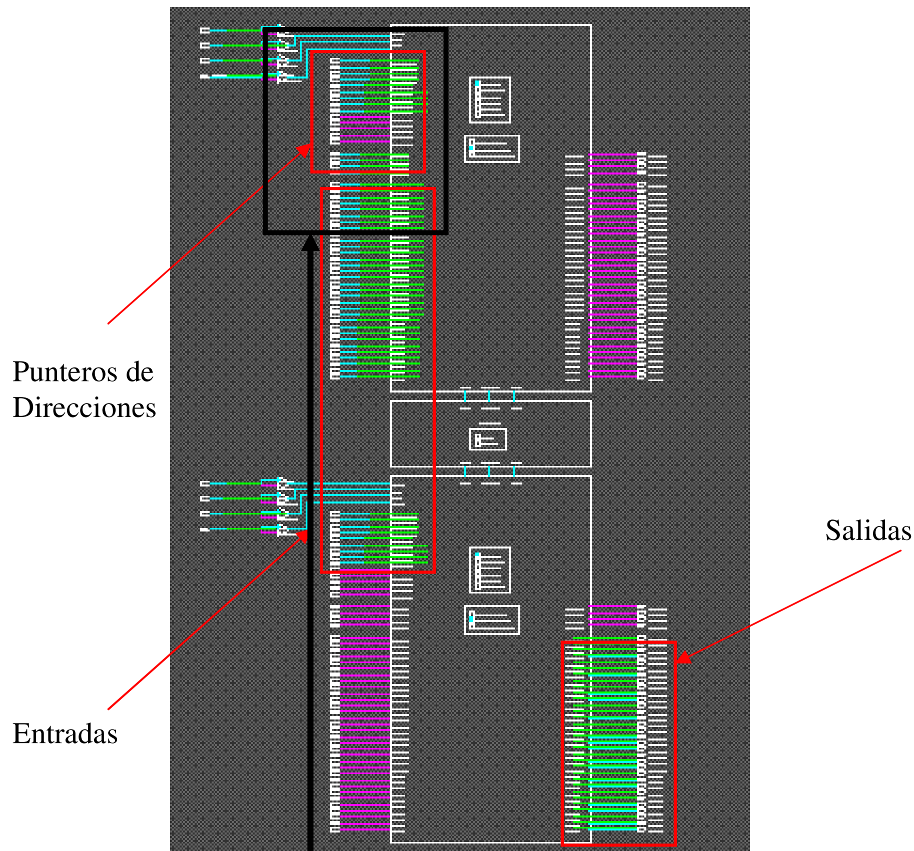
Figura 4.2.6.5 Conexiones FIFO Mejorada

Ahora el número de conexiones es enormemente menor.
Si nos fijamos en la síntesis que realiza el Xilinx se puede ver que se corresponde con nuestra implementación:
Si nos centramos en uno de los punteros de la FIFO "contadorE".

```
ENTITY FIFO10wordaes IS
PORT (entrada:IN word; WF,lec:IN bit;clk: IN bit; salida: out
word;vacia: out bit);
END FIFO10wordaes;
ARCHITECTURE estructural OF FIFO10wordaes IS
signal mem:arrayfifo;
signal contadorE,contadorS : natural range 0 to tamanofifo:=0;
begin
 process (clk,WF,lec)
       begin
              if clk='1' and clk'event then
                     if WF = '1' then
                     -- si no esta llena
                            if (tamanofifo - (contadorE - contadorS+ 1
)) /= 0 and (tamanofifo - (contadorE - contadorS+ 1 )) /= tamanofifo
then
                     mem(contadorE) <= word_vector(entrada);
                     contadorE <= (contadorE + 1) mod 10;
              end if;
                     end if;
                     if lec = '1' then
              -- si no esta vacia
                            if contadorS/=contadorE then
                                   salida<=vector_word(mem(ContadorS));

                                   contadorS <= (contadorS + 1)mod 10;
                            end if;
                     end if;
              end if;
       end process;
vacia <= '1' when contadorE = contadorS
       else '0';
end estructural;
```

Tabla 4.2.6.6 Código Vhdl FIFO Mejorada.



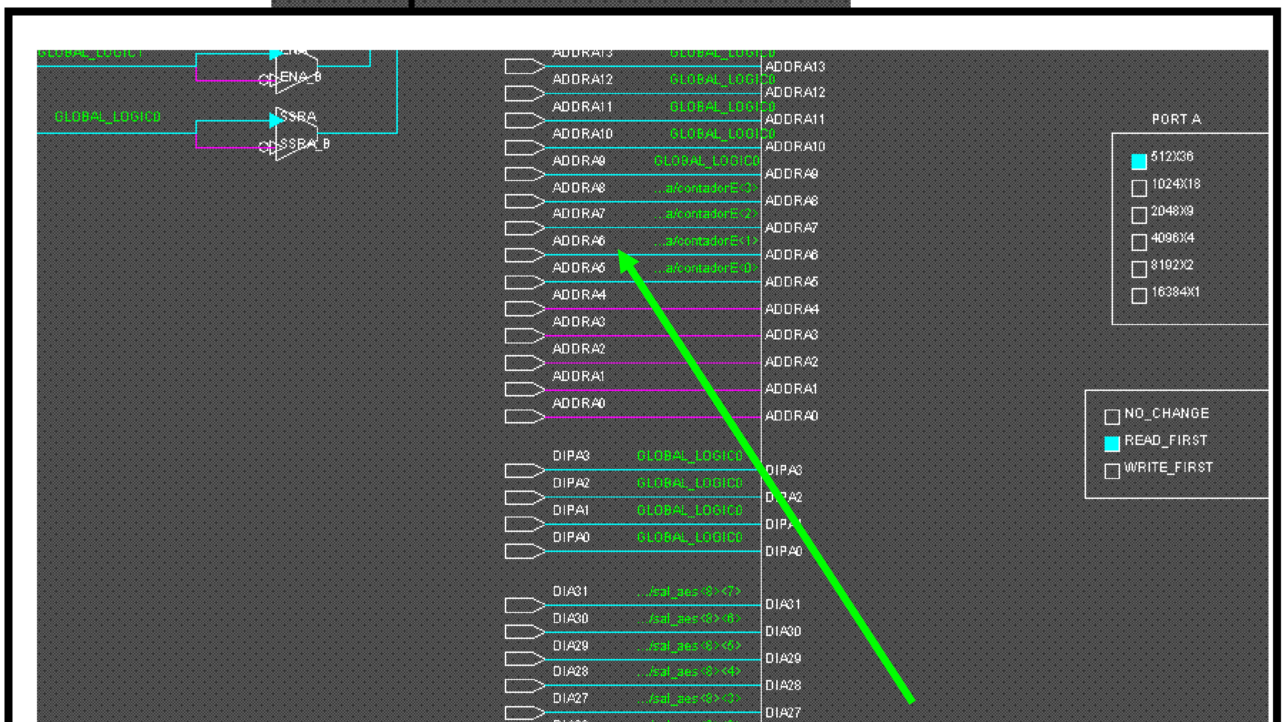


Figura 4.2.6.7 Interior FIFO Mejorada.

Se puede ver que la herramienta de síntesis utiliza esa misma señal "ContadorE" para seleccionar la dirección de memoria.

4.3. Procesador de direcciones

Esta unidad lee la dirección de una clave y la procesa para generar la clave del padre, del hijo y del hermano.

Recibe como entradas la dirección “dir” y una señal de control DirOp por la cual se calcula la dirección que se quiere de la siguiente forma:

Si DirOp = 00 (hijo izquierdo)
dir = desplazamientoDer(dir) + 0
Si DirOp = 01 (hijo derecho)
dir = desplazamientoIzq(dir) + 1
Si DirOp = 10 (dirección padre)
dir = 1 + desplazamientoIzq(dir)
Si DirOp = 11 (dejamos pasar)
dir = dir
Donde + es concatenación

Tabla 4.3.1 Módulo Procesador Direcciones.

Por la estructura de árbol podemos observar fácilmente que para calcular los hijos de un nodo basta con hacer un desplazamiento a la izquierda y concatenarle un 0 si es el izquierdo o un 1 si es el derecho.
Para hallar el padre se concatena a un 1 un desplazamiento a la derecha del hijo en cuestión.

4.4. El procesador de LR

EL módulo procesador LR recibe como entradas un código LR (2 bit) y una señal de control LRPC.
LRPC es una señal de 2 bit donde el más significativo indica si se une o desune el hijo (uso o desuso) y el menos significativo corresponde al bit menos significativo de la dirección actual, para saber si el hijo es derecho o izquierdo.

El módulo sigue la siguiente lógica:

Si LRPC = 00 (la desunión viene del hijo izquierdo) LR(1) = 0 (0x)
Si LRPC = 01 (la desunión viene del hijo derecho)
LR(0) = 0 (x0)
Si LRPC = 10 (la unión viene del hijo izquierdo)
LR(1) = 1 (1x)
Si LRPC = 11 (la unión viene del hijo derecho)
LR(0) = 1 (x1)

Tabla 4.4.1 Módulo Procesador LR

De donde obtendremos que:

LR(¬LRPC(0)) = LRPC(1)

4.5. Controlador

4.5.1. Funcionamiento

El controlador permanentemente comprueba que nuevas instrucciones que hay en la cola. Si encuentra una, el controlador la coge, la decodifica y controla su ejecución. Finalmente el resultado se introduce en la cola de salida.

El controlador tiene tres niveles:

- El primer nivel es el controlador principal que obtiene una instrucción, la procesa para identificar las entradas, reconoce la operación a realizar y manda realizarla.
- El segundo nivel es el nivel de operación que reconoce la operación a ejecutar y sigue el algoritmo correspondiente.
- El tercer nivel es el nivel lógico de gestión de los componentes que unidos realizan la operación. Así, el controlador va modificando las señales de control de los distintos componentes para obtener el comportamiento deseado.

El controlador sigue el siguiente esquema de conexión.

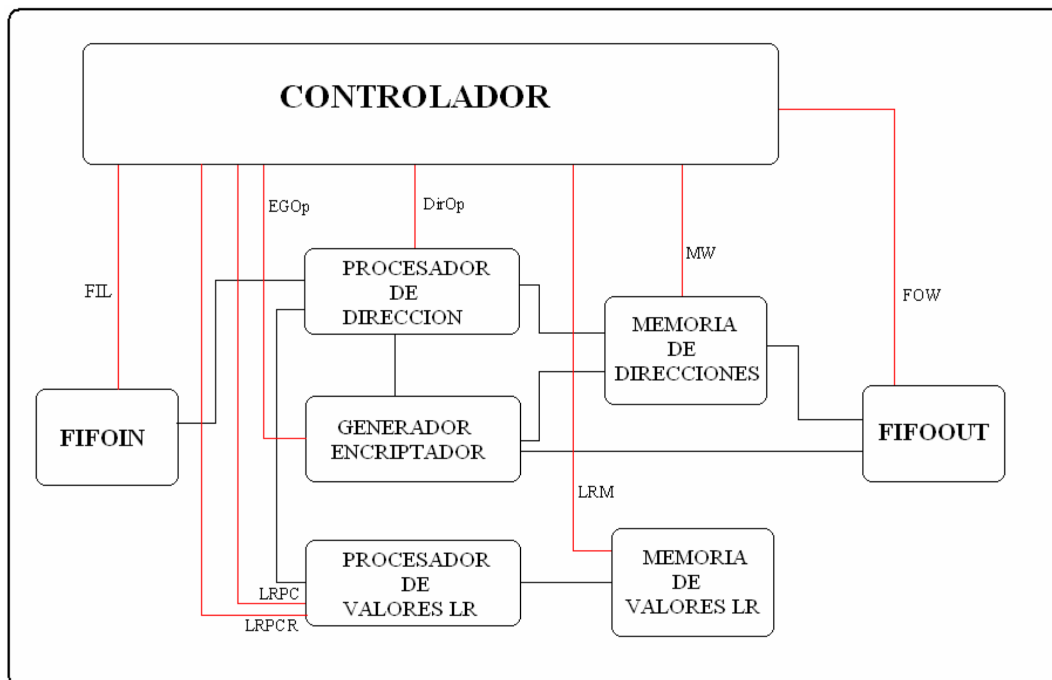


Figura 4.5.1 Esquema Controlador del Sistema

Donde las señales en rojo son las señales de control que se enumeran a continuación.

4.5.2. Señales de control

1s: Selector del multiplexor 1.

2s: Selector del multiplexor 2.

3s: Selector del multiplexor 3.

4s: Selector del multiplexor 4.

5s: Selector del multiplexor 5.

6s: Selector del multiplexor 6.

AW: Bit de permiso de escritura en el registro Address.

MW: Bit de permiso de escritura en memoria principal.

EGW: Bit de permiso de escritura en el EG.

KRW: Bit de permiso de escritura en el KR.

LRM: Bit de permiso de escritura en la memoria LR.

LRW: Bit de permiso de escritura en el registro LR.

AW: Bit de permiso de escritura en el registro Address.

FOW: Bit de permiso de escritura en el registro Address.

DirOp : bits de control del módulo procesador de direcciones.

EGOp : código de operación que se entrega al módulo Generador/ Encriptador.

LRPC: bit de control del módulo procesador de campos LR.

LRPCR: reset de la memoria LR.

4.5.3 Tablas de señales en operaciones

A continuacion se muestran el estado de cada señal de control para cada estado de cada operación

Inicio:

STa	Operación	1s	2s	3s	4s	5s	6s	AW	MW	EGW	LRMW	LRW	KR	FOW	DirOp	EGOp	LRMRes	LRPC	STa
0	Iniciar E/G Iniciar LRM	x	x	x	x	x	x	0	0	0	0	0	0	0	X	00	1	x	F
F	Devolver el control																		

Tabla 4.5.2.1 Señales de Control Inicio

Unión (userId, kd):

STA	Operación	1s	2s	3s	4s	5s	6s	AW	MW	EGW	LRMW	LRW	KR	FOW	DirOp	EGOp	LRPC	STa
0	Adress = userId;	0	x	x	x	x	x	1	0	0	0	0	0	0	11	01	x	1
1	Almacenar (kd,Adress);	x	01	0	x	x	X	0	1	0	0	0	0	0	x	01	x	2
2	Si adress/= raiz Adress = padre(adres); EG= generaClave; //si disponible LR= LRM(adress) Si no	1 x	x x	x x	x x	x x	x x	1 x	0 x	1 x	0 x	1 x	x	0 x	10 x	11 01	x x	3 F
3	encripta(EG,kd) //si disponible	x	x	x	1	1	01	0	0	1	0	0	0	1	x	10	x	4
4	encripta(EG,Mem(adress)) //si disponible	x	01	x	0	1	01	0	0	1	0	0	0	1	x	10	x	5
5	Mem(adress) = EG ActualizaLR	X	01	1	x	x	x	0	1	0	1	0	0	0	x	01	1a	2
F	Devolver el control																	

Tabla 4.5.2.2 Señales de Control Unión.

DesUnión (userId)

STa	Operación	1s	2s	3s	4s	5s	6s	AW	MW	EGW	LRMW	LRW	FOW	DirOp	EGOp	LRPC	STa
0	Adress = userId;	0	x	x	x	x	x	1	0	0	0	0	0	11	01	x	1
1	Si adress/= raiz Adress = padre(adress); EG = generaClave; //si disponible LR = LRM(adress) Si no	1 x	x x	x x	x x	x x	x x	1 x	0 x	1 x	0 x	1 x	0	10 x	11 01	x x	2 F
2	Si lr = 01 //si disponible encripta(EG,MEM(hizq)) Si lr = 10 //si disponible encripta(EG,MEM(der)) Si lr = 11 //si disponible encripta(EG,MEM(der))	1 1 1	00 00 00	x x x	0 0 0	1 1 1	01 01 01	0 0 0	0 0 0	0 0 0	0 0 0	0 0 0	1 1 1	00 01 01	10 10 10	x x x	3 3 4
3	Mem(adress) = EG ActualizarLR	x	01	1	x	x	X	0	1	0	1	0	0	x	01	0a	1
F	Devolver el control																
4	encripta(EG,MEM(hizq)) //si disponible	1	00	x	0	1	01	0	0	0	0	0	1	00	01	x	3

Tabla 4.5.2.3 Señales de Control Desunión.

Actualiza_KG

STa	Operación	1s	2s	3s	4s	5s	6s	AW	MW	EGW	LRMW	LRW	FOW	DirOp	EGOp	LRPC	STa
0	EG = generaClave;	x	x	x	x	1	x	0	0	1	0	0	0	x	11	x	1
1	FIFO Out = Encripta(EG, mem(raiz))	x	10	x	0	0	01	0	0	0	0	0	1	x	10	x	2
2	Mem(raiz) = EG	x	10	1	x	x	x	0	1	0	0	0	0	x	01	x	F
F	Devolver le control																

Tabla 4.5.2.4 Señales de Control ActualizaKg.

Entrega_KG g

STa	Operación	1s	2s	3s	4s	5s	6s	AW	MW	EGW	LRMW	LRW	FOW	Dir Op	EG Op	LR PC	STa
0	FIFO OUT = Mem(raiz)	x	10	x	x	0	x	0	0	0	0	0	1	x	01	x	F
F	Devolver le control																

Tabla 4.5.2.5 Señales de Control EntregaKG.

Resincronizar (userId)

STA	Operación	1s	2s	3s	4s	5s	6s	AW	MW	EGW	LRMW	KR	LRW	FOW	Dir Op	EG Op	LR PC	STa
0	Adress = userId;	0	x	x	x	x	x	1	0	0	0	0	0	0	11	01	x	1
1	Kr = mem(adress)	x	01	x	x	X	x	0	0	0	0	1	0	0	x	x	x	2
2	Si adress/= raiz		x	x	x	x	x		0	0	0	0	0	0		01	x	3
	Adress = padre(adres);	1						1							10			
	Si no	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	F
3	FiFO Out= encripta(mem(adress), kr);	x	01	x	0	1	10	0	0	0	0	0	0	1	x	01	x	2
F	Devolver el control																	

Tabla 4.5.2.6 Señales de Control Resincronizar.

4.6. Resultados para el Sistema Completo

Para el sistema completo no hemos tenido que modificar el ciclo de reloj, en cuanto al área de la placa ocupada ahora es el 33%.

```
Found area constraint ratio of 100 (+ 5) on block rutadatos, actual
ratio is 33.
```

A continuación podemos ver el área ocupada marcada en rojo. También se puede observar como se aprovechan las memorias de la placa.

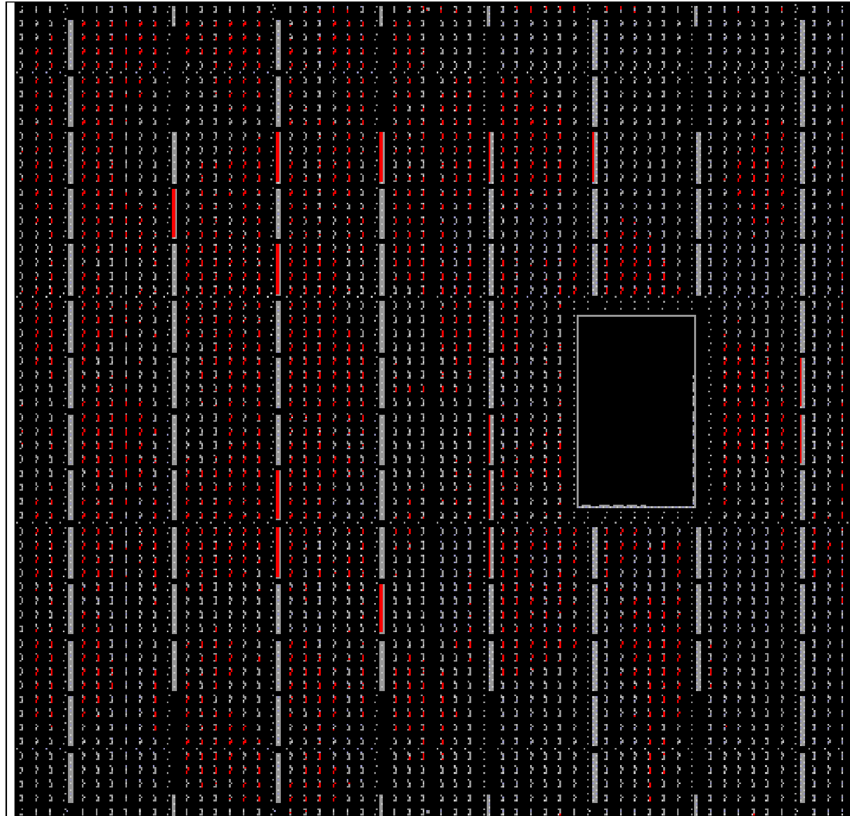


Figura 4.6.1 Área placa Sistema.

Podemos ver donde está situada la lógica de control en la figura 4.6.2:

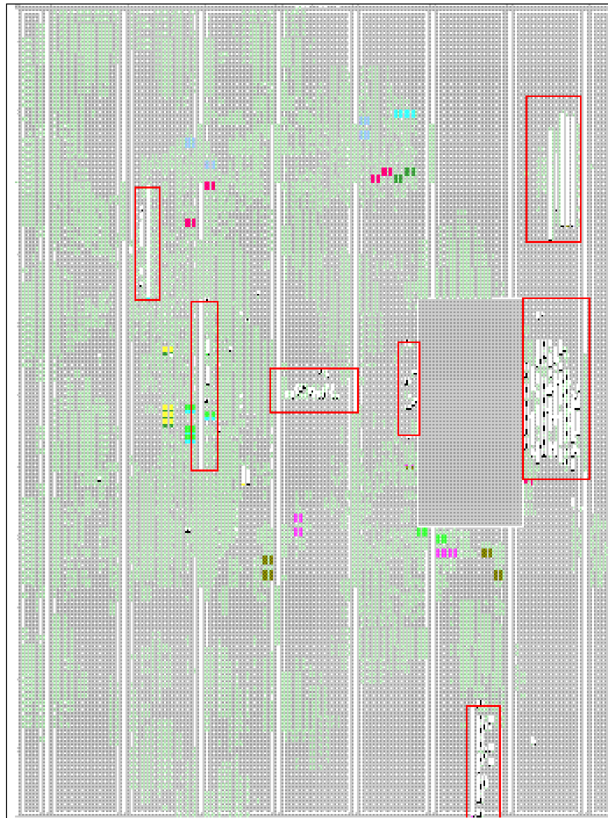


Figura 4.6.2 Área placa Control del Sistema.

Como es de suponer la mayor parte de la placa está ocupada por el Generador/Encriptador, en la figura 4.6.3 esta parte se muestra en color oscuro:

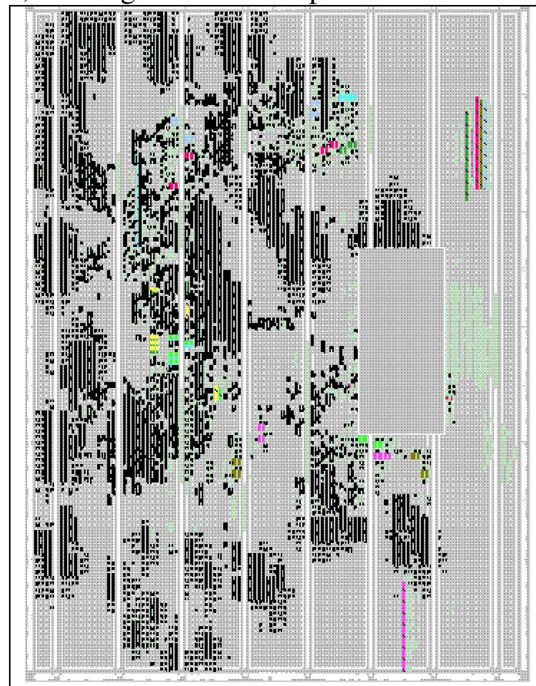


Figura 4.6.3 Area placa Generador/Encriptador.

Simulación

Para comprobar que el sistema funciona correctamente vamos a simular la ejecución de las instrucciones mostradas en el ejemplo de funcionamiento del apartado 3.7.

Para ello hacemos una simulación *Behavioral* con el programa ModelSim.

Tenemos el árbol de claves vacío:

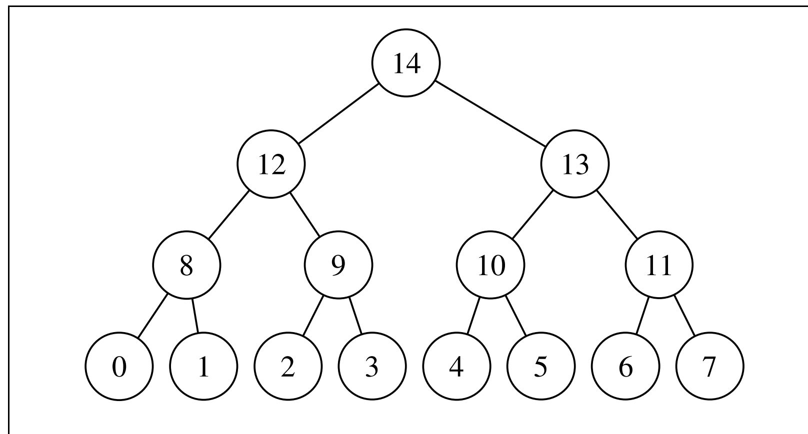


Figura 4.6.4 Árbol de Claves vacio.

Primero inicializamos el sistema: Instrucción:"000000000 ... 0"

Unión del usuario 3: "00100110000001100...0", (001 unión, 0011 id usuario3, el resto clave 03..0)

Señal	Valor
/tproyec_vhd/salida	EB6D
/tproyec_vhd/clk	1
/tproyec_vhd/uut/ctrl/op	010
/tproyec_vhd/uut/ctrl/dirop	11
/tproyec_vhd/uut/ctrl/egop	01
/tproyec_vhd/uut/fo/mem	{3854BEE17C97C93567A351EABE15D853A 38
(0)	3854BEE17C97C93567A351EABE15D853A
(1)	38CCF5BAE3AFAE61D7D267713E0543D4E
(2)	343C6A2624B614CF3BADF1ECE1154F41A
(3)	XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
(4)	XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
(5)	XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
(6)	XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
(7)	XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
(8)	XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
(9)	XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
/tproyec_vhd/uut/mem/tabla	{00000000000000000000000000000000 00000
(0)	00000000000000000000000000000000
(1)	00000000000000000000000000000000
(2)	00000000000000000000000000000000
(3)	03000000000000000000000000000000
(4)	00000000000000000000000000000000
(5)	00000000000000000000000000000000
(6)	00000000000000000000000000000000
(7)	00000000000000000000000000000000
(8)	00000000000000000000000000000000
(9)	F795BD4A52E29ED713D313FA20E98DBC
(10)	00000000000000000000000000000000
(11)	00000000000000000000000000000000
(12)	868B60B4094B650025D529B4D6045852
(13)	00000000000000000000000000000000
(14)	26A2F9E3F5371DE74BF794B26AF5A36C
(15)	00000000000000000000000000000000
/tproyec_vhd/uut/ctrl/estado	1
/tproyec_vhd/uut/sar	9

Figura 4.6.4 Captura de Simulación.

Observamos como se escriben las nuevas claves en el árbol de claves
en la posición 3 0300... que es la clave que le pasamos en la instrucción
en la posición 9 (padre de 3) f795bd4a52e29ed713d313fa20e98dbc ... que es la 1ª clave generada
en la posición 12 (padre de 9) 868b60b4094b650025d529b4d6045852... la 2ª clave generada
en la posición 14 (padre de 12) 26a2f9e3f5371de74bf794b26af5a36c... la 3ª clave generada

En la FIFO de salida se escriben:
3854B... :El 3 inicial se corresponde al id del usuario destino y es la clave del padre (F795..) encriptada con la clave del usuario 3 (0300..)
38CCF... :La clave de 12 (868B..) encriptada con la clave de 3

También podemos observar los valores de la señal salida, 6921 7a4d eb6d estos nos servirán para comprobar que la simulación Post Place & Route es correcta.

Unión del usuario 2: "00100100000001000...0", (001 unión, 0010 id usuario2, el resto clave 02..

Señal	Valor
proyec_vhd/salida	B190
proyec_vhd/clk	1
proyec_vhd/uut/ctrl/op	001
proyec_vhd/uut/ctrl/dirop	10
proyec_vhd/uut/ctrl/egop	10
proyec_vhd/uut/fo/mem	{3854BEE17C97C93567A351EABE15D853A 3
(0)	3854BEE17C97C93567A351EABE15D853A
(1)	38CCF5BAE3AFAE61D7D267713E0543D4E
(2)	343C6A2624B614CF3BADF1ECE1154F41A
(3)	2F47E453043644801EE30D4464708E0F2
(4)	9389EBEE8191752A1379348E614D55641
(5)	2D8AFF9E2F318D62364BF638B8D142A70
(6)	C3797F0884FAE561C74C8A06A0972F2FB
(7)	2832CCBDD07703930AA278996F11D45C9
(8)	E1269E7DDDF8BD84CF8EE6F55C7D03AE3
(9)	XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
proyec_vhd/uut/mem/tabla	{00000000000000000000000000000000 0000
(0)	00000000000000000000000000000000
(1)	00000000000000000000000000000000
(2)	02000000000000000000000000000000
(3)	03000000000000000000000000000000
(4)	00000000000000000000000000000000
(5)	00000000000000000000000000000000
(6)	00000000000000000000000000000000
(7)	00000000000000000000000000000000
(8)	00000000000000000000000000000000
(9)	6063ADC92BAC1D871D90E88B1E2774E2
(10)	00000000000000000000000000000000
(11)	00000000000000000000000000000000
(12)	94BD3082CC00A632D9F854331A607DB7
(13)	00000000000000000000000000000000
(14)	0A56991047CFBFCDE3AABEF3B163F689
(15)	00000000000000000000000000000000
proyec_vhd/uut/ctrl/estado	100
proyec_vhd/uut/sar	8

Figura 4.6.5 Captura de Simulación.

Ahora vemos como además de los mensajes generados para el usuario 2 se producen otros para 9, C y E, esto es porque el sistema sabe que estas claves estaban en uso (por el usuario 3) entonces al ser generada una nueva clave 9 ésta se encripta con la antigua clave 9, de este modo el usuario 3 puede obtener la nueva clave.

El mensaje almacenado en fo/mem(3) se corresponde con la nueva clave 9 encriptada con la clave de 3.
mem(4) con la nueva clave de 9 encriptada con la antigua 9.
mem(5) con la nueva de 12 encriptada con la de 3.
mem(6) con la nueva de 12 encriptada con la antigua 12.
mem(7) con la nueva de 14 encriptada con la de 3.
mem(8) con la nueva de 14 encriptada con la antigua 14.

Comprobamos si realmente el sistema controla el uso de las claves, mirando el contenido de la tabla lr:

/tproyec_vhd/uut/lrmem/tabla	{00 0(
(0)	00
(1)	00
(2)	00
(3)	00
(4)	00
(5)	00
(6)	00
(7)	00
(8)	00
(9)	11
(10)	00
(11)	00
(12)	01
(13)	00
(14)	10
(15)	00

Figura 4.6.6 Captura de Simulación de la memoria LR

Observamos que la clave 9 está usada tanto por la derecha como por la izquierda, la 12 por la derecha y la 14 por la izquierda:

En forma de árbol:

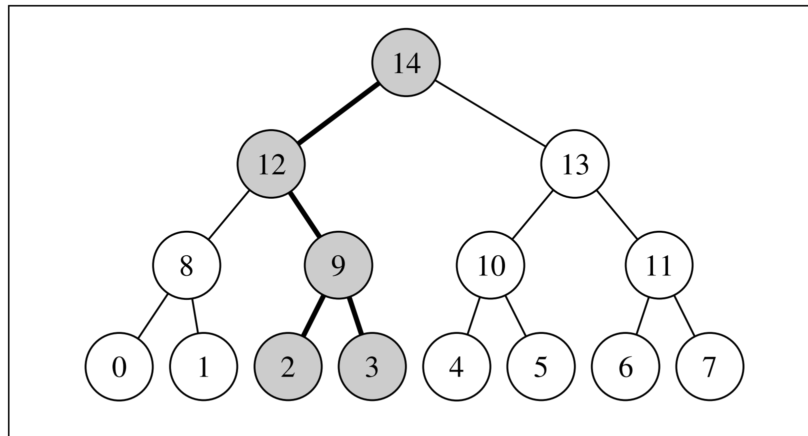


Figura 4.6.7 Árbol de Claves Actual.

Unión del usuario 0: "00100000000000000...0", (001 unión, 0000 id usuario2, el resto clave 00..0)

Señal	Valor	Forma de onda
proyec_vhd/salida	09A6	B190 673A A6E8 5E2B 79ED
proyec_vhd/clk	1	
proyec_vhd/uut/ctrl/op	001	001
proyec_vhd/uut/ctrl/dirop	10	10
proyec_vhd/uut/ctrl/egop	01	10 10 10
proyec_vhd/uut/fo/mem	{03A47B1A7BB490DD0F0AB02C5247C6CA7	{3854... {3854BE... {03A... {03A47B... {03A...
(0)	03A47B1A7BB490DD0F0AB02C5247C6CA7	3854BEE17C97C9... 03A47B1A7BB490DD0F0AB0
(1)	C106E4BB969108E1EBDE08C974A117314	38CCF5BAE3AFAE61D7D... C106E4BB969108E1
(2)	0C8D88A4A33679B628578C03115FB02E4	343C6A2624B614CF3BADF1ECE1154... 0C8D88A
(3)	E4DF71CC6A007F570261700C2B76CC656	2F47E453043644801EE30D4464708E0F2
(4)	9389EBEE8191752A1379348E614D55641	9389EBEE8191752A1379348E614D55641
(5)	2D8AFF9E2F318D62364BF638B8D142A70	2D8AFF9E2F318D62364BF638B8D142A70
(6)	C3797F0884FAE561C74C8A06A0972F2FB	C3797F0884FAE561C74C8A06A0972F2FB
(7)	2832CCBDD07703930AA278996F11D45C9	2832CCBDD07703930AA278996F11D45C9
(8)	E1269E7DDDF8BD84CF8EE6F55C7D03AE3	E1269E7DDDF8BD84CF8EE6F55C7D03AE3
(9)	02B1C870815AE8E129E578F003F6B40A9	XXXX... 02B1C870815AE8E129E578F003F6B40A
proyec_vhd/uut/mem/tabla	{00000000000000000000000000000000 000C	{0000... {000000... {0000000000000... {0000000
(0)	00000000000000000000000000000000	00000000000000000000000000000000
(1)	00000000000000000000000000000000	00000000000000000000000000000000
(2)	02000000000000000000000000000000	02000000000000000000000000000000
(3)	03000000000000000000000000000000	03000000000000000000000000000000
(4)	00000000000000000000000000000000	00000000000000000000000000000000
(5)	00000000000000000000000000000000	00000000000000000000000000000000
(6)	00000000000000000000000000000000	00000000000000000000000000000000
(7)	00000000000000000000000000000000	00000000000000000000000000000000
(8)	E23ABFC6F33EC4AC4773AC6321CDF439	00000... E23ABFC6F33EC4AC4773AC6321CDF43
(9)	6063ADC92BAC1D871D90E88B1E2774E2	6063ADC92BAC1D871D90E88B1E2774E2
(10)	00000000000000000000000000000000	00000000000000000000000000000000
(11)	00000000000000000000000000000000	00000000000000000000000000000000
(12)	3CF7D890010A66BBA7B0C9B727658DE3	94BD3082CC00A6... 3CF7D890010A66BBA7B0C9
(13)	00000000000000000000000000000000	00000000000000000000000000000000
(14)	479BC0D8AB0CB238EA790683008D2095	0A56991047CFBFCDE3AABEF3B163F... 479BC0D
(15)	00000000000000000000000000000000	00000000000000000000000000000000
proyec_vhd/uut/ctrl/estado	001	100 100 101 100 101
proyec_vhd/uut/sar	E	8 C E

Figura 4.6.8 Captura de Simulación.

El mensaje en mem(9) va dirigido a 0 y consiste en la nueva clave 8 encriptada con la clave de 0, la clave 8 al no estar utilizada previamente no es necesario encriptarla de nuevo.
mem(0) es la nueva clave para 12 encriptada con la de 0.
Como esta sí que estaba utilizada se encripta con su antigua clave, de este modo tanto el usuario 2 como el 3 pueden desencriptarla. Este es mem(1).
mem(2) es la nueva clave para 14 encriptada con la de 0.
mem(3) es la nueva clave para 14 con la antigua clave 14.

Desunión del usuario 3: "00100110...0", (010 desunión, 0011 id usuario3)

Es necesario actualizar las claves 9, 12 y 14 y distribuirlas de modo que el usuario 3 no pueda desencriptarlas. Entonces encriptamos la nueva clave 9 con la clave de 2, la nueva clave 12 con la clave 8 y la nueva 9 (que 2 posee pero no 3), la nueva 14 con la nueva 12.

Señal	Valor
royec_vhd/salida	E51F
royec_vhd/clk	1
royec_vhd/uut/ctrl/op	010
royec_vhd/uut/ctrl/dirop	10
royec_vhd/uut/ctrl/egop	01
royec_vhd/uut/fo/mem	{03A47B1A7BB490DD0F0AB02C5247C6CA7
(0)	03A47B1A7BB490DD0F0AB02C5247C6CA7
(1)	C106E4BB969108E1EBDE08C974A117314
(2)	0C8D88A4A33679B628578C03115FB02E4
(3)	E4DF71CC6A007F570261700C2B76CC656
(4)	2706E8EFA980A73C86FF0D38C8C2572C2
(5)	82CCB088F7B57B715CC32CACF5C410901
(6)	99E155678C01ECA97D06C18083D30F826
(7)	C0861588CA20069C904CFD6D2EB2A79F8
(8)	E1269E7DDDF8BD84CF8EE6F55C7D03AE3
(9)	02B1C870815AE8E129E578F003F6B40A9
royec_vhd/uut/mem/tabla	{00000000000000000000000000000000 0000
(0)	00000000000000000000000000000000
(1)	00000000000000000000000000000000
(2)	02000000000000000000000000000000
(3)	03000000000000000000000000000000
(4)	00000000000000000000000000000000
(5)	00000000000000000000000000000000
(6)	00000000000000000000000000000000
(7)	00000000000000000000000000000000
(8)	E23ABFC6F33EC4AC4773AC6321CDF439
(9)	61680AE1EFBF30E1CBC1B34D58B74C10
(10)	00000000000000000000000000000000
(11)	00000000000000000000000000000000
(12)	15D84C440F23A5ACAB2484B1AC00A1AD
(13)	00000000000000000000000000000000
(14)	007B48E3EF49DBC2BA82EC59275DC966
(15)	00000000000000000000000000000000
royec_vhd/uut/ctrl/estado	001
royec_vhd/uut/sar	F

Figura 4.6.9 Captura de Simulación.

Se puede comprobar que los cuatro mensajes resaltados en la captura se corresponden a los mencionados más arriba.

Desunión del usuario 2: "00100100...0", (010 desunión, 0010 id usuario2)
Esta operación es similar a la comentada antes, una única consideración: no es necesario generar una nueva clave para 9 ya que es una clave que se queda sin utilizar, por tanto es necesario: generar una clave nueva para 12 y encriptarla con 8, y generar una nueva para 14 y encriptarla con la nueva 12.

Señal	Valor
/tproyec_vhd/salida	BDF8
/tproyec_vhd/clk	1
/tproyec_vhd/uut/ctrl/op	001
/tproyec_vhd/uut/ctrl/dirop	10
/tproyec_vhd/uut/ctrl/egop	11
/tproyec_vhd/uut/fo/mem	{03A47B1A7BB490DD0F0AB02C5247C6CA7
(0)	03A47B1A7BB490DD0F0AB02C5247C6CA7
(1)	C106E4BB969108E1EBDE08C974A117314
(2)	0C8D88A4A33679B628578C03115FB02E4
(3)	E4DF71CC6A007F570261700C2B76CC656
(4)	2706E8EFA980A73C86FF0D38C8C2572C2
(5)	82CCB088F7B57B715CC32CACF5C410901
(6)	99E155678C01ECA97D06C18083D30F826
(7)	C0861588CA20069C904CFD6D2EB2A79F8
(8)	85475DADF84F3E45601DBC63541AB3CE8
(9)	C33119985AE62577919BE614F0DCB5A5D
/tproyec_vhd/uut/mem/tabla	{00000000000000000000000000000000 000
(0)	00000000000000000000000000000000
(1)	00000000000000000000000000000000
(2)	02000000000000000000000000000000
(3)	03000000000000000000000000000000
(4)	00000000000000000000000000000000
(5)	00000000000000000000000000000000
(6)	00000000000000000000000000000000
(7)	60000000000000000000000000000000
(8)	E23ABFC6F33EC4AC4773AC6321CDF439
(9)	007B48E3EF49DBC2BA82EC59275DC966
(10)	00000000000000000000000000000000
(11)	00000000000000000000000000000000
(12)	231091AB4EDE855A2D62ADC946F6E7B0
(13)	00000000000000000000000000000000
(14)	F14E4B3E2A0C9A04D92466FFE051CA0F
(15)	00000000000000000000000000000000
/tproyec_vhd/uut/ctrl/estado	011
/tproyec_vhd/uut/sar	B

Figura 4.6.10 Captura de Simulación.

Observamos que se hacen las dos operaciones requeridas, además se ha cambiado la clave de 9, pero esto no es una clave válida, es simplemente basura, pero no importa porque la clave 9 está marcada como no utilizada.

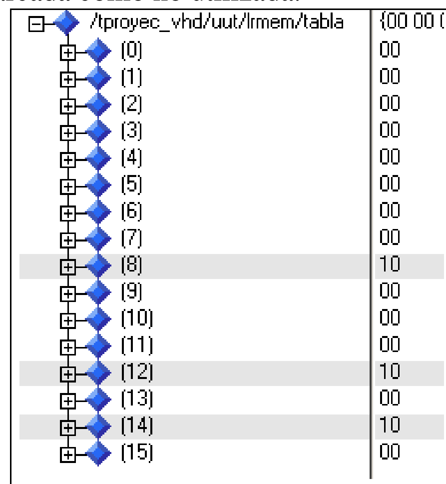


Figura 4.6.11 Captura de Simulación de la memoria LR

Que en forma de árbol es:

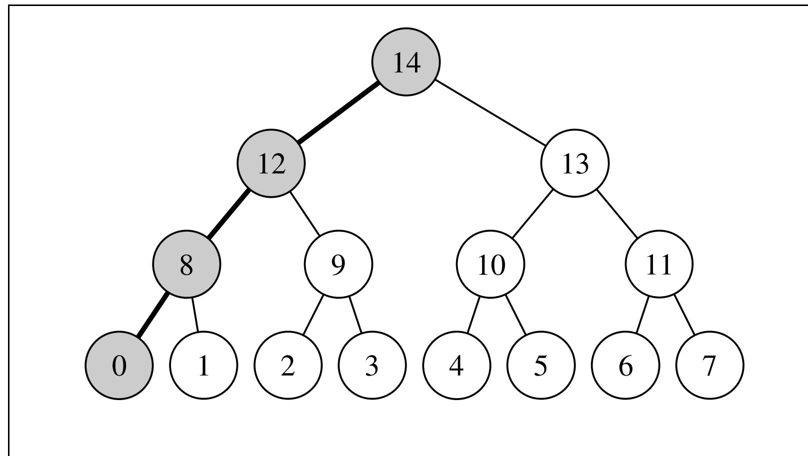
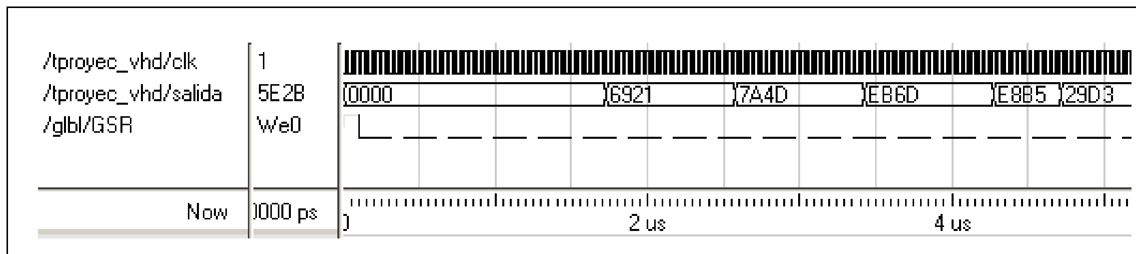


Figura 4.6.12 Árbol de Claves Actual.

Como ya vimos en el apartado 3.7.

Simulación *Post Place and Route*:
Ahora hacemos esta simulación, también en el ModelSim para comprobar que el sistema se implementará correctamente en Hardware:



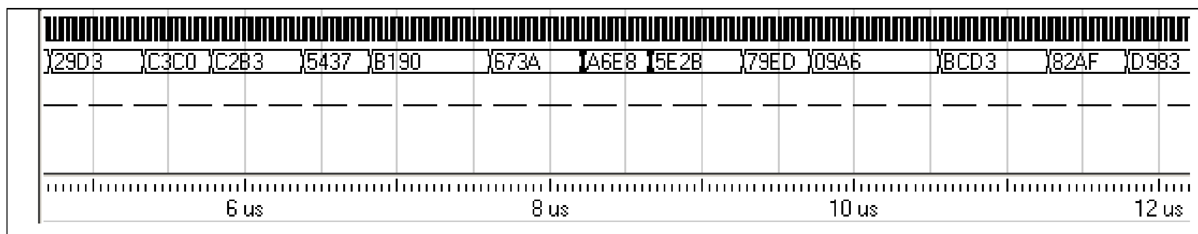


Figura 4.6.14a Captura de Simulación.

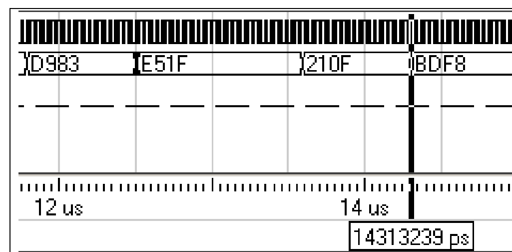


Figura 4.6.14b Captura de Simulación.

Podemos observar que la salida coincide con las simulaciones anteriores.
Como detalle podemos mostrar como varía la salida poco a poco antes de tomar el valor correcto si agrandamos mucho la onda figura 4.6.15.

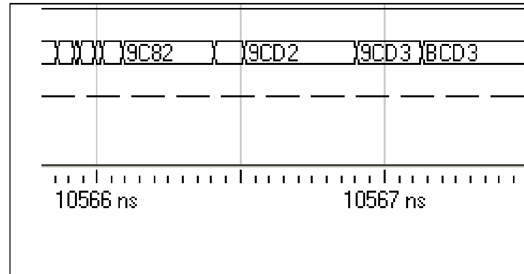


Figura 4.6.15 Detalle Captura de Simulación.

4.7. Mejoras respecto a versiones anteriores

Incluir dos AES

Una solución para mejorar los tiempos, es incluir otro módulo AES en el Generador/Encriptador, de forma que funcionen por separado el encriptador y el generador de nuevas claves. De esta forma el generador estaría funcionando continuamente de forma automática y guardando las claves generadas en la FIFO. Tendremos, por tanto, que modificar la ruta de datos del Generador /Encriptador, añadiendo el nuevo módulo y diferenciando encriptador y generador.

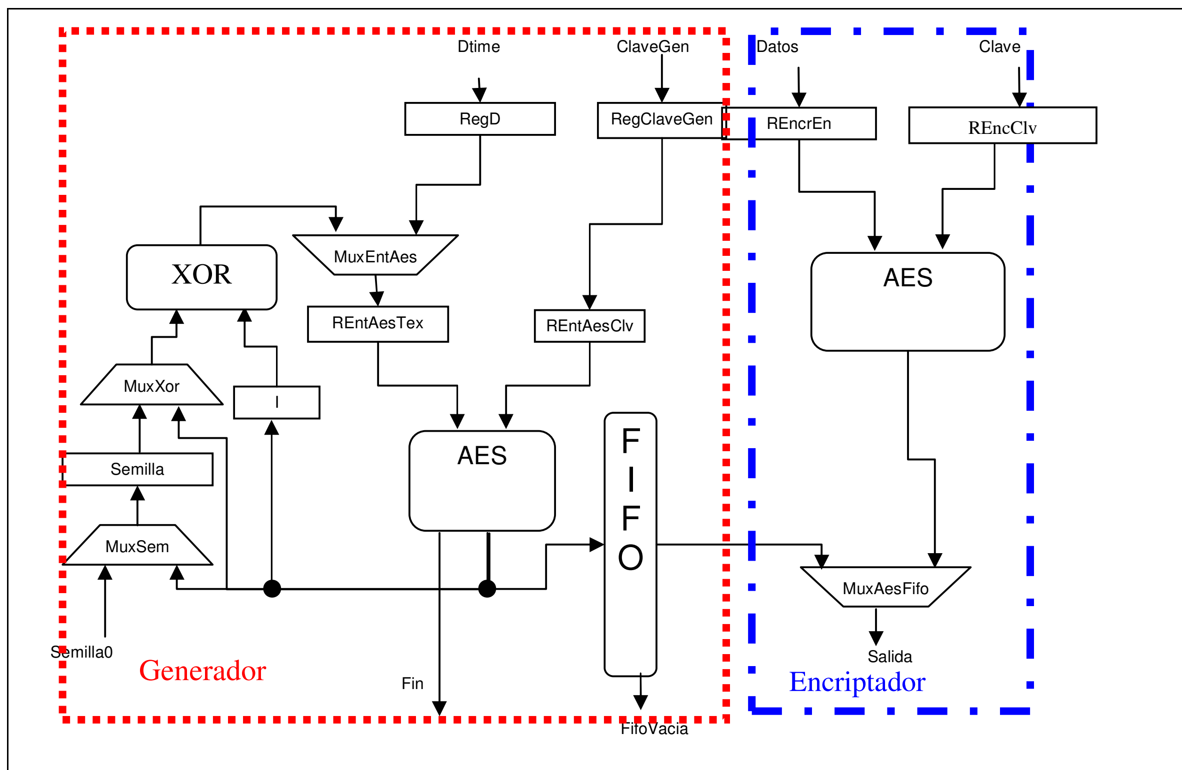


Figura 4.7.1 Ruta Datos Generador/Encriptador Mejorado.

También será necesario incluir un nuevo controlador que se encargue del generador. Este controlador estará su vez dirigidoo por el controlador principal. Con esto hemos logrado a su vez simplificar la lógica de control del controlador principal, ya que este sólo se tiene que encargar directamente de la función de encriptación y lectura, y delegar el resto de las funciones al nuevo controlador. De esta forma conseguimos que el generador esté en continuo funcionamiento.

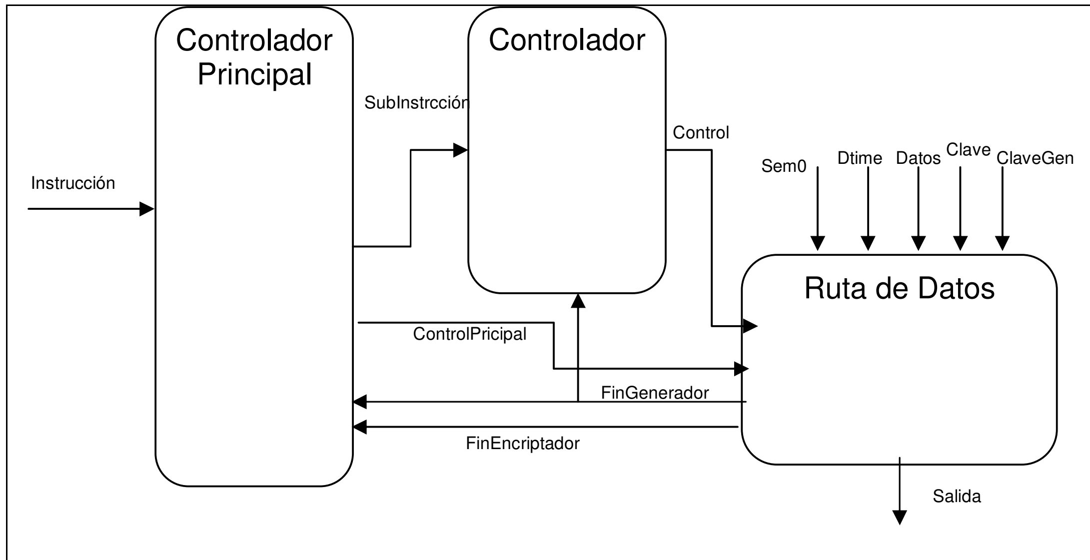


Figura 4.7.2 MóduloGenerador/Encriptador Mejorado.

Todo esto es posible ya que el espacio ocupado por este módulo era el 25%, ahora al añadir un nuevo AES y otro controlador aumenta a 42%, quedando espacio de sobra para el resto del sistema, de forma que incluso se podría pensar en la inclusión de otro AES. El reloj sin embargo no sufre modificaciones.
Todos los cambios comentados anteriormente se han realizado dentro del módulo Generador/Encriptador por lo que no han sido necesarios cambios en el resto del sistema, que sigue funcionando de igual manera, los que se encargan de la gestión son los dos controladores internos.
Para comprobar la diferencia de rendimiento de la versión con un AES a la versión con dos, simulamos dos uniones, un entregarKg, un actualizarKg, un resincronizar y dos desuniones.
Simulamos el sistema con un aes.

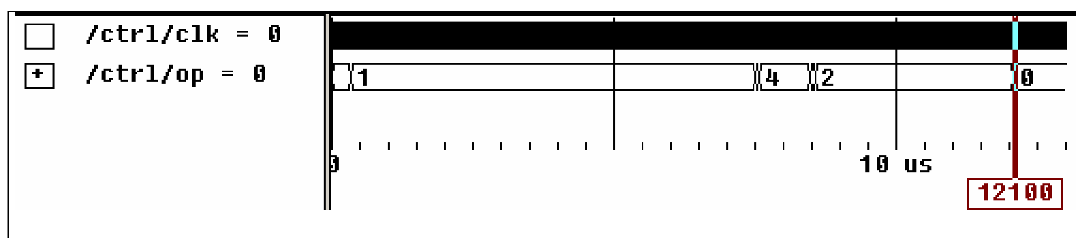


Figura 4.7.3 Detalle de Simulación.

Vemos que tarda 12.1 us.

Ahora simulamos la versión con dos aes.

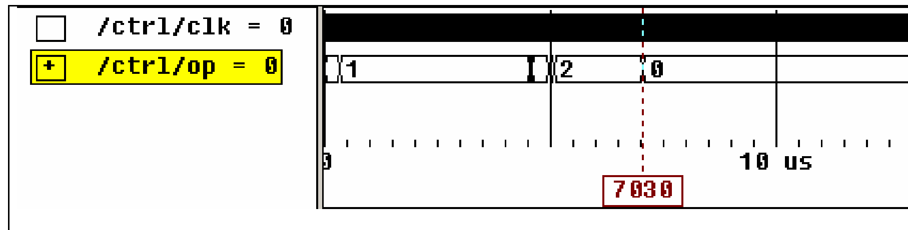


Figura 4.7.4 Detalle de Simulación.

Vemos que ahora tarda 7.03 us en ejecutar las mismas instrucciones.
Lo que nos da un speedup de 1.7 aproximadamente.

4.8. Resultados de la versión final.

Después de todas mejoras y modificaciones hemos llegado a una versión final con las siguientes características.
-. Realiza 6 tipos diferentes de instrucciones.
-. Un ciclo de reloj para el sistema de 20ns.
-. Espacio ocupado en la placa es el 50% con un total de 5182 Slices.
Sólo se muestran en el informe de Xilinx los elementos no comentados anteriormente, se has subrayado los términos más interesantes en la tabla 4.8.1.

```
=========================================================================
*                           HDL Synthesis                               *
=========================================================================

Synthesizing Unit <rutaaes>.
    Related source file is "E:/proyec/proy.vhd".
Unit <rutaaes> synthesized.
Synthesizing Unit <genencryp>.
    Related source file is "E:/proyec/proy.vhd".
Unit <genencryp> synthesized.
Synthesizing Unit <lrProc>.
    Related source file is "E:/proyec/rutadatos.vhd".
Unit <lrProc> synthesized.
Synthesizing Unit <lrM>.
    Related source file is "E:/proyec/rutadatos.vhd".
    Found 2-bit 16-to-1 multiplexer for signal <salida>.
    Found 32-bit register for signal <tabla>.
    Summary:
       inferred  32 D-type flip-flop(s).
       inferred   2 Multiplexer(s).
Unit <lrM> synthesized.
Synthesizing Unit <dirProc>.
    Related source file is "E:/proyec/rutadatos.vhd".
    Found 4-bit 4-to-1 multiplexer for signal <salida>.
    Summary:
       inferred   4 Multiplexer(s).
Unit <dirProc> synthesized.
Synthesizing Unit <rutadatos>.
    Related source file is "E:/proyec/rutadatos.vhd".
Unit <rutadatos> synthesized.

=========================================================================
HDL Synthesis Report

Macro Statistics
# FSMs                                     : 4
```

```
# Block RAMs                                    : 3
 10x128-bit dual-port block RAM                 : 1
 10x132-bit dual-port block RAM                 : 1
 10x135-bit dual-port block RAM                 : 1
# LUT RAMs                                      : 1
 16x128-bit single-port distributed RAM: 1
# ROMs                                          : 40
 256x8-bit ROM                                  : 40
# Adders/Subtractors                   : 9
 31-bit adder                          : 1
 32-bit adder                          : 2
 5-bit subtractor                      : 2
 6-bit adder                           : 4
# Counters                             : 8
 31-bit up counter                     : 2
 32-bit up counter                     : 2
 4-bit modulo-%d up counter            : 4
# Registers                            : 288
 1-bit register                        : 35
 11-bit register                       : 1
 2-bit register                        : 24
 31-bit register                       : 1
 4-bit register                        : 2
 5-bit register                        : 1
 8-bit register                        : 224
# Comparators                          : 4
 31-bit comparator equal               : 1
 31-bit comparator not equal           : 1
 4-bit comparator equal                : 2
# Multiplexers                         : 54
 1-bit 4-to-1 multiplexer              : 12
 2-bit 16-to-1 multiplexer             : 1
 2-bit 4-to-1 multiplexer              : 1
 2-bit 8-to-1 multiplexer              : 1
 31-bit 8-to-1 multiplexer             : 1
 4-bit 4-to-1 multiplexer              : 3
 5-bit 4-to-1 multiplexer              : 1
 8-bit 12-to-1 multiplexer             : 2
 8-bit 4-to-1 multiplexer              : 32
# Xors                                 : 416
 1-bit xor2                            : 256
 1-bit xor8                            : 16
 8-bit xor2                            : 110
 8-bit xor3                            : 2
 8-bit xor4                            : 32

==============================================================*
Low Level Synthesis                          *
==============================================================
Building and optimizing final netlist ...
Found area constraint ratio of 100 (+ 5) on block rutadatos,
actual ratio is 50.
==============================================================*
Final Report                              *
==============================================================Final Results
RTL Top Level Output File Name     : rutadatos.ngr
Top Level Output File Name         : rutadatos
Output Format                      : NGC
Optimization Goal                  : Speed
Keep Hierarchy                     : NO

Design Statistics
# IOs                              : 158

Macro Statistics :
# RAM                              : 4
#      10x128-bit dual-port block RAM: 1
#      10x132-bit dual-port block RAM: 1
#      10x135-bit dual-port block RAM: 1
#      16x128-bit single-port distributed RAM: 1
# ROMs                             : 40
#      256x8-bit ROM               : 40
# Registers                        : 283
```

```
#      1-bit register            : 22
#      11-bit register           : 1
#      2-bit register            : 24
#      31-bit register           : 1
#      4-bit register            : 10
#      5-bit register            : 1
#      8-bit register            : 224
# Multiplexers                   : 54
#      1-bit 4-to-1 multiplexer  : 12
#      2-bit 16-to-1 multiplexer : 1
#      2-bit 4-to-1 multiplexer  : 1
#      2-bit 8-to-1 multiplexer  : 1
#      31-bit 8-to-1 multiplexer : 1
#      4-bit 4-to-1 multiplexer  : 3
#      5-bit 4-to-1 multiplexer  : 1
#      8-bit 12-to-1 multiplexer : 2
#      8-bit 4-to-1 multiplexer  : 32
# Adders/Subtractors             : 9
#      31-bit adder              : 1
#      32-bit adder              : 2
#      5-bit subtractor          : 2
#      6-bit adder               : 4
# Comparators                    : 4
#      31-bit comparator equal   : 1
#      31-bit comparator not equal : 1
#      4-bit comparator equal    : 2
# Xors                           : 50
#      1-bit xor8                : 16
#      8-bit xor3                : 2
#      8-bit xor4                : 32

Cell Usage :
# BELS                           : 15073
#      GND                       : 1
#      INV                       : 43
#      LUT1                      : 155
#      LUT1_L                    : 12
#      LUT2                      : 163
#      LUT2_D                    : 4
#      LUT2_L                    : 519
#      LUT3                      : 906
#      LUT3_D                    : 175
#      LUT3_L                    : 303
#      LUT4                      : 1377
#      LUT4_D                    : 527
#      LUT4_L                    : 5444
#      MUXCY                     : 247
#      MUXF5                     : 2744
#      MUXF6                     : 1286
#      MUXF7                     : 642
#      MUXF8                     : 291
#      VCC                       : 1
#      XORCY                     : 233
# FlipFlops/Latches              : 3313
#      FD                        : 5
#      FDCE                      : 64
#      FDE                       : 3113
#      FDR                       : 8
#      FDRE                      : 112
#      FDS                       : 11
# RAMS                           : 30
#      RAM16X8S                  : 16
#      RAMB16_S36_S36            : 14
# Clock Buffers                  : 1
#      BUFGP                     : 1
# IO Buffers                     : 157
#      IBUF                      : 137
#      OBUF                      : 20
=========================================================================

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

Selected Device : 2vpx20ff896-7

 Number of Slices:                      5182  out of   9792     52%
```

Number of Slice Flip Flops:	3313	out of	19584	16%
Number of 4 input LUTs:	9601	out of	19584	49%
Number of bonded IOBs:	158	out of	556	28%
Number of BRAMs:	14	out of	88	15%
Number of GCLKs:	1	out of	16	6%

Tabla 4.8.1 Informe Versión Final

Esta sería situación de la placa para esta versión final.

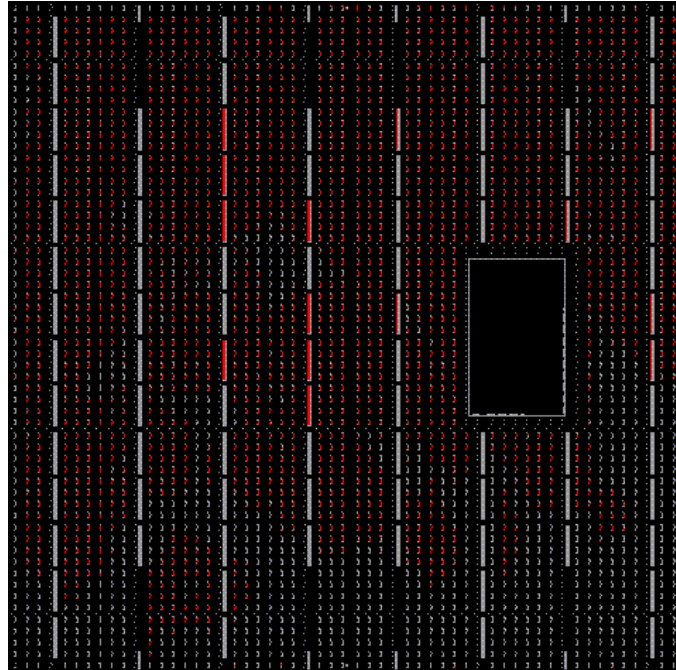


Figura 4.8.2 Area placa Memorias Versión Final.

-. Posee 4 controladores, controlador principal, dos controladores en el módulo generador/encriptador y un contador en el AES.

```
Synthesizing Unit <contador>.
    Related source file is "E:/proyec/proy.vhd".
    Found 1-bit 4-to-1 multiplexer for signal <regload>.
    Found 1-bit 4-to-1 multiplexer for signal <mux1>.
    Found 1-bit 4-to-1 multiplexer for signal <mux2>.
    Found 8-bit 12-to-1 multiplexer for signal <pot>.
    Found 32-bit up counter for signal <tmp>.
    Summary:
       inferred   1 Counter(s).
       inferred  11 Multiplexer(s).
Unit <contador> synthesized.
Synthesizing Unit <controlprincipal>.
Related source file is "E:/proyec/proy.vhd".
    Found finite state machine <FSM_0> for signal <paso>.
    -----------------------------------------------------------------------
    | States             | 2                                              |
    | Transitions        | 12                                             |
    | Inputs             | 6                                              |
    | Outputs            | 4                                              |
    | Clock              | reloj (rising_edge)                            |
    | Power Up State     | 00                                             |
    | Encoding           | automatic                                      |
    | Implementation     | LUT                                            |
    -----------------------------------------------------------------------
    Found 1-bit register for signal <leer>.
    Found 5-bit register for signal <control>.
    Found 1-bit register for signal <instrSub>.
```

```
    Found 1-bit 4-to-1 multiplexer for signal <$n0017> created at line 424.
    Found 1-bit 4-to-1 multiplexer for signal <$n0018> created at line 424.
    Found 5-bit 4-to-1 multiplexer for signal <$n0019> created at line 424.
    Found 1-bit 4-to-1 multiplexer for signal <$n0021> created at line 437.
    Found 1-bit 4-to-1 multiplexer for signal <$n0022> created at line 426.
    Found 1-bit 4-to-1 multiplexer for signal <$n0023> created at line 426.
    Summary:
       inferred   1 Finite State Machine(s).
       inferred   7 D-type flip-flop(s).
       inferred  10 Multiplexer(s).
Unit <controlprincipal> synthesized.


Synthesizing Unit <controlaes>.
    Related source file is "E:/proyec/proy.vhd".
    Found finite state machine <FSM_1> for signal <paso00>.
    -----------------------------------------------------------------------
    | States             | 3                                              |
    | Transitions        | 3                                              |
    | Inputs             | 0                                              |
    | Outputs            | 2                                              |
    | Clock              | reloj (rising_edge)                            |
    | Reset              | $n0012 (positive)                              |
    | Reset type         | synchronous                                    |
    | Reset State        | 00                                             |
    | Power Up State     | 00                                             |
    | Encoding           | automatic                                      |
    | Implementation     | LUT                                            |
    -----------------------------------------------------------------------
    Found finite state machine <FSM_2> for signal <paso01>.
    -----------------------------------------------------------------------
    | States             | 3                                              |
    | Transitions        | 5                                              |
    | Inputs             | 1                                              |
    | Outputs            | 2                                              |
    | Clock              | reloj (rising_edge)                            |
    | Clock enable       | $n0015 (positive)                              |
    | Reset              | $n0016 (positive)                              |
    | Reset type         | synchronous                                    |
    | Reset State        | 00                                             |
    | Power Up State     | 00                                             |
    | Encoding           | automatic                                      |
    | Implementation     | LUT                                            |
    -----------------------------------------------------------------------
    Found 1-bit register for signal <leer>.
    Found 11-bit register for signal <control>.
    Summary:
       inferred   2 Finite State Machine(s).
       inferred  12 D-type flip-flop(s).
Unit <controlaes> synthesized.


Synthesizing Unit <Controlador>.
    Related source file is "E:/proyec/rutadatos.vhd".
    Found finite state machine <FSM_3> for signal <estado>.
    -----------------------------------------------------------------------
    | States             | 8                                              |
    | Transitions        | 89                                             |
    | Inputs             | 18                                             |
    | Outputs            | 8                                              |
    | Clock              | clk (rising_edge)                              |
    | Power Up State     | 000                                            |
    | Encoding           | automatic                                      |
    | Implementation     | LUT                                            |
    -----------------------------------------------------------------------
    Found 1-bit register for signal <AW>.
    Found 1-bit register for signal <EGW>.
    Found 2-bit register for signal <LRPC>.
    Found 1-bit register for signal <LRMR>.
    Found 1-bit register for signal <KRW>.
    Found 1-bit register for signal <LRMW>.
    Found 2-bit register for signal <EGOp>.
    Found 1-bit register for signal <s1>.
    Found 2-bit register for signal <s2>.
    Found 1-bit register for signal <s3>.
    Found 2-bit register for signal <s4>.
    Found 1-bit register for signal <LRW>.
```

```
    Found 1-bit register for signal <s5>.
    Found 2-bit register for signal <s6>.
    Found 2-bit register for signal <s7>.
    Found 1-bit register for signal <MW>.
    Found 1-bit register for signal <FOW>.
    Found 2-bit register for signal <DirOp>.
    Found 1-bit register for signal <lecFI>.
    Found 1-bit register for signal <AW2>.
    Found 2-bit 8-to-1 multiplexer for signal <$n0057>.
    Found 1-bit 4-to-1 multiplexer for signal <$n0063>.
    Found 31-bit 8-to-1 multiplexer for signal <$n0065>.
    Found 31-bit adder for signal <$n0066> created at line 594.
    Found 2-bit 4-to-1 multiplexer for signal <$n0084> created at line 401.
    Found 1-bit register for signal <dj>.
    Found 1-bit register for signal <djsig>.
    Found 1-bit register for signal <fin_instruccion>.
    Found 31-bit register for signal <ret>.
    Found 1-bit register for signal <segenc>.
    Found 1-bit register for signal <Th>.
    Found 1-bit register for signal <utilizado>.
    Summary:
       inferred   1 Finite State Machine(s).
       inferred  64 D-type flip-flop(s).
       inferred   1 Adder/Subtractor(s).
       inferred  36 Multiplexer(s).
Unit <Controlador> synthesized.
```

Tabla 4.8.3 Informe Versión Final.

Los controladores están situados en las siguientes regiones:

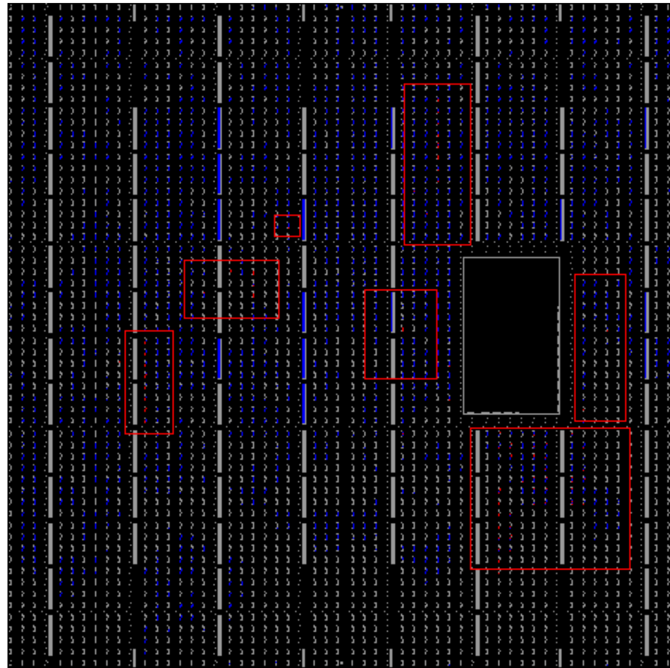


Figura 4.8.3 Area placa Control Versión Final.

-. Cuatro memorias RAM.

```
 Synthesizing Unit <fifo10wordaes>.

     Related source file is "E:/proyec/proy.vhd".
     Found 10x128-bit dual-port distributed RAM for signal <mem>.

    -----------------------------------------------------------------------
     | aspect ratio       | 10-word x 128-bit                   |          |
     | clock              | connected to signal <clk>           | rise     |
     | write enable       | connected to internal node          | high     |
     | address            | connected to signal <contadorE>     |          |
     | dual address       | connected to signal <contadorS>     |          |
     | data in            | connected to internal node          |          |
     | data out           | not connected                       |          |
     | dual data out      | connected to internal node          |          |
     | ram_style          | Auto                                |          |
     -----------------------------------------------------------------------
 INFO:Xst:1442 - HDL ADVISOR - The RAM contents appears to be read asynchronously. A synchronous
 read would allow you to take advantage of available block RAM resources, for optimized device
 usage and improved timings. Please refer to your documentation for coding guidelines.
     Found 128-bit register for signal <salida>.
     Found 5-bit subtractor for signal <$n0003> created at line 388.
     Found 6-bit adder for signal <$n0007> created at line 388.
     Found 6-bit adder for signal <$n0008> created at line 388.
     Found 4-bit comparator equal for signal <$n0010> created at line 402.
     Found 4-bit modulo-10 up counter for signal <contadorE>.
     Found 4-bit modulo-10 up counter for signal <contadorS>.
     Summary:
       inferred   1 RAM(s).
       inferred   2 Counter(s).
       inferred 128 D-type flip-flop(s).
       inferred   3 Adder/Subtractor(s).
       inferred   1 Comparator(s).
 Unit <fifo10wordaes> synthesized.

Synthesizing Unit <FIFOIN>.
    Related source file is "E:/proyec/rutadatos.vhd".
    Found 10x135-bit dual-port distributed RAM for signal <mem>.
    -----------------------------------------------------------------------
    | aspect ratio       | 10-word x 135-bit                   |          |
    | clock              | connected to signal <clk>           | rise     |
    | write enable       | connected to internal node          | high     |
    | address            | connected to signal <contadorE<3:0>> |          |
    | dual address       | connected to signal <contadorS<3:0>> |          |
    | data in            | connected to signal <entrada>       |          |
    | data out           | not connected                       |          |
    | dual data out      | connected to internal node          |          |
    | ram_style          | Auto                                |          |
    -----------------------------------------------------------------------
INFO:Xst:1442 - HDL ADVISOR - The RAM contents appears to be read asynchronously. A synchronous
read would allow you to take advantage of available block RAM resources, for optimized device
usage and improved timings. Please refer to your documentation for coding guidelines.
WARNING:Xst:1772 - You have explicitly defined initial contents for this RAM, which are currently
ignored when the RAM is implemented with LUT resources, leading to incorrect circuit behavior.
Changing the RAM description so that it is read synchronously will allow implementation on block
RAM resources for which we provide full initial contents support.
    Found 128-bit register for signal <clave>.
    Found 3-bit register for signal <op>.
    Found 4-bit register for signal <userid>.
    Found 32-bit adder for signal <$n0009> created at line 342.
    Found 32-bit adder for signal <$n0010> created at line 342.
    Found 31-bit comparator equal for signal <$n0013> created at line 352.
    Found 31-bit comparator not equal for signal <$n0018> created at line 352.
    Found 31-bit up counter for signal <contadorE>.
    Found 31-bit up counter for signal <contadorS>.
    Summary:
       inferred   1 RAM(s).
       inferred   2 Counter(s).
       inferred 135 D-type flip-flop(s).
       inferred   2 Adder/Subtractor(s).
       inferred   2 Comparator(s).
Unit <FIFOIN> synthesized.

Synthesizing Unit <FIFO10word>.    Related source file is "E:/proyec/rutadatos.vhd".
```

```
    Found 10x132-bit dual-port distributed RAM for signal <mem>.
    -----------------------------------------------------------------------
    | aspect ratio       | 10-word x 132-bit                    |          |
    | clock              | connected to signal <clk>            | rise     |
    | write enable       | connected to internal node           | high     |
    | address            | connected to signal <contadorE>      |          |
    | dual address       | connected to signal <contadorS>      |          |
    | data in            | connected to internal node           |          |
    | data out           | not connected                        |          |
    | dual data out      | connected to internal node           |          |
    | ram_style          | Auto                                 |          |
    -----------------------------------------------------------------------
INFO:Xst:1442 - HDL ADVISOR - The RAM contents appears to be read asynchronously. A
synchronous read would allow you to take advantage of available block RAM resources, for
optimized device usage and improved timings. Please refer to your documentation for coding
guidelines.
    Found 128-bit register for signal <salida>.
    Found 4-bit register for signal <salidaDir>.
    Found 5-bit subtractor for signal <$n0004> created at line 270.
    Found 6-bit adder for signal <$n0008> created at line 270.
    Found 6-bit adder for signal <$n0009> created at line 270.
    Found 4-bit comparator equal for signal <$n0013> created at line 277.
    Found 4-bit modulo-10 up counter for signal <contadorE>.
    Found 4-bit modulo-10 up counter for signal <contadorS>.
    Summary:
       inferred   1 RAM(s).
       inferred   2 Counter(s).
       inferred 132 D-type flip-flop(s).
       inferred   3 Adder/Subtractor(s).
       inferred   1 Comparator(s).
Unit <FIFO10word> synthesized.

Synthesizing Unit <arbolMem>.
    Related source file is "E:/proyec/rutadatos.vhd".
    Found 16x128-bit single-port distributed RAM for signal <tabla>.
    -----------------------------------------------------------------------
    | aspect ratio       | 16-word x 128-bit                    |          |
    | clock              | connected to signal <clk>            | rise     |
    | write enable       | connected to signal <WR>             | high     |
    | address            | connected to signal <dir>            |          |
    | data in            | connected to internal node           |          |
    | data out           | connected to internal node           |          |
    | ram_style          | Auto                                 |          |
    -----------------------------------------------------------------------
    Summary:
       inferred   1 RAM(s).
Unit <arbolMem> synthesized.
```

Tabla 4.8.5 Informe Versión Final

-. Memorias ROM que implementa la SBox.
-. Aparecen marcadas en rojo en la Figura 4.8.7.

```
Synthesizing Unit <sbox>.
    Related source file is "E:/proyec/proy.vhd".
    Found 256x8-bit ROM for signal <$n0001> created at line 57.
    Summary:
     inferred   1 ROM(s).
Unit <sbox> synthesized.
```

Tabla 4.8.6 Informe Versión Final

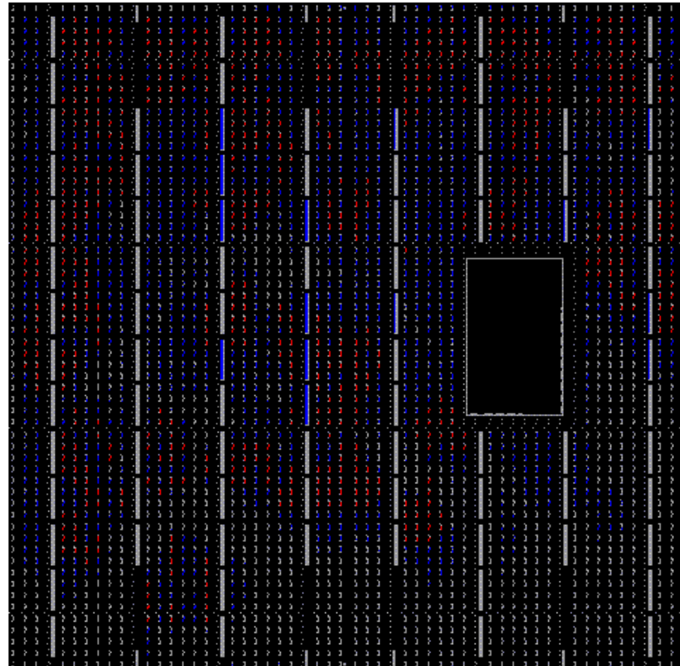


Figura 4.8.7 Área placa SBox Versión Final.

Podemos comprobar, como se ve a continuación, que aunque el módulo Generador/Encriptador ocupa la mayoría del sistema, las conexiones con el resto de los componentes están relativamente cerca. Esto ayuda a que el retardo de las conexiones sea menor.

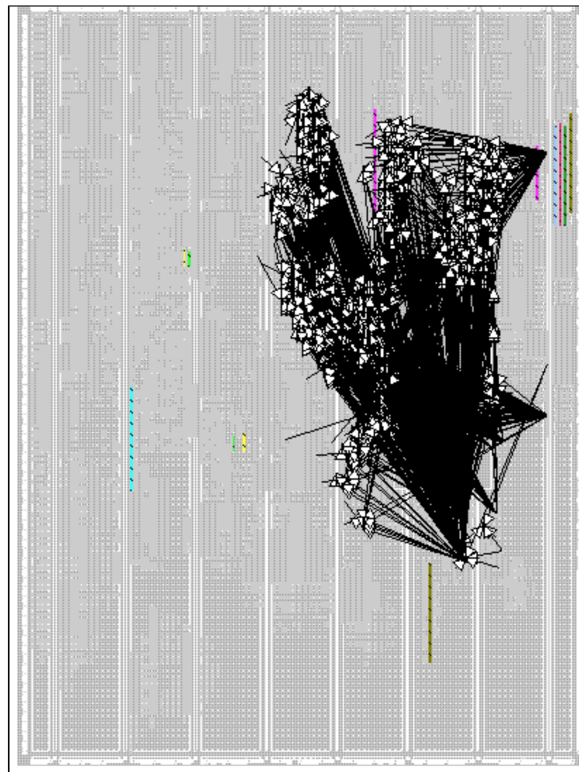


Figura 4.8.8 Conexiones placa Generador/Encriptador Versión Final

Como se acaba de mencionar el sistema completo sólo ocupa el 50% de la placa mientras que el AES combinacional no cabía en ella. La posibilidad de incluir un AES

combinacional que mejoraría el cálculo de las encriptaciones en 80% (para un ciclo de 20ns), pero habría que utilizar una placa más grande, y sería necesario el uso de un segundo reloj. Otra posibilidad sería modificar uno de los AES, el que esté más saturado, para que se realicen dos ciclos o vueltas consecutivas del AES en el mismo ciclo de reloj, así de los 10 ciclos el 2º y 3º, el 4º y el 5º, el 6º y el 7º y el 8º y el 9º se realizarían en un solo ciclo de reloj. Esto hace un total de 4 ciclos de reloj más el 1º y el 10º que son diferentes y un ciclo más por la carga de los registro reduciría a 7 ciclos el cálculo de la encriptación, pero que aumentaría seguramente el ciclo de reloj.
Estas posibles mejoras no producirían un cambio importante, ya que se tarda 24 ciclos en generar una nueva clave y 13 en encriptar, y de media se realizar algo más de 2 encriptaciones por generación (según la operación que se realice), lo que da lugar a que los tiempos totales de encriptaciones y generaciones sean muy parejos.

4.9. Mejoras para versiones futuras

- S-Box:

Hemos visto que las s-boxes son lo que más ocupa en nuestro diseño, una posible mejora sería una implementación más eficiente de éstas. Nosotros la implementamos mediante una tabla, pero la s-box del AES consiste en hallar el inverso multiplicativo en GF(256) y después hacer la operación xor con 0x63. Es posible descomponer esta operación en operaciones en GF(16) mediante tablas mucho más pequeñas que las utilizadas por nosotros o a su vez descomponerla en operaciones en GF(4) y en GF(2).

Otra posibilidad sería no realizar las 16 operaciones con la s-box simultáneamente para poder reutilizarlas.

Ambas opciones mejorarían el área del circuito pero empeorarían su velocidad.

- Optimización del control:

También sería posible buscar la operación más costosa que nos marca el tiempo de ciclo y partir ésta en dos ciclos, pudiendo así reducir el tiempo de ciclo.

- Diferente reloj para el generador:

Dado que el tiempo de ciclo del aes podría ser más corto que el del sistema, se podría utilizar un reloj diferente para cada uno, pudiendo así el generador realizar sus operaciones más rápidamente.

4.10. Conclusiones

Hemos conseguido implementar el sistema con los requisitos deseados, con éxito:

- En cuanto a requerimientos de velocidad; ya que el sistema permite realizar una unión en aproximadamente 2us. Esto también demuestra que la decisión de implementar el sistema en FPGA ha sido acertada.
- En cuanto a escalabilidad; porque, aunque nuestro sistema soporta sólo 8 usuarios, simplemente aumentando la memoria de claves podríamos manejar un número de usuarios mucho mayor sin gran pérdida de rendimiento, al tener las

operaciones de unión y desunión una complejidad de *O(log n).* Es de esperar, por ejemplo, que para un sistema con un millón de usuarios el sistema tarde unos 14us en realizar una unión.

- Por lo comentado anteriormente, el sistema demuestra ser práctico al ser el tiempo de unión y desunión de los usuarios lo suficientemente pequeño como para que éstos no puedan apreciarlo.

También hemos conseguido una implementación con una buena relación velocidad/área, gracias a la implementación del AES de forma secuencial que reduce drásticamente el área con una pérdida de velocidad aceptable. Este ahorro tiene dos consecuencias:

- La disminución de área permite incluir dos módulos AES, que mejora enormemente el rendimiento del sistema, al ser el uso del módulo para la encriptación y generación de claves un cuello de botella del sistema.

Al saturar menos los recursos de la FPGA, la herramienta de implementación consigue realizar mejor el enrutado, lo que produce un mejor rendimiento del sistema.

5.Bibliografía

[1] A. Shoufan and S. A. Huss A Scalable Rekeying. "Processor for Multicast Pay-TV on Reconfigurable Platforms". Workshop on Application Specific Processors, International Conference on Hardware/Software Codesign and System Synthesis, Stockholm, Sweden, September 2004.
[2] S. Setia, S. Koussih, S. Jajodia. "Kronos: A Scalable Group Re-Keying Approach for Secure Multicast". Center for Secure Information Systems George Masson University.
[3] www.wikipedia.org.
[4] Vincent Rijmen. "Eficient Implementation of the Rijndael S-box". Katholieke Universiteit Leuven, Dept. ESAT.
[5] AJ Elbirt, W Yip, B Chetwynd, C Paar. "An FPGA Implementation and Performance Evaluation of the AES Block Cipher Candidate Algorithm Finalists". Electrical and Computer Engineering Department.
[6] Von Eilert Backhus, Harald Würfel, Oliver Riesener und Prof. Dr. Stefan Wolter. "AES in FPGAs Implementierung des Advanced Encryption Standards in Hardware".

Los alumnos abajo firmantes autorizan a la Universidad Complutense a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la propia memoria, como el código, la documentación y/o el prototipo desarrollado:

Abelardo Palomino Guzmán, Ángel Manuel Romero Zamora, Alfonso Solbes Bosch