UNIVERSIDAD CARLOS III DE MADRID
ESCUELA POLITÉCNICA SUPERIOR

INGENIERÍA INDUSTRIAL
PROYECTO FIN DE CARRERA

"IMPLEMENTACIÓN HARDWARE DE PROTOCOLOS DE AUTENTIFICACIÓN PARA TECNOLOGÍA RFID"

Autor: Hugo Izquierdo Donoso
Tutor: Enrique San Millán Heredia

JUNIO 2012

ÍNDICE DE CONTENIDOS

CAPÍTULO 1 11
INTRODUCCIÓN 11
1.1 INTRODUCCIÓN DEL PROYECTO 12
1.2 OBJETIVOS DEL PROYECTO 13
1.3 ESTRUCTURA DEL CONTENIDO 13
1.4 PROJECT SUMMARY 14
CAPÍTULO 2 15
TECNOLOGÍA RFID 15
2.1 TECNOLOGÍA RFID 16
2.2 HISTORIA RFID 17
2.3 FUNCIONAMIENTO RFID 19
2.4 LA ETIQUETA RFID 21
2.5 COMPOSICIÓN DE LA TAG 24
2.6 EL LECTOR RFID 24
2.7 COLISIONES 26
2.8 FRECUENCIAS EN RFID 26
2.9 ÁREAS DE APLICACIÓN PARA RFID 28
2.10 CONCLUSIONES 29
2.11 AMENAZAS DEL SISTEMAS RFID 30
2.12 PROTOCOLOS DE AUTENTIFICACIÓN PARA RFID 30
2.14.1 PROTOCOLO GAO 31
2.14.2 PROTOCOLO Y.C LEE 31
2.14.3 PROTOCOLO CHEN AND DENG 32
CAPÍTULO 3 35
METODOLOGÍA DEL PROYECTO 35
3.1 ARQUITECTURA 36

3.2 VHDL Y SIMULACIÓN 36
3.3 SÍNTESIS 36
CAPÍTULO 4 38
PROTOCOLO DE AUTENTIFICACIÓN HMAC 38
4.1 DESCRIPCIÓN DEL PROTOCOLO 39
4.2 AMENAZAS PARA EL PROTOCOLO HMAC 40
4.3 DIAGRAMA DE FLUJO DEL PROTOCOLO HMAC 41
4.4 DESCRIPCIÓN FUNCIÓN HASH 43
4.5 DIAGRAMA DE FLUJO FUNCIÓN HASH 44
4.6 DIAGRAMA DE BLOQUES 46
4.7 REGISTROS USADOS EN LA FUNCIÓN HASH 47
4.8 DIFERENTES DESARROLLOS DE BITS 51
4.9 SIMULACIONES 52
4.10 RESULTADOS SÍNTESIS 63
4.10.1 RESULTADOS SÍNTESIS ALU 64
4.10.2 RESULTADO SÍNTESIS FUNCIÓN HASH 66
4.10.3 RESULTADO SÍNTESIS PROTOCOLO HMAC 66
4.11 CONCLUSIONES PROTOCOLO HMAC 69
CAPÍTULO 5 70
PROTOCOLO DE AUTENTIFICACIÓN AZUMI 70
5.1 DESCRIPCIÓN PROTOCOLO AZUMI 71
5.2 DIAGRAMA DE FLUJO PROTOCOLO AZUMI 73
5.3 REGISTROS USADOS EN EL PROTOCOLO 75
5.4 DIFERENTES DESARROLLOS DE BITS 77
5.5 DIAGRAMA DE BLOQUES 79
5.6 IMPLEMENTACIÓN PRNG'S 79
5.7 SIMULACIONES 83
5.8 RESULTADOS SÍNTESIS PROTOCOLO AZUMI 87

5.8.1 RESULTADOS SÍNTESIS PRNG 88
5.8.2 RESULTADO SÍNTESIS AZUMI_16 BITS 90
5.8.3 RESULTADO SÍNTESIS AZUMI_32 BIT 93
5.8.4 COMPARACIÓN AZUMI 16 – 32- 64 BITS 96
5.9 CONCLUSIÓN PROTOCOLO AZUMI 97
CAPÍTULO 6 99
COMPARACIÓN PROTOCOLOS HMAC – AZUMI 99
6.1 COMPARACIÓN RESULTADOS HMAC-AZUMI 100
CAPÍTULO 7 103
CONCLUSIONES Y LÍNEAS FUTURAS 103
7.1 CONCLUSIÓN 104
7.2 LÍNEAS FUTURAS 106
CAPÍTULO 8 107
PRESUPUESTO DEL PROYECTO 107
8.1 PRESUPUESTO DEL PROYECTO 108
CAPÍTULO 9 110
BIBLIOGRAFÍA DEL PROYECTO 110
9.1 BIBLIOGRAFÍA 111
ANEXOS 113
Tutorial ModelSim-Synopsys 113
Códigos 113

ÍNDICE DE FÍGURAS

Figura 1. Evolución RFID 19
Figura 2. Funcionamiento RFID 19
Figura 3. Comunicación RFID 20
Figura 4. Arquitectura Tag RFID 21
Figura 5. Bloques Tag RFID 21
Figura 6. Etiquetas RFID 23
Figura 7. Chip Etiqueta 24
Figura 8. Sistema RFID 25
Figura 9. Ancho de Frecuencia RFID 26
Figura 10. Aplicaciones Según Frecuencia RFID 27
Figura 11. Regiones RFID 28
Figura 12. Aplicaciones RFID 29
Figura 13. Protocolo Y.C Lee 31
Figura 14. Protocolo Chen & Deng 34
Figura 15. Diagrama de flujo: Desarrollo del proyecto 37
Figura 16. Diagrama de Flujo: HMAC 41
Figura 17. Diagrama de flujo HMAC 128 bits 42
Figura 18. Código C función Hash 43
Figura 19. Diagrama flujo: Función Hash 45
Figura 20. Hardware Protocolo Hmac 46
Figura 21. Hardware Función Hash 47
Figura 22. Bits Protocolo opción 1 51
Figura 23. Bits Protocolo opción 2 51
Figura 24. Bits protocolo opción 3 52
Figura 25. Hardware Alu 64
Figura 26. Protocolo Final HMAC 69
Figura 27. Funcionamiento Azumi 73
Figura 28. Diagrama de flujo Azumi 74
Figura 29. Protocolo 16 bits Azumi 77
Figura 30. Protocolo 32 Bits Azumi 78
Figura 31. Protocolo Azumi 64 bits 78
Figura 32. Hardware Azumi 79
Figura 33. Hardware PRNG 1 80
Figura 34. Hardware PRNG 2 81

Figura 35. Hardware PRNG 3..... 81
Figura 36. Hardware PRNG 4..... 82

ÍNDICE DE TABLAS

Tabla 1. Variables Protocolo Chen & Deng 32
Tabla 2. Valores Función A (Hash) 54
Tabla 3. Valores Función B (hash) 55
Tabla 4. Valores Función C (hash) 56
Tabla 5. Valores Función D (hash) 57
Tabla 6. Resultados Síntesis Alu 65
Tabla 7. Resultados Síntesis Función Hash 66
Tabla 8. Resultados de la síntesis Protocolo HMAC 67
Tabla 9. Número de Ciclos HMAC 68
Tabla 10. Variables Protocolo Azumi 72
Tabla 11. Resultado Síntesis PRNG's 88
Tabla 12. Número de ciclos PRNG's 89
Tabla 13. Resultado Síntesis Azumi 16 Bits 91
Tabla 14. Número de ciclos PRNG's 92
Tabla 15. Comparación Azumi - Prng's 93
Tabla 16. Resultado Síntesis Azumi 32 Bits 94
Tabla 17. Número de ciclos PRNG's (64 bis) 95
Tabla 18. Comparación Azumi (32 bits) - Prng's 95
Tabla 19. Puertas Equivalentes Protocolo Azumi 97
Tabla 20. Comparación Protocolos Azumi-HMAC 100
Tabla 21. Puertas Equivalentes Azumi-HMAC 100
Tabla 22. Número de ciclos HMAC-Azumi 101
Tabla 23. Selección Protocolo 105

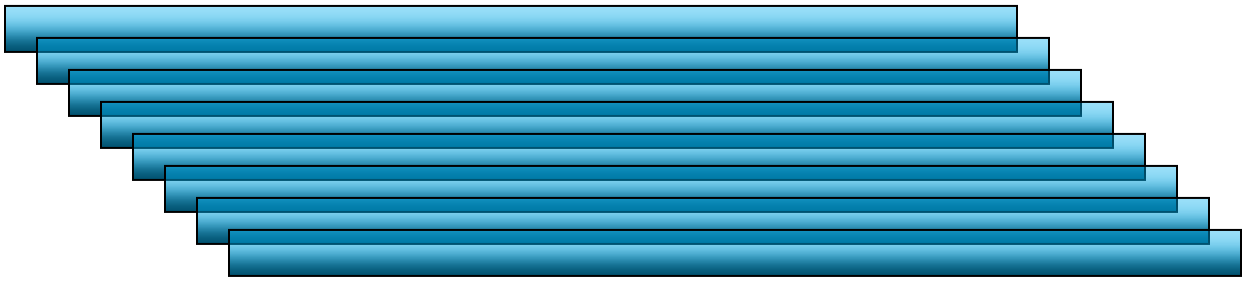
ÍNDICE DE GRÁFICAS

Gráfica 1. Puertas Equivalentes Alu_hash 65
Gráfica 2. Puertas Equivalentes HMAC 67
Gráfica 3. Comparación Protocolo HMAC 68
Gráfica 4. Área PRNG's 88
Gráfica 5. Número de ciclos PRNG's 89
Gráfica 6. Comparación PRNG's 90
Gráfica 7. Puertas Equivalentes PRNG's 91
Gráfica 8. Número de ciclos PRNG's 92
Gráfica 9. Comparación Azumi 16 bits 93
Gráfica 10. Puertas Equivalentes 32 bits 94
Gráfica 11. Número de ciclos PRNG's 32 bits 95
Gráfica 12. Comparación Azumi 32 bits 96
Gráfica 13. Puertas Equivalentes Azumi según Nº Bits 97
Gráfica 14. Puertas Equivalentes HMAC-Azumi 101
Gráfica 15. Comparación Protocolo HMAC-Azumi 102

ÍNDICE DE SIMULACIONES

Simulación 1. Función A (Función Hash) 53
Simulación 2. Función B (Función hash) 54
Simulación 3. Función C (Función hash) 56
Simulación 4. Función D (Función Hash) 57
Simulación 5. Variable Estado 58
Simulación 6. Actualización de la variable Estado 59
Simulación 7. Función Hash 60
Simulación 8. Protocolo Hmac 1 61
Simulación 9. Protocolo Hmac 2 62
Simulación 10. Protocolo Hmac 64 bits 62
Simulación 11. Protocolo Hmac 128 bits 63
Simulación 12. Prng 1 80
Simulación 13. Prng 2 81
Simulación 14. Prng 3 82
Simulación 15. Prng 4 83
Simulación 16. Protocolo Azumi Parte 1 84
Simulación 17. Protocolo Azumi Parte 2 85
Simulación 18. Protoclo Azumi Parte 3 86
Simulación 19. Protocolo Azumi 32 bits 87

CAPÍTULO 1



INTRODUCCIÓN

1.1 INTRODUCCIÓN DEL PROYECTO

Actualmente los Sistemas de Identificación Automáticas están tomando mucha importancia en las actividades cotidianas de la sociedad. Con estos sistemas conseguimos identificar cualquier producto, máquina, alimento, etc, a través de un código el cual nos da información acerca de su precio, fabricación, localización o cualquier dato que se considere necesario.

Los RFID son Sistemas de Identificación Automática que se caracterizan por transmisión entre lector y receptor a través de datos de radio frecuencia. En estos sistemas se establece dos líneas importantes de investigación como son el precio y la seguridad de la comunicación de los datos, es en éste último es en el cual se va a centrar este proyecto.

El objetivo del proyecto es implementar protocolos de encriptación seguros y ligeros para RFID, estos protocolos deberán tener que cumplir una serie de restricciones a la hora de implementar el diseño así como un cierto nivel de seguridad para poder asegurar la comunicación.

El algoritmo de los protocolos ya ha sido diseñado, por lo que en el proyecto se implementará esos algoritmos en código VHDL para poder evaluar si esos protocolos cumplen las restricciones de las etiquetas RFID de bajo coste.

Una vez evaluado los distintos protocolos implementados se tratará de comparar los distintos diseños para obtener la mejor opción para la aplicación en cuestión.

Se tomará como referencia la restricción de área para las tarjetas de bajo costes, las cuales está establecida en 3000-4000 puertas equivalentes. Además de este valor se obtendrán otros datos del circuito como son el número de ciclos o la potencia dinámica total.

A la hora de implementar el circuito se tendrá especial interés en el espacio ocupado, por ello se realizará un análisis exhaustivo de área ocupada justificando cada registro usado. De esta manera se asegurará que el circuito diseñado ocupará el menor área posible.

En cuanto a los protocolos los cuales se va a implementar serán:

a) Protocolo Hmac el cual se basa en una función Hash.
b) Protocolo Azumi el cual se basa en generadores pseudo-aleatorios PRNG.

Los algoritmos en los cuales se basan estos dos protocolos están establecidos en un número de bits específico, además también se han cambiado el número de bits de estos protocolos para poder obtener una comparación más amplia de los algoritmos. En temas de seguridad de datos mientras mayor sea el número de bits el modo de comunicación se considera más seguro, es por

ello que se ha intentado que el número de bits con el cual funciona el protocolo sea lo más grande posible.

Una vez obtenida los mejores diseños para cada protocolo se realizará la comparación entre los dos, obteniendo el más apropiado para implementarlo en un circuito o el más apropiado para alguna aplicación específica.

1.2 OBJETIVOS DEL PROYECTO

Los objetivos que se establecen para este proyecto son los siguientes:

a) Búsqueda de protocolos de autentificación que sean válidos para la tecnología RFID de bajo coste.
b) Implementación de los protocolos seleccionados.
c) Buscar diferentes arquitecturas en los protocolos que se implementarán.
d) Síntesis de los protocolos implementados.
e) Comparación de los protocolos y obtener el que mejor se adapte a este tipo de tecnología.

1.3 ESTRUCTURA DEL CONTENIDO

En cuanto a la primera parte del proyecto se presenta una introducción acerca de la tecnología RFID, en este apartado se presenta además de la historia de esta tecnología y sus características, algunos protocolos de encriptación en los cuales se basarán los implementados.

Posteriormente se muestran los dos protocolos que se han implementado. Dentro de cada protocolo se explica sus características, incluyendo diagrama de flujo, diagrama de bloques y justificación de los registros usados. Una vez realizado la descripción y comportamiento del protocolo se presentará las simulaciones en las cuales se muestra el funcionamiento del mismo. Después de asegurarse que la simulación del protocolo es la correcta se realizará la síntesis del circuito, de ahí se obtendrán los datos necesarios para ver si el protocolo implementado es válido o no.

Después de implementar los dos protocolos se hará una comparación entre estos, obteniéndose algunas conclusiones en función de las características que tenga cada diseño.

Este proyecto tiene como parte final una conclusión acerca de los datos extraídos de las implementaciones así como propuestas futuras y presupuesto del proyecto.

1.4 PROJECT SUMMARY

This Project is based on RFID (Radio Frecuency Identification). RFID has a lot of applications but currently it has two disadvantages. The first downside is that it is more expensive than other communication technology such as barcodes. The other problem with RFID is that it is less safe if we compare it with other kinds communication technology. We'll focus on this second disadvantage.

We are going to look for some authentication protocols for RFID and we'll do the design, implementation and synthesis of them. We'll check if these protocols are suitable for RFID otherwise the hardware architecture will be changed in order to improve the designs and to be able to use them in RFID.

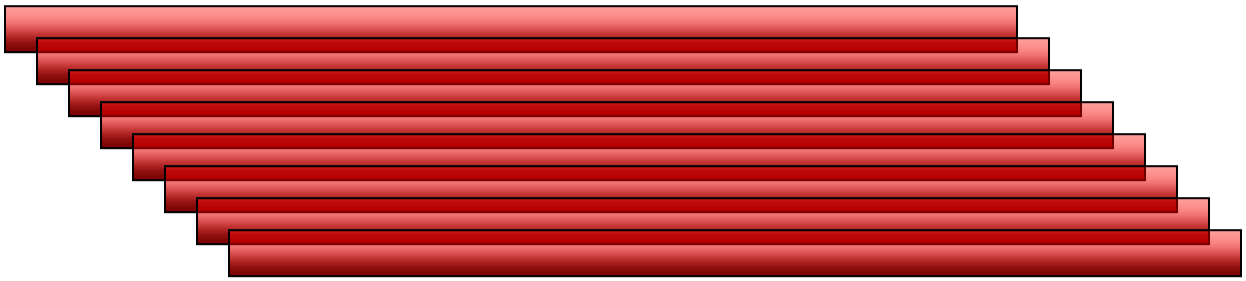
Two protocols will be designed, the first named HMAC which is based on hash function to generate random numbers. The second protocol implemented will be Azumi which is based on PRNG to generate random numbers.

There is a standard for this technology which says that there are some rules about the implementation inside the chip's tag. In this chip there is limited space therefore the most important restriction is that the circuit can't be bigger than 3000-4000 equivalent gates.

Also in the project there will be datas about timing and power, although the timing and power aren't very important in the standard, this data will be used to compare the protocols designed.
The goals of this project are to:

a) Find some protocols to use in RFID.
b) Implement these protocols.
c) Look for different architecture to improve the design.
d) Do the protocol synthesis. Make sure the design is right for the RFID, otherwise try to modify the protocol.
e) Finally, both protocols will be compared with each other and conclusions with be drawn about the protocols which have been designed and implemented.

CAPÍTULO 2



TECNOLOGÍA RFID

2.1 TECNOLOGÍA RFID

RFID (siglas de Radio Frequency IDentification, en español Identificación por radiofrecuencia) es un sistema de almacenamiento y recuperación de datos remoto que usa dispositivos denominados etiquetas, transpondedores o tags RFID. El propósito fundamental de la tecnología RFID es transmitir la identidad de un objeto (similar a un número de serie único) mediante ondas de radio. Las tecnologías RFID se agrupan dentro de las denominadas Auto ID (Automatic Identification, o Identificación Automática).

La tecnología RFID[1] está orientada a almacenamiento de datos y su transmisión, en el mercado existen varias tecnologías para introducir fácilmente la información de un producto en un sistema informático. La más popular es sin duda el código de barras, el cual es muy barato y fácil de implantar. No obstante presenta numerosos inconvenientes:

a) Puede almacenar poca información. Por ejemplo los códigos de barras usados en los productos alimentarios contienen sólo 12 dígitos.
b) Son relativamente difíciles de leer: es necesario orientar perfectamente el lector con el código para obtener una lectura correcta.
c) No pueden ser modificados una vez impresos.
d) Se deterioran fácilmente

Debido a esto, desde hace años se está popularizando una tecnología de identificación por radiofrecuencia (RFID) que presenta numerosas ventajas frente a los códigos impresos:

a) Es difícil destruir
b) Es fácil de leer, incluso es capaz de leer varios productos a la vez.
c) Contiene mayor cantidad de información
d) No necesita contacto directo, debido a que la lectura se realiza por radiofrecuencia.
e) Permite la lectura del objeto en movimiento.

[1] Para más información acerca de RFID consultar [1] [2] y [3] de bibliografía

Una de las pocas desventajas que encontramos en la tecnología RFID frente al código de barras es su coste, debido a que el código de barras es un sistema que se puede realizar por impresión mientras que la tecnología RFID necesita de un circuito integrado.

La gran mayoría de las etiquetas RFID son pasivas, que son mucho más baratas de fabricar y no necesitan batería. En 2004, estas etiquetas tenían un precio desde 0,40$, en grandes pedidos, para etiquetas inteligentes, según el formato, y de 0,95$ para tags rígidos usados frecuentemente en el sector textil encapsulados en PPs o epoxi. El mercado de RFID universal de productos individuales será comercialmente viable con volúmenes muy grandes de 10.000 millones de unidades al año, llevando el coste de producción a menos de 0,05$ según un fabricante. La demanda actual de chips de circuitos integrados con RFID no está cerca de soportar ese coste. Los analistas de las compañías independientes de investigación como Gartner and Forrester Research convienen en que un nivel de precio de menos de 0,10$ (con un volumen de producción de 1.000 millones de unidades) sólo se puede lograr en unos 6 u 8 años,lo que limita los planes a corto plazo para una adopción extensa de las etiquetas RFID pasivas. Otros analistas creen que esos precios serían alcanzables dentro de 10-15 años.

2.2 HISTORIA RFID

Se ha sugerido que el primer dispositivo conocido similar a RFID pudo haber sido una herramienta de espionaje inventada por Léon Theremin para el gobierno soviético en 1945. El dispositivo de Theremin era un dispositivo de escucha secreto pasivo, no una etiqueta de identificación, por lo que esta aplicación es dudosa. Según algunas fuentes, la tecnología usada en RFID habría existido desde comienzos de los años 1920, desarrollada por el MIT y usada extensivamente por los británicos en la Segunda Guerra Mundial (fuente que establece que los sistemas RFID han existido desde finales de los años 1960 y que sólo recientemente se había popularizado gracias a las reducciones de costos).

Una tecnología similar, el transpondedor de IFF, fue inventada por los británicos en 1939, y fue utilizada de forma rutinaria por los aliados en la Segunda Guerra Mundial para identificar los aeroplanos como amigos o enemigos. Se trata probablemente de la tecnología citada por la fuente anterior.

Las actividades comerciales empezaron en 1960, Sensormatic and Checkpoint junto con otras compañías equipos electrónicos para evitar robos. Estas tag solían usar 1 bit, por tanto solo se podía usar como "detección" o "no-detección" de la tarjeta.

En los años 70, investigadores, compañías, laboratorios y universidades estuvieron trabajando en la investigación de la tecnología RFID, hubo notables avances gracias a la investigación conjunta de estos grupos de investigación.

La autoridad portuaria de New York y New Jersey probaron los sistemas diseñados por General Electric con resultados favorables, aunque la aplicación para el transporte todavía no usaba este tipo de tecnología.

En los 80, hubo muchos grupos de investigación que poco a poco se iban interesando por la tecnología RFID. Las aplicaciones más interesantes en EEUU fueron para el transporte, acceso de personal y en menor medida para el control de animales. Más tarde en EEUU se empezó a utilizar para el control en carreteras. Por otro lado en Europa, las aplicaciones se centraban en las industrias y negocios.

En los años 90, gracias a la tecnología CMOS, permitió grandes avances en el diseño de circuitos integrados en las etiquetas

Entre los años 2000 y 2010, se siguen investigando en las distintas aplicaciones así como numerosas investigaciones para la mejora de la tecnología y la expansión de la tecnología RFID en el mundo actual.

Decade	Event
1940 - 1950	Radar refined and used, major World War II development effort. RFID invented in 1948.
1950 - 1960	Early explorations of RFID technology, laboratory experiments.
1960 - 1970	Development of the theory of RFID. Start of applications field trials.
1970 - 1980	Explosion of RFID development. Tests of RFID accelerate. Very early adopter implementations of RFID.
1980 - 1990	Commercial applications of RFID enter mainstream.
1990 - 2000	Emergence of standards. RFID widely deployed. RFID becomes a part of everyday life.

Figura 1. Evolución RFID

2.3 FUNCIONAMIENTO RFID

El modo de uso de la tecnología RFID es similar al tradicional código de barras. Al producto que se desea identificar se le añade una etiqueta y se utiliza un lector conectado a un ordenador para obtener la información de identificación automáticamente. No obstante, las similitudes terminan ahí: tanto la etiqueta como el lector son totalmente diferentes. El principio de funcionamiento es el siguiente: el lector emite una señal electromagnética que al ser recibida por la etiqueta hace que ésta responda mediante otra señal en la que se envía codificada la información contenida en la etiqueta.

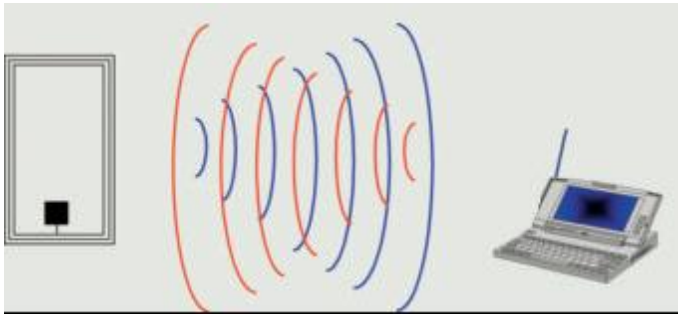


Figura 2. Funcionamiento RFID

El funcionamiento de la tecnología RFID se resumen a continuación:

a) El lector manda una señal de interrogación a la etiqueta.
b) La etiqueta utiliza usa la energía de esta señal para funcionar, y su frecuencia como reloj.
c) La etiqueta lee los datos del lector, en caso de que existan.
d) El RFID contesta con su propia información.
e) Un protocolo de autentificación permite gestionar la respuesta simultánea de múltiples etiquetas.
f) Un protocolo de seguridad permite dar seguridad a la información que se transmite entre el lector y la tarjeta.
g) La información recibida se integra con el resto de sistemas de información.

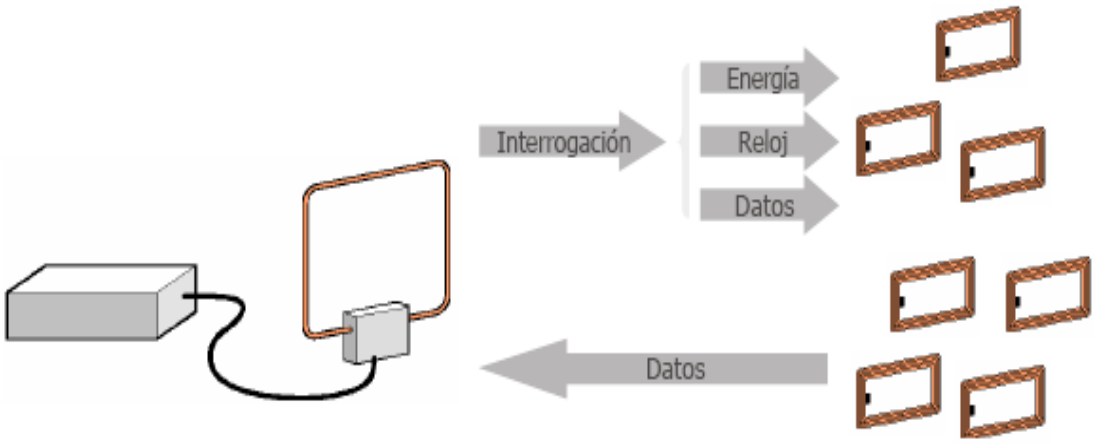


Figura 3. Comunicación RFID

2.4 LA ETIQUETA RFID

En una etiqueta RFID podemos distinguir tres elementos: la antena, el circuito integrado y el elemento almacenador de energía.

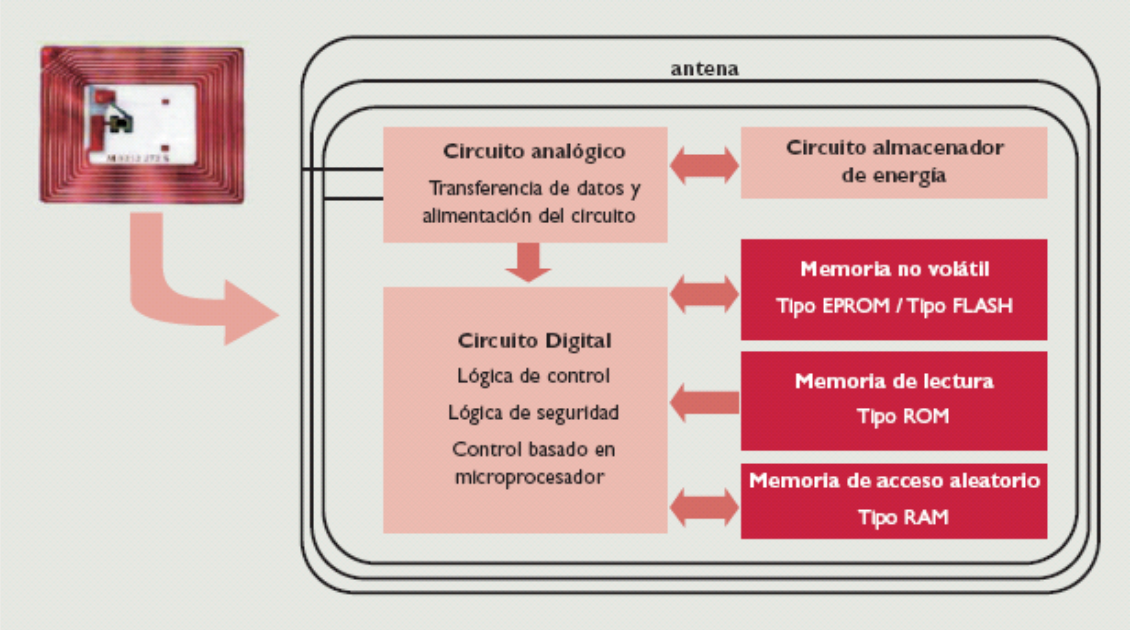


Figura 4. Arquitectura Tag RFID

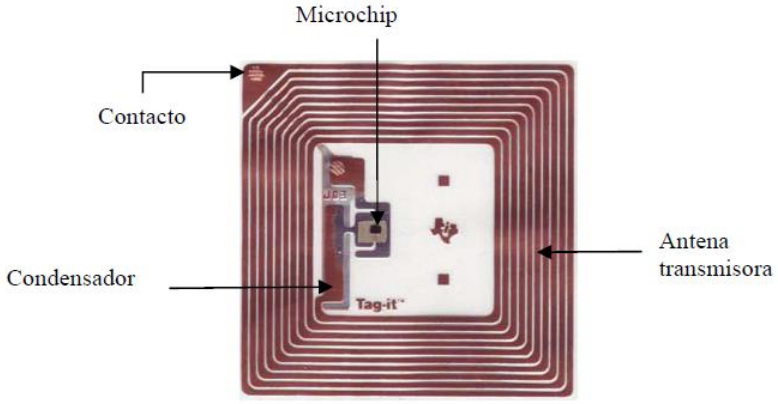


Figura 5. Bloques Tag RFID

La antena permite realizar la comunicación entre la etiqueta y el lector. Su tamaño limita la distancia máxima a la que puede realizarse la lectura.

El circuito integrado es un circuito mixto analógico- digital. La parte analógica se encarga de controlar la alimentación y la comunicación por radiofrecuencia. Por otro lado, la parte digital gestiona la información almacenada en la etiqueta.
Por último, es necesario incluir un elemento para alimentar al circuito. En función del elemento usado existen dos tipos de etiquetas: las activas y las pasivas.

En las etiquetas pasivas, el elemento almacenador de energía es un condensador, el cual se carga con la energía emitida por el lector y luego utiliza dicha energía para responder. Por ello, la potencia de emisión está limitada, por lo que la distancia entre el lector y la etiqueta no puede ser muy elevada. La ventaja obvia de este tipo de etiquetas es el ahorro de espacio, la duración prácticamente ilimitada de la etiqueta y su menor coste. Debido a esto, éstas son las etiquetas más usadas, aplicándose en campos tan diversos como la identificación de animales, llaves de contacto de automóviles, identificación de productos en cadenas de montaje, control de accesos, cronometraje de carreras, etc. Obviamente, los componentes mencionados han de protegerse del ambiente exterior, por lo que en función de la aplicación habrá de elegirse el encapsulado adecuado. El más sencillo es el que se ha mostrado en la Figura 2 consistente en una lámina de plástico. No obstante los fabricantes ofrecen innumerables encapsulados, incluso a medida del cliente. Por ejemplo existen etiquetas en formato tarjeta de crédito para control de accesos, encapsuladas en una ampolla de vidrio para identificar animales, etiquetas en forma de clavo para palets, encapsulados resistentes a altas temperaturas para etiquetar equipos que tengan que soportar condiciones adversas, etc.

A diferencia de los tags pasivos, los activos poseen su propia fuente autónoma de energía, que utilizan para dar corriente a sus circuitos integrados y propagar su señal al lector. Estos tags son mucho más fiables (tienen menos errores) que los pasivos debido a su capacidad de establecer sesiones con el reader. Gracias a su fuente de energía son capaces de transmitir señales más potentes que las de los tags pasivos, lo que les lleva a ser más eficientes en entornos dificultosos para la radiofrecuencia como el agua (incluyendo humanos y ganado, formados en su mayoría por agua), metal (contenedores, vehículos). También son efectivos a distancias mayores pudiendo generar respuestas claras a partir de recepciones débiles (lo contrario que los tags

pasivos). Por el contrario, suelen ser mayores y más caros, y su vida útil es en general mucho más corta.

Muchos tags activos tienen rangos efectivos de cientos de metros y una vida útil de sus baterías de hasta 10 años. Algunos de ellos integran sensores de registro de temperatura y otras variables que pueden usarse para monitorizar entornos de alimentación o productos farmacéuticos. Otros sensores asociados con ARFID incluyen humedad, vibración, luz, radiación, temperatura y componentes atmosféricos como el etileno. Los tags, además de mucho más rango (500 m), tienen capacidades de almacenamiento mayores y la habilidad de guardar información adicional enviada por el transceptor.

Actualmente, las etiquetas activas más pequeñas tienen un tamaño aproximado de una moneda. Muchas etiquetas activas tienen rangos prácticos de diez metros, y una duración de batería de hasta varios años.

En la siguiente figura se muestra el interior de las etiquetas utilizadas para la tecnología RFID.

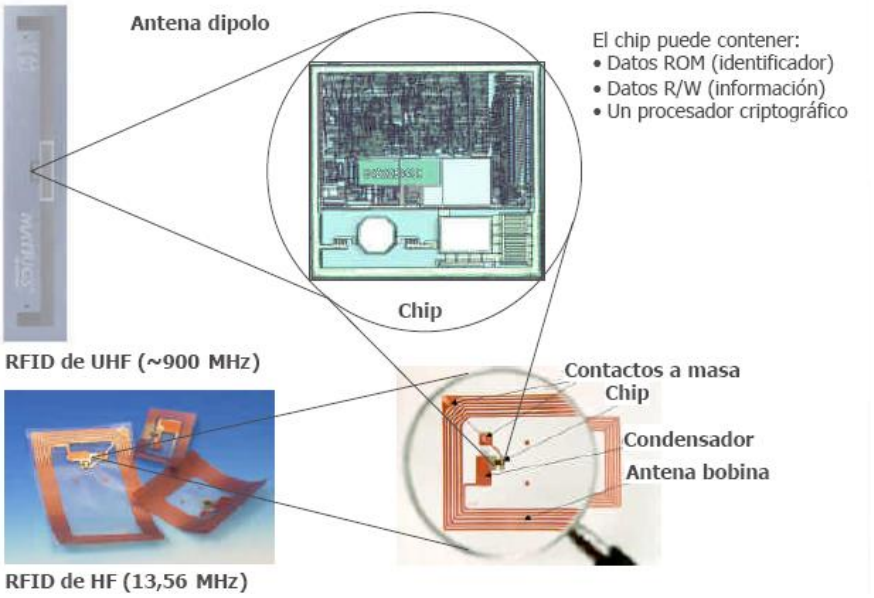


Figura 6. Etiquetas RFID

2.5 COMPOSICIÓN DE LA TAG

En el siguiente esquema se muestra un esquema de cómo se puede insertar en chip en el conjunto de la tag [4].

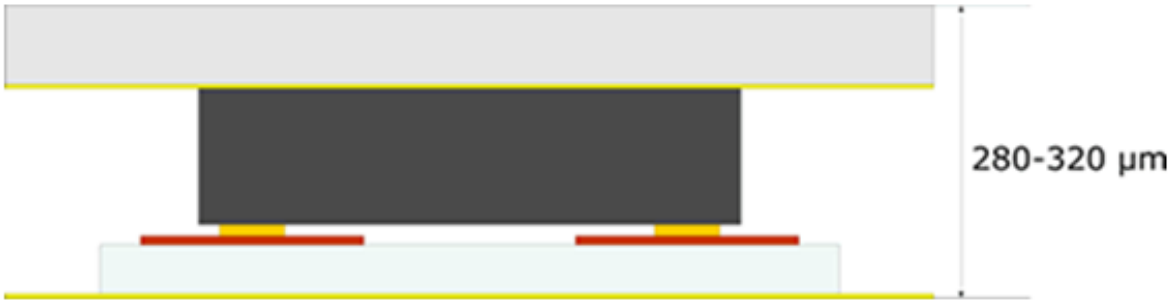


Figura 7. Chip Etiqueta

Las diferentes partes de las que se componen la tag son:

a) Cara frontal (papel, material sintético, …) donde se imprime información además de dar protección al chip.
b) Chip RFID: la zona donde se encuentra el circuito y se almacena la información.
c) Bumps: son los soportes del chip a la tag, se suele aplicar unas gotas de adhesivos especiales para cumplir con las características eléctricas.
d) Antena impresa: sirve para captar la onda electromagnética RFID.
e) Capa dieléctrica de 50 micras, normalmente de PET, que hace de soporte de la antena con el chip.
f) Adhesivo para unir la etiqueta a su destino.

2.6 EL LECTOR RFID

La estructura del equipo de lectura es similar a la de las etiquetas: es necesaria una antena para comunicarse con la etiqueta y un circuito para gestionar la comunicación. Este circuito dispone de un interfaz estándar, como por ejemplo RS-232 o CompactFlash, para conectarse a un ordenador o a una PDA. Además existen en el mercado equipos lectores con la antena integrada y equipos que admiten antenas externas, que pueden seleccionarse en función de la aplicación.

En este último caso, existen dos tipos de antenas, las antenas tipo "cuadro", similares a las usadas en los grandes almacenes para evitar robos y las antenas con núcleo de ferrita. Las primeras, al tener un mayor tamaño pueden alcanzar distancias de lectura del orden de un metro, aunque con poca direccionabilidad. Las segundas en cambio tienen un tamaño más reducido por lo que su rango se reduce a unos pocos centímetros. Sin embargo su menor tamaño permite su uso en equipos portátiles. Además son más direccionales, por lo que se pueden leer etiquetas que estén próximas entre sí.

El último componente en un sistema de identificación por RFID es el sistema de proceso de datos. Las etiquetas sólo de lectura devuelven un código único grabado "a fuego" al fabricar el chip (un tamaño típico es 64 bits.) Las etiquetas de lectura/escritura, que son más caras que las anteriores, permiten leer y/o escribir una longitud mayor de información (valores típicos son 32, 256 o 2048 bits). Por tanto será necesario usar algún sistema de bases de datos que realice la correlación entre la información devuelta por la etiqueta y el resto de información del producto. A modo de conclusión, en la siguiente figura se resume en forma gráfica el sistema completo RFID.

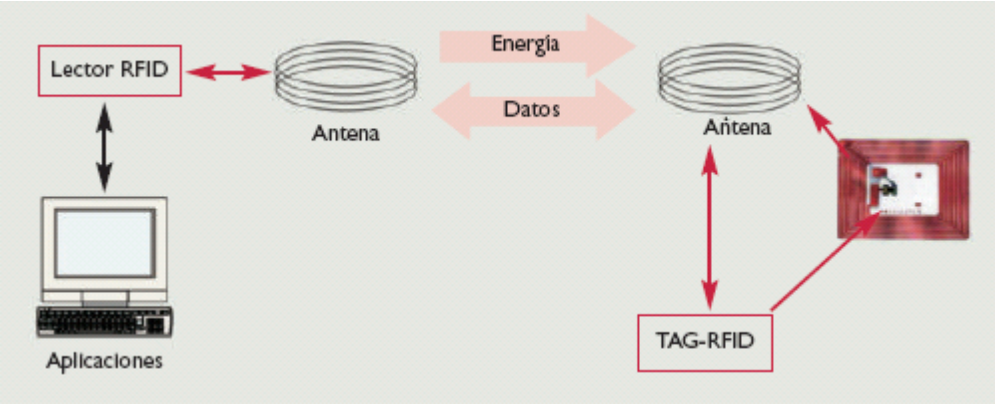


Figura 8. Sistema RFID

2.7 COLISIONES

Las etiquetas más sencillas necesitan ser leídas de una en una, ya que si se sitúa más de una dentro del rango de alcance de la antena del lector, ambas interferirán entre sí y no podrá realizarse la lectura. Obviamente este tipo de etiquetas son más baratas y para muchas aplicaciones son más que suficientes. Sin embargo, existen aplicaciones en las que puede ser necesario leer varias etiquetas a la vez. Ejemplos de este tipo de aplicaciones son los sistemas de cronometraje, en los que varios corredores pasan simultáneamente por la antena lectora; el etiquetado de equipajes, etc. Para este tipo de aplicaciones se han desarrollado etiquetas anticolisión que permiten la lectura de varias etiquetas situadas en el rango de alcance de la antena. Además la lectura es bastante rápida

2.8 FRECUENCIAS EN RFID

Las frecuencias utilizadas para esta tecnología RFID es la mostrada en la figura siguiente:

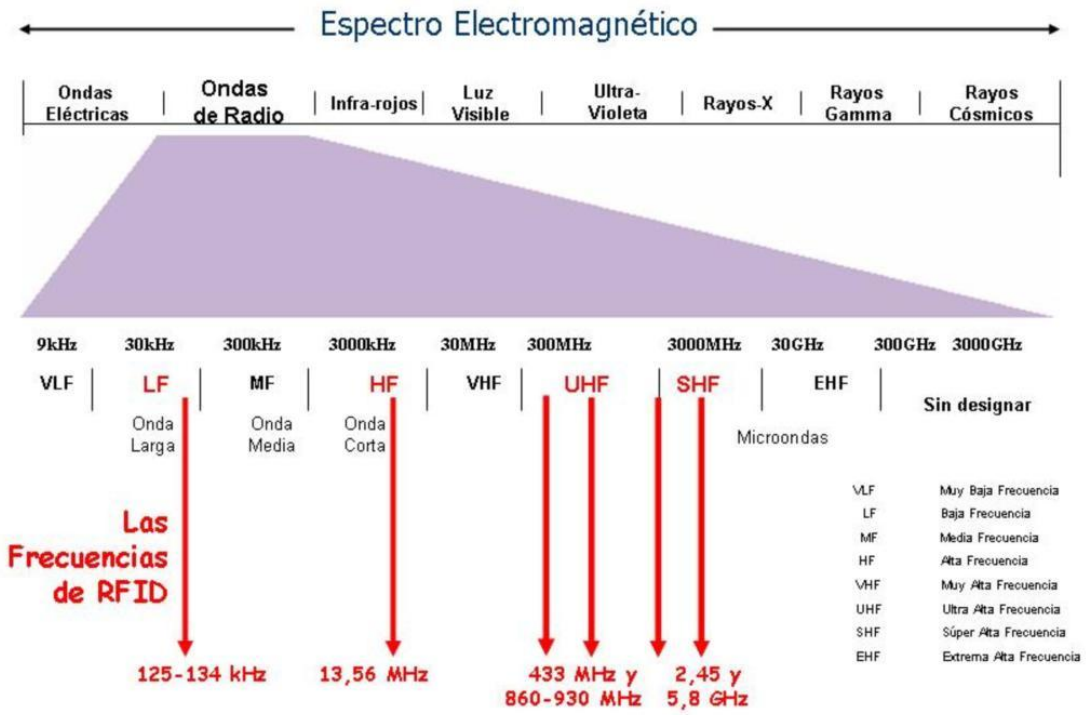


Figura 9. Ancho de Frecuencia RFID

Los sistemas de comunicaciones están estandarizados por organismos internacionales (UIT, FCC, CCITT, etc.) que regulan su uso y operatividad. Además, estos organismos son también los encargados de repartir el espectro de frecuencia entre todas las aplicaciones que lo necesitan. Para las aplicaciones de RFID, se han asignado tres bandas principales de frecuencias, que se resumen en la Tabla 1, donde se indican además las características típicas de los sistemas que usan dicha banda, así como sus aplicaciones típicas.

Banda de frecuencia	Características	Aplicaciones típicas
Baja 100-500 KHz	Lectura para corta y media distancia Sistemas con tags económicos Velocidad de lectura baja	Control de acceso Identificación de animales Control de existencias Inmovilizadores de automóviles
Intermedia 10-15 MHz	Lectura para corta y media distancia Potencialmente barato Velocidad de lectura media	Control de acceso Tarjetas inteligentes
Alta 850-950 MHz 2.4-5,8 GHz	Lectura para corta y media distancia Velocidad de lectura alta Línea de vista requerida Tecnología cara	Supervisión en sistemas ferroviarios y automotriz.Acceso y control de peaje

Figura 10. Aplicaciones Según Frecuencia RFID

En la tabla anterior se han mostrado unos márgenes de frecuencia muy amplios en los
que se encuadran los rangos asignados para RFID en cada uno de los países. Así, en función de la zona en la que estemos situados tendremos que usar un rango de frecuencia u otro, lo que implica usar etiquetas distintas. En la Figura 4 se muestran las tres regiones en las que se divide el planeta en función de las regulaciones de radiofrecuencia: Europa y África (Región 1), América (Región 2) y lejano oriente con Australasia (Región 3). Afortunadamente los fabricantes disponen de versiones de cada una de sus etiquetas para las distintas regiones. Por ejemplo las etiquetas Ucode HSL de Philips están disponibles para una banda de 869 MHz (región 1) y para una banda de 900 MHz (región 2)

Figura 11. Regiones RFID

Otro parámetro importante que varía entre regiones es la máxima potencia de emisión, la cual está directamente relacionada con la distancia máxima de lectura de las etiquetas. Siguiendo el ejemplo anterior, en la región 1, donde para la banda de 869 MHz se admite una potencia máxima de emisión de 0,5 W, la distancia máxima para un tamaño típico de antenas es de 4m. En cambio las mismas etiquetas en la región 2, en donde la potencia máxima permitida es de 4 W, pueden leerse desde 8,4 m.

2.9 ÁREAS DE APLICACIÓN PARA RFID

El uso potencial de RFID es prácticamente ilimitado en cada sector de la industria, comercio y servicios donde existen datos que deben ser leídos o comprobados. Las áreas principales de aplicación de RFID son:

a) Transporte y logística.
b) Fabricación y procesamiento.
c) Seguridad de personas.
d) Identificación y trazabilidad alimentaria animal.
e) Rastreo postal.
f) Verificación y control de equipaje.
g) Control de peaje y medios de pago electrónico.
h) Sustitución o uso simultáneo y compartido con códigos de barras.
i) Vigilancia electrónica.
j) Control de accesos y un largo etc.

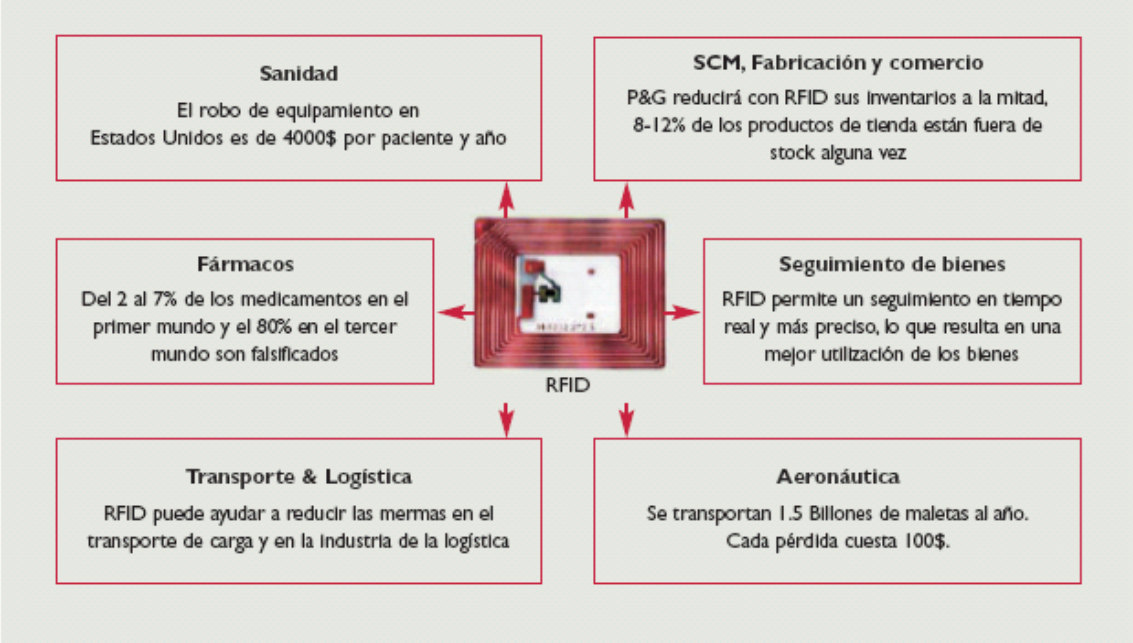


Figura 12. Aplicaciones RFID

2.10 CONCLUSIONES

Hasta ahora se ha realizado una introducción a la tecnología RFID, la cual presenta numerosas ventajas frente a los métodos tradicionales de identificación como el código de barras. Se ha mostrado que su característica fundamental consiste en la utilización de etiquetas "inteligentes" capaces de almacenar mayor cantidad de información de forma segura y perdurable. No obstante, aunque esta tecnología se vislumbra como el futuro en las aplicaciones de identificación, aún necesita superar algunas barreras para que se popularice aún más. Aparte de los problemas de estandarización que se resolverán con el tiempo, el principal escollo es el precio de las etiquetas, que es claramente desfavorable frente al coste cero del código de barras. No obstante, la mayor facilidad de uso hace que el coste total de la aplicación pueda ser menor. Por ejemplo, una aplicación que impactaría en nuestra vida diaria sería el uso de etiquetas RFID para identificar los productos de un supermercado. Con este sistema bastaría con pasar por la antena lectora para que se registrasen todos los productos del carro. El día en que el precio de las etiquetas sea menor que el coste del tiempo empleado en pasar el producto por el lector de código de barras, veremos este tipo de etiquetas cada vez que hagamos la compra.

Por tanto para que esta tecnología se considera que hay dos posibles ramas de investigación: por un lado la reducción del coste de fabricación, y por otro lado el aumento de la seguridad en el envío de datos, y es en este último aspecto en el cual se va a centrar este proyecto.

2.11 AMENAZAS DEL SISTEMAS RFID

Debido a la comunicación inalámbrica en el cual se basa esta tecnología, esta es vulnerable a los ataques. Los principales riesgos para el RFID son [5] [6]:

a) Suplantación de identidad: La identidad falsa actúa al principio como lector y envía una señal a la tag para iniciar la comunicación, este recibe una respuesta de la tag. Cuando el lector verdadero envía una señal a la tag el falso lector envía la respuesta que tenía de la tag, el lector falso obtendrá la autentificación. Esto se puede evitar utilizando autentificación de encriptación de datos.
b) Repetición de ataques: Cuando el lector envía una señal a la tag, el lector falso obtendrá la respuesta de la tag por espionaje. En la próxima vez que se quiera realizar la comunicación el lector falso enviará la información robada de la tag y obtendrá la autentificación. Esto se puede solucionar a través de temporizadores y contadores.
c) Rastreo: El intruso hace un seguimiento de la tag en su respuesta, intentando conseguir la privacidad de la tag.
d) Desincronización: El ataque intenta destruir la información de la back-database de la tag destruyendo la utilización del protocolo.

2.12 PROTOCOLOS DE AUTENTIFICACIÓN PARA RFID

Para conseguir la suficiente seguridad en la cual el robo de información no sea posible, se usan protocolos de autentificación, con los cuales se quiere conseguir que la transmisión de la información se realice de forma segura.

En este proyecto se implementarán 2 protocolos de autentificación, los cuales son basados en otros sistemas de autentificación más antiguos y menos seguros.

El protocolo HMAC[6] se basa en sistemas de autentificación como son The GAo, The Chen y Y.C.Lee

Por otro lado el protocolo Azumi tiene su origen en sistemas de autentificación como es Chen and Deng [13].

2.14.1 PROTOCOLO GAO

En 2004 Gao propuso un protocolo de seguridad para RFID. En él, se tiene el identificador del lector (ID_r) y el identificador de la tarjeta (ID_t).

El ID_r es almacenado en la memoria de la tag, y en el back-database se sabe el ID_t. Hay un generador de números aleatorios implementado en la tag. Este protocolo no puede asegurar la privacidad del usuario, en el momento en el que la tag envía un mensaje fijo $h(ID_t)$ al lector. Si algún intruso espía el mensaje, podría rastrear a la tag.

2.14.2 PROTOCOLO Y.C LEE

En 2008 Y.C LEE, propuso un protocolo en el cual mejoraba los defectos que tenía el protocolo Chen, en este nuevo protocolo se impedía el seguimiento y la suplantación de identidad, esto se conseguía a través de tener diferentes valores hash durante cada autentificación.

En la siguiente figura se muestra el protocolo.

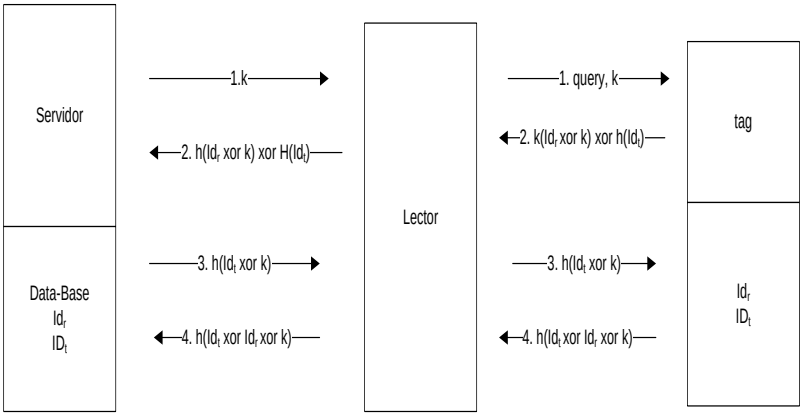


Figura 13. Protocolo Y.C Lee

Aquí como k, $h(ID_r \text{ xor } k) \text{ xor } h(Idt)$, $h(ID_t \text{ xor } k)$ y $h(Id_t \text{ xor } ID_r \text{ xor } k)$ siempre cambian durante cada identificación, esto dificulta que el atacante pueda seguir a la tag con el cual descrifrar el mensaje. Al mismo tiempo, debido a que el número aleatorio k es cambiado cada vez, el

protocolo puede resistir ataques de espionaje. Para que este protocolo sea lo suficientemente seguro se considera que necesita 4 cálculos de la función hash, debido al área limitada de la tag y a la baja capacidad de esta se considera que este protocolo es ineficiente.

2.14.3 PROTOCOLO CHEN AND DENG

En 2009 Chen y Deng propusieron un protocolo de autentificación en el cual usaba un generador de número pseudo-aleatorios y un código de redundancia cíclica, tal y como recomienda el estándar EPC-C1G2.

Usará las siguientes señales:

Variables	Valor
X_t (j)	Valor x de la tag t registrada en la database j
$N_t(j)$, $K_t(j)$	$N_t(j)$ es una palabra y $k_t(j)$ es la llave de cada t
CRC()	Función de codigo de redundancia ciclica
EPCt	Número de identificación para cada tag t
Idr	Número de identificación para cada lector
RND	Número aleatorio
xor	Operación xor
M_{req}	Mensage solicitado del lector
M_{resp}	Mensage respondido del lector

Tabla 1. Variables Protocolo Chen & Deng

Se componen de dos fases, una fase de registro y otra de comunicación.

Fase Registro

Las tag y lectores deben registrar en la database por separado a través de un entorno seguro. La tarjeta envía el EPC a la database. La database responde con N_t y $k_t(j)$ por cada tag que pide registrarse. El lector es registrado en la database bajo un único ID_r. Después del registro la database responde con (N_t,K_t) para todas las tarjetas asignadas que pueden ser accesibles al lector ID_r.

Fase de Comunicación.

Una vez terminada la etapa de la fase de registro la tag y el lector pueden comunicarse. Solamente números aleatorios, operaciones xor, y funciones ligeras CRC son utilizadas para el cambio en los mensajes en la comunicación, como es solicitado por el EPC-C1G2 estándar. Esta fase se pude dividir en 5 pasos.

a) Cuando el lector quiere acceder a la tarjeta envía un mensaje de petición M_{req}, CRC/N_t xor RND_1) y RND_1 a la tag.
b) Un vez recibido, la tag usa el valor almacenado N_t' para calcular $CRC(N_t$ xor $RND_1)$. Además la tag puede autentificar al lector a través de la siguiente verificación.
   $CRC(N_t$' xor $RND_1)$?= $CRC(N_t$ xor $RND_1)$.
   Si esta comparación no es la misma, la tag interpreta que un intruso quiere accede a los datos.
   La tag calcula un número aleatorio RND_2 y calcula:
   $X= k'_t$ xor EPC_t xor RND_2
   $Y= CRC(RND_2$ xor N'_t xor $X)$
c) La tag envía $\{RND_2,X,Y\}$ al lector.
d) Una vez recibido el mensaje, el lector calcula Y, usando los valores RND_2 y X obtenido de la tag. Si el Y calculado no coincide con el valor Y recibido se considera que el mensaje es enviado por un intruso y el lector no seguirá respondiendo. En caso contrario el lector usa K_t y $\{RND_2,X\}$ para obtener el valor identificador de la tag T.
   $EPC_t= K_i$ xor RND_2 xor X
e) Cuando el lector obtienes el valor EPC_t, y la autentificación de la tarjeta ha sido confirmado y lector envía un respuesta M_{resp} a la tag.

En el siguiente diagrama se muestra un proceso en las comunicaciones durante este protocolo:

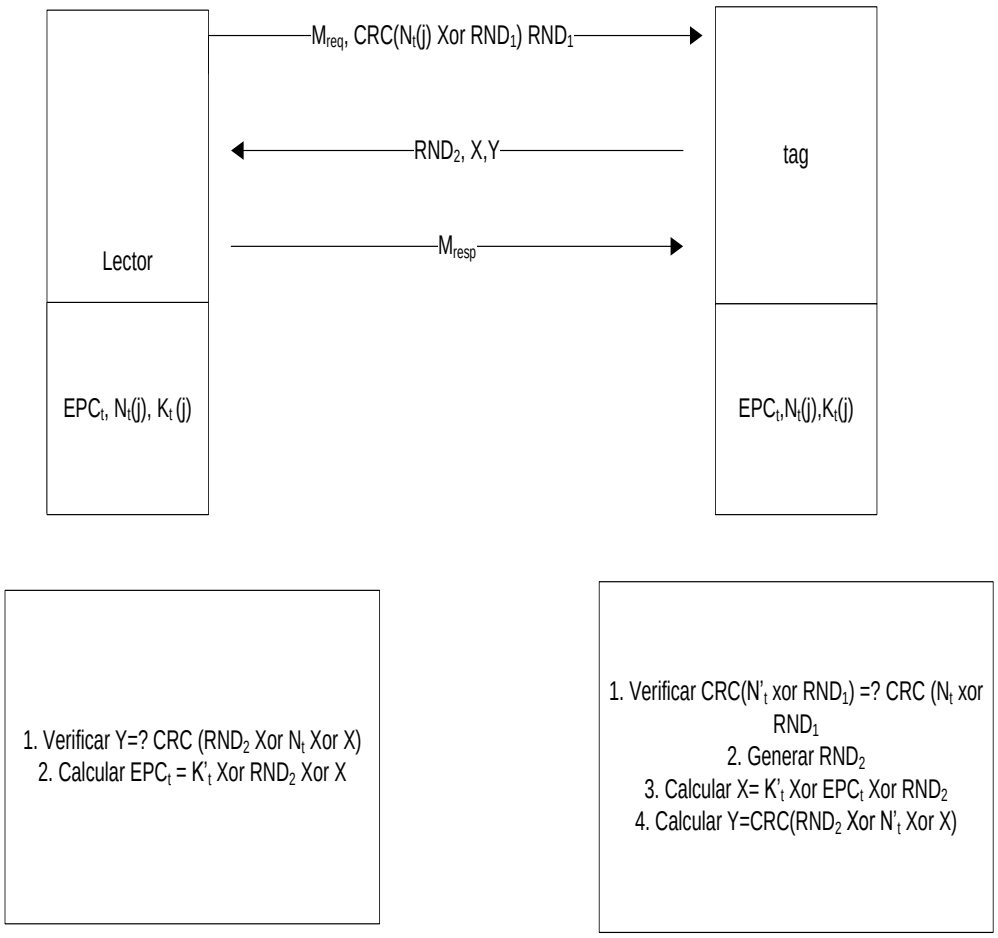
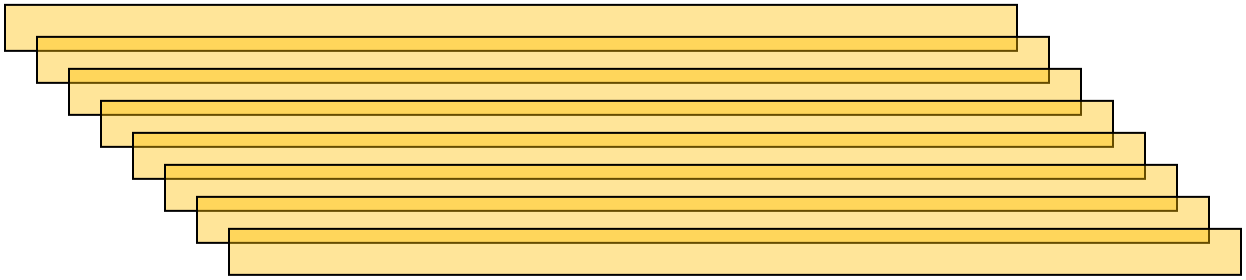


Figura 14. Protocolo Chen & Deng

CAPÍTULO 3



METODOLOGÍA DEL PROYECTO

3.1 ARQUITECTURA

Para la arquitectura de los protocolos diseñados se tomarán en cuenta las siguientes características:

a) Área: Se tiene la restricción del estándar EPC-C1G2 [20] [21] que solo nos permite 3000-4000 puertas equivalentes para las etiquetas de bajo coste para RFID. Por tanto será el valor más restrictivo de los estudiados ya que limita el circuito.
b) Tiempo: Se estudiará el número de ciclos que tarda cada circuito, sabiendo que la frecuencia aproximada según el estándar es de 100Khz se podrá obtener el tiempo total de cálculo del protocolo.
c) Consumo: Este valor se utilizará para comparar ambos protocolos y poder obtener cual tiene menores valores tanto de pérdidas como de potencia dinámica.

3.2 VHDL Y SIMULACIÓN

La implementación de los protocolos se hará en VHDL para posteriormente realizar la simulación y comprobar que el circuito diseñado cumple con las características de la bibliografía.

3.3 SÍNTESIS

Para la realización de la síntesis de los circuitos implementados se utilizará la herramienta Synopsys [17] [18] [19]. Se ha utilizado una biblioteca de fabricación, gracias a esto se conoce el layout de las celdas y por tanto se tendrán unos valores muy exactos en consumo, retraso y área. Hay que destacar que el área es el factor más importante en este proyecto ya que es el valor que limita el diseño del circuito. Synopsys nos da información acerca del área total del circuito, debido a que nos interesa conocer el valor de puertas equivalentes se pasará el valor de área total a número de puertas equivalentes. Se conoce que el área de la puerta equivalentes más pequeña es la puerta NAND, para obtener el número de puertas equivalentes se divide el valor total del circuito entre lo que ocupa una puerta NADN, el resultado de esta división nos dará el número de puertas equivalentes. En este proyecto se ha trabajado con una tecnología de 90nm, en la cual se considera que cada puerta NAND ocupa 3.136 um^2 .

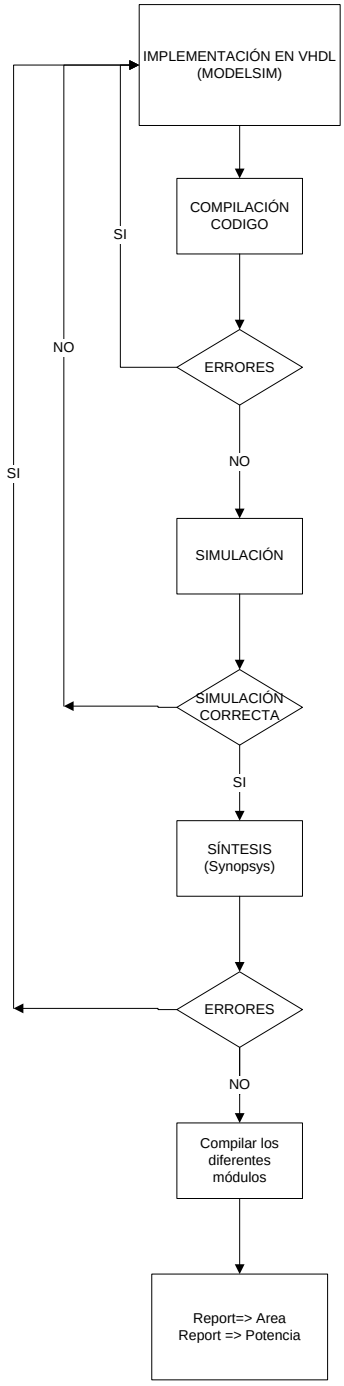
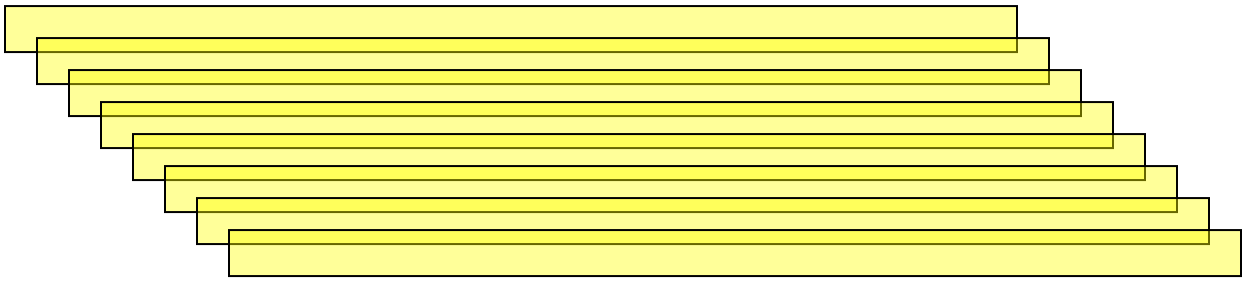


Figura 15. Diagrama de flujo: Desarrollo del proyecto

CAPÍTULO 4



PROTOCOLO DE AUTENTIFICACIÓN HMAC

4.1 DESCRIPCIÓN DEL PROTOCOLO

Este protocolo [6] tiene la característica de usar una función Hash, para la encriptación de los datos. Las tres variables con las cuales va a trabajar el protocolo serán:

a) Nr: Número aleatorio propio del lector
b) Nt: Número aleatorio propio de la tag
c) ID: Número identificador de la tag.

Lo ideal para este protocolo sería implementar un generador de números aleatorios en la tag, debido al excesivo espacio que ocupa la función hash esto no es posible, así cada número Nt que se usa en la tag está guardado en la memoria de la tag y no es generado.

Los pasos a realizar en el protocolo serán los siguientes:

a) En el primer paso el lector genera un número aleatorio Nr y realiza un comunicación con la tag.
b) La tag genera un número aleatorio Nt y realiza el cálculo a= Hid(0, Nr,Nt), donde H(x) es la función hash e Id es el número de identificación de la tag. Una vez calculado a se envía (a, Nt) al lector. En la implementación del protocolo el número Nt ya es un número guardado en la tag.
c) Una vez recibido el lector los valores de la tag comprobara si el valor recibido es el correcto. Para ello calcula b= Hid(0, Nr,Nt), si a=b el lector enviará a,Nr y Nt a la back-database.
d) La back-databse busca cualquier valor ID_j que hace $Hid_j(0,Nr,Nt)=Hid(0,Nr,Nt)$. En caso de existir un valor esa tarjeta pasará el protocolo de autentificación. La back-database envía c=Hid(1,Nr,Nt) y ID al lector. En caso de que los valores comparados no coincidan el protocolo es abortado.
e) Cuando el lector recibe c y ID de la back-database envía c a la tag y almacena Id en la memoria. La tag una vez recibido c, calcula d=Hid(1,Nr,Nt), lo compara con c en caso de no coincidir la autentificación del lector será abortada. En caso de coincidir el proceso de autentificación ha sido correcto.

4.2 AMENAZAS PARA EL PROTOCOLO HMAC

A continuación se muestra la respuesta del protocolo a distintas amenazas [6] [7] [8].

Suplantación de identidad:

Debido a que el lector genera un número aleatorio en cada autentificación no es posible para el atacante hacer este tipo de suplantación, por lo tanto se considera este protocolo seguro para este tipo de ataques.

Replay attack.

Después de que el lector envía la query y N_r a la tag, y esta responde al lector, el atacante obtendrá la respuesta Hid(0,N_r,N_t) y N_t. Debido a la función de filtrado previa del lector antes de la autentificación de la tag, este protocolo se considera seguro para este tipo de ataques.

Seguimiento

Aquí es donde el atacante intercepta el mensaje de respuesta de la tag, obtiene Hid(0,Nr,Nt) y N_t. Analizando la respuesta el atacante puede realizar un seguimiento de la tag. Como la tag genera un número aleatorio por cada autentificación y la función hash se hace de una sola manera, el atacante no puede establecer cuáles de las tag está respondiendo al lector. Por tanto este protocolo se considera seguro para el seguimiento.

Desincronización

La ID de cada tag es fija, incluido en una pérdida de conexión, esto podría ser un inconveniente para el protocolo. En este protocolo se establece que la Id es estrictamente confidencial y nunca es revelada. Si la tag es atacada, el valor Id podría ser obtenido por el atacante, esto podría llevar a que el intruso actuara como tarjeta y poder comunicarse con el lector. Sin embargo, esto no es considerado como un ataque ya que podría ocurrir con cualquier otro protocolo en el cual se revela la ID.

4.3 DIAGRAMA DE FLUJO DEL PROTOCOLO HMAC

En el siguiente diagrama se muestra el flujo del código para un protocolo de 32 bits.

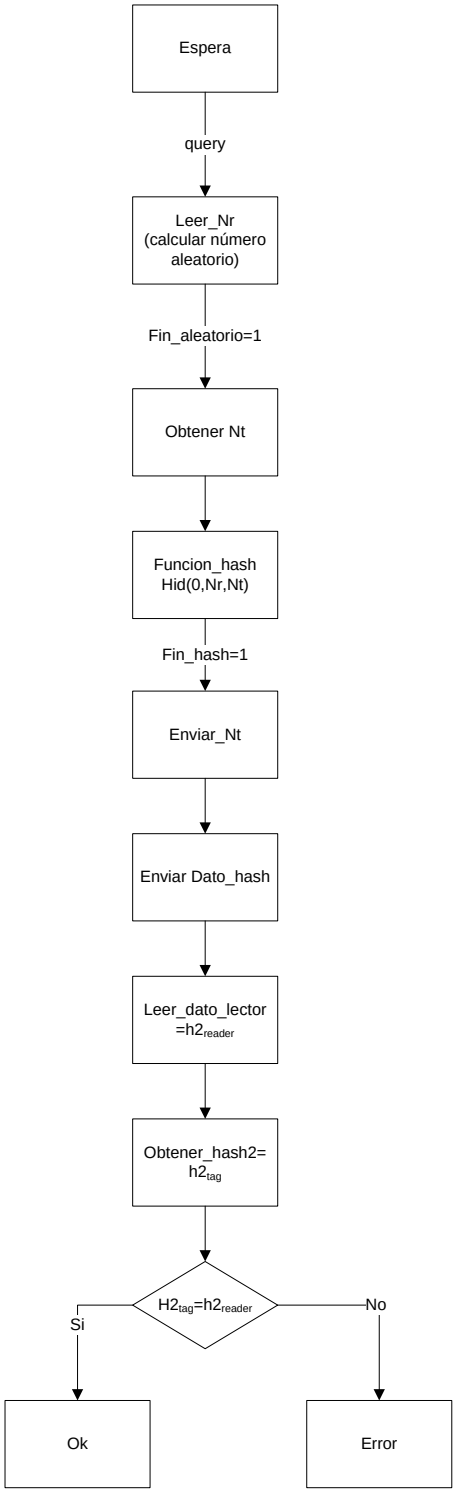


Figura 16. Diagrama de Flujo: HMAC

Aunque el protocolo implementado de la bibliografía se refiere a un protocolo de 32 bit, también se ha implementado el protocolo para mayor número de bits, se comprobará si cumple las restricciones de área para poder ser implementado en el tipo de etiquetas a las cuales se refiere este proyecto. Hay que destacar que a mayor número de bits mayor seguridad del protocolo.
A continuación se muestra el diagrama de flujo para el protocolo usando 128 bits.

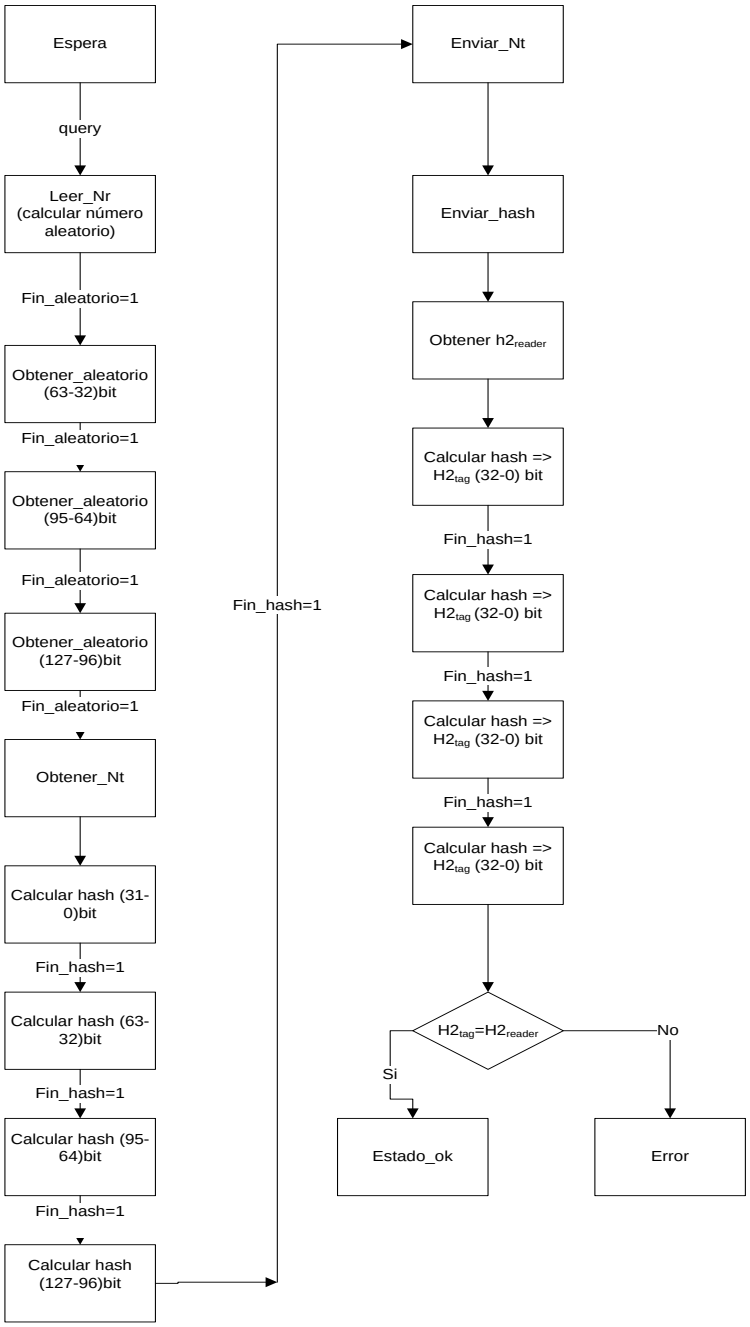


Figura 17. Diagrama de flujo HMAC 128 bits

4.4 DESCRIPCIÓN FUNCIÓN HASH

Se ha implementado una función hash [9] [10] de 128 bit la cual asegura la seguridad y además puede ser implementada en las tag de bajo coste, debido al espacio que ocupa.

Código implementado.

A continuación se muestra el código en C que ha sido implementado en VHDL.

```
/*********************************************/
Process the input a1 modifying the
accumulated hash a0 and the state
/*********************************************/
void tav(unsigned long *state, unsigned long
*a0, unsigned long *a1)

{
unsigned long h0,h1;
int i,j,r1,r2,nstate;

/* Initialization */
 r1=32; r2=8; nstate=4;
 h0=*a0; h1=*a0;

/* A - Function */
for(i=0;i<r1;i++){h0=(h0<<1)+((h0+(*a1))>>1);}
/* B - Function */
for(i=0;i<r1;i++){h1=(h1>>1)+(h1<<1)+h1+(*a1);}

/* C and D - Function */
for(j=0;j<nstate;j++) {
 for(i=0;i<r2;i++)
    {
    /* C - Function */
    h0^=(h1+h0)>>3;
    h0=((((h0>>2)+h0)>>2)+(h0<<3)
          +(h0<<1))^0x736B83DC;
    /* D - Function */
    h1^=(h1^h0)>>1;
    h1=(h1>>4)+(h1>>3)+(h1<<3)+h1;
    } // round-r2
     state[j]+=h0;
     state[j]^=h1;
} // state

/* a0 updating */
 *a0=h1+h0;

}


/*********************************************/
Initialization of the state and a0 with
random values obtained from www.random.org
/*********************************************/
void init_state(unsigned long *state, unsigned
long *a0)

{
     state[0]=0xa92be51d;
     state[1]=0xba9b1ef0;
     state[2]=0xc234d75a;
     state[3]=0x845c2e03;
        a0[0]=0x768c7e74;
}
```

Figura 18. Código C función Hash

Las funciones A y B actúan de filtro para evitar que posibles atacantes tengan acceso a algún bit de la variable estado. De esta forma dificulta el acceso del atacante a las entradas del sistema.
La salida de tamaño 128 bit cumple el compromiso entre rapidez y robustez.

Hardware hash

La arquitectura de la función se divide en dos principales bloques:

a) Memoria: Aquí se guardan todas la variables que serán necesarias para la implementación del código: estado (128 bit), registro acumulado hash a0 (32 bit), variable interna h0 (32 bit), variable interna h1(32 bit), entrada a1(32 bit)
b) ALU: En esta unidad se hacen desplazamientos de bit además de operaciones con XOR. Se necesita un registro auxiliar para guardar los resultados.

4.5 DIAGRAMA DE FLUJO FUNCIÓN HASH

En el siguiente diagrama de flujo se muestra el código de la función hash implementada en VHDL.

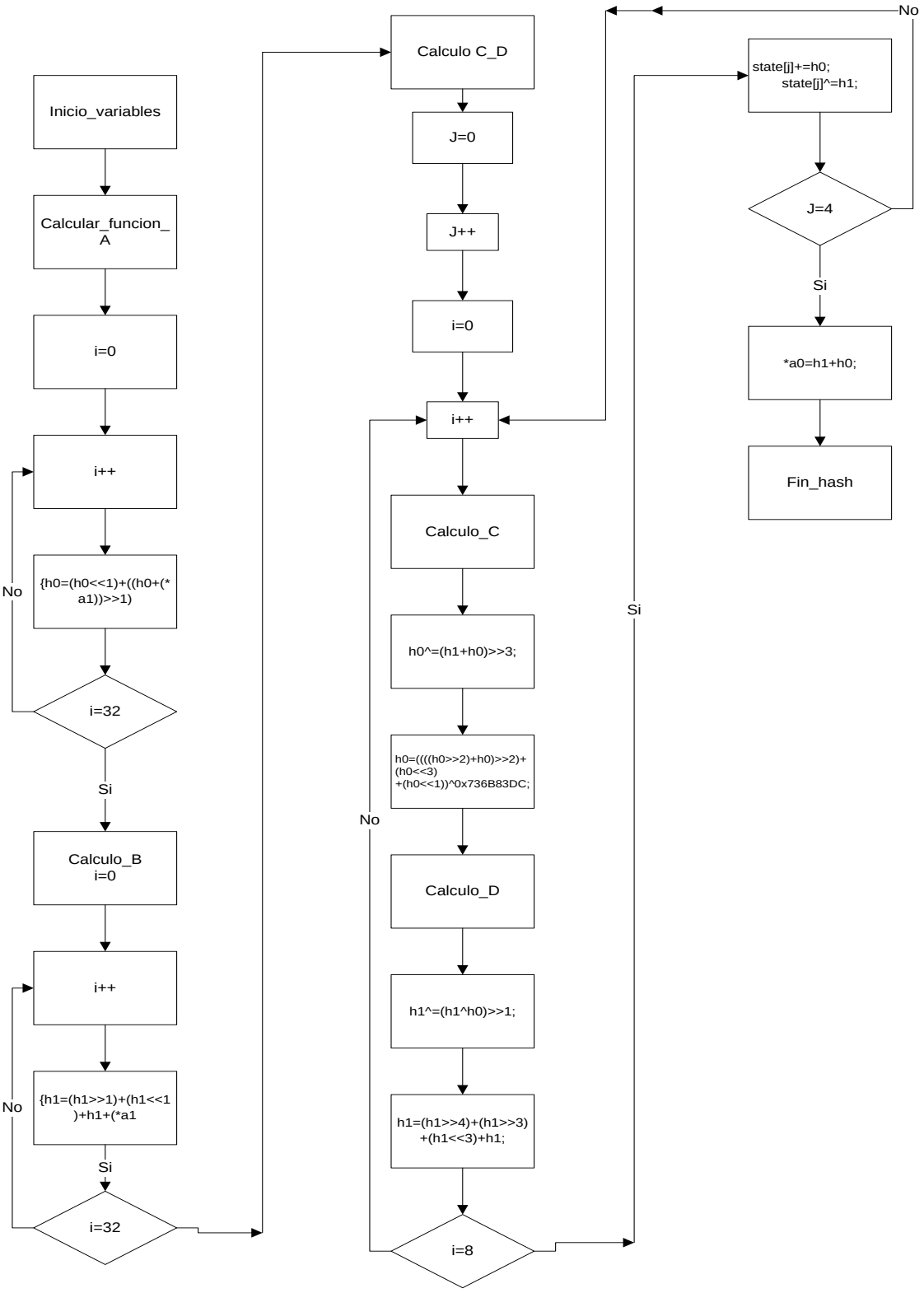


Figura 19. Diagrama flujo: Función Hash

4.6 DIAGRAMA DE BLOQUES

A continuación se muestra el diagrama de bloques del protocolo HMAC implementado.

La función Alu se compone de un sumador y una función XOR, a la hora de la síntesis el esquema mostrado sería el siguiente:

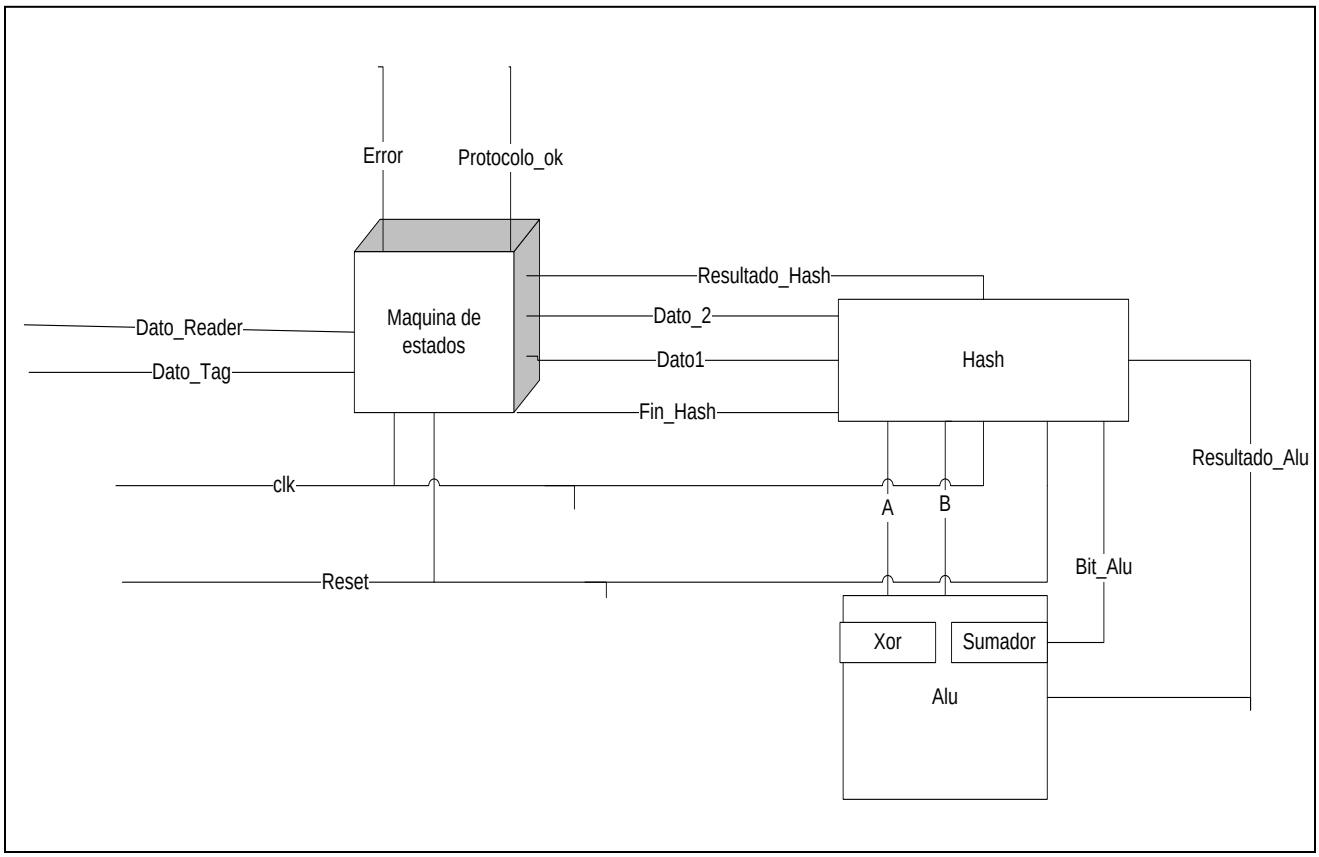


Figura 20. Hardware Protocolo Hmac

Se ha implementado solo una función Xor y un Sumador, de esta forma la función hash utilizará la misma Alu cada vez que necesite realizar cada operación. De esta forma se incrementa el tiempo de cálculo pero se reduce el área implementada en el circuito. Esto es necesario ya que el número de puertas equivalentes que se tiene de máximo es de 4000 puertas.

En la siguiente figura se muestra el diagrama hardware de la función hash:

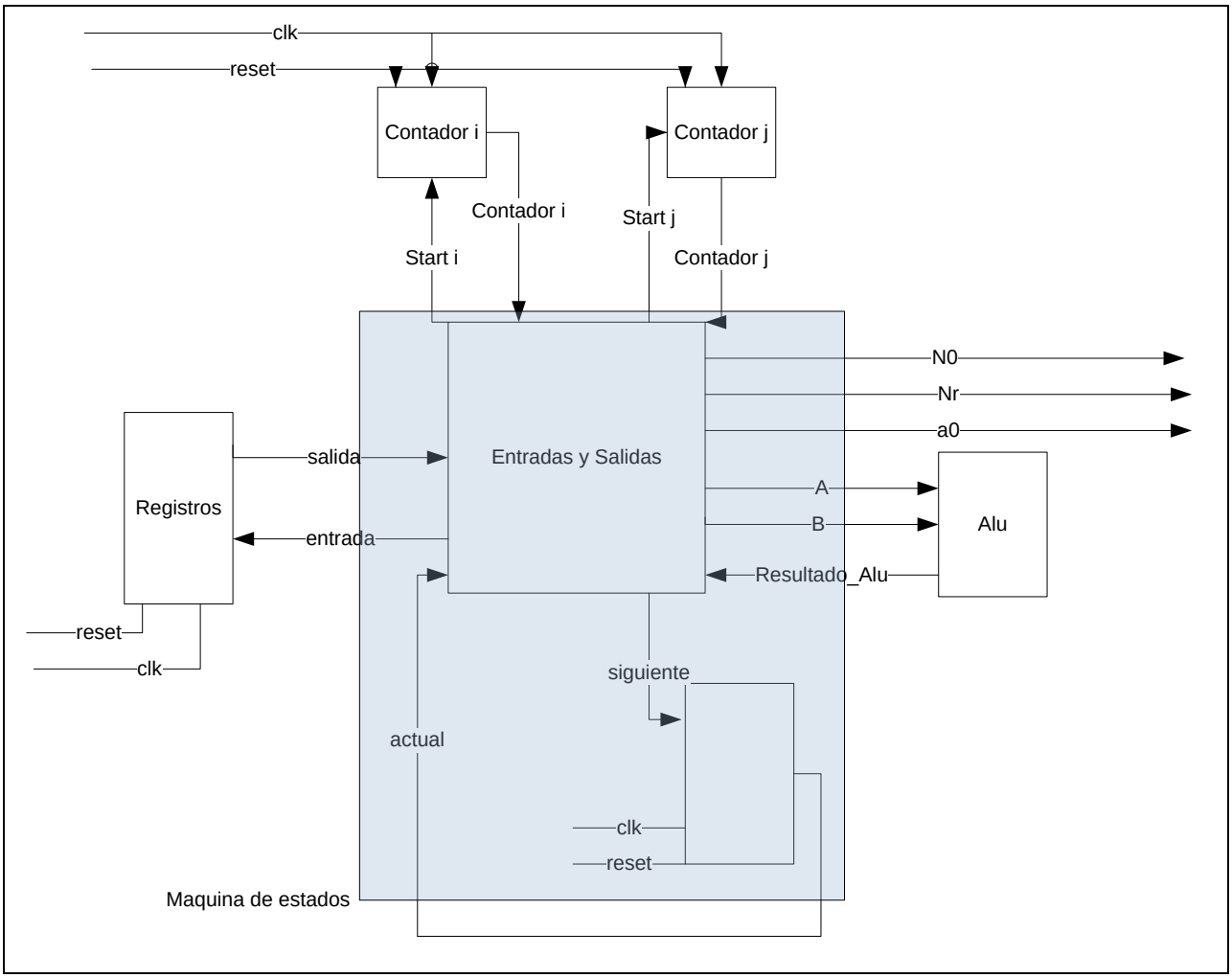
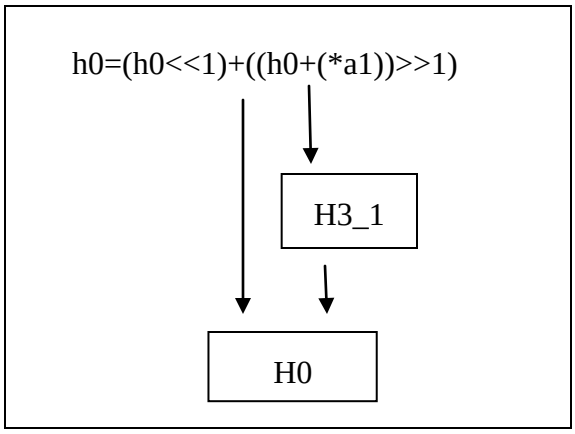


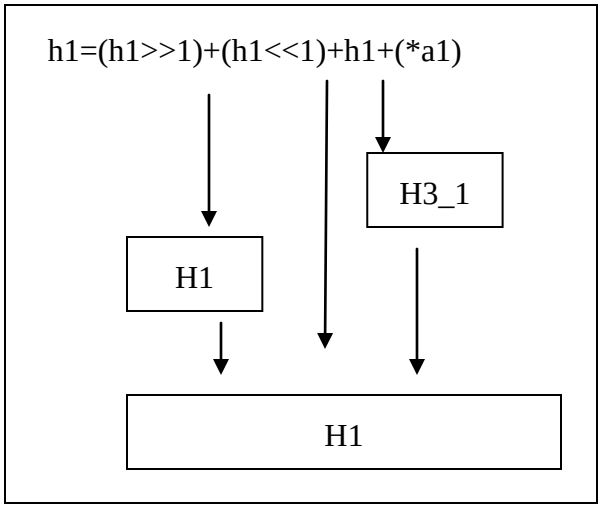
Figura 21. Hardware Función Hash

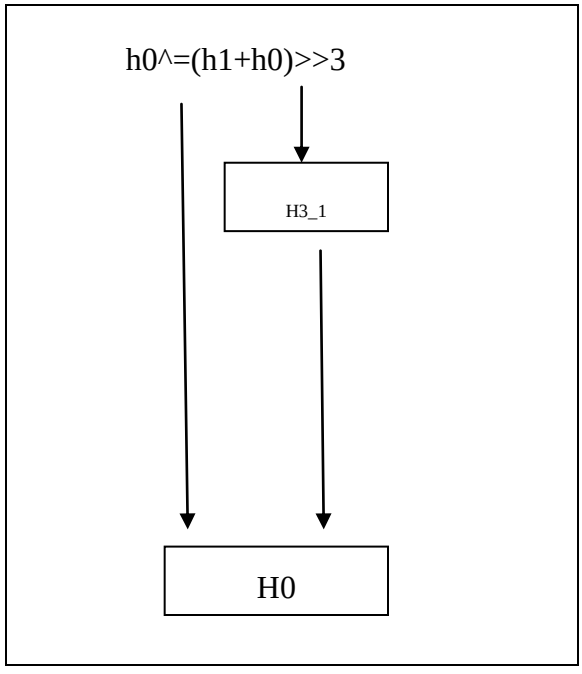
4.7 REGISTROS USADOS EN LA FUNCIÓN HASH

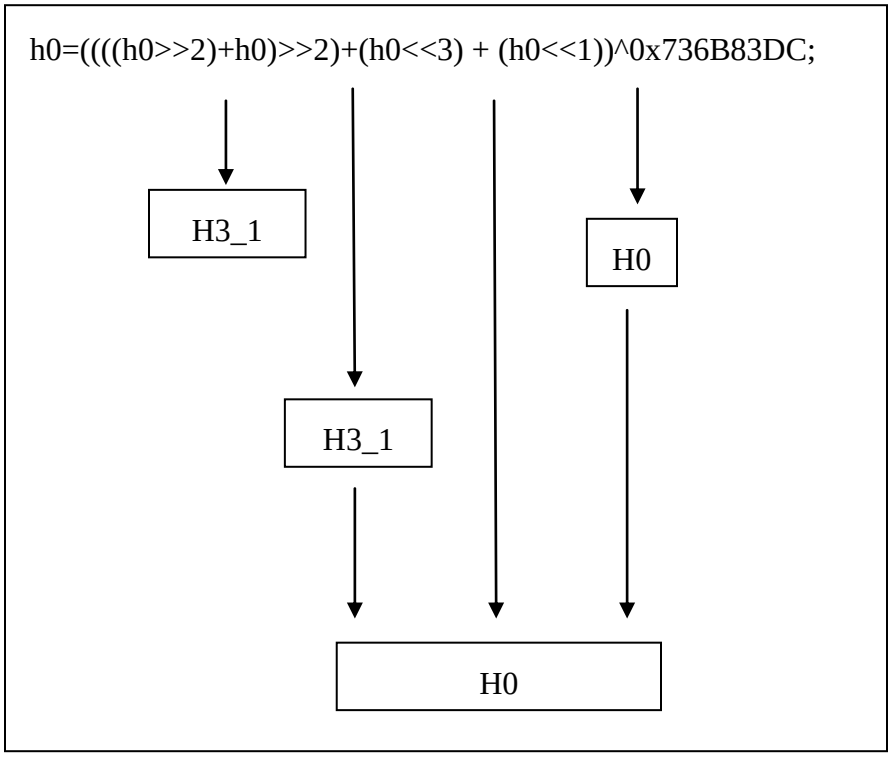
Debido que la restricción de área está en 4000 puertas equivalentes, hace que la implementación del circuito se haga ocupando el mínimo espacio posible, de esa forma se muestra la manera en la cual se ha creado los registro y como se han usado para no tener que crear más registros y de esa manera ocupar el mínimo espacio posible.

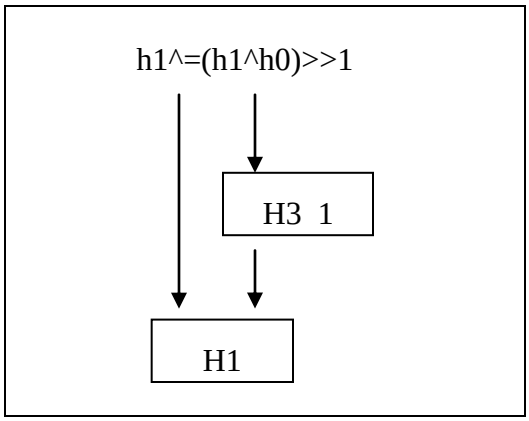
En lo siguiente esquemas se muestra que operación se ha realizado y en que registro se ha guardado.

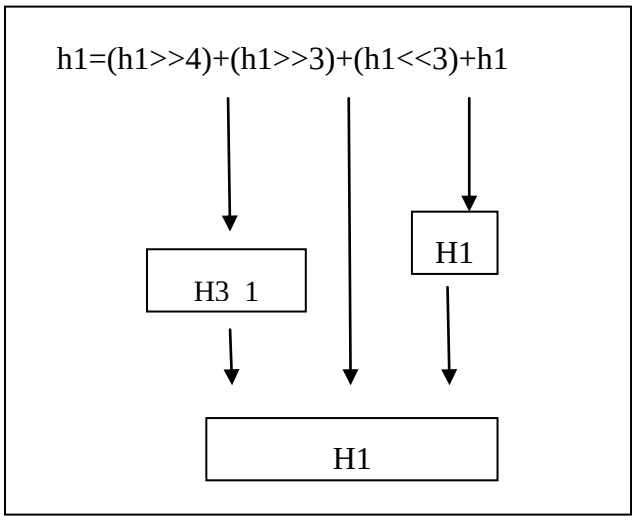

h0=(h0<<1)+((h0+(*a1))>>1)
H3_1
H0

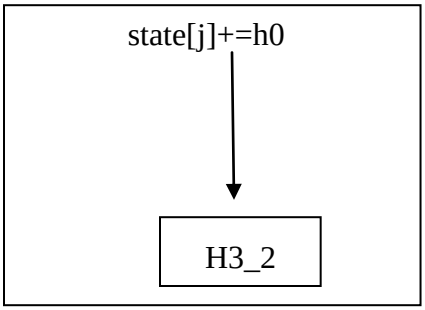

h1=(h1>>1)+(h1<<1)+h1+(*a1)
H3_1
H1
H1

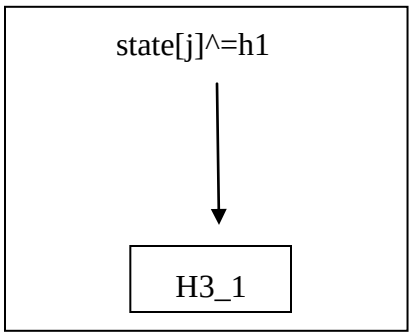

h0^=(h1+h0)>>3
H3_1
H0


h0=((((h0>>2)+h0)>>2)+(h0<<3) + (h0<<1))^0x736B83DC;
H3_1
H0
H3_1
H0


h1^=(h1^h0)>>1
H3 1
H1







En total se han usado 3 registros para las funciones A,B,C,D, y un registro para la función del registro estado h3_2.

4.8 DIFERENTES DESARROLLOS DE BITS

La función hash está diseñada para salida de 128 bit, por tanto no se va a modificar su arquitectura y se mantendrán registros de 32 bits. Las diferentes opciones que se pueden dar son en la Alu y en la máquina de estados. A continuación se muestra una tabla con los diferentes diseños que se han implementado y sintetizado.

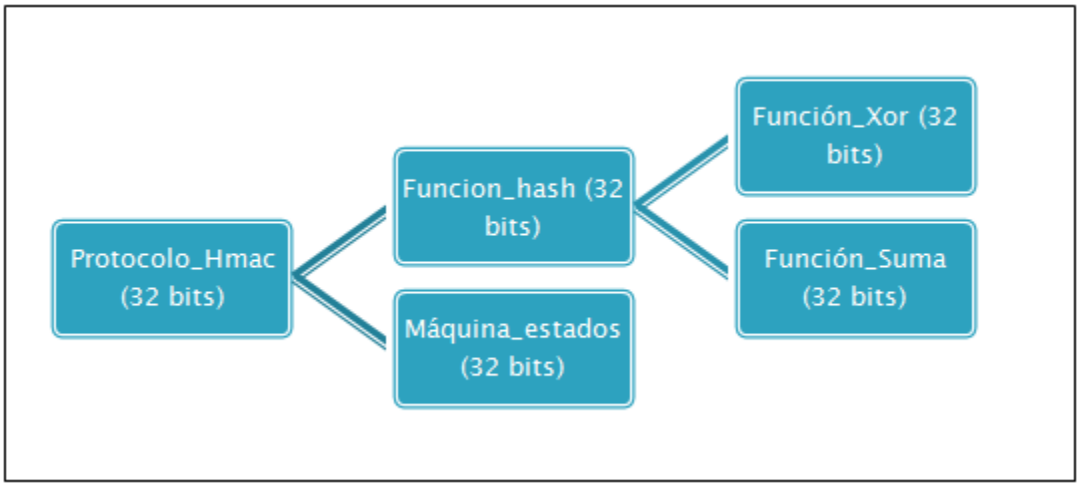


Figura 22. Bits Protocolo opción 1

En el siguiente caso para protocolo de 64 bits, al ser la función hash de 32 bits habrá que dividir los registros del protocolo en parte alta y parte baja y tener que realizar la función hash al menos dos veces para tener el resultado total.

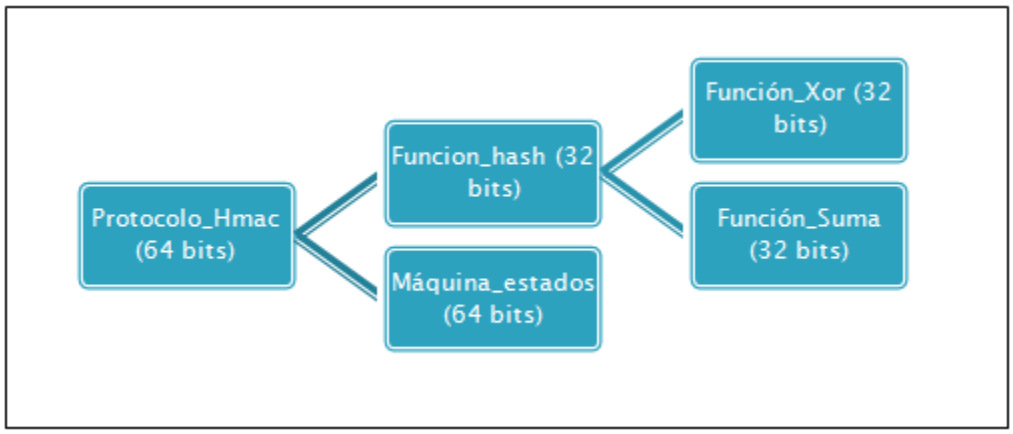


Figura 23. Bits Protocolo opción 2

Para el siguiente caso de 128 bits ocurra igual que para 64 bits pero este caso con mayor número de bits. Por tanto los registros del protocolo habrá que dividirlos en 4 partes y realizar la función hash al menos 4 veces para obtener el resultado final.

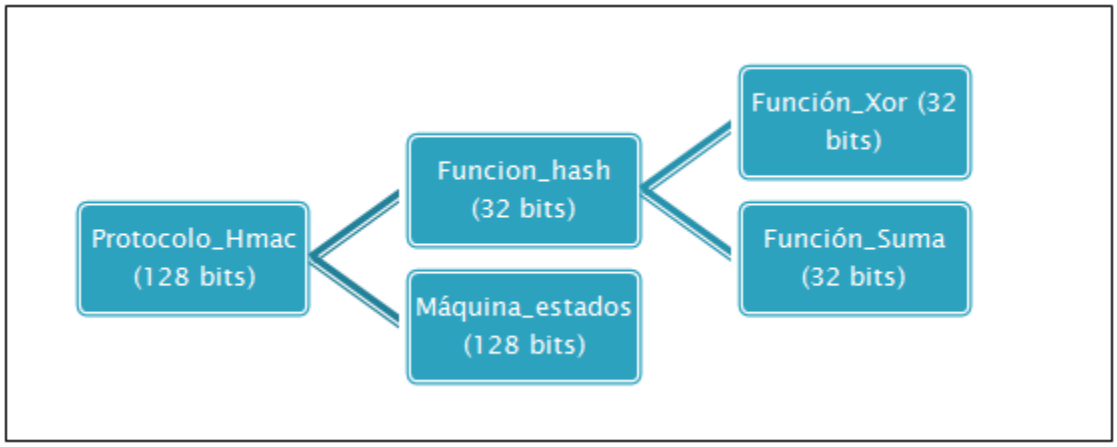


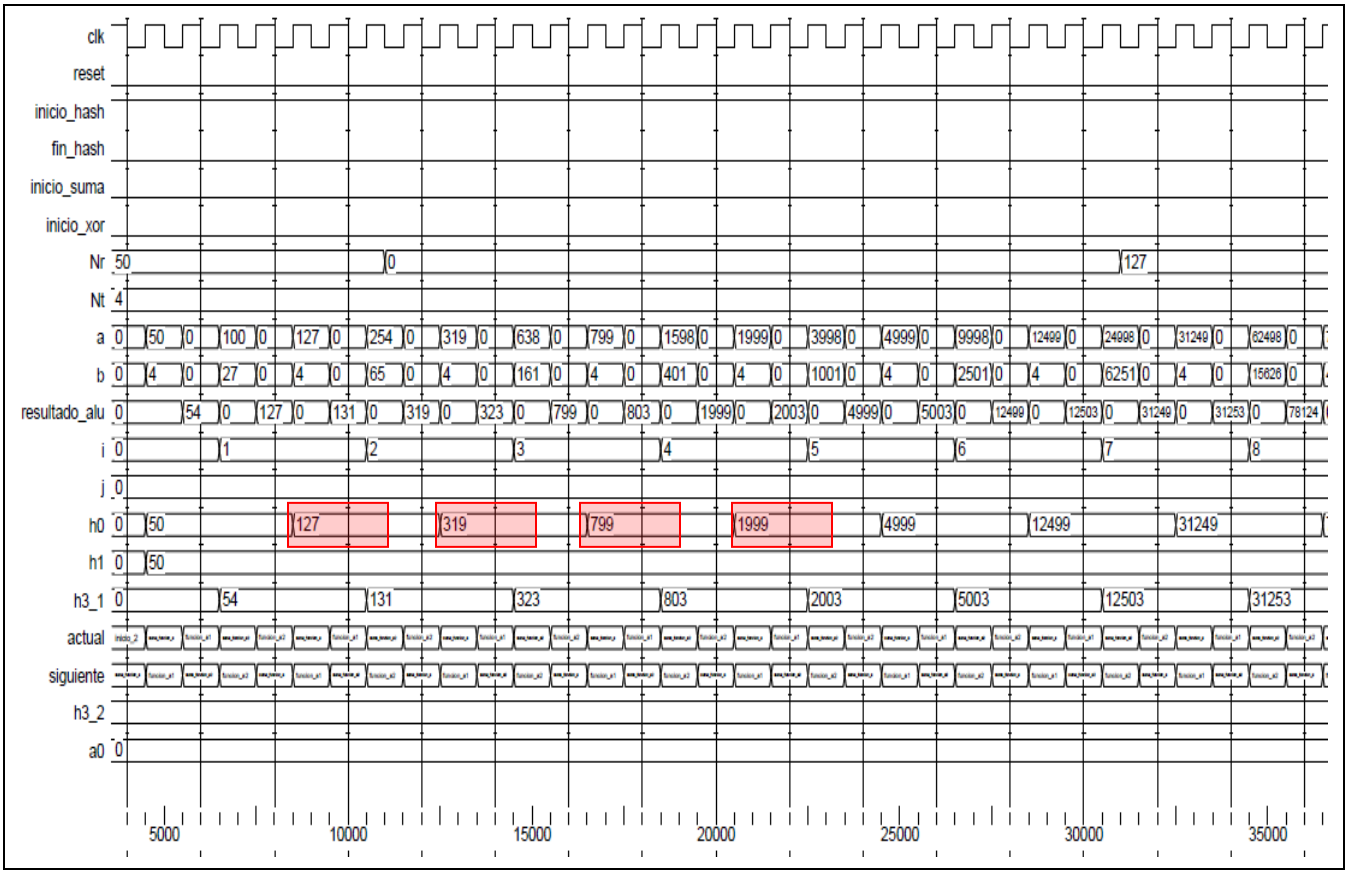
Figura 24. Bits protocolo opción 3

Se recuerda el interés por conseguir el mayor número de bits posibles en el protocolo, ya que a mayor número de bits se considera la seguridad del protocolo mayor.

4.9 SIMULACIONES

Simulación función hash

En las siguientes simulaciones se muestra el funcionamiento de las distintas etapas de la función hash.



Simulación 1. Función A (Función Hash)

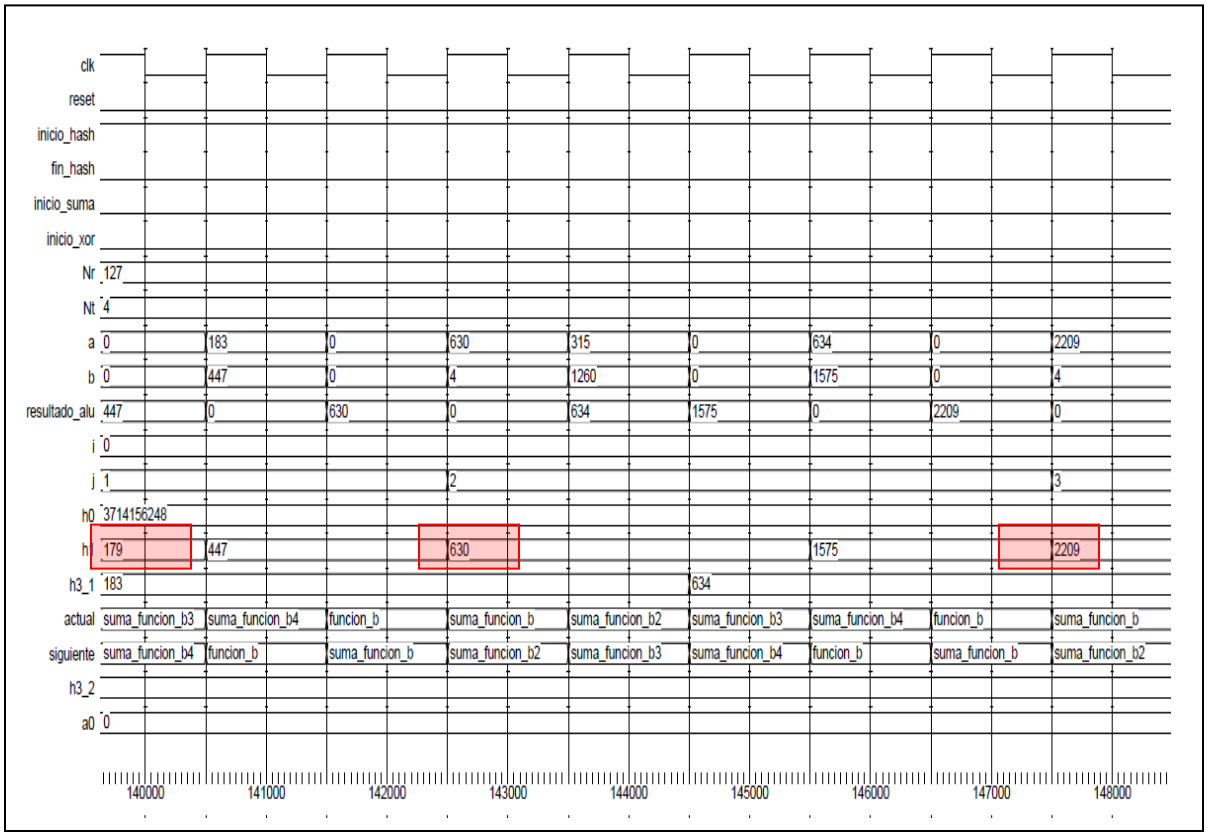
El valor h0 obtiene los valores para la función A. Se comprueba como los valores los cuales se deben obtener coinciden con los de la simulación.

Funcion A	h0<<1	h0	h0+a	(h0+a)>>1	h0total
0	100	50	54	27	127
1	254	127	131	65	319
2	638	319	323	161	799
3	1598	799	803	401	1999
4	3998	1999	2003	1001	4999
5	9998	4999	5003	2501	12499
6	24998	12499	12503	6251	31249
7	62498	31249	31253	15626	78124
8	156248	78124	78128	39064	195312
9	390624	195312	195316	97658	488282
10	976564	488282	488286	244143	1220707
11	2441414	1220707	1220711	610355	3051769
12	6103538	3051769	3051773	1525886	7629424
13	15258848	7629424	7629428	3814714	19073562

14	38147124	19073562	19073566	9536783	47683907
15	95367814	47683907	47683911	23841955	119209769
16	238419538	119209769	119209773	59604886	298024424

Tabla 2. Valores Función A (Hash)

En la siguiente simulación se muestra el funcionamiento de la parte del cálculo de B.



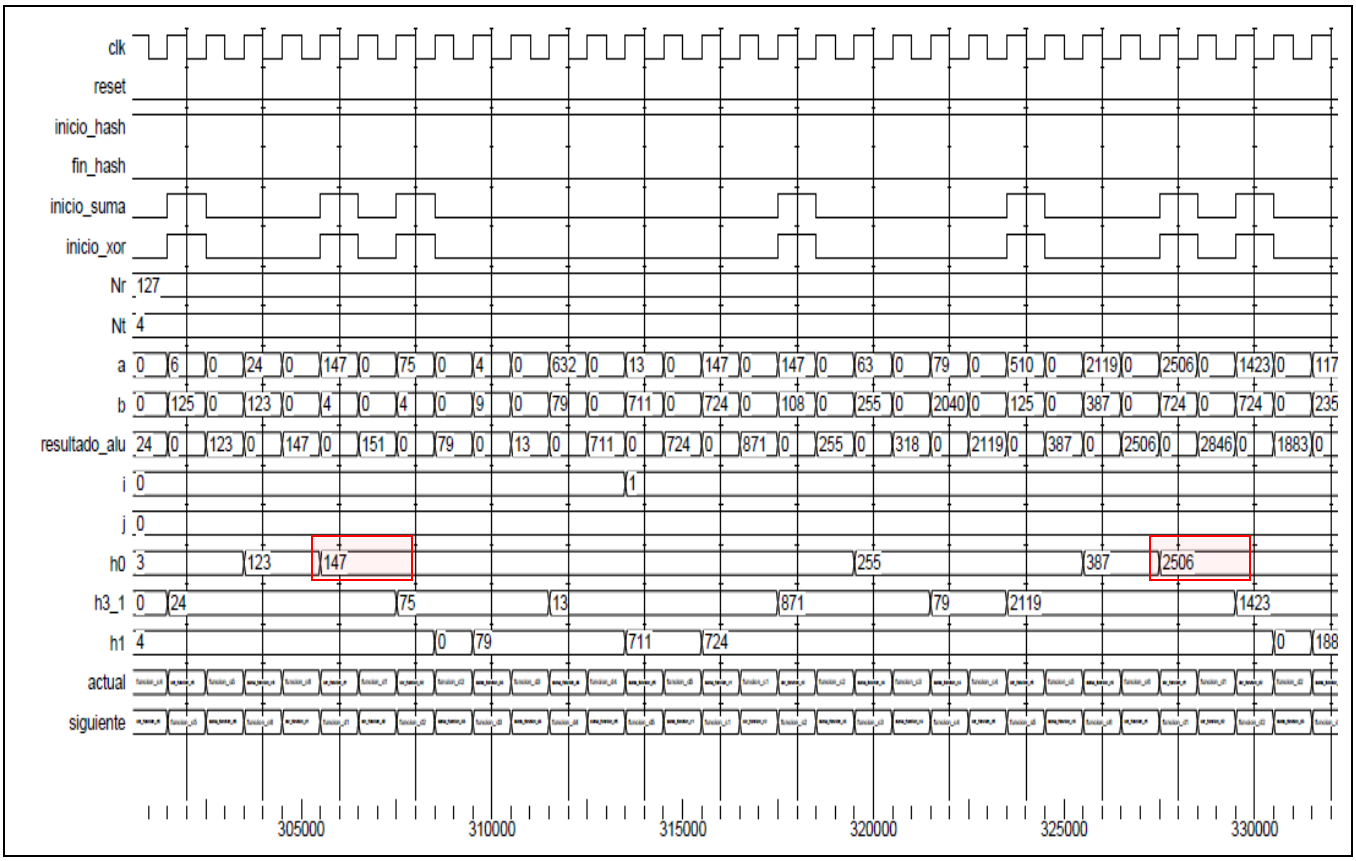
Simulación 2. Función B (Función hash)

En el valor h1 se obtiene los valores de la función B. Al igual que para la función A se comprueba el valor de la simulación con los valores calculados, observando que son válidos los valores obtenidos en la simulación.

Funcion B	h1>>1	h1<<1	h1+a1	h1total
0	25	100	54	179
1	89	358	183	630
2	315	1260	634	2209
3	1104	4418	2213	7735
4	3867	15470	7739	27076
5	13538	54152	27080	94770
6	47385	189540	94774	331699
7	165849	663398	331703	1160950
8	580475	2321900	1160954	4063329
9	2031664	8126658	4063333	14221655
10	7110827	28443310	14221659	49775796
11	24887898	99551592	49775800	174215290
12	87107645	348430580	174215294	609753519
13	304876759	1219507038	609753523	2134137320
14	1067068660	4268274640	2134137324	3174513328
15	1587256664	2054059360	3174513332	2520862060
16	1260431030	746756824	2520862064	233082622

Tabla 3. Valores Función B (hash)

Para las siguientes simulaciones se muestra los valores para la función C y D, estas dos funciones se hacen seguidas y dentro del mismo paso.



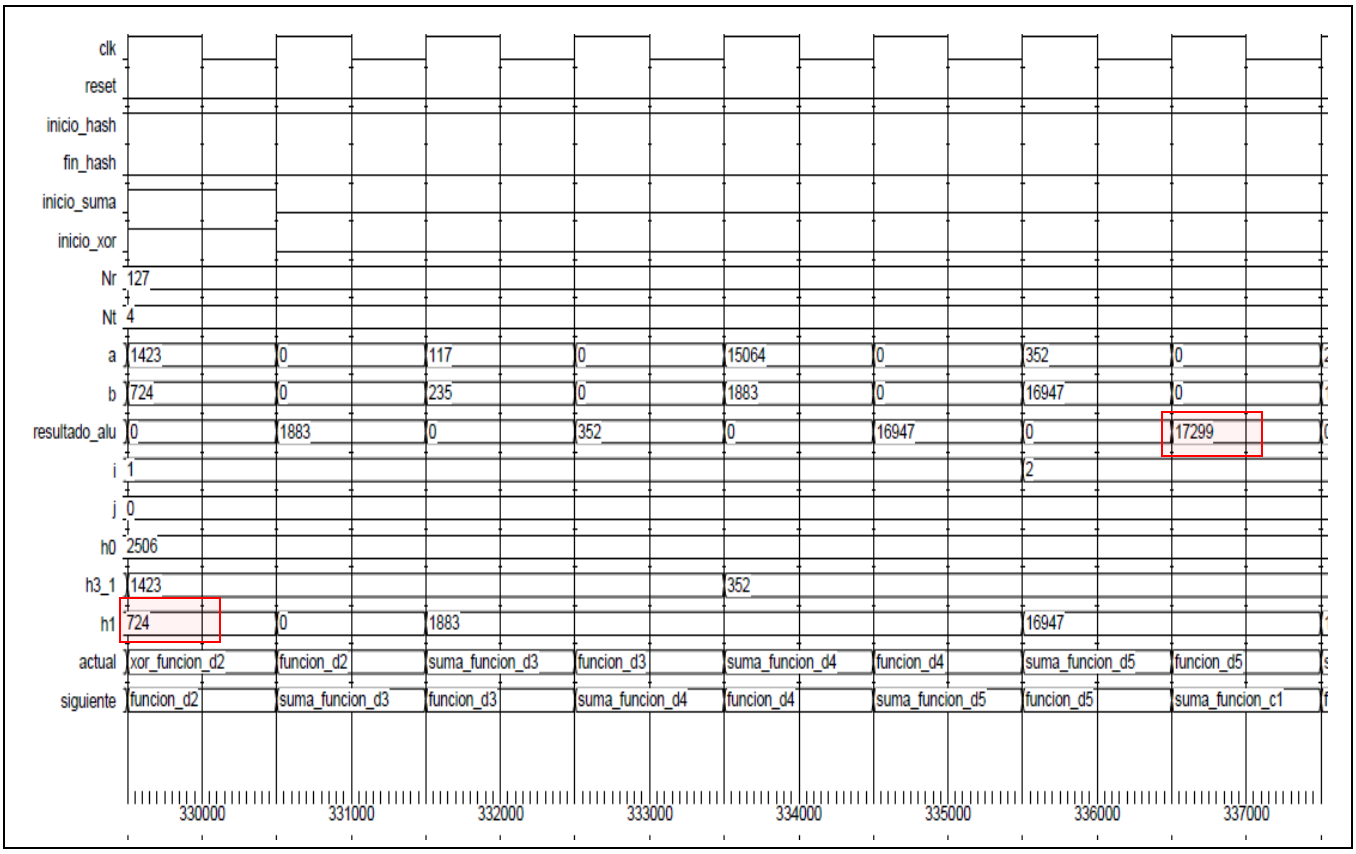
Simulación 3. Función C (Función hash)

Se vuelve a guardar en h0 los valores correspondientes a la función C

Funcion C	h1+h0 >>3	h0-1^xor	h0>>2	(h0>>2 + h0)>>2	h0<<3	h0<<1	h0<<1 Xor aleatorio	h0total
0	0	3	0	0	24	6	123	147
1	108	255	63	79	2040	510	387	2506
2	2475	97	24	30	776	194	191	997
3	29423	28938	7234	9043	231504	57876	57961	298508
4	45942	277882	69470	86838	2223056	555764	555657	2865551
5	1889263	3631712	907928	1134910	29053696	7263424	7263421	37452027
6	42777841	12043274	3010818	3763523	96346192	24086548	24086633	124196348

Tabla 4. Valores Función C (hash)

La siguiente simulación se corresponde para los valores de la función D.



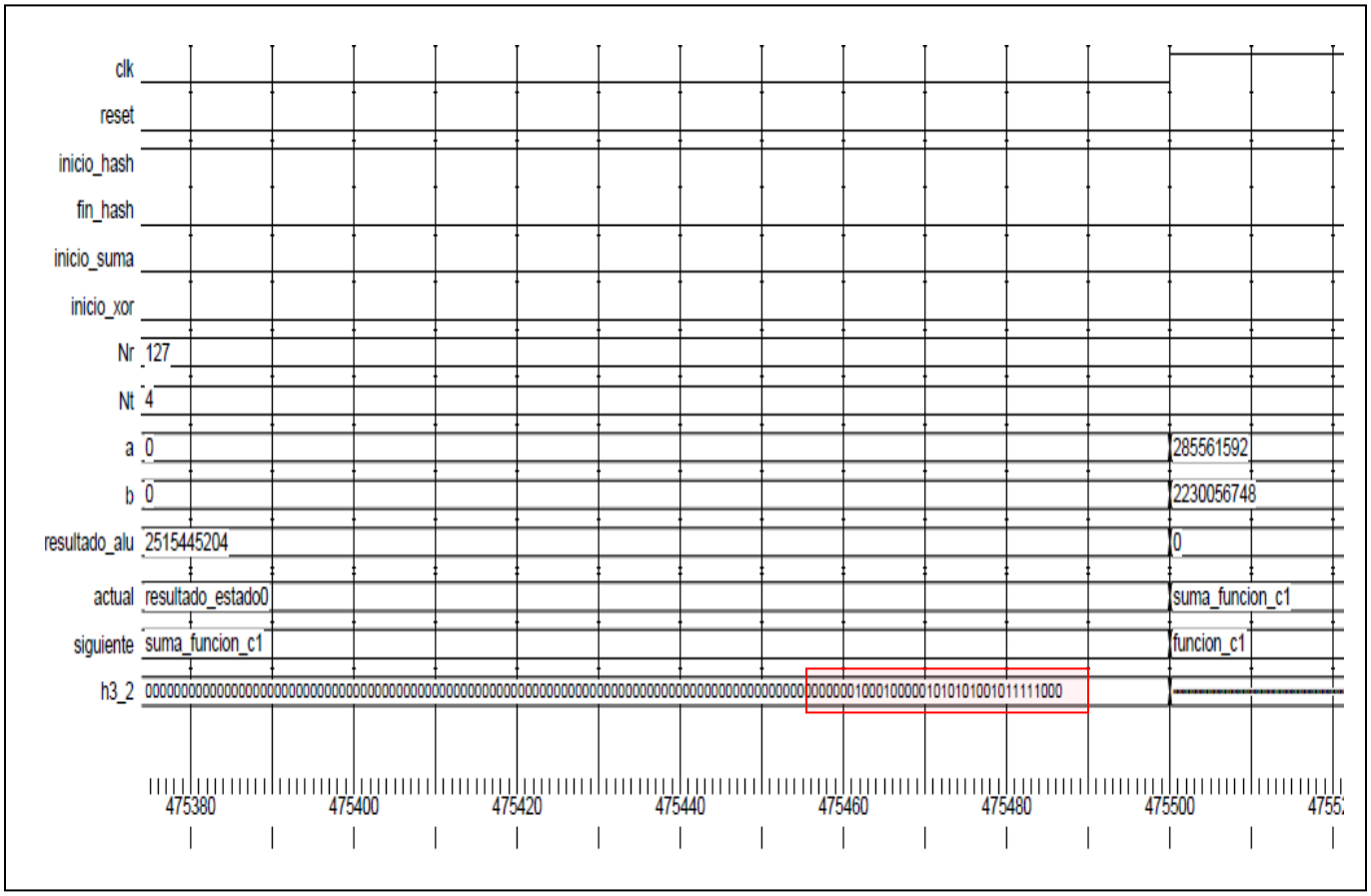
Simulación 4. Función D (Función Hash)

Se vuelven a guardar en h1 los valores correspondientes de la función h1.

Funcion D	XOR/2	XOR	h1>>4	h1>>3	h1<<3	H1total
0	75	79	4	9	632	724
1	1423	1883	117	235	15064	17299
2	8251	25512	1594	3189	204096	234391
3	233165	7514	469	939	60112	69034
4	1399314	1333176	83323	166647	10665408	12248554
5	21024392	33172322	2073270	4146540	265378576	304770708

Tabla 5. Valores Función D (hash)

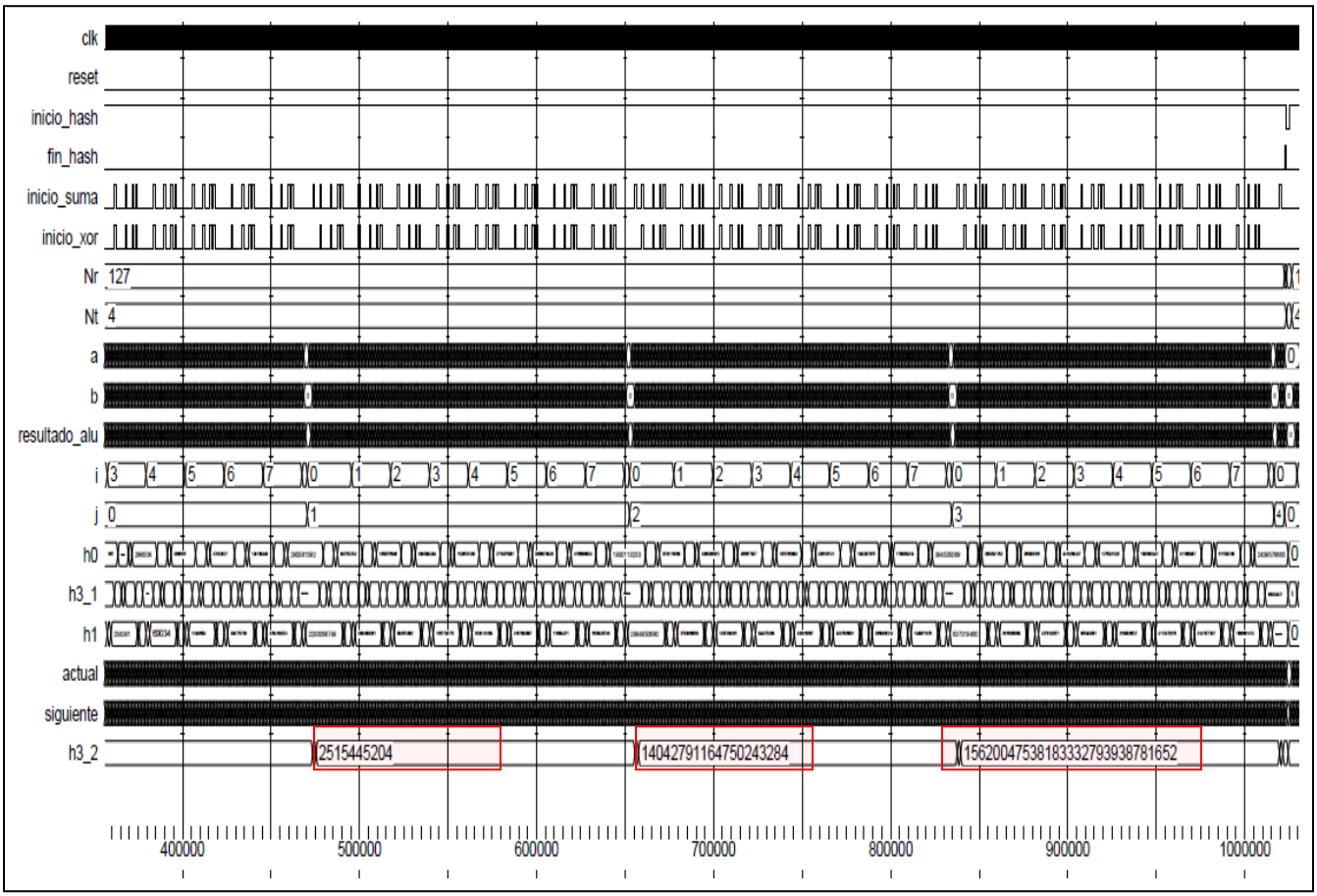
Una vez que se termina la función C y D, se vuelven a calcular los valores de la variables estado, en esta simulación se muestra el cálculo para los primeros 32 bits.



Simulación 5. Variable Estado

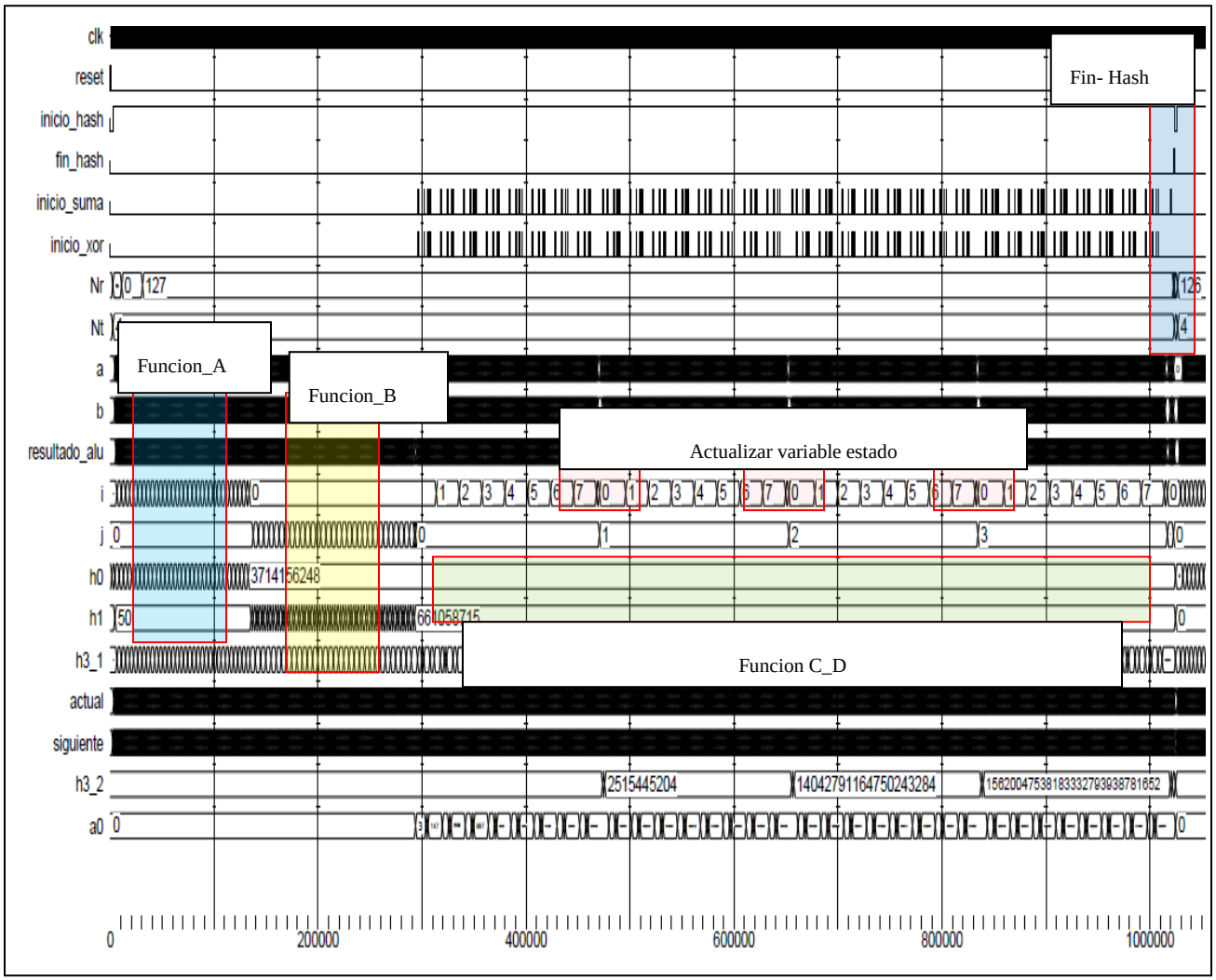
Obteniendo una representación del tiempo más grande, se observa la actualización de la variable estado para sus cuatro grupos de 32 bits, obteniendo el resultado final de la variable de 128 bits. En la primera vez que se actualiza este valor, se calcula solos los primeros 32 bits.

En la siguiente simulación se muestra la actualización de los distintos grupos de bits para la variable estado.



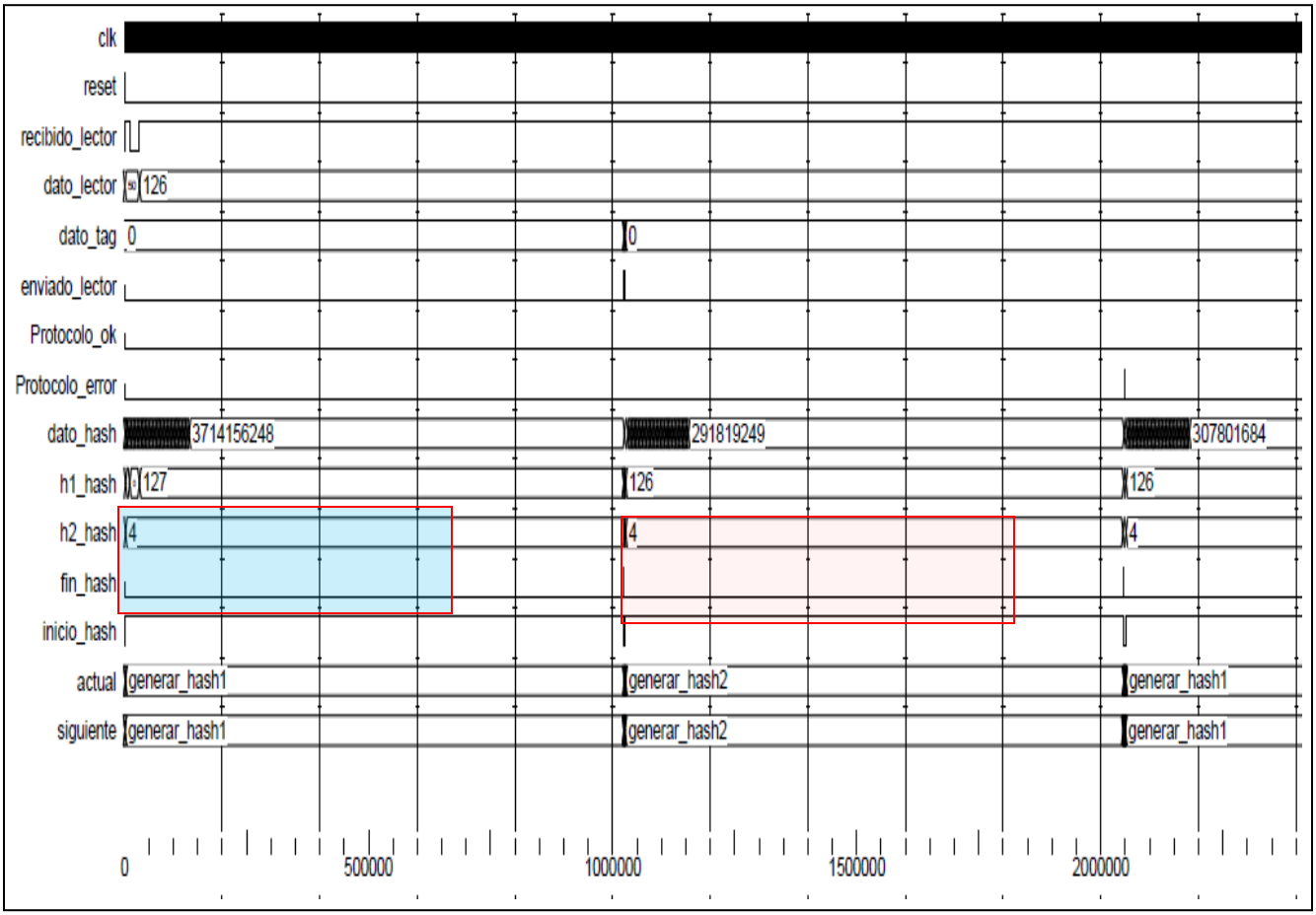
Simulación 6. Actualización de la variable Estado

A continuación se muestra el funcionamiento de la función hash entera, calculando las funciones A,B,C, D y obteniendo la variable estado.



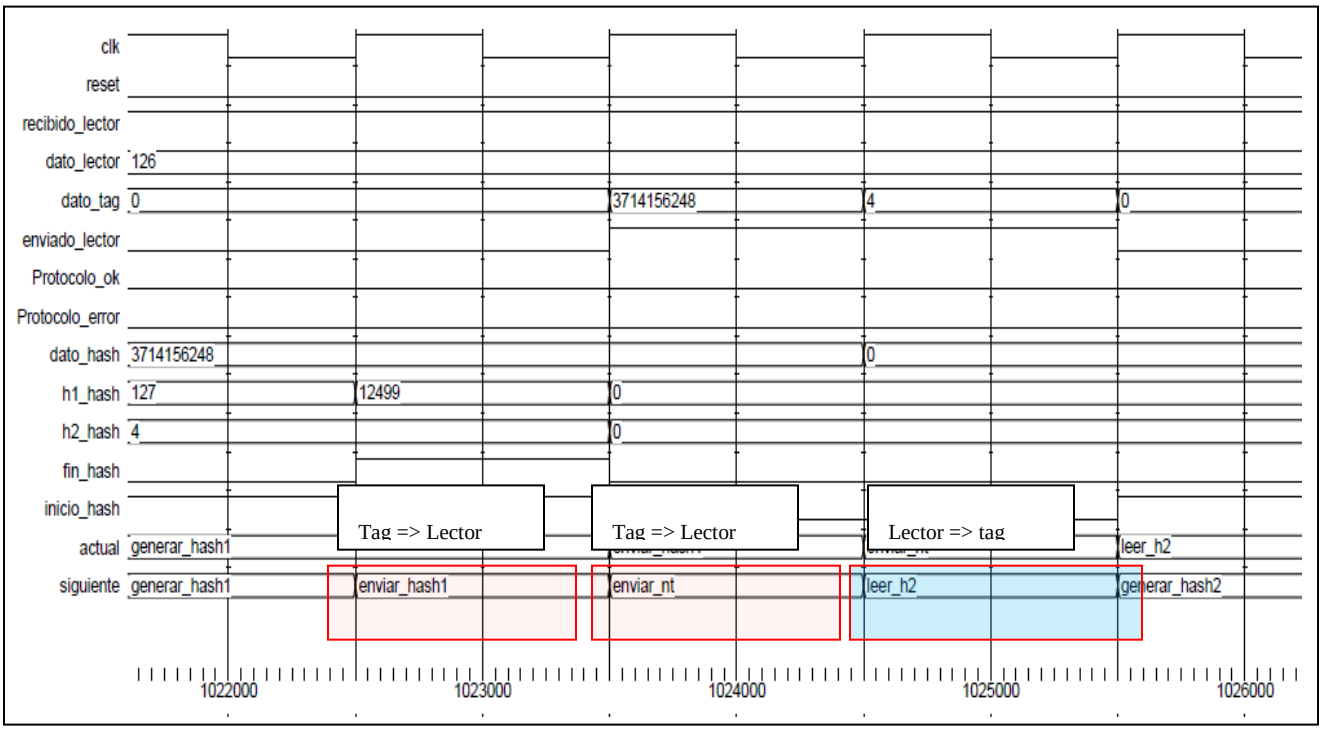
Simulación 7. Función Hash

Una vez implementado la función hash dentro del protocolo de 32 bits, el funcionamiento es el siguiente.



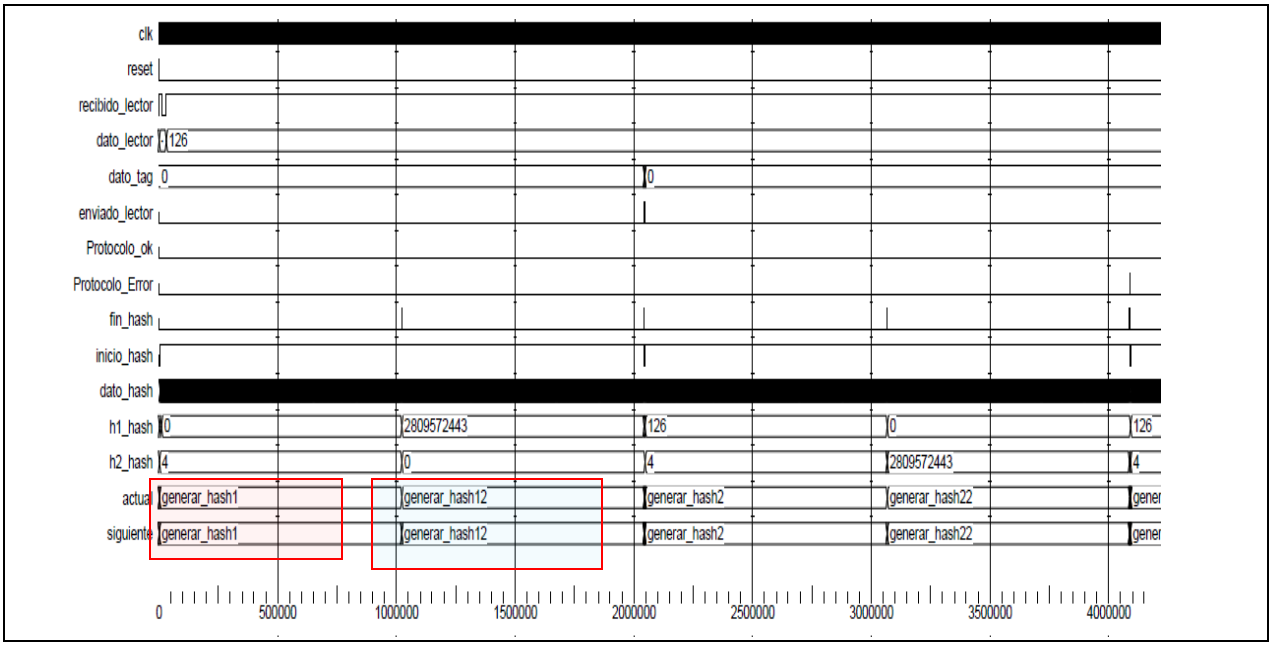
Simulación 8. Protocolo Hmac 1

Se observa cómo se necesita realizar dos veces la función hash, la primera vez para generar el valor la tag y la segunda para la comprobación con el valor recibido del lector.



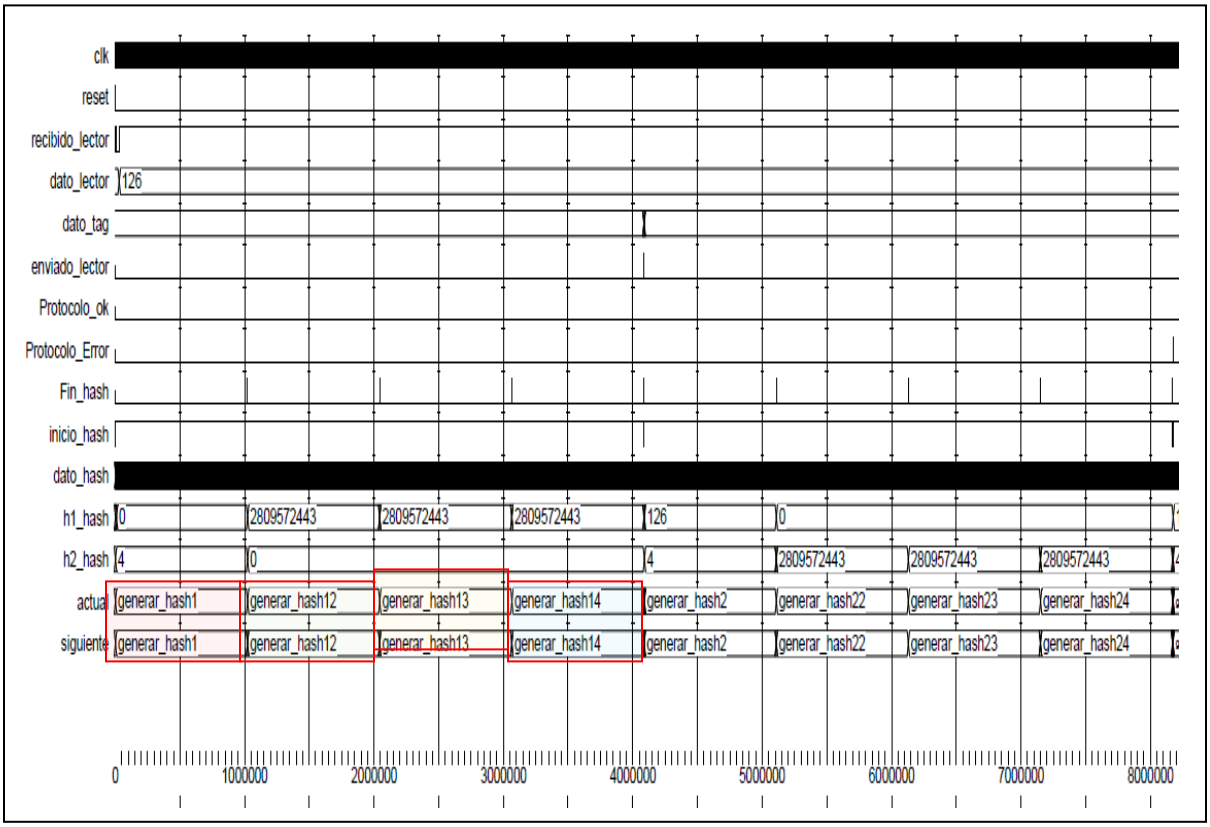
Simulación 9. Protocolo Hmac 2

Para protocolos de 64 bits, se necesitan realizar dos veces la función hash, ya que nos dará salida de 32 bits.



Simulación 10. Protocolo Hmac 64 bits

En el caso de protocolo de 128 bit se necesitaría realizar la función hash 4 veces para obtener el valor final.



Simulación 11. Protocolo Hmac 128 bits

Al necesitar realizar 4 veces la función hash el tiempo de cálculo se incrementa, de esta manera se consigue no incrementar el área de la función hash en perjuicio de incrementar el número de ciclos.

4.10 RESULTADOS SÍNTESIS

Una vez que se ha obtenido las simulaciones correctamente habrá que realizar la síntesis del circuito, con esta síntesis se obtiene las variables de área y potencia. La variable de ciclos se puede obtener a través de la simulación.

Como ya se ha comentado la obtención de la síntesis se realiza a través del programa Synopsys, en el cual se ha utilizado una biblioteca de 90nm. En esta biblioteca el área de una puerta NAND es de 3,136 um^2. Este valor es importante ya que una vez que tengamos el área total del circuito,

dividiendo ese valor por el valor del área de la puerta NAND obtendremos el valor de las puertas equivalentes.

$$Puertas\ equivalentes = \frac{\text{Área total}}{\text{Área Puerta NAND}}$$

4.10.1 RESULTADOS SÍNTESIS ALU

La función hash va a ser implementada en 32 bit, por ello el primer estudio que se realiza es decidir sobre el número de bits de la Alu el cual se va a utilizar. En la Alu se ha implementado tanto la operación de suma como la de Xor.

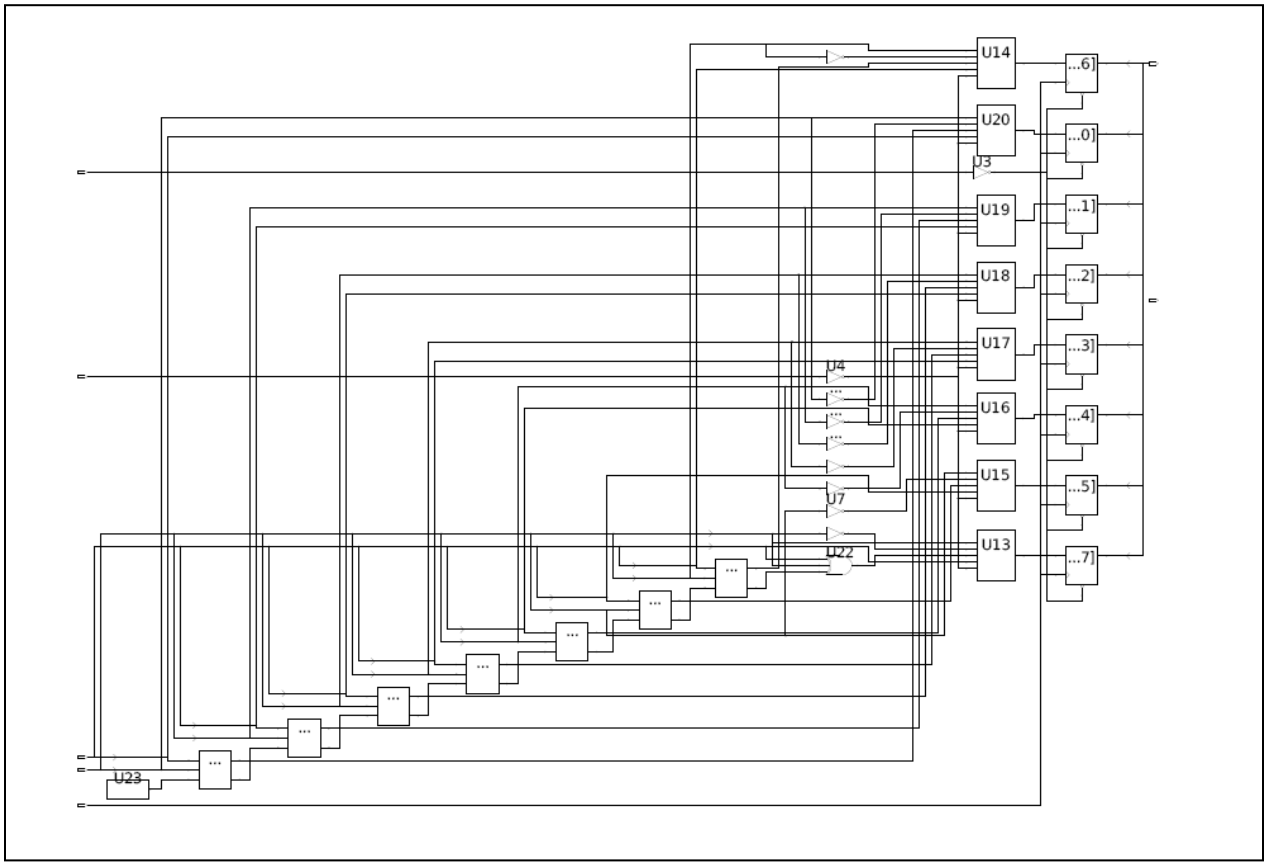


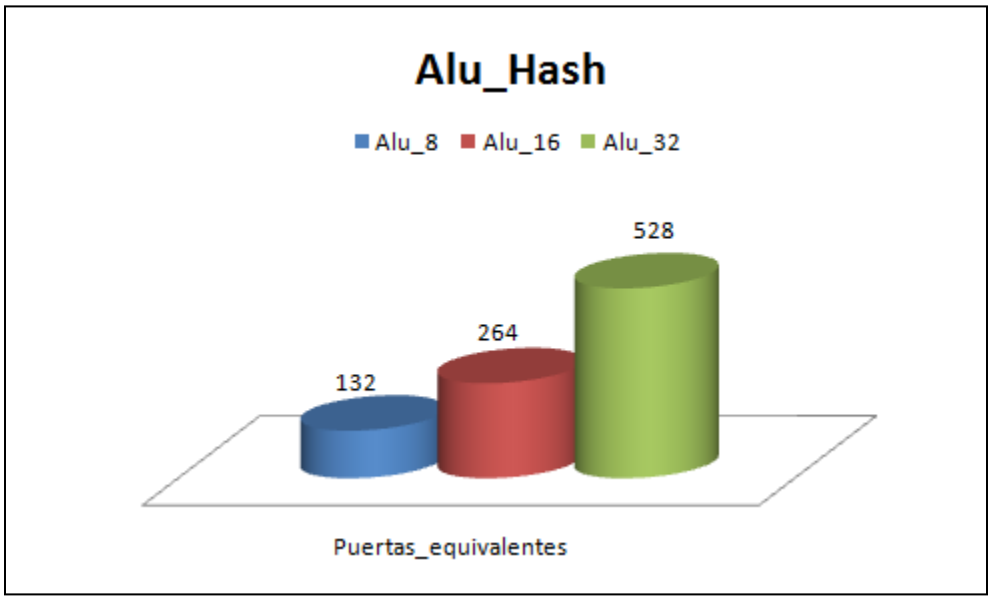
Figura 25. Hardware Alu

Los diseños de Alu implementado han sido en 32 bits, 16 bits y 8 bits. Más pequeño que 8 bits no sería útil ya que el tiempo de cálculo para la función luego sería mayor, y el hacerlo mayor de

32 tampoco tendría sentido ya que la función hash está establecida en 32 bits, por tanto las operaciones con mayor número de bits que se realizarán serán de 32 bits.

Resltados de síntesis ALU			
	Alu_8	**Alu_16**	**Alu_32**
Área Combinacional (μm^2)	301,84	602,896	1205,008
Área no combinacional (μm^2)	112,86	225,792	451,584
Área total (μm^2)	414,73	828,68	1656,592

Tabla 6. Resultados Síntesis Alu



Gráfica 1. Puertas Equivalentes Alu_hash

Debido a que la función hash será de 32 bits, el hecho de utilizar Alu de menor número de bits implica un aumento del número de ciclos para realizar las operaciones. Se realiza un estudio entre las ventajas y desventajas de reducir el número de bits de 32 bits a 16 bits:

a) El reducir la Alu de 32 bits a 16 bits se consigue:
    a. Reducir área Alu 50%
    b. Aumentar el tiempo en un 45%

Se demuestra que la reducción de área no justifica el aumento del tiempo que implica el hacer la Alu más pequeña de 32 bits, ya que la Alu sólo representa un 14 % del total del protocolo sería un ahorro de un 8 % en área del protocolo total.
Se utilizará la Alu_32 bits ya que el número de puertas equivalentes no es excesivamente grande y de esa manera el tiempo de cálculo para la función hash será menor.

4.10.2 RESULTADO SÍNTESIS FUNCIÓN HASH

El diseño implementado de la función hash esta especialmente diseñado para 32 bits. Una vez implementado el circuito el resultado de la síntesis es el siguiente.

Resultados de síntesis función hash	
	Hash
Número de puertos	199
Número de nodos	1609
Número de celdas	1485
Área Combinacional (μm^2)	5358,63991
Área no combinacional (μm^2)	4857,664
Área total (μm^2)	10216,304
Potencia Interna (nW)	122,073
Potencia conmutación (nW)	96,652
Potencia dinámica total (nW)	218,7263
Pérdidas de potencia (nW)	16,828

Tabla 7. Resultados Síntesis Función Hash

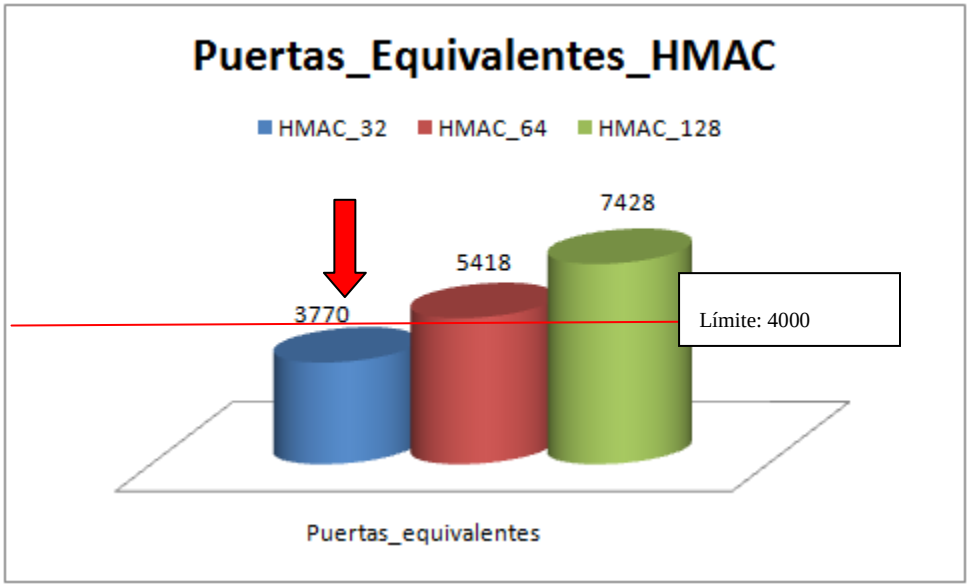
Se observa que el área ocupada es bastante grande por tanto el número de puertas usadas para la máquina de estados no podrá ser muy grande. Obteniendo el número de puertas equivalentes se tendría 3258. En este caso ya ha sido implementada la Alu de 32 bits. Por tanto para terminar el protocolo solo faltaría añadir la máquina de estados.

4.10.3 RESULTADO SÍNTESIS PROTOCOLO HMAC

Como a mayor número de bits mayor seguridad se han implementado 3 protocolos HMAC con distintos número de bits (32, 64, 128). Hay que recordar que para los 3 protocolos el bloque Hash+Alu es el mismo (32 bit) lo que se modifica es el número de bits usado en la máquina de estados.

Resltados de síntesis Protocolo HMAC			
	HMAC_32	HMAC_64	HMAC_128
Número de puertos	70	266	394
Número de nodos	300	765	1696
Número de celdas	5	5	5
Área Combinacional (μm^2)	6514,255	9241,458	11532,54
Área no combinacional (μm^2)	5310,816	7750,81	11764,12
Área total (μm^2)	11825,072	16992,268	23296,66
Potencia Interna (nW)	234,139	278,7	284,45
Potencia conmutación (nW)	57,733	75,89	97,2
Potencia dinámica total (nW)	291,872	354,59	381,65
Pérdidas de potencia (nW)	18,835	27,45	33,6

Tabla 8. Resultados de la síntesis Protocolo HMAC



Gráfica 2. Puertas Equivalentes HMAC

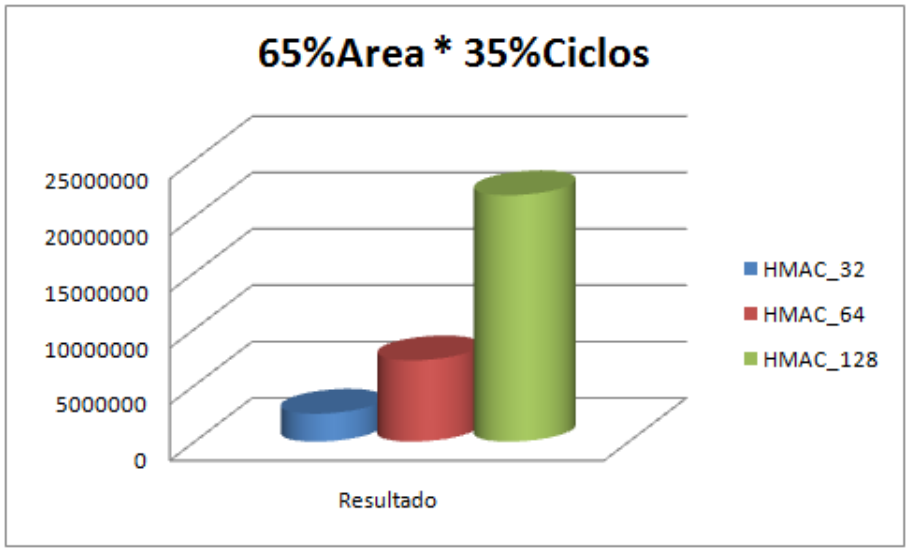
La restricción que se tiene en cuanto a puertas equivalentes es de 4000 puertas, de los 3 diseños implementados solo el de 32 bits es el que cumpliría con esa restricción, por el contrario se perdería seguridad en comparación con los de 64 y 128 bits.

También se ha realizado un estudio del tiempo de cómputo de cada diseño.

Número de ciclos para Protocol HMAC			
	HMAC_32	HMAC_64	HMAC_128
Ciclos	2050	4090	8175
Tiempo a 100 Khz (ms)	21	41	82

Tabla 9. Número de Ciclos HMAC

Para el caso de 64 y 128 bits tienen que utilizar la función hash varias veces ya que está establecida en 32 bits, por tanto habrá que dividir los registros de estos protocolos en N/2 (64 bits) o N/4 (128 bits) para poder obtener el resultado final del cálculo hash. Para el caso de 32 bits es el que menos tiempo tarda ya que simplemente utiliza una vez en el caso en que necesita realizar el cálculo de la función hash.



Gráfica 3. Comparación Protocolo HMAC

En esta gráfica se representa una media de las 2 variables que se considera en este caso importante, como son el área y ciclos. Se da más importancia al área ocupada ya que es una restricción más importante que la del tiempo necesario.

Debido a la restricción de las puertas equivalentes se ha decidido que la mejor opción para el protocolo es usar máquina de estados de 32 bits.

La función hash utilizada será también de 32 bits ya que ha sido diseñada con ese número de bits, la Alu que utiliza la función hash es también de 32 bits, ya que hay suficiente espacio como para aprovechar los 32 bits de la hash y si la Alu se reduce a menos bits el tiempo de cálculo sería bastante mayor.

Por ello el circuito elegido quedaría con Protocolo (32 bits)+ Hash (32 bits) + Alu (32 bits).

4.11 CONCLUSIONES PROTOCOLO HMAC

En este apartado se ha realizado un estudio sobre la implementación del protocolo HMAC. En este protocolo se incluye una máquina de estados, la función hash y una Alu con operaciones de suma y Xor.

Al hacer el primer estudio sobre el número de bits de la Alu se considera que lo mejor sería utilizar el mismo número de bits que use la función hash, la cual está establecida en 32 bits.

Al hacer el estudio sobre el número de bits del protocolo se considera que el único viable sería utilizar un protocolo con 32 bits, ya que un protocolo con mayor número de bits ocuparía mucho espacio y estaría por encima de las 4000 puertas equivalentes. Se recuerda que la restricción impuesta para las etiquetas a las cuales se dirige este proyecto está en torno a 3000-4000 puertas.

Por tanto el diseño seleccionado para una posible implementación sería la siguiente:

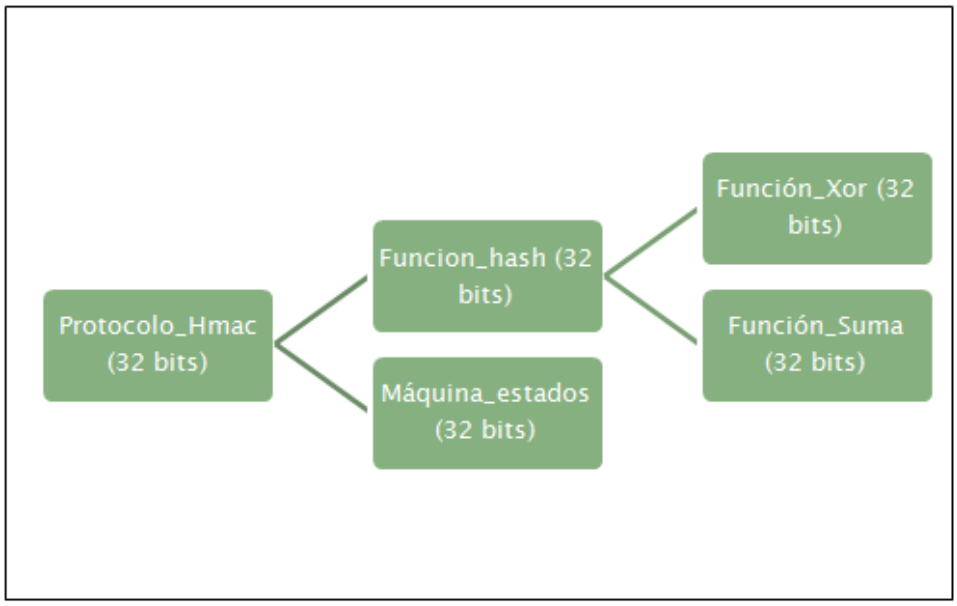
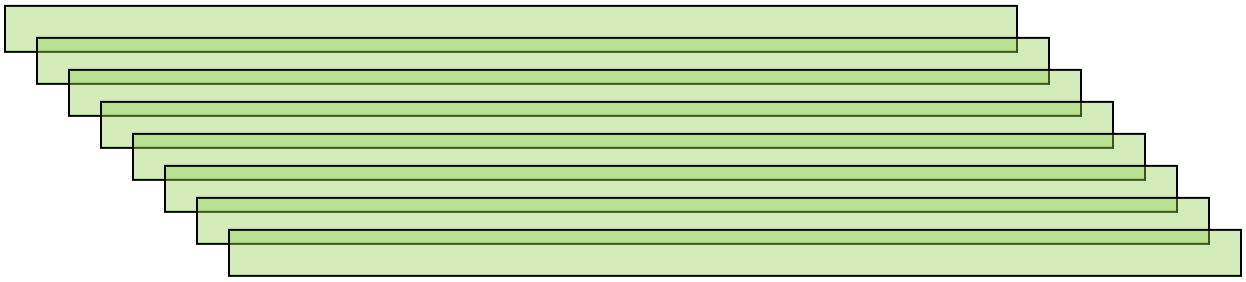


Figura 26. Protocolo Final HMAC

CAPÍTULO 5



PROTOCOLO DE AUTENTIFICACIÓN AZUMI

5.1 DESCRIPCIÓN PROTOCOLO AZUMI

Pasos del protocolo Azumi [11] [12] [13]:

1) El lector envía un mensaje a la tag, en ese momento empieza el protocolo de autentificación. Los datos enviados serán M_{req} y RND_{lector}
2) Un vez recibido el mensaje, la tag generará dos números aleatorios (RND_t, RND_t').
3) A través de operaciones de XOR y el generador de números pseudoalatorios se realizará las siguientes operaciones para obtener A,B,C.
    a. A= PRNG (N_t XOR RND_t XOR RND XOR ID16)
    b. B=PRNG ($EPC16_t$ XOR RND_t')
    c. C= RND_t' XOR K16
4) Para el cálculo de A tenemos N_T que será el doble de bits que las demás variables, se realizará primero el XOR de la parte baja de N_T, para posteriormente concatenarlo con la parte alta de N_T e introducirlo en el PRNG
    a. A=PRNG(N_t(high) & N_t(low) XOR RND_t XOR RND XOR ID16)
5) Una vez calculadas estas variables, la tag enviará al lector el siguiente dato {RND_t A B C}
6) El lector una vez recibido el dato de la tag realiza dos operaciones:
    a. Identificación de la tag: Para el caso de que hubiese varias tag a las cuales se quiere acceder.
    b. Autentificación de la tag: El lector obtendrá un valor de A propio y lo comparará con el valor enviado por la tag, en el caso de que la comparación de valores distintos el protocolo es abortado. Si los dos valores son iguales el lector calculará el siguiente valor D.
        i. D= PRNG(N_t XOR RND_{lector} XOR RND_t XOR EPC16 XOR RND_t).
    c. Actualización de los datos N16 y K16 almacenadas por el lector.
7) El lector envía D a la tag.
8) La tag una vez que recibe D del lector calcula su propio valor D y lo comparará con el valor enviado del lector.
    a. D= PRNG(N_t XOR RND_{lector} XOR RND_t XOR EPC16 XOR RND_t). En este caso igual que para el cálculo de A, N_t es del doble de bit que las demás variables por tanto se realizará primero la parte baja para luego concatenarlo con la parte alta.

b. D= PRNG(N_t(high) & N_t(low) XOR RND_{lector} XOR RND_t XOR EPC16 XOR RND_t)

9) Si $D_{lector}=D_{tag}$, el proceso de autentificación es completado.

10) Por último la tag realiza la actualización de las siguientes variables.

a. N16 = PRNG (N16 XOR RND')

b. K16 = PRNG (K16 XOR RND')

La comunicación entre la tag y el lector se hará a través de un canal de N bit para los cuales los distintos datos se representarán de la siguiente manera [13] [14].

Variable	Valor
N_t	Password de la tag de 32 bit
N16	Este valor se obtiene de hacer la xor a los dos bloques en los que se divide N_t
K_t	Password kill de la tag de 32 bit
K16	Este valor se obtiene de hacer la xor a los dos bloques en los que se divide K_t
EPC	Identificación EPC de la tag de 96 bit
EPC16	Haciendo XOR de los 6 bloques de la EPC se obtiene EPC16
ID16	Número de identificación del lector de 16 bit
PRNG()	Generador de número seudo-aleatorio de 16 bit
RND	Número aleatorio de 16 bit
RND'	Número aleatorio secundario de 16 bit
XOR	Función xor
M_{req}	Mensaje solicitado por el lector

Tabla 10. Variables Protocolo Azumi

En la siguiente figura se muestra un esquema con la comunicación que se realiza en el protocolo así como las operaciones que realiza tanto la tag como el lector.

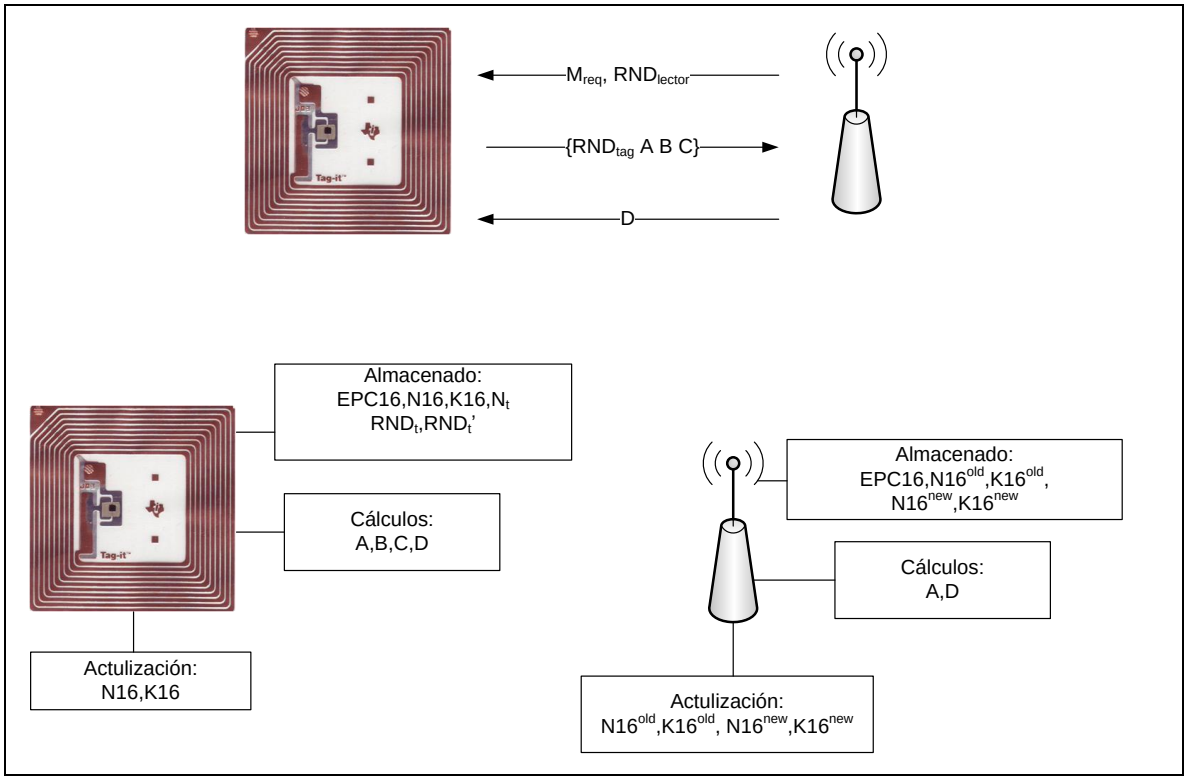


Figura 27. Funcionamiento Azumi

La parte del lector no implica ningún problema el realizar mucho cálculos ya que esta parte se considera con mayores recursos que la tag. En esta última si habrá que tomar aspectos como es el área ocupada ya que es una restricción importante a la hora de implementar el protocolo en el chip.

5.2 DIAGRAMA DE FLUJO PROTOCOLO AZUMI

A continuación se muestra el diagrama de flujo del protocolo implementado. En este caso se representa el diagrama de flujo para el protocolo de 16 bits, en la implementación en vhdl se ha realizado también para 32 y 64 bits.

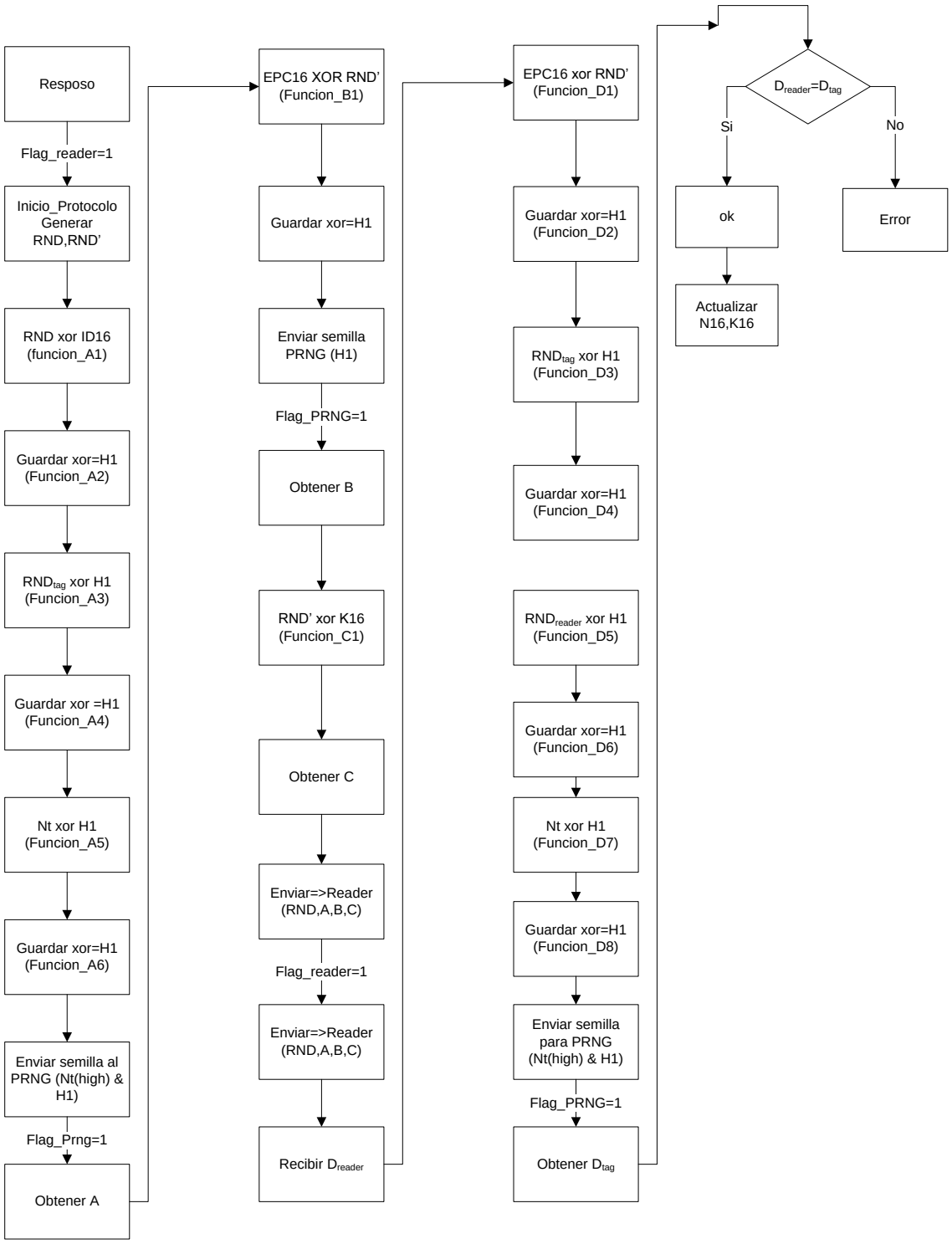
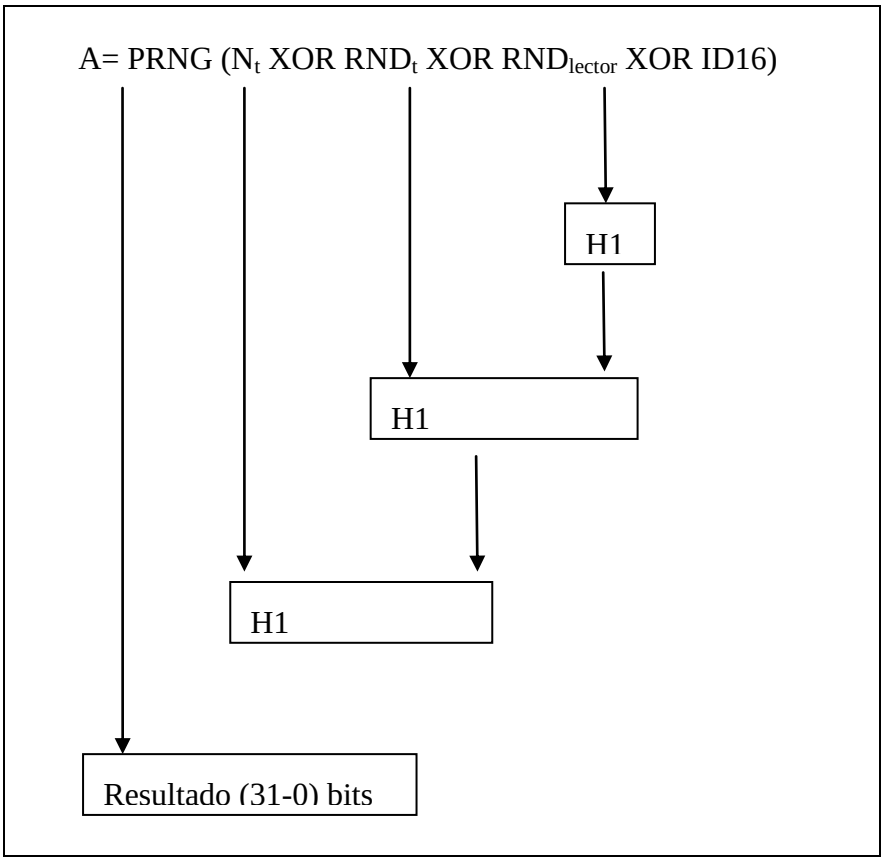
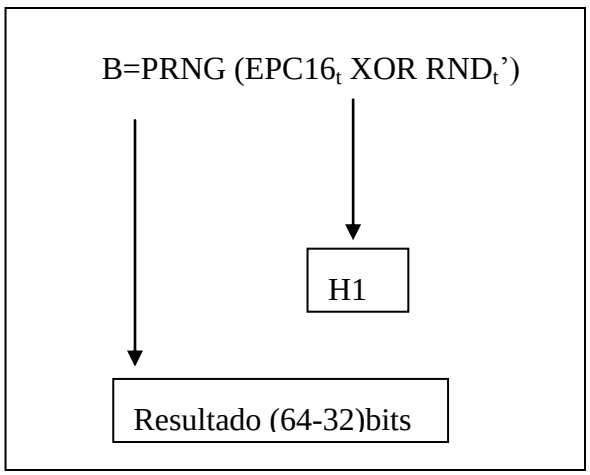


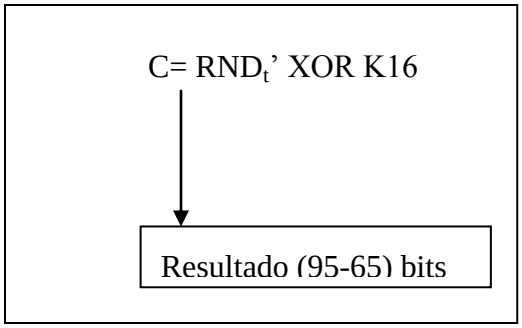
Figura 28. Diagrama de flujo Azumi

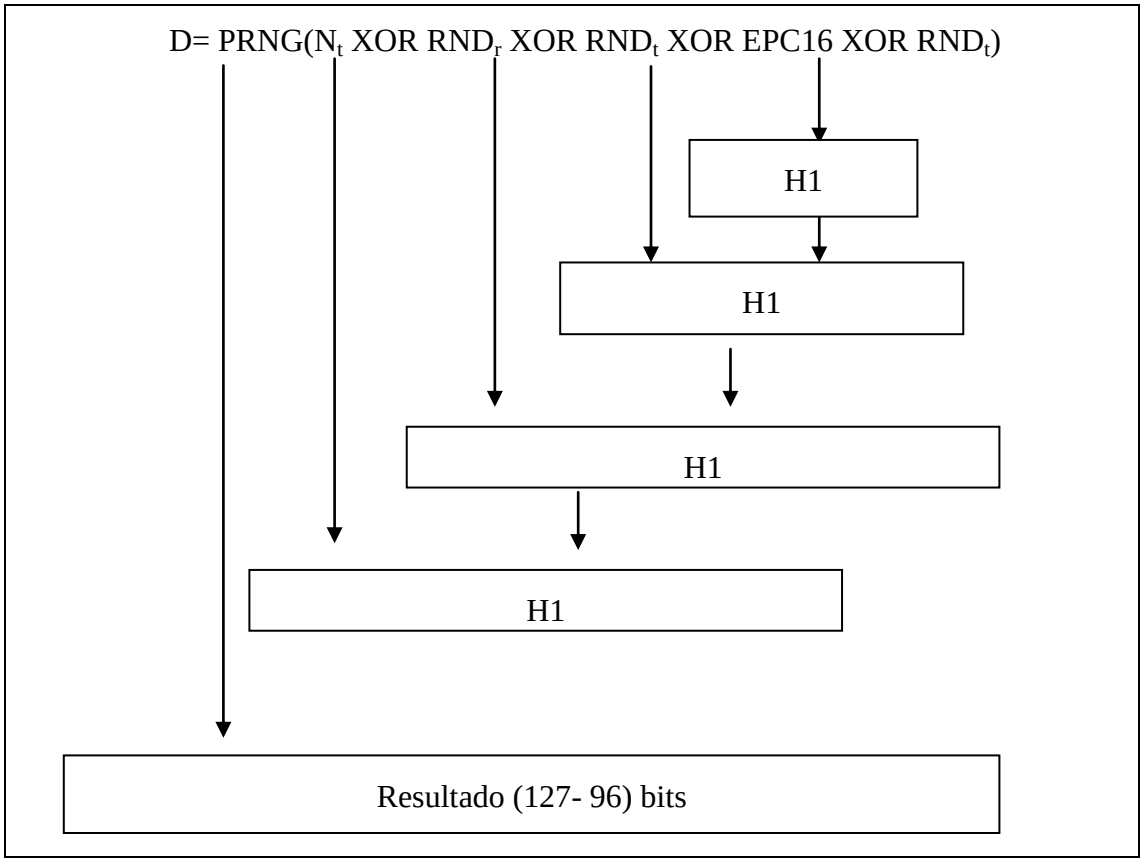
5.3 REGISTROS USADOS EN EL PROTOCOLO

A continuación se muestra la forma que han sido usados los registros en el protocolo, se ha utilizado el menor número de registros posibles de manera que el circuito ocupe el menor área posible. De esta manera se intenta cumplir con la restricción de área impuesta en este proyecto de 3000- 4000 puertas equivalentes.









El registro Resultado será de 256 bits, ya que en los primeros 128 bit se obtienen el dato D enviado por el lector y en los otros 256 bit se guardará el resultado D calculado por la tag. De esta manera se podrá hacer la comparación.

5.4 DIFERENTES DESARROLLOS DE BITS

Para la implementación del protocolo existen diferentes combinaciones posibles en función del número de bits de la máquina de estados, de la función Xor, del sumador o de los Prng.

En la siguiente tabla se muestran las implementaciones hechas y los cuales se han sintetizado para poder calcular cual es el mejor en función del área y tiempo.

Para el protocolo de 16 es el protocolo el cual ha sido obtenido de la bibliografía.

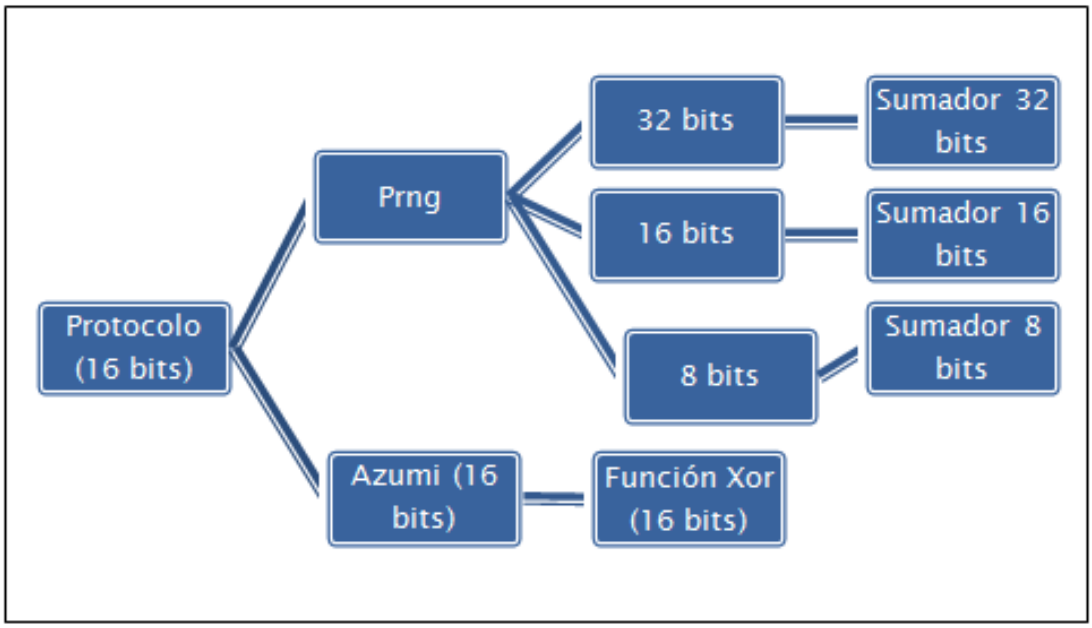


Figura 29. Protocolo 16 bits Azumi

Para usar el protocolo con 32 bits se necesitan también doblar el número de bits del generador pseudo-aleatorio, ya que este generador implementado para N bits tiene una salida de N/2 bits.

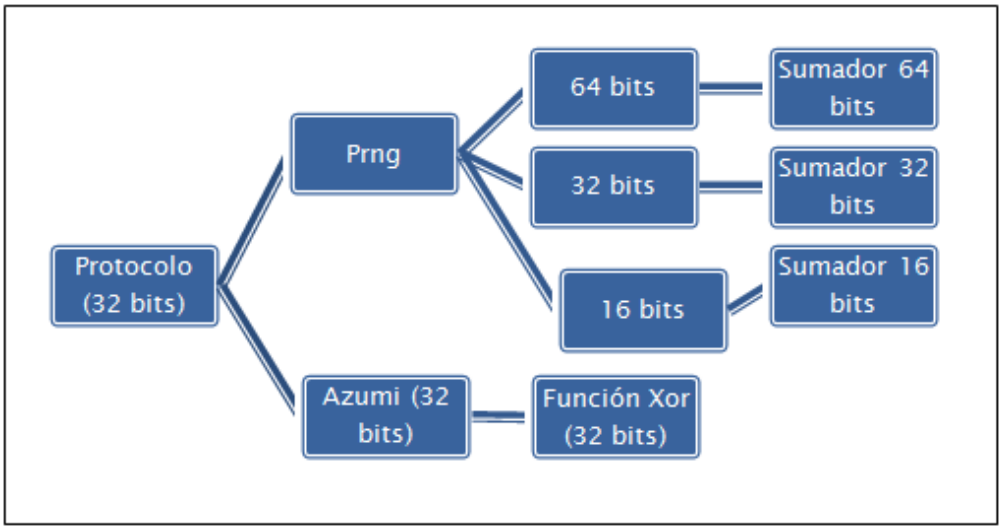


Figura 30. Protocolo 32 Bits Azumi

Debido que a mayor número de bits la seguridad del protocolo será mayor, se ha intentando probar con un número de bits elevado e intentar que ocupe lo menos posible para que pueda ser implementado en la tag.

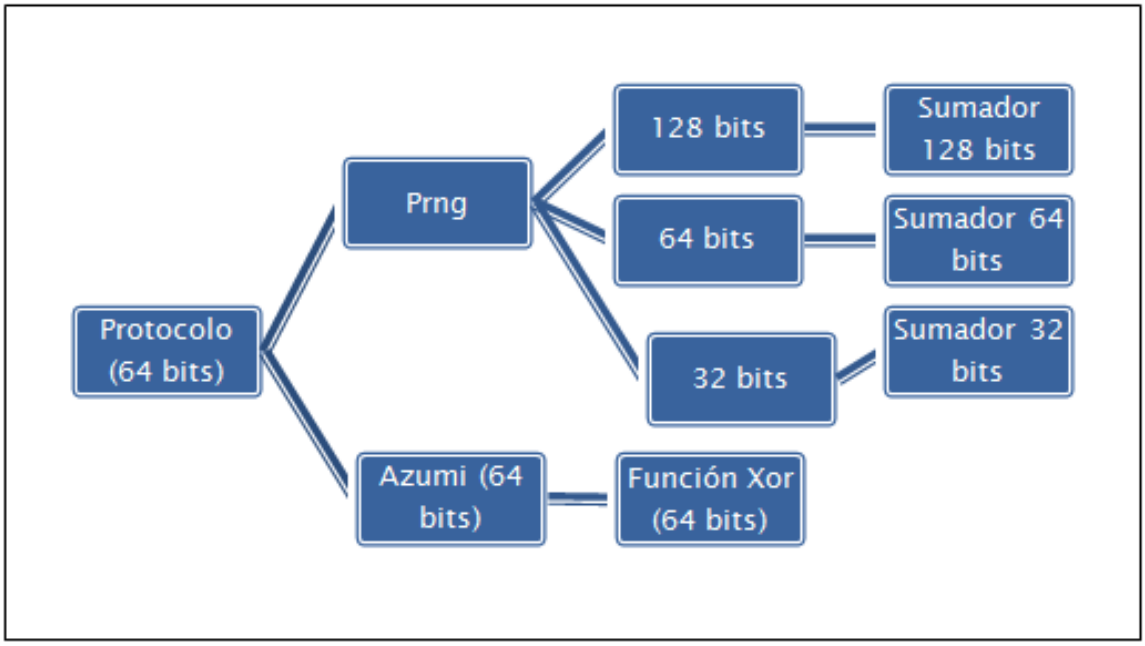


Figura 31. Protocolo Azumi 64 bits

5.5 DIAGRAMA DE BLOQUES

En el siguiente esquema se muestra el diagrama de bloques total del protocolo en función de las conexiones entre los diferentes bloques. En este caso se muestra el diagrama de bloques para 16 bits, ya que para 32 bit y 64 bit el diagrama es el mismo simplemente cambia el número de bits.

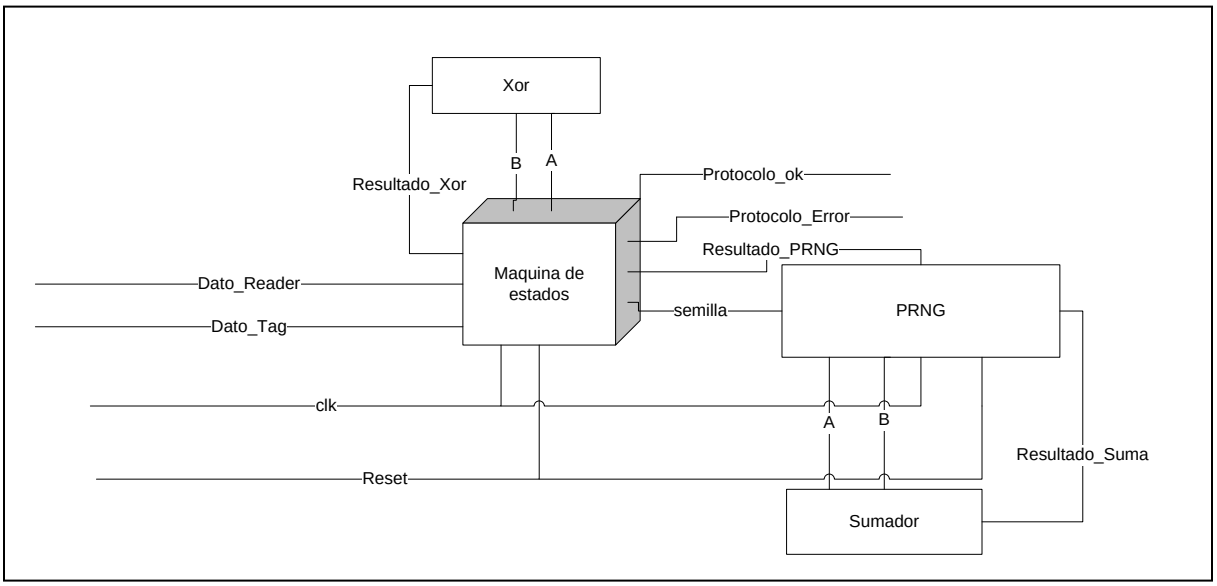


Figura 32. Hardware Azumi

Se observa cómo tanto la función Xor como el Sumador se han implementado por separado, esto ha sido así debido a que el generador PRNG sólo utiliza sumadores y es la máquina de estados la que sólo utiliza funciones Xor para calcular la semilla del PRNG.

5.6 IMPLEMENTACIÓN PRNG'S

Para la implementación de este protocolo son necesarios algoritmos de cálculo de números pseudoaleatorios. En este proyecto se han utilizado algoritmos los cuales ya han sido diseñados e implementado y simplemente se ha modificado ciertas partes del código para adaptarlo al protocolo.

Se han utilizado 4 diseños para poder calcular cual se adapta mejor al protocolo, tomando en cuenta los datos de área y tiempo. Estos 4 algoritmos se distingues por un lado por el número de bits, y por otro lado por la utilización de uno o varios sumadores.

A continuación se muestran los esquemas de los distintos Prng's usados junto con sus simulaciones.

a) Prng.1

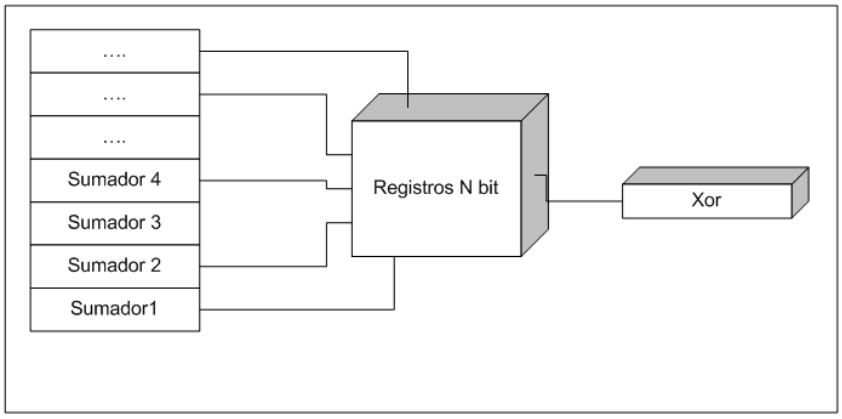
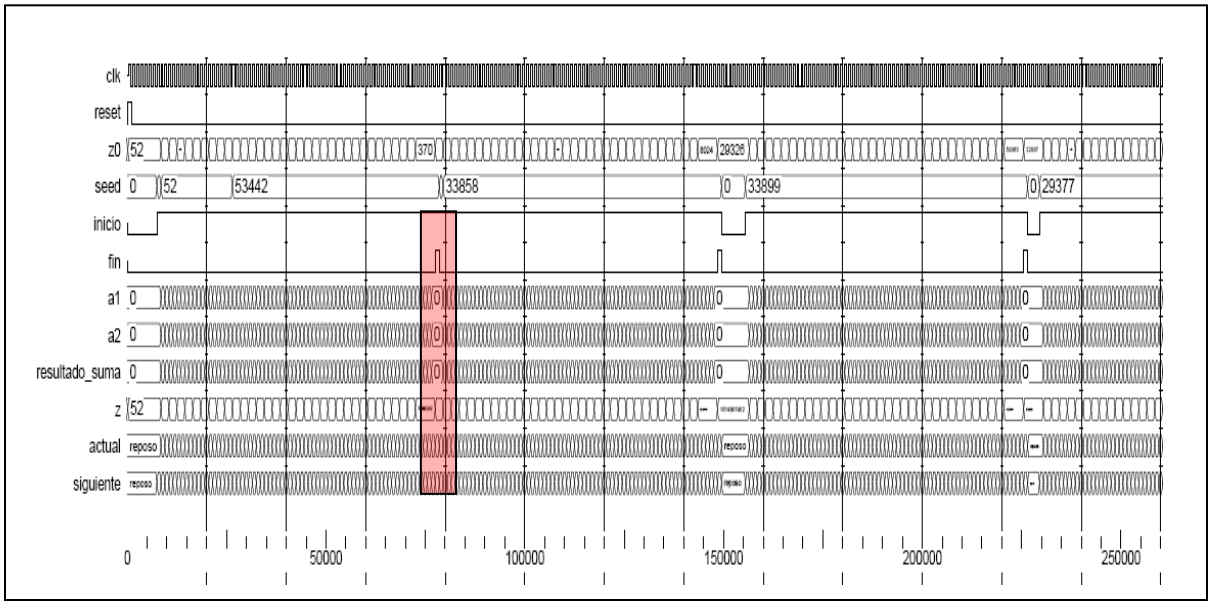


Figura 33. Hardware PRNG 1

Este diseño será el más rápido ya que es el que más sumadores usa, por el contrario será el que más área ocupa.



Simulación 12. Prng 1

b) Prng 2.

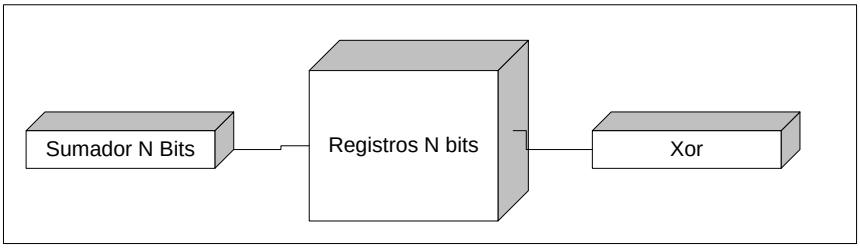
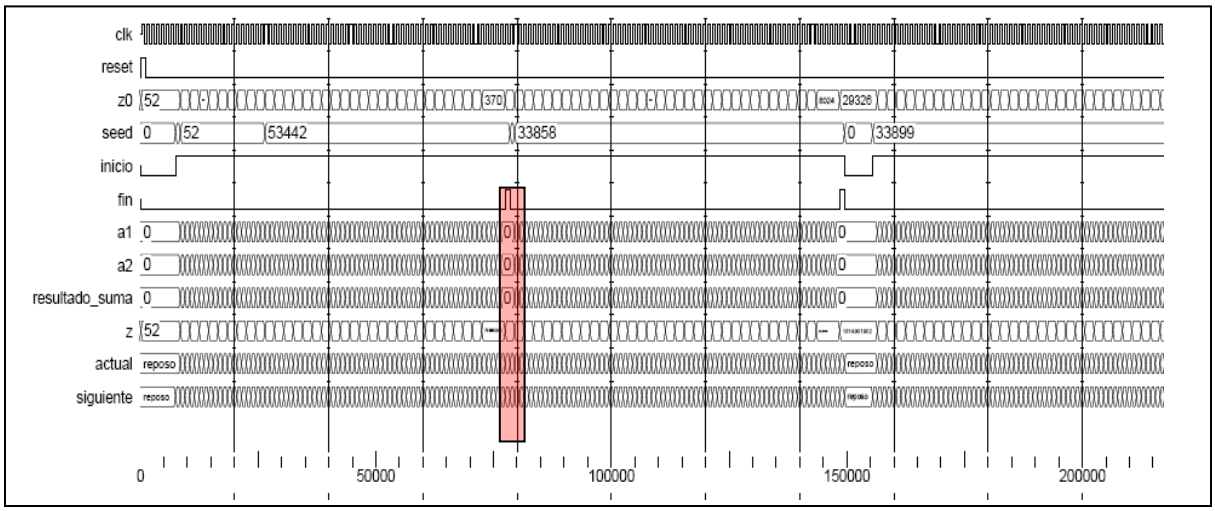


Figura 34. Hardware PRNG 2

En este diseño usará el mismo número de bits que el anterior pero esta vez sólo usa un sumador, se tendrá que usar el mismo sumador varias veces para obtener el mismo el resultado, por tanto tendrá un tiempo de procesamiento mayor.



Simulación 13. Prng 2

c) Prng 3.

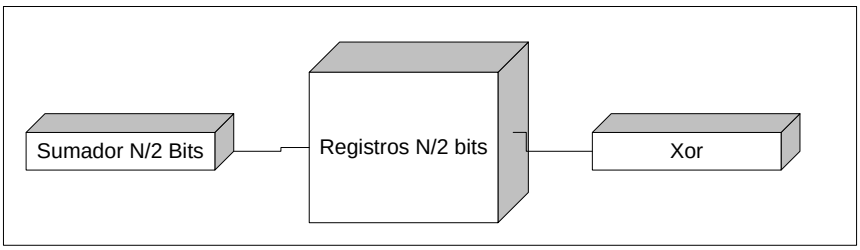
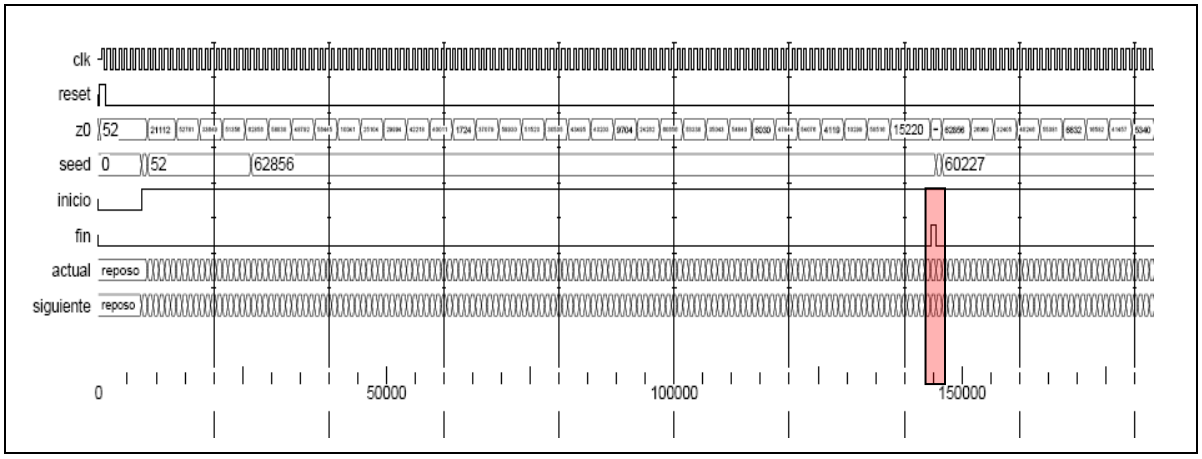


Figura 35. Hardware PRNG 3

En este diseño se hace con un sumador de N/2 bits, para realizar una suma habrá que hacerlo en dos pasos, realizando primero la parte baja y posteriormente la parte alta.



Simulación 14. Prng 3

d) Prng 4.

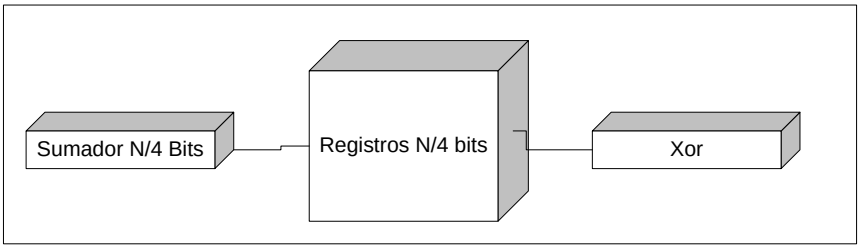
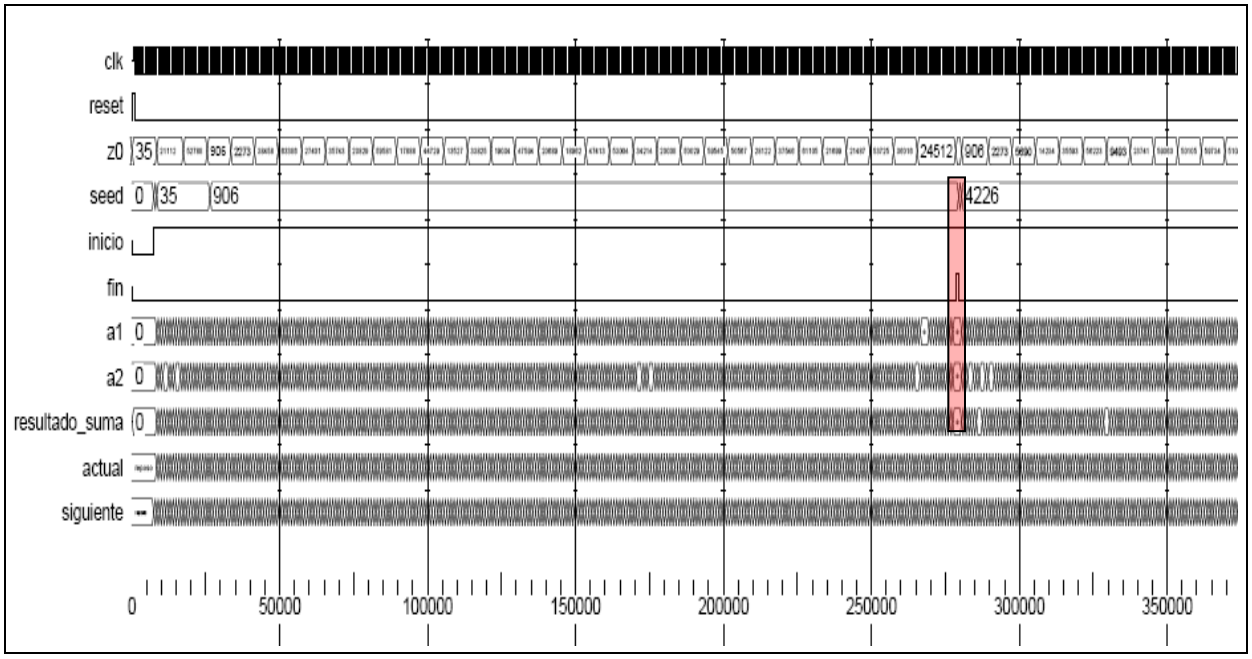


Figura 36. Hardware PRNG 4

En este diseño se calcula con un sumador de N/4 bits. Será el diseño que más tardará ya que se necesitará 4 ciclos para realizar una suma, por otro lado será el que menos ocupe.



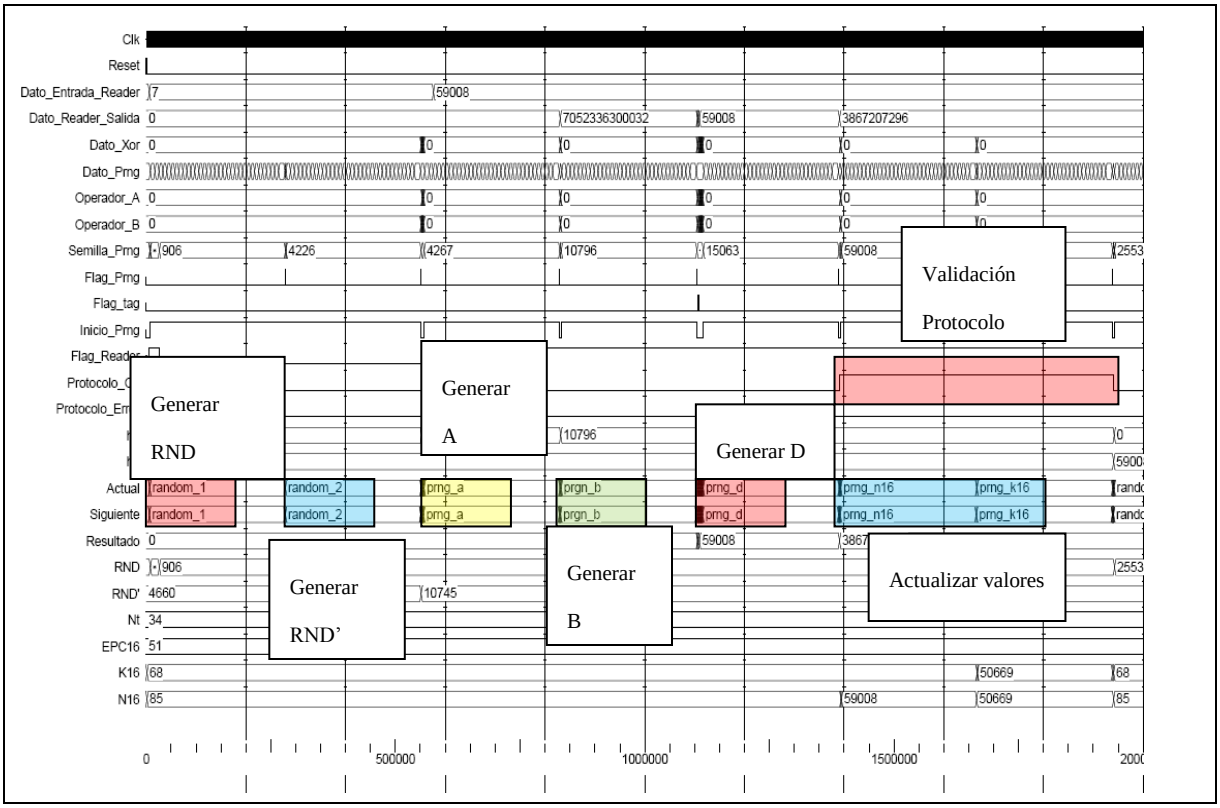
Simulación 15. Prng 4

5.7 SIMULACIONES

A continuación se muestra las diferentes simulaciones del protocolo. Con estas simulaciones se pretende conseguir el funcionamiento correcto del protocolo además de obtener el número de ciclos el cual necesitamos para realizar los cálculos.

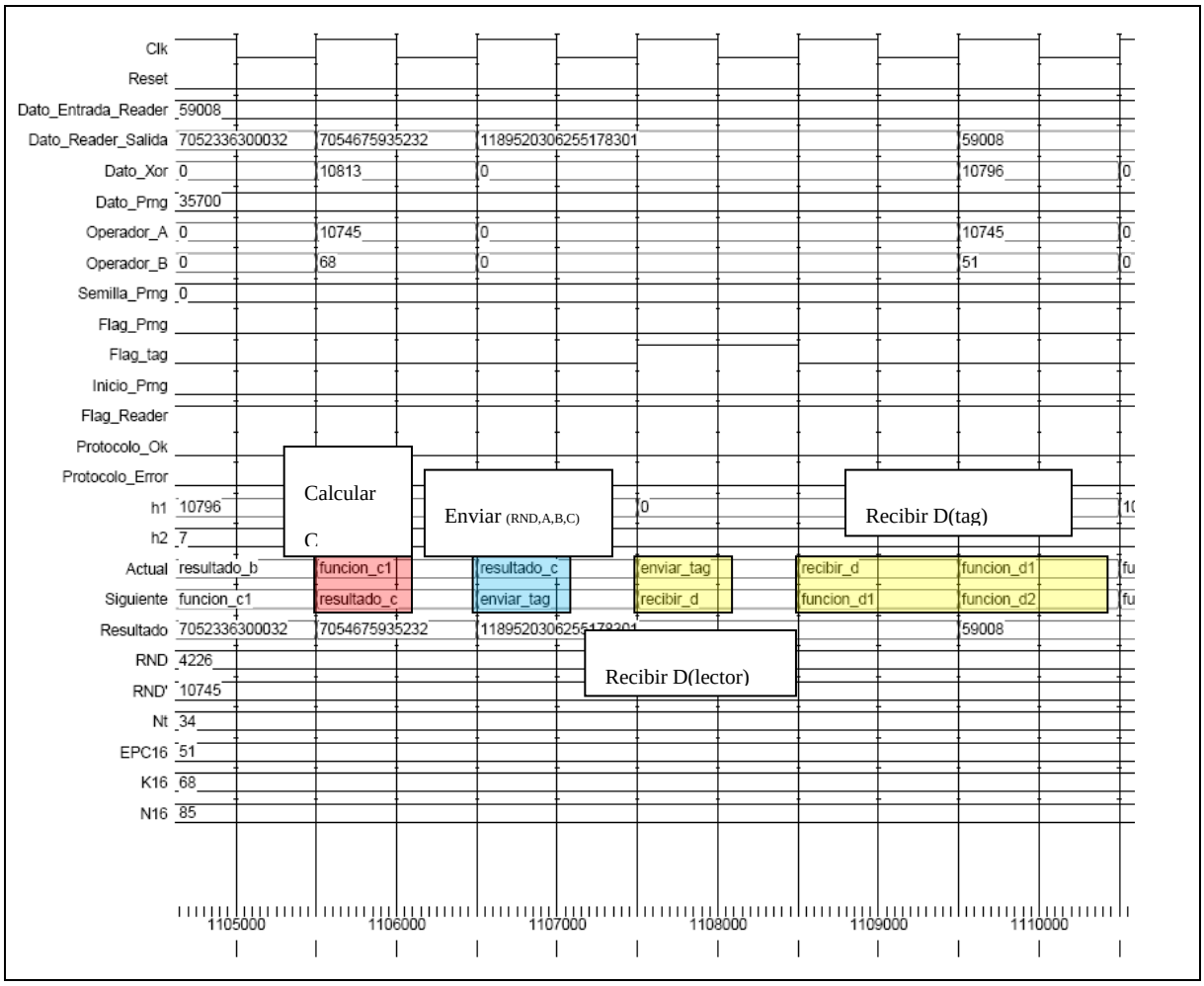
Simulaciones Protocolo

En la siguiente simulación se muestra el funcionamiento del protocolo entero.



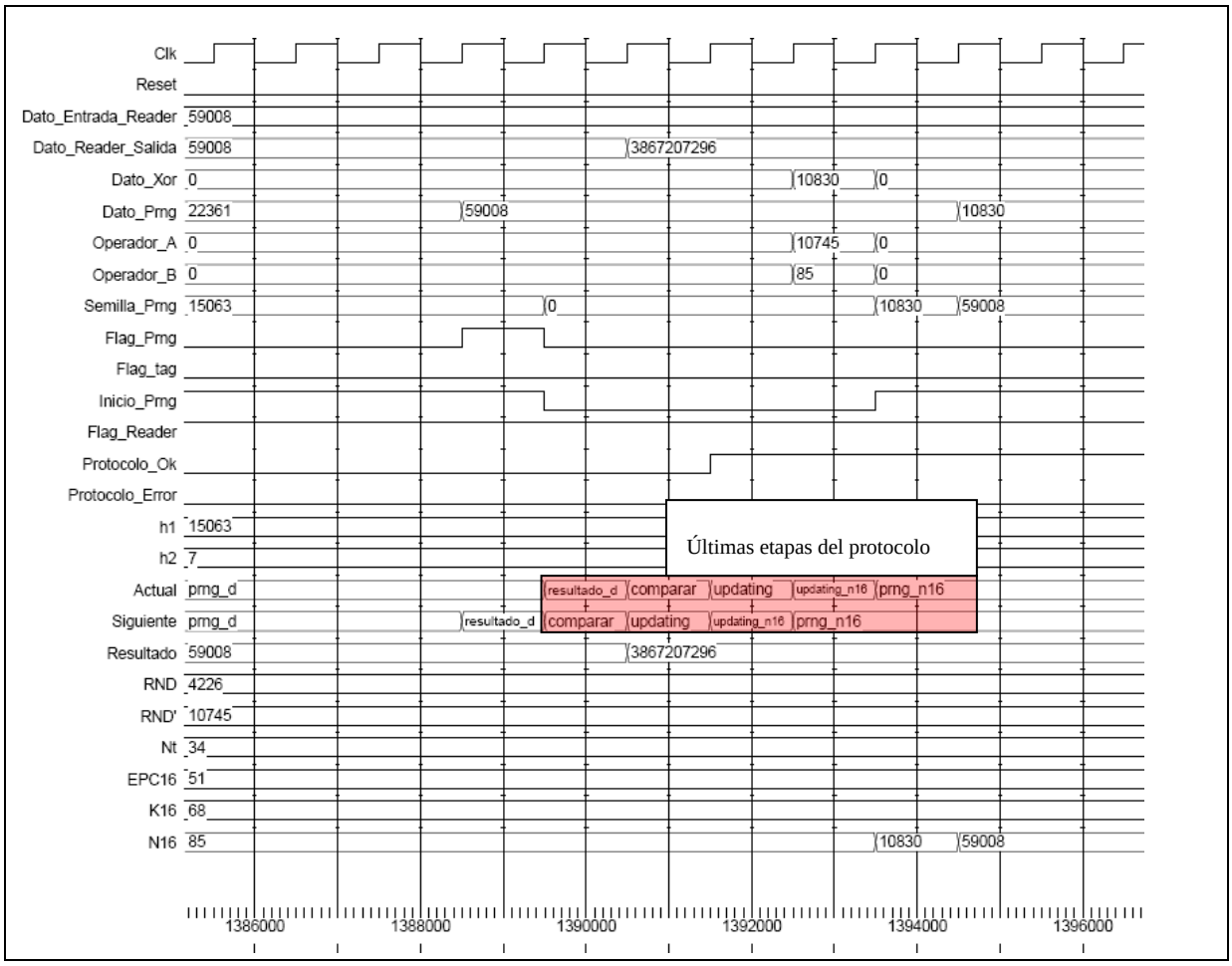
Simulación 16. Protocolo Azumi Parte 1

En la siguiente simulación se muestra como para calcular la función C no hace falta calcular el número pseudoaleatorio por tanto tardaremos un ciclo en calcular C.



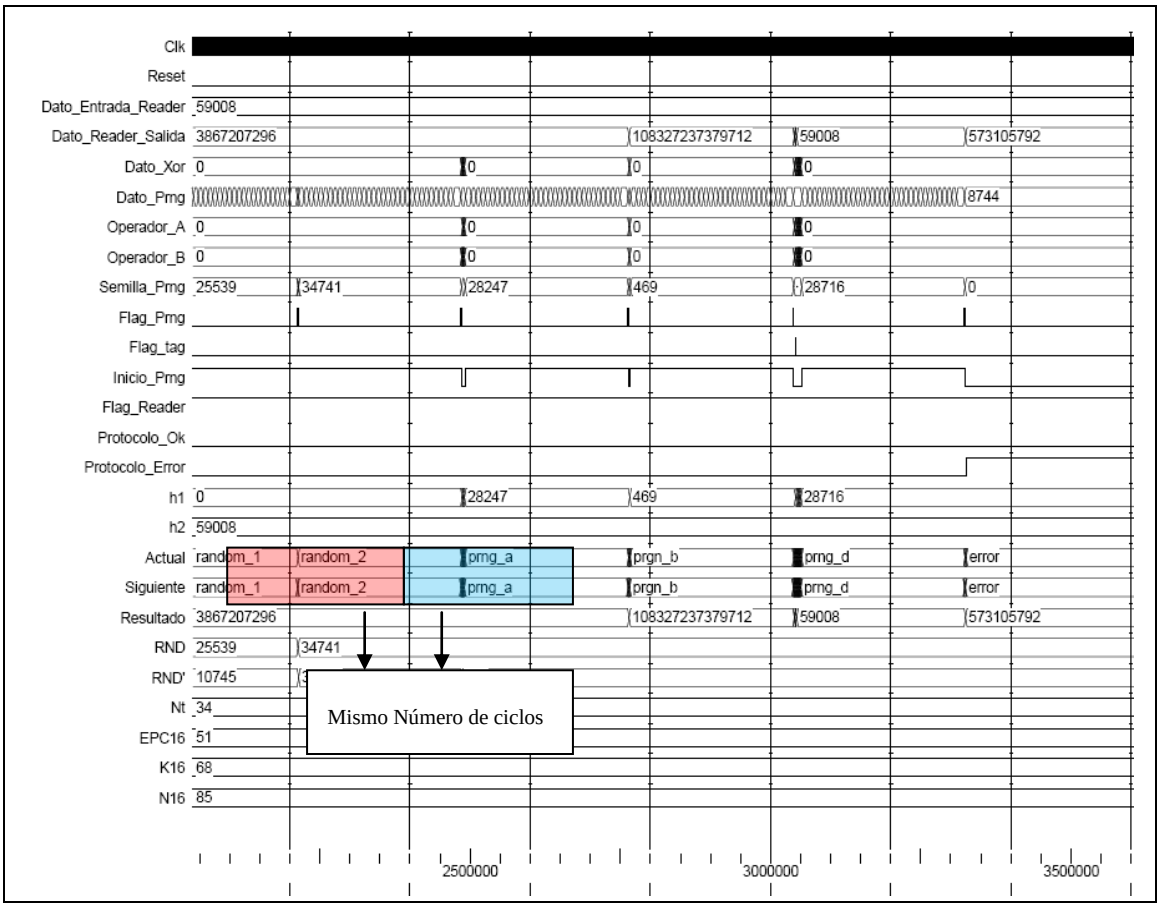
Simulación 17. Protocolo Azumi Parte 2

Las últimas etapas del protocolo una vez que se han enviado los datos se centra en la actualización de las variables $N16_t$ y $K16_t$



Simulación 18. Protoclo Azumi Parte 3

Las etapas de Random1, random2, tardan lo mismo que la generación de números PRNG, ya que se utiliza este generador para el cálculo de los números aleatorios. Esto se hace así de manera que se aproveche el espacio ocupado por el generador PRNG y el protocolo no ocupe demasiado.



Simulación 19. Protocolo Azumi 32 bits

En esta simulación se muestra el protocolo implementado con 32 bit a diferencia de 16 bit. Se observa que el tiempo menor que para 16 bit, por el contrario el espacio necesario será mayor como se mostrará más adelante.

5.8 RESULTADOS SÍNTESIS PROTOCOLO AZUMI

Al igual que para el protocolo Hmac la síntesis se ha realizado con el programa Synopsys. Se ha utilizado la misma librería de 90 nm por tanto el área de una puerta NAND es la misma y para obtener las puertas equivalentes se divide el área total entre los 3,136 um^2 que ocupa una puerta NAND. Con la síntesis también se obtendrá los consumos de potencia.

5.8.1 RESULTADOS SÍNTESIS PRNG

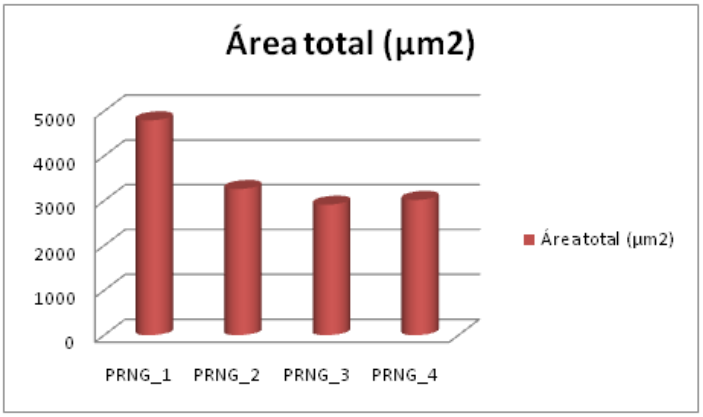
A continuación se muestra el resultado de la síntesis para los 4 generadores los cuales se han implementado en el protocolo.

a) PRNG_1: Varios sumadores
b) PRNG_2: Un sumador de N bits.
c) PRNG_3: Un sumador de N/2 bits.
d) PRNG_4: Un sumador de N/4 bits.

Resltados de síntesis de los generadores pseudo-aleatorio				
	PRNG_1	PRNG_2	PRNG_3	PRNG_4
Número de puertos	148	148	52	52
Número de nodos	590	526	504	517
Número de celdas	399	393	386	405
Área Combinacional (μm^2)	4168,528	2127	1564,863	1652,671
Área no combinacional (μm^2)	626,416	1134,448	1346,911	1364,944
Área total (μm^2)	4794,944	3261,448	2911,775	3017,616
Potencia Interna (nW)	76,129	69,194	66,07	65,38
Potencia conmutación (nW)	37,15	23,969	20,162	15,583
Potencia dinámica total (nW)	113,2796	93,154	86,233	80,918
Pérdidas de potencia (nW)	8,0863	4,273	4,297	4,81

Tabla 11. Resultado Síntesis PRNG's

En cuanto al área total se observa como el PRNG_1 es el que más ocupa debido a que es el que más sumadores incorpora, al contrario que los otros 3 los cuales usan un solo sumador.
En cuanto a los valores de potencia el PRNG_1 es el que más potencia dinámica total consume así como el mayor de pérdidas.



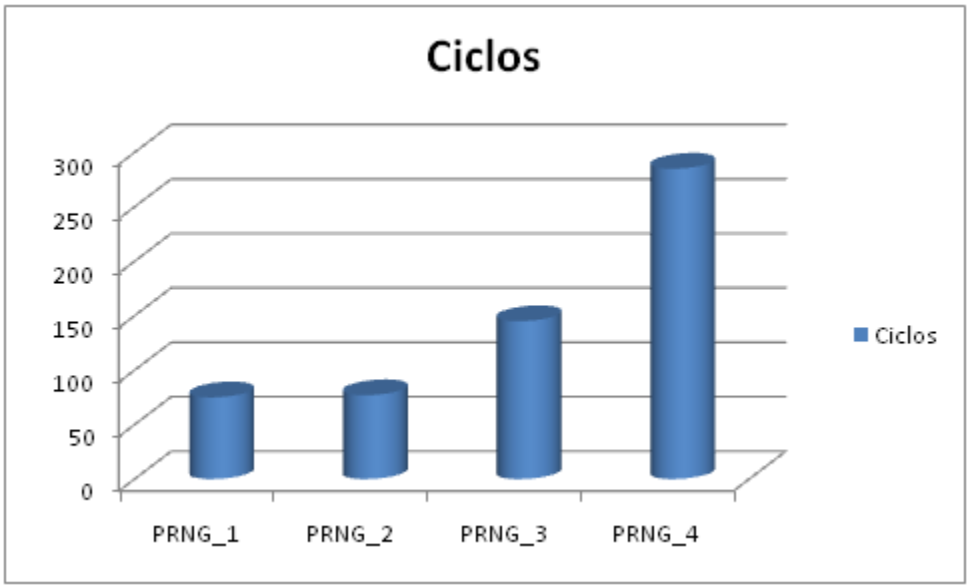
Gráfica 4. Área PRNG's

En esta gráfica se muestra como los 3 generadores que usan un único sumador tienen un área muy similar.
Los generadores 3 y 4 deberían ocupar bastante menos área que el generador 2 ya que usan un sumador de N/2 y N/4 respectivamente, esto no ocurre debido a que el número con el cual lo estamos usando 16 bits no es lo suficiente grande como para que se haga notar esa diferencia. Si se estuviera usando por ejemplo para un protocolo de 512 bits el generador 4 sería el más adecuado ya que el 2 y el 3 ocuparían más.

También se hace un análisis del tiempo que tarda en realizar el cálculo cada generador, la tag tendrá una velocidad de aproximadamente 100 Khz.

Número de ciclos para cada PRNG				
	PRNG_1	PRNG_2	PRNG_3	PRNG_4
Ciclos	75	77	145	285
Tiempo a 100 Khz (µs)	750	770	1450	2850

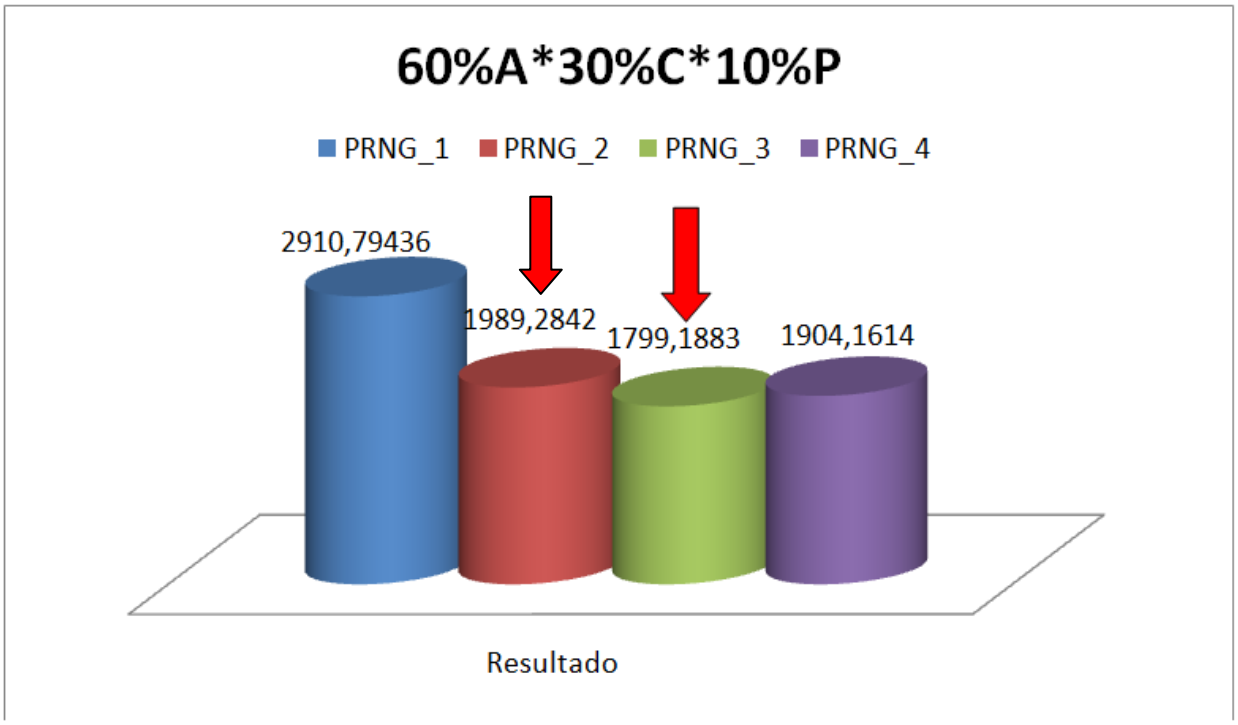
Tabla 12. Número de ciclos PRNG's



Gráfica 5. Número de ciclos PRNG's

En este caso el generador 1 y 2 serían los más adecuados ya que serán los que menos tiempo tarden. El 3 y el 4 al tener que realizar N veces el número de sumas el tiempo se incrementa.

Ya que se tiene 3 variables las cuales son interesantes a la hora de tomar una decisión de cuál es el mejor generador, se ha calculado un valor el cual se representa una media entre el valor de área, tiempo y potencia. Se considera que el valor de área es el más importante y por ello se le da un 60% del valor, el siguiente valor que más importancia se le da será el número de ciclos con un 30% y por último el valor de la potencia dinámica que se establece con un valor del 10% sobre el total.



Gráfica 6. Comparación PRNG's

Se puede observar que el PRNG_3 es el valor menor comparado con los otros generadores, por tanto se puede decir que en este caso sería el generador pseudo-aleatorio más recomendable para que el protocolo Azumi de 16 bits.

5.8.2 RESULTADO SÍNTESIS AZUMI_16 BITS

Se ha integrado los diferentes generadores pseudo-aleatorios dentro del protocolo para ver el comportamiento total del protocolo y confirmar que el generador PRNG_3 es el más adecuado y observar el número de puertas equivalentes que ocupa el protocolo.

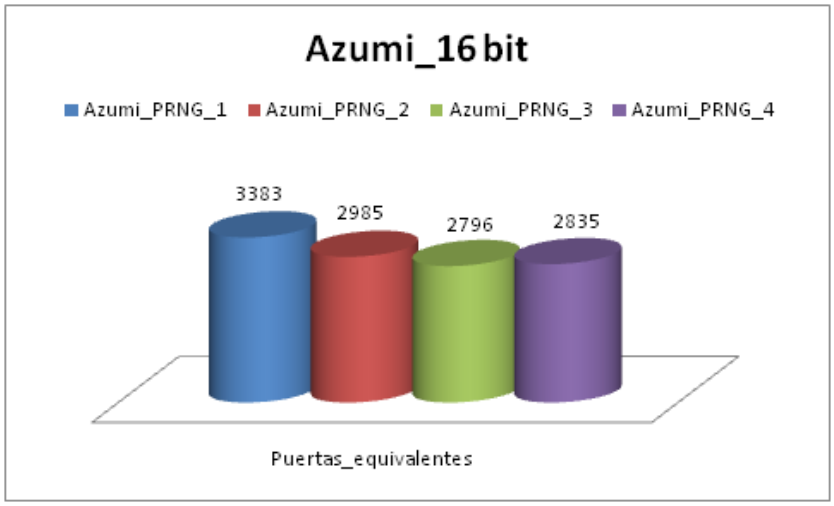
Se recuerda que el número total de puertas equivalentes para las tag de bajo costes es de 3000-4000 puertas para la parte de seguridad.

En el caso de las bibliotecas usadas en el programa Synopsys la puerta NAND ocupa 3,136 μm^2. Por tanto una vez se tenga el área total en μm^2 se divide en el valor de la puerta NAND para obtener el valor de puertas equivalente.

Resultados de síntesis del protocolo Azumi_16 bits para distintos PRNG				
	Azumi_PRNG_1	**Azumi_PRNG_2**	**Azumi_PRNG_3**	**Azumi_PRNG_4**
Número de puertos	102	102	102	102
Número de nodos	185	281	185	185
Número de celdas	4	5	4	4
Área Combinacional (μm^2)	6813,744	5056,015	4226,543	4330,851
Área no combinacional (μm^2)	3795,344	4304,944	4541,712	4551,904
Área total (μm^2)	10609,088	9360,96	8768,256	8892,72
Potencia Interna (nW)	187,794	193,9887	186,001	184,874
Potencia conmutación (nW)	34,169	32,647	21,811	17,555
Potencia dinámica total (nW)	221,917	226,636	207,81	202,42
Pérdidas de potencia (nW)	12,311	14,854	13,8636	14,493

Tabla 13. Resultado Síntesis Azumi 16 Bits

En cuanto al área total se observa como el PRNG_3 es el que menos ocupa como ya se demostró anteriormente. En cuanto a la potencia dinámica total el PRNG_1 y el PRNG_2 son los que más potencia consumen. El área total hay que convertirlo en número de puertas equivalentes para comprobar si cumple la restricción de 4000 puertas como máximo.



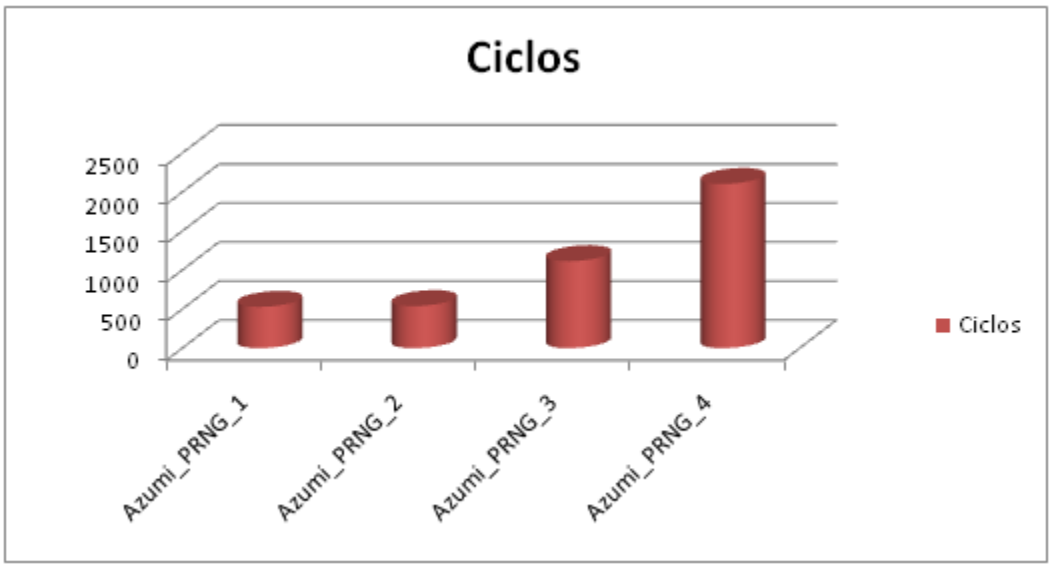
Gráfica 7. Puertas Equivalentes PRNG's

Se comprueba que los 4 diseños no superan las 4000 puertas equivalentes por tanto los 4 diseños podrían ser válidos.

En el anterior análisis sobre el número de ciclos se tomó en cuenta el número de ciclos que tarda cada generador en realizar el cálculo 1 vez, en este caso integrado dentro del protocolo habrá que tomar en cuenta que el protocolo necesita realizar el cálculo de número pseudo-aleatorio un número mayor de veces.

Número de ciclos para cada PRNG				
Columna1	Azumi_PRNG_1	Azumi_PRNG_2	Azumi_PRNG_3	Azumi_PRNG_4
Ciclos	530	540	1115	2105
Tiempo a 100 Khz (µs)	5300	5400	1150	21050

Tabla 14. Número de ciclos PRNG's



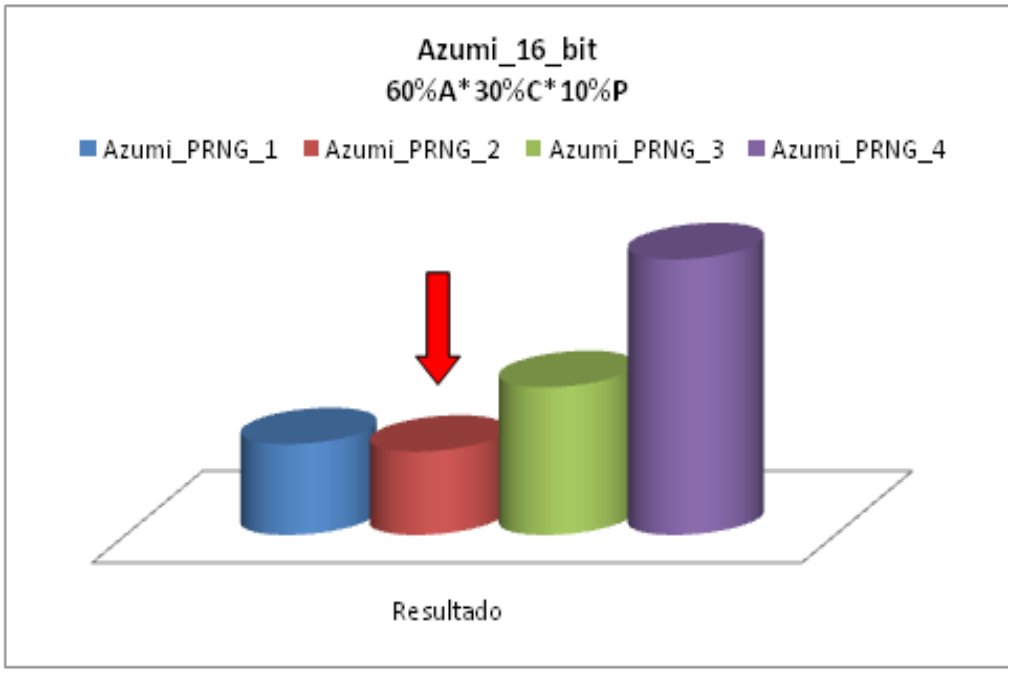
Gráfica 8. Número de ciclos PRNG's

En este caso al utilizar más veces la función pseudo-aleatoria se observa como la distancia entre el 3 y 4 comparada con el 1 y 2 es favorable para estos últimos.

Al igual que antes se hace un análisis tomando en cuenta las área, tiempo y potencia dinámica total.

Valor (60%Area * 30% Ciclos * 10% Potencia)				
	Azumi_PRNG_1	Azumi_PRNG_2	Azumi_PRNG_3	Azumi_PRNG_4
Resultado	22460374,81	20621276,76	36570174,78	68204439,45

Tabla 15. Comparación Azumi - Prng's



Gráfica 9. Comparación Azumi 16 bits

Anteriormente para el caso del estudio de los PRNG's por separado se tenía que el valor menor era para el caso del PRNG_3. En este caso este generador ya no es el más adecuado, esto es debido a que en este caso el factor de número de ciclos es peor en este generador, al realizar los generadores pseudoaleatorios varias veces este factor que hace desfavorable para el PRNG_3 aumenta, y de ahí que en este caso sea el PRNG_2 el más adecuado.

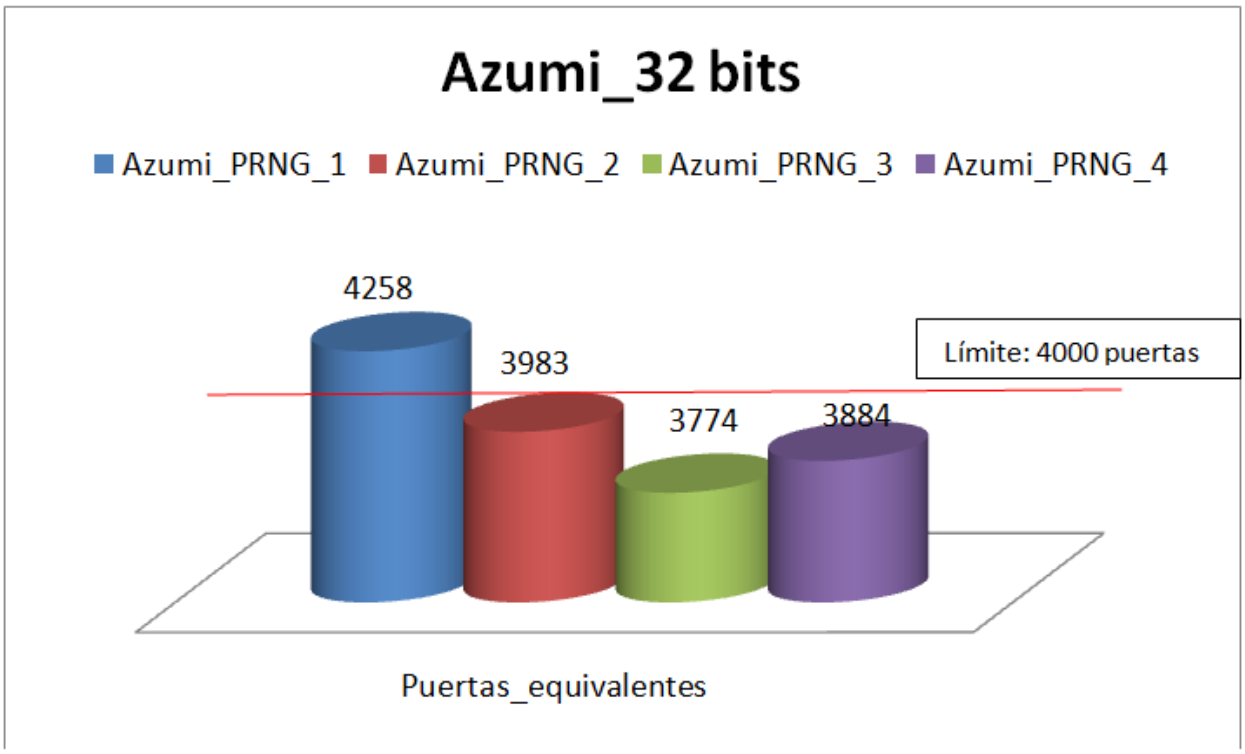
5.8.3 RESULTADO SÍNTESIS AZUMI_32 BIT

Se ha realizado la síntesis del protocolo con 32 bit incrementando el número de bits de los generadores pseudo-aleatorios a 64 bits.

Resultados de síntesis del protocolo Azumi_32 bits para distintos PRNG				
	Azumi_PRNG_1	Azumi_PRNG_2	Azumi_PRNG_3	Azumi_PRNG_4
Número de puertos	198	198	198	198
Número de nodos	356	474	356	356
Número de celdas	4	5	4	4
Área Combinacional (μm^2)	7628,422	5674,592	4714,788	4872,45
Área no combinacional (μm^2)	5727,754	6817,664	7123,24	7308
Área total (μm^2)	13356,176	12492,256	11838,028	12180,45
Potencia Interna (nW)	253,4	257,112	247,52	245,85
Potencia conmutación (nW)	28,78	25,813	17,46	13,93
Potencia dinámica total (nW)	282,18	282,924	264,98	259,78
Pérdidas de potencia (nW)	17,45	20,625	18,84	19,95

Tabla 16. Resultado Síntesis Azumi 32 Bits

En este caso sigue siendo de menor área el generador 3. Habrá que comprobar el número de puertas equivalentes que ocupa cada diseño.



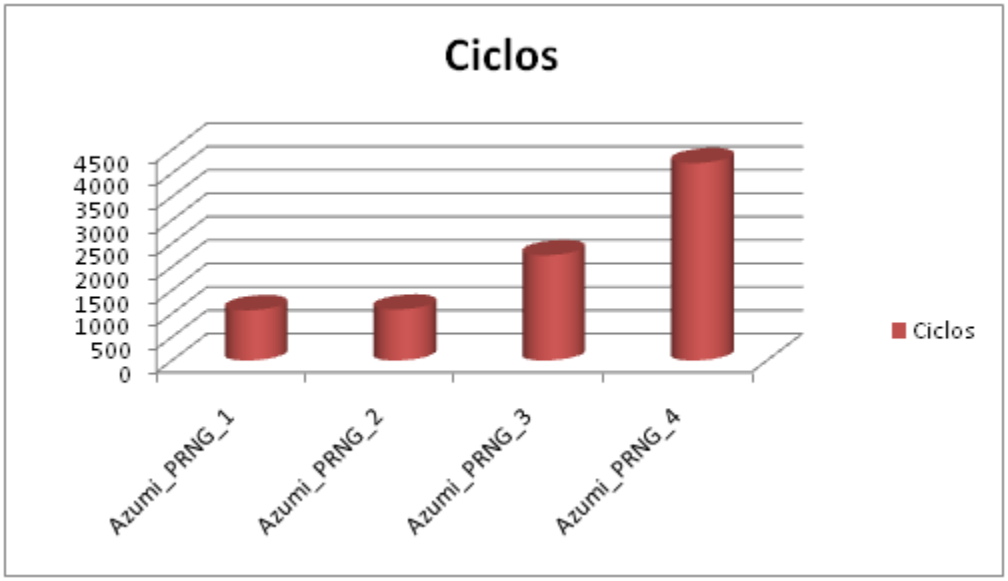
Gráfica 10. Puertas Equivalentes 32 bits

En este caso el generador 1 ya no podría implementarse en la tag ya que supera las 4000 puertas. El generador 3 sería el que menos ocupa siendo el 2 el cual está en el límite con 3983 puertas.

Al comparar el número de ciclos se observa como de nuevo al dividir el sumador en N/2 bits se obtiene mayor número de ciclos siendo el generador 4 el que más tarda en realizar el cálculo.

Número de ciclos para cada PRNG				
	Azumi_PRNG_1	Azumi_PRNG_2	Azumi_PRNG_3	Azumi_PRNG_4
Ciclos	1075	1090	2250	4220
Tiempo a 100 Khz (µs)	5300	5400	1150	21050

Tabla 17. Número de ciclos PRNG's (64 bis)

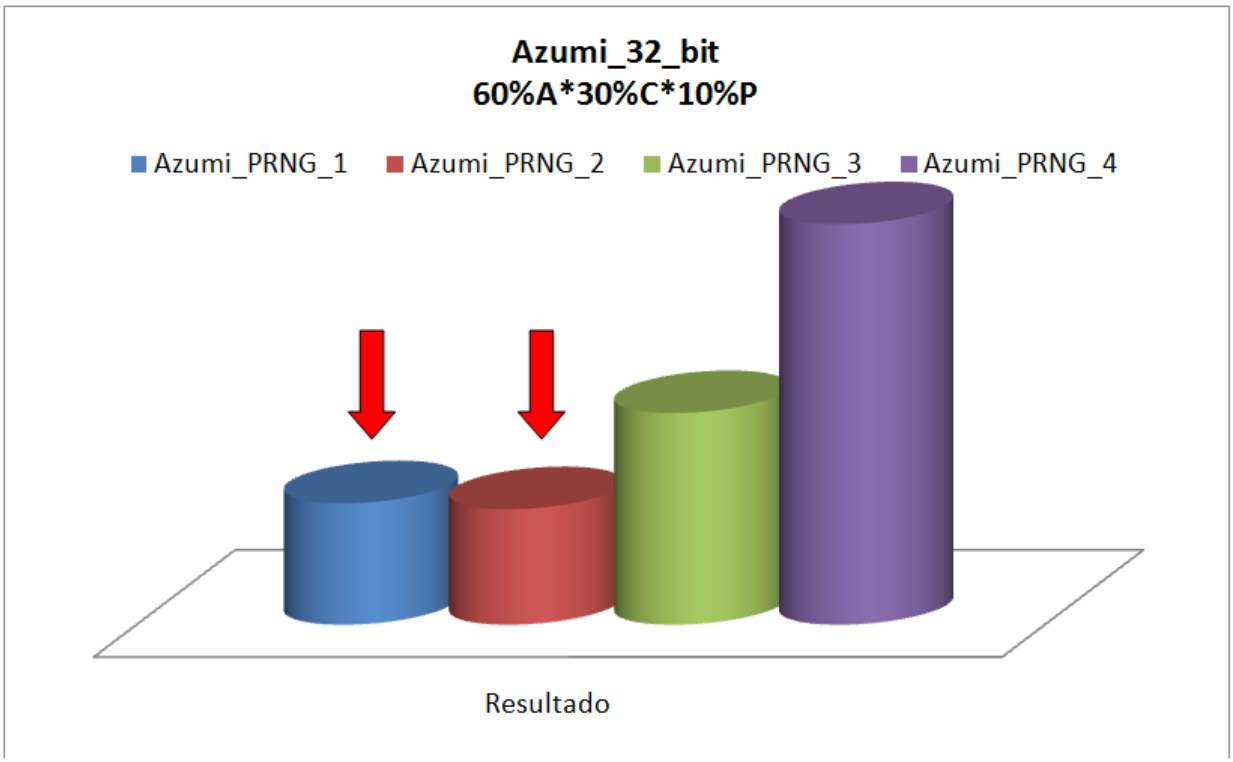


Gráfica 11. Número de ciclos PRNG's 32 bits

Al igual que en los anteriores casos se realiza una comparación tomando en cuenta los valores de área, ciclos y potencia dinámica total.

Valor (60%Area * 30% Ciclos * 10% Potencia)				
	Azumi_PRNG_1	Azumi_PRNG_2	Azumi_PRNG_3	Azumi_PRNG_4
Resultado	72927165,14	69344124,3	127042046,7	240355465,4

Tabla 18. Comparación Azumi (32 bits) - Prng's



Gráfica 12. Comparación Azumi 32 bits

Realizando un análisis de estos datos tenemos varias conclusiones:

a) Aunque el generador 1 no es un valor muy alto en la media de las 3 variables, las puertas equivalentes está por encima de 4000, por tanto es un generador que descartamos.
b) El generador 4 podría ser implementado pero debido a los ciclos que se necesitan para el cálculo no sería un diseño adecuado si lo comparamos con el 2 y 3.
c) Entre el 2 y 3 los dos diseños están por debajo de las 4000 puertas se establece como el diseño del generador 2 como mejor que el 3 debido a un menor tiempo de cálculo.

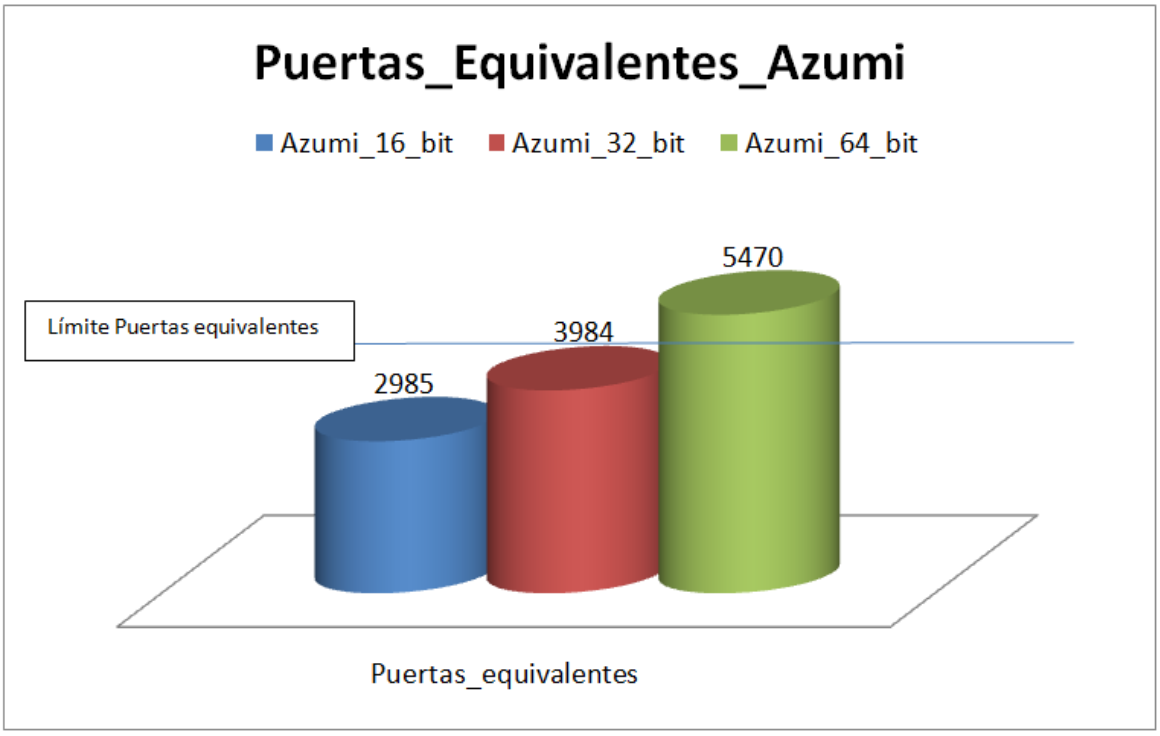
Por tanto al igual que en el protocolo con 16 bits se establece como el PRNG_2 la mejor opción para el protocolo azumi con 32 bits.

5.8.4 COMPARACIÓN AZUMI 16 – 32- 64 BITS

En este apartado se muestra una comparación del protocolo Azumi usando el PRNG_2 para distintos números de bits del protocolo. A mayor número de bits del protocolo mayor será la seguridad por tanto se puede establecer un número de bits hasta que no se pueda cumplir con la restricción de área, ya que es este valor el que limita la fabricación del circuito.

Número de puertas equivalentes			
	Azumi_16_bit	Azumi_32_bit	Azumi_64_bit
Puertas_equivalentes	2985	3984	5470

Tabla 19. Puertas Equivalentes Protocolo Azumi



Gráfica 13. Puertas Equivalentes Azumi según Nº Bits

El protocolo con 64 bits ocupa 5470 puertas por tanto es un diseño que no podría ser utilizado para nuestro caso.

Debido a que con 32 bits está por debajo de las 4000 puertas se considera este protocolo más seguro que con 16 y por ello se establece como el más idóneo dentro del estudio del protocolo Azumi.

5.9 CONCLUSIÓN PROTOCOLO AZUMI

Se ha realizado un estudio de los distintos generadores pseudo-aleatorios que se podrían usar dentro del protocolo Azumi. Posteriormente se ha realizado un análisis del protocolo Azumi con distinto números de bits para ver el comportamiento de los generadores pseudo-aleatorios dentro del protocolo.

Finalmente se ha realizado una comparación entre el protocolo Azumi con16,32 y 64 bits usando el generador pseudo-aleatorio 2. Se ha descartado con 64 bits debido a que supera las 4000 puertas establecidas como límite y se considera que el protocolo Azumi con 32 bits y usando el generador pseudo-aleatorio 2 es el más adecuado atendiendo a los valores de área, tiempo y potencia dinámica total.

CAPÍTULO 6

COMPARACIÓN PROTOCOLOS HMAC – AZUMI

6.1 COMPARACIÓN RESULTADOS HMAC-AZUMI

Una vez que se han seleccionado los mejores diseños de cada protocolo se va a realizar una comparación de estos dos protocolos para comparar las ventajas de un protocolo frente al otro.

En la siguiente tabla se muestran las características de cada protocolo.

Características Protocolos		
Columna1	**HMAC**	**Azumi**
Número bits	32	32
Generar número aleatorios	hash	PRNG
Número de bits=> Generador_aleatorio	32	32
Operaciones	Xor + Suma	Xor + Suma
Número bits Alu	32	32

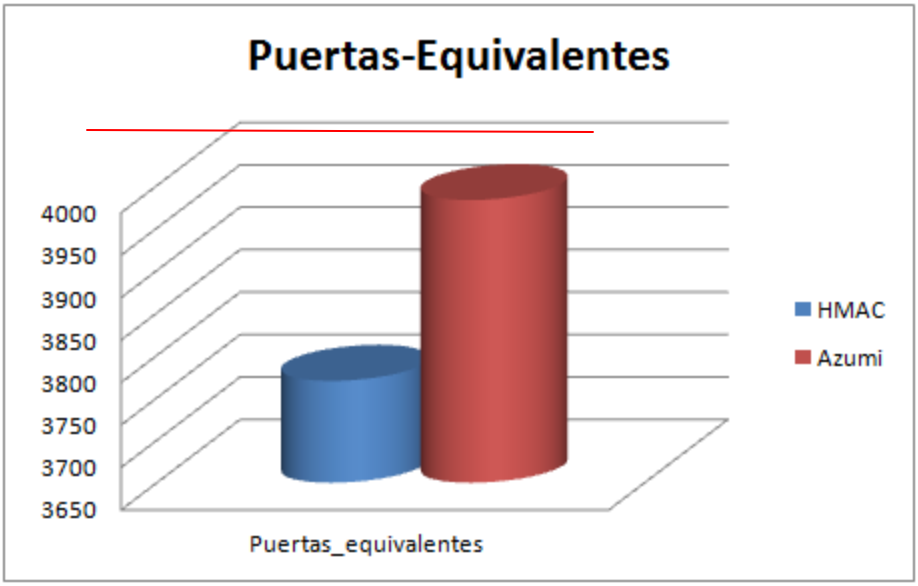
Tabla 20. Comparación Protocolos Azumi-HMAC

Comparando los dos protocolos se observa que no existen grandes diferencias en el número de bits utilizados. Las diferencias están en la forma de obtener los números aleatorios y también hay que destacar que en el protocolo Hmac es capaz de realizar operaciones de Xor y suma al igual que Azumi.

A la hora de realiza la comparación de los dos protocolos se tomará en cuenta las variables de puertas equivalentes, ciclos y potencia dinámica total.

Número de puertas equivalentes (HMAC-Azumi)		
Columna1	**HMAC**	**Azumi**
Puertas_equivalentes	3770	3983

Tabla 21. Puertas Equivalentes Azumi-HMAC



Gráfica 14. Puertas Equivalentes HMAC-Azumi

Se comprueba que los dos protocolos cumplen con la restricción de 3000-4000 puertas equivalentes, aunque el Azumi ocupe un poco más que el Hmac se puede considerar que en este caso no sería una gran desventaja para este protocolo, ya que estaría por debajo de las 4000 puertas.

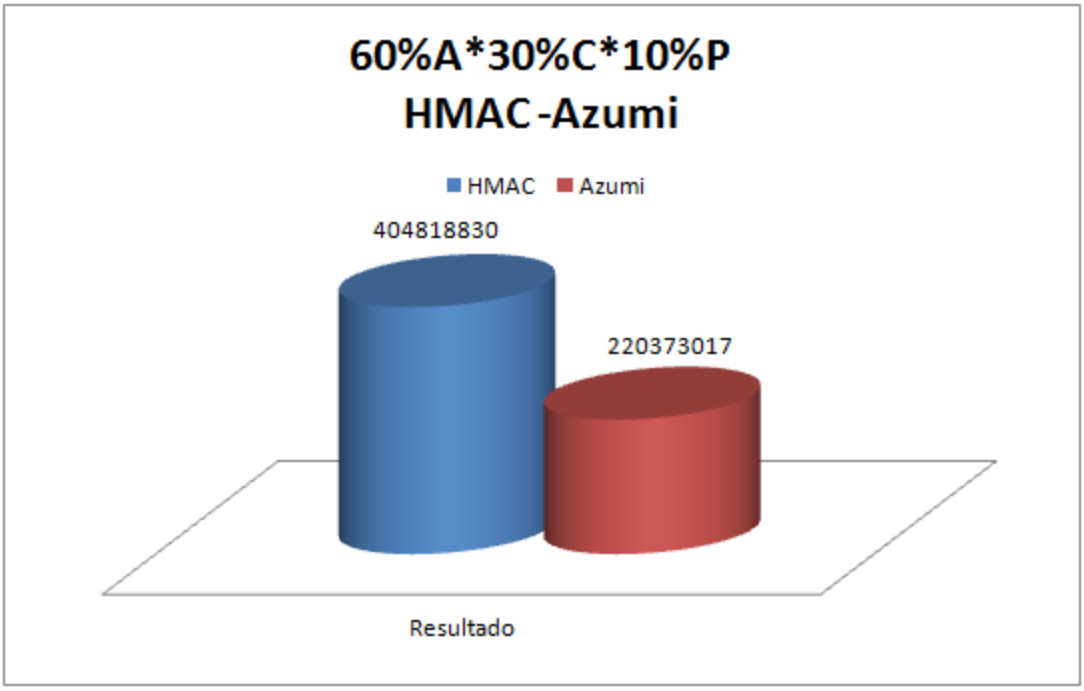
Es importante también saber el número de ciclos que tarda cada protocolo ya que nos dará el tiempo de cálculo de cada uno.

Número de ciclos para (HMAC-Azumi)		
Columna1	**HMAC**	**Azumi**
Ciclos	2050	1090
Tiempo a 100 Khz (ms)	21	11

Tabla 22. Número de ciclos HMAC-Azumi

Para el caso del protocolo Hmac tiene casi el doble de ciclos, por tanto habrá que tener en cuenta que al utilizar este protocolo tardará el doble de veces que el Azumi.

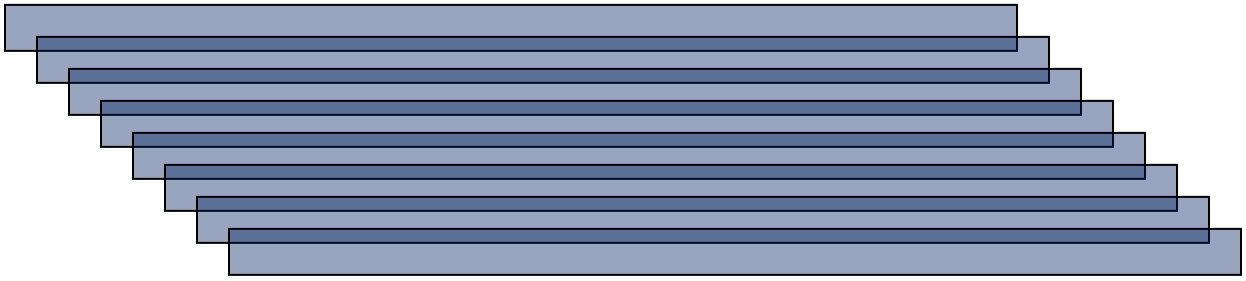
Al tener en cuenta también la potencia consumida, se hará una media entre las tres variables utilizando un 60% para el área utilizada, un 30% para el número de ciclos y un 10% para la potencia dinámica total.



Gráfica 15. Comparación Protocolo HMAC-Azumi

Se observa que en la utilización de estos recursos el Azumi obteniendo un valor medio gastaría menos que el protocolo Hmac. Esto es así ya que las variables de puertas equivalentes y potencia son muy parecidas para los protocolos y es el número de ciclos el que hace que este valor sea menos para el protocolo Azumi que para el protocolo Hmac.

CAPÍTULO 7



CONCLUSIONES Y LÍNEAS FUTURAS

7.1 CONCLUSIÓN

El objetivo del proyecto era el de buscar diferentes protocolos de encriptación los cuales fueran válidos para las tarjetas de bajo coste de RFID. Se puede decir que este objetivo se ha cumplido con éxito ya que se han encontrado algunos diseños los cuales podrían ser implementados en este tipo de aplicaciones.

Para la comparación de los distintos diseños, se ha valorado como prioridad el número de puertas equivalentes. Debido a que existe una restricción de 3000-4000 puertas equivalentes este será la restricción en la cual nos hemos fijado para descartar los diseños que no sean válidos. También se ha tomado otras dos variables para poder comparar mejor los distintos diseños. Por un lado se ha tomado la variable de ciclos, la cual nos dará una aproximación del tiempo necesario para calcular los datos, se sabe que estas tag operan en torno a 100kHz. La otra variable que se ha tomado a la hora de comparar es la potencia dinámica total. A través de la media de estas tres variables dando una mayor importancia a las puertas equivalentes frente a las otras dos, se ha podido seleccionar el mejor diseño de los implementados.

Se han buscado dos protocolos de los cuales una vez estudiados cada uno por separado se ha seleccionado cual era la mejor opción.

Antes de comprobar si los objetivos del proyecto se han cumplido, es necesario destacar los siguientes puntos.

a) Los dos protocolos podrían utilizar un nivel de seguridad de 32 bits. Mientras mayor sea el número de bits más seguro se considera el protocolo. Al ser este valor igual para los dos este no será un valor el cual nos haga decantar un mejor protocolo frente al otro.
b) El protocolo Hmac utilizará una función hash.
c) El protocolo Azumi utilizará un generador PRNG.
d) Los dos protocolos cumplen con la restricción de 4000 puertas equivalentes como máximo. En este aspecto se podría decir que el protocolo Hmac es un poco mejor que el Azumi ya que ocupa un número de puertas menos que el Azumi. Aunque como se tiene la seguridad de que el Azumi no supera las 4000 puertas se podría considerar igual de válido en este aspecto frente al Hmac, ya que se dispondría de hasta 4000 puertas para ocupar ese espacio.

A la hora de seleccionar un diseño u otro se elegirían estas dos alternativas.

a) En el caso de necesitar un protocolo el cual su característica principal sea la rapidez en la realización de los cálculos, en este caso el mejor protocolo a implementar sería el Protocolo Azumi.

b) En el caso de necesitar un protocolo en el cual la característica principal sea la seguridad, en este caso la mejor opción sería la implementación del Protocolo Hmac. Esto es debido a que la función hash es de 32 bits y los PRNG's son de 16 bits, por tanto se puede considerar que para ataques de fuerza bruta sería más adecuado el protocolo HMAC que es el que incluye la función hash.

Selección_Protocolo		
Características	**HMAC**	**Azumi**
Rapidez	✗	★
Seguridad	★	✗

Tabla 23. Selección Protocolo

A continuación se da una respuesta a cada objetivo del proyecto.

a) *Objetivo: Búsqueda de protocolos de autentificación que sean válidos para la tecnología RFID de bajo coste.*

Los dos protocolos HMAC y Azumi se consideran válidos para la tecnología RFID de bajo coste debido a que cumplen las restricciones de estándar.

b) *Objetivo: Implementación de los protocolos seleccionados.*

Se han implementado ambos protocolos en VHDL.

c) *Objetivo: Buscar diferentes arquitecturas en los protocolos que se implementarán.*

En el protocolo HMAC se ha realizado diferentes arquitecturas tanto en la función hash como en el protocolo completo.

De la misma manera, se ha realizado diferentes arquitecturas en el protocolo Azumi, tanto en los generadores PRNG como en el protocolo completo.

d) *Objetivo: Síntesis de los protocolos implementados.*

Se ha podido realizar este punto gracias a una bibliteca de fabricantes el cual nos ha dado un valor muy aproximado de área, tiempo y consumo.

e) Objetivo: Comparación de los protocolos y obtener el que mejor se adapte a este tipo de tecnología.

Para una aplicación que se considere la seguridad como factor prioritario sería más valido el protocolo HMAC, mientras que para una aplicación donde el tiempo de cálculo sea prioritario se elegiría el Azumi.

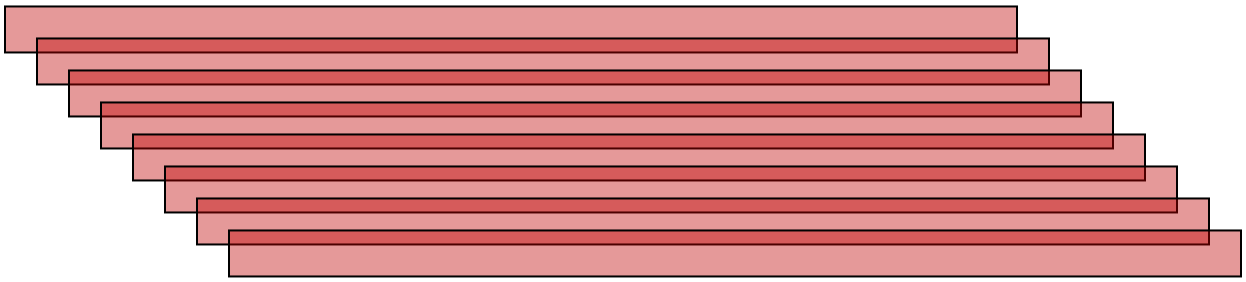
Por tanto se puede considerar que se han cumplido satisfactoriamente los objetivos del proyecto.

7.2 LÍNEAS FUTURAS

Se proponen como trabajos futuros los siguientes puntos:

a) Búsqueda e implementación de nuevos protocolos ligeros que puedan ser aplicados a las tarjetas RFID de bajo coste.
b) Realización de un estudio de seguridad sobre los protocolos implementados así como las modificaciones que se han realizado.
c) Búsqueda e implementación de un generador de números aleatorios de reducido tamaño, de modo que pueda ser integrado en los protocolos realizados en este proyecto o en otros protocolos que cumplan los requisitos de la tecnología RFID.
d) Realizar la síntesis de los protocolos implementados con otro tipo de tecnología a diferencia de los 90nm usados en este proyecto.
e) Búsqueda e implementación de otro tipo de generadores pseudo-aleatorios, con el fin de poder ser integrado en el protocolo Azumi o en otro tipo de protocolo que sea investigado.

CAPÍTULO 8



PRESUPUESTO DEL PROYECTO

8.1 PRESUPUESTO DEL PROYECTO

En este capítulo se presenta los costes que se han generado para este proyecto. En él se incluye tanto los costes materiales como los personales [15] [16].

Para los costes personales se toma el coste de ingeniero mes en 1700 euros, como la dedicación no ha sido de 8 horas por día se considera que el coste por mes trabajado será un 30% aproximadamente.

En cuanto a los costes materiales se tomarán un ordenador y los programas usados. Para obtener los costes imputables al proyecto se aplicará la siguiente fórmula de amortización de los mismos.

$$\frac{A \times C \times D}{B}$$

A = número de meses desde la fecha de facturación desde que el equipo es usado.

B = periodo de depreciación (meses).

C = coste del equipo (sin IVA).

D = % del uso que se dedica al proyecto.

Se ha tomado como el período de depreciación de los equipos informáticos 60 meses, en cuanto a las licencias de los programas se ha tomado 12 meses.

Los costes indirectos se consideran un 20% de los costes totales del proyecto.

PRESUPUESTO DEL PROYECTO HUGO IZQUIERDO DONOSO				
GASTOS PERSONALES				
Personal	DEDICACIÓN (hombre mes)	COSTES/MES	Porcentaje Dedicación	TOTAL (euros)
Ingeniero	6	1700	40%	4080

GASTOS MATERIALES					
Equipo	Coste (Euros)	% Uso dedicado al proyecto	Dedicación (meses)	Período de depreciación	Coste imputable
PC	500	30	6	60	15
Licencia ModelSim	1000	100	6	12	500
Licencia Synopsys	1800	100	6	12	900
Total costes Materiales (Euros)	1415				

Resumen de Costes	(Euros)
Personal	4080
Amortización	1415
Costes indirectos (20%)	1099
Total	**6594**

Por tanto el presupuesto total del proyecto asciende a la cantidad de *seis mil quinientos noventa y cuatro* euros.

CAPÍTULO 9

BIBLIOGRAFÍA DEL PROYECTO

9.1 BIBLIOGRAFÍA

[1] RFID : applications, security, and privacy Garfinkel, Simson, Adison-Wesley [2006].

[2] RFID : la tecnología de identificación por radiofrecuencia Muñoz Frías, José Daniel.

[3] RFID security , Thornton, Frank. Syngress [2006].

[4] M5_DS_IPJ_MONZA5_agChip_Datasheet.

[5] Xingxin Gao; Zhe Xiang; Hao Wang; Jun Shen; Jian Huang; Song Song;An approach to security and privacy of RFID system for supply chain; IEEE International Conference on 13-15Sept.2004 Page(s):164-168.

[6] An Authentication Protocol for RFID Tag and Its Simulation; Wang Shang-ping; Ma Qiao-mei, Zhang Ya-ling and Li You-sheng.

[7] Ju Weicheng and Yu Chengfang, "An Anti-Collision RFID Algorithm Based on the Dynamic Binary," Journal OF FuDan University, Shanghai, vol.44, pp.46–50, 2005.

[8] Guo Aixia, Li Feng, Xiong Juntao, "Research and Simulation on RFID Authentication Protocol," Research and Exploration in Laboratory, vol.29, no.3, pp.61–65, 2010.

[9] An Efficient Authentication Protocol for RFID Systems Resistant to Active Attacks; Pedro Peris-Lopez, Julio Cesar Hernandez-Castro, Juan M. Estevez-Tapiador, and Arturo Ribagorda.

[10] Ranasinghe, D., Engels, D., Cole, P.: Low-cost RFID systems: Confronting security and privacy. In: Auto-ID Labs Research Workshop (2004)).

[11] EPCGlobal: Class-1 Generation-2 UHF Air Interface Protocol Standard version 1.0.9: "Gen 2".

[12] EPCGlobal: EPC Generation-1 Tag Data Standards version 1.1.

[13] CryptanalysisofanEPCClass-1Generation-2standardcompliant authenticationprotocol Pedro Peris-Lopez, JulioC.Hernandez-Castro, JuanM.E.Tapiado , JanC.A.vanderLubbe.

[14] Chien, H.Y., Chen, C.H., 2007. Mutual authentication protocol for RFID conforming to EPC class-1 generation-2 standards. Computer Standards and Interfaces, Elsevier Science Publishers 29 (2), 254–259.

[15] De la Iglesia Toribios, E.: "*Implementación y análisis de dos protocolos de audentificación seguros y ligeros para RFID*", Proyecto Fin de Carrera, Universidad Carlos III de Madrid, Julio 2010.

[16] Plantilla presupuesto de Memoria de la Universidad Carlos III de Madrid.

[17] Design Compiler® Tutorial Using Design Vision™.

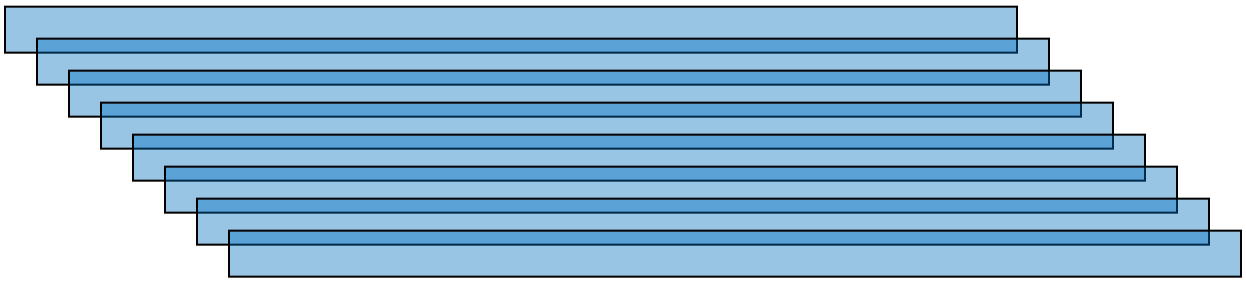
[18] "Design Compiler. Reference Manual: Optimization and timing analysis", Synopsys Inc., octubre 1999.

[19] "Expanding the Synopsys prime time. Solution with power analysis", Synopsys Inc., Junio 2006.

[20] "Lightweight Cryptography in Radio Frequency Identification (RFID) Systems Author: Pedro Peris López"

[21] Class-1 Generation-2 UHF air interface protocol standard version 1.0.9: "Gen2". http://www.epcglobalinc.org/standards/, 2005.

ANEXOS



Tutorial ModelSim-Synopsys

Códigos

Tutorial ModelSim-Synopsys

A continuación se muestra un tutorial sobre los pasos a realizar a la hora de trabajar con ModelSim y Synopsys. Para ello se mostrará los diferentes pasos a través de un ejemplo el cual será un contador de 4 bits.

Al abrir el programa ModelSim, lo primero que habrá que realizar será hacer un nuevo proyecto.

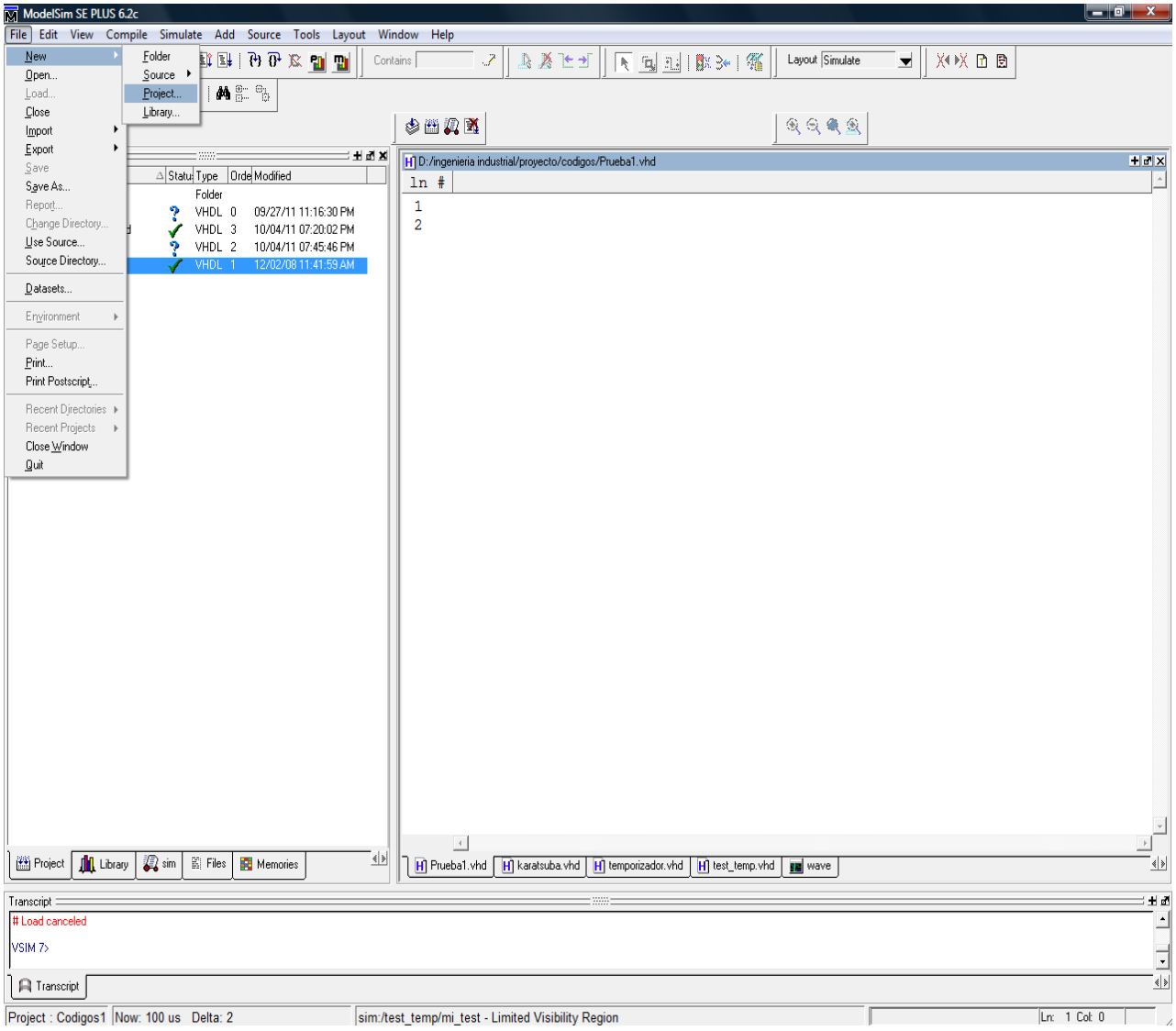


Figura 37. Inicio ModelSIm

Una vez creado el proyecto, se añaden los archivos VHDL los cuales se quieren añadir o crear un nuevo archivo VHDL en el proyecto.

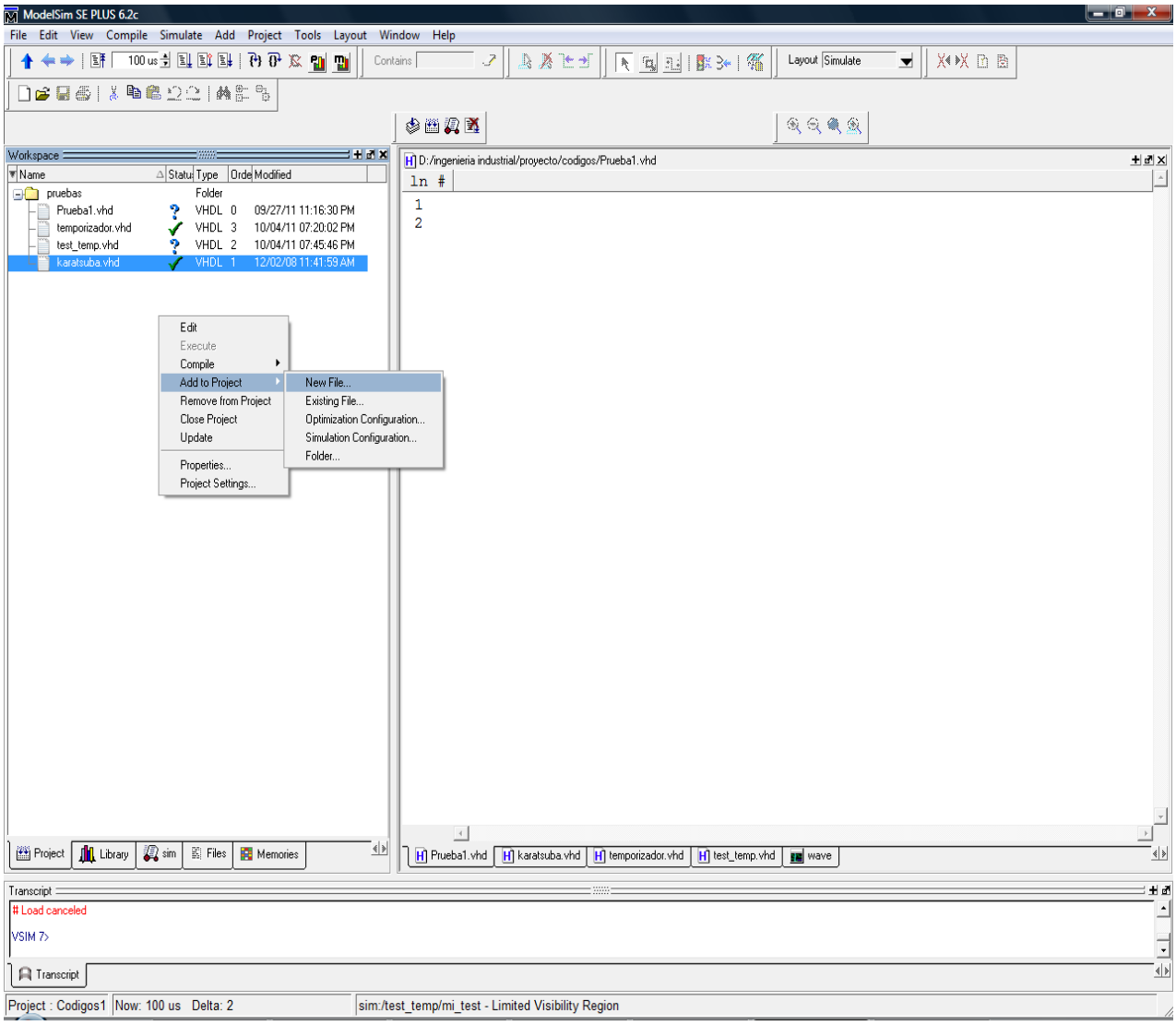


Figura 38. Insertar archivos ModelSim

En primer lugar utilizamos la herramienta ModelSim para crear el diseño y poder simular. Ejemplo temporizador:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
```

```
entity Temporizador is
   Port ( clk : in STD_LOGIC;
   reset : in STD_LOGIC;
   activacion : in STD_LOGIC;
   tcumplido : out std_logic);
end Temporizador;

architecture Temporizador_a of Temporizador is
signal contador: std_logic_vector (4 downto 0);
begin
process(clk, reset)

begin
   if reset = '1' then
      contador <= "00000";
   elsif clk'event and clk='1' then
      if activacion = '1' then
         contador <= contador + '1';
            if contador="01010" then
            contador<="00000";
            else
            end if;
      else
      contador <= "00000";
      end if;
end if;

end process;
tcumplido <= '1' when contador="00101" else '0';
end Temporizador_a;
```

Una vez que se tiene el diseño del circuito se crea el test-bench el cual nos servirá para hacer la simulación de las señales.

```
library ieee;
    use ieee.std_logic_1164.all;
    use ieee.std_logic_unsigned.all;
    use ieee.std_logic_arith.all;

 entity test_temp IS
  end test_temp;

 architecture test_tempa of test_temp IS

     COMPONENT temporizador4
        Port ( clk : in STD_LOGIC;
          reset : in STD_LOGIC;
        activacion : in STD_LOGIC;
        tcumplido : out std_logic);

       END COMPONENT;

       SIGNAL activacion,tcumplido: STD_LOGIC;
       SIGNAL CLK,RESET: STD_LOGIC;
```

```
  begin
  mi_test: temporizador4 PORT MAP (clk,reset,activacion,tcumplido);

      --SIMULACION DEL RESET
PROCESS
  BEGIN
    reset <= '0';
    WAIT FOR 150 ns;
    reset <= '1';
    WAIT FOR 1000 ns;
    reset <= '0';
    WAIT;
END PROCESS;

-- SIMULACION CLK
PROCESS

   BEGIN
      wait for 0.5 us;
      clk<='1';
      wait for 0.5 us;
      clk<='0';
   end process;

-- SIMULACION SEÑAL ACTIVACION
process
   begin
      ACTIVACION<='0';
      WAIT FOR 1 us;
      activacion<='1';
      WAIT FOR 50 us;
      activacion<='0';
      wait for 80 us;
      activacion<='1';
      WAIT;
   END PROCESS;


END TEST_tempa;
```

Antes de realizar la simulación es necesario compilar el circuito, tanto el circuito diseñado como el test

Se selecciona simulación y se indica el circuito test-bench que se quiere probar.

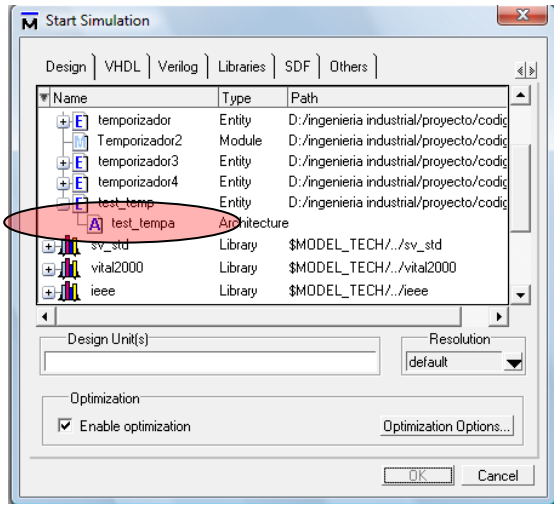


Figura 39. Inicio Simulación ModelSim

En el siguiente paso se añaden las señales que se quieren muestrear y se selecciona el tiempo de simulación. A partir de ahí se puede comprobar si el circuito diseñado es correcto.

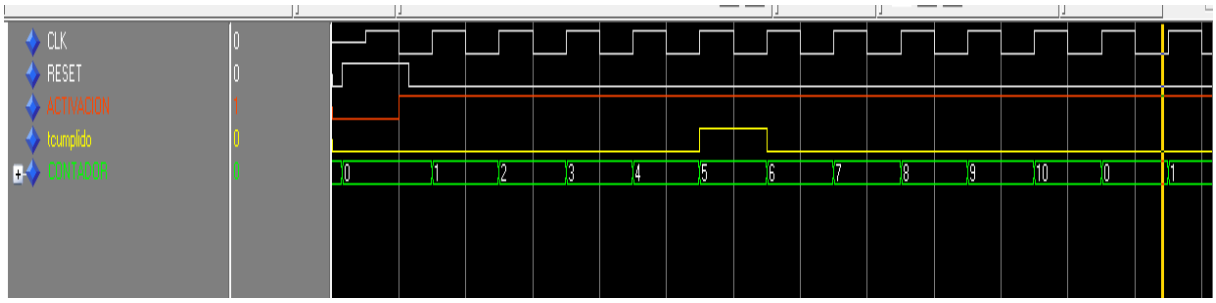


Figura 40. Simulación ModelSim

A partir de este paso para realizar la síntesis del circuito habrá que utilizar la herramienta Synopsys.

Una vez instalado la maquina virtual en el ordenador habrá que instalar el VmWare player, herramienta gratuita que se puede obtener de internet.

Al abrir el VmWare, se selecciona la maquina virtual la cual se ha instalado.
La pantalla de inicio una vez entrado en la máquina virtual es la siguiente.

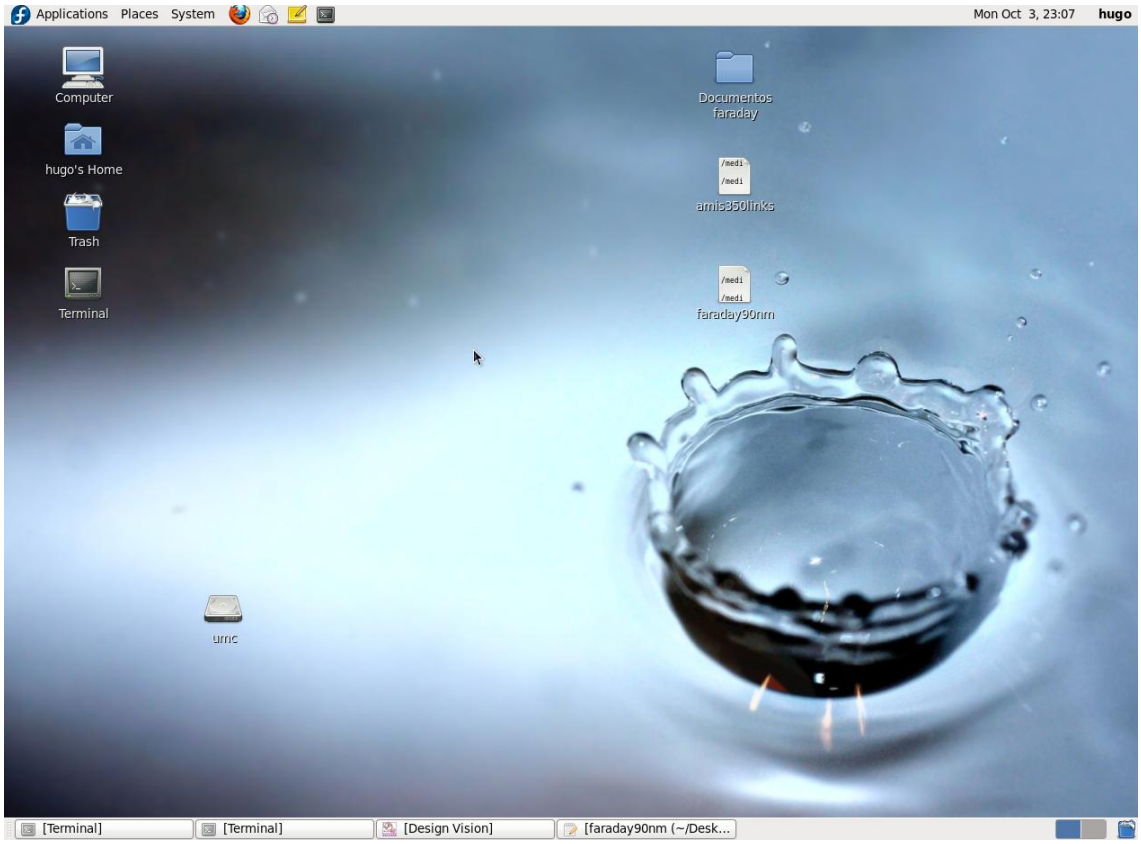


Figura 41. Inicio Synopsys

Para cargar el programa se abre el terminal y se introduce “design visión”. Una vez cargado el programa aparecerá la siguiente imagen de inicio.

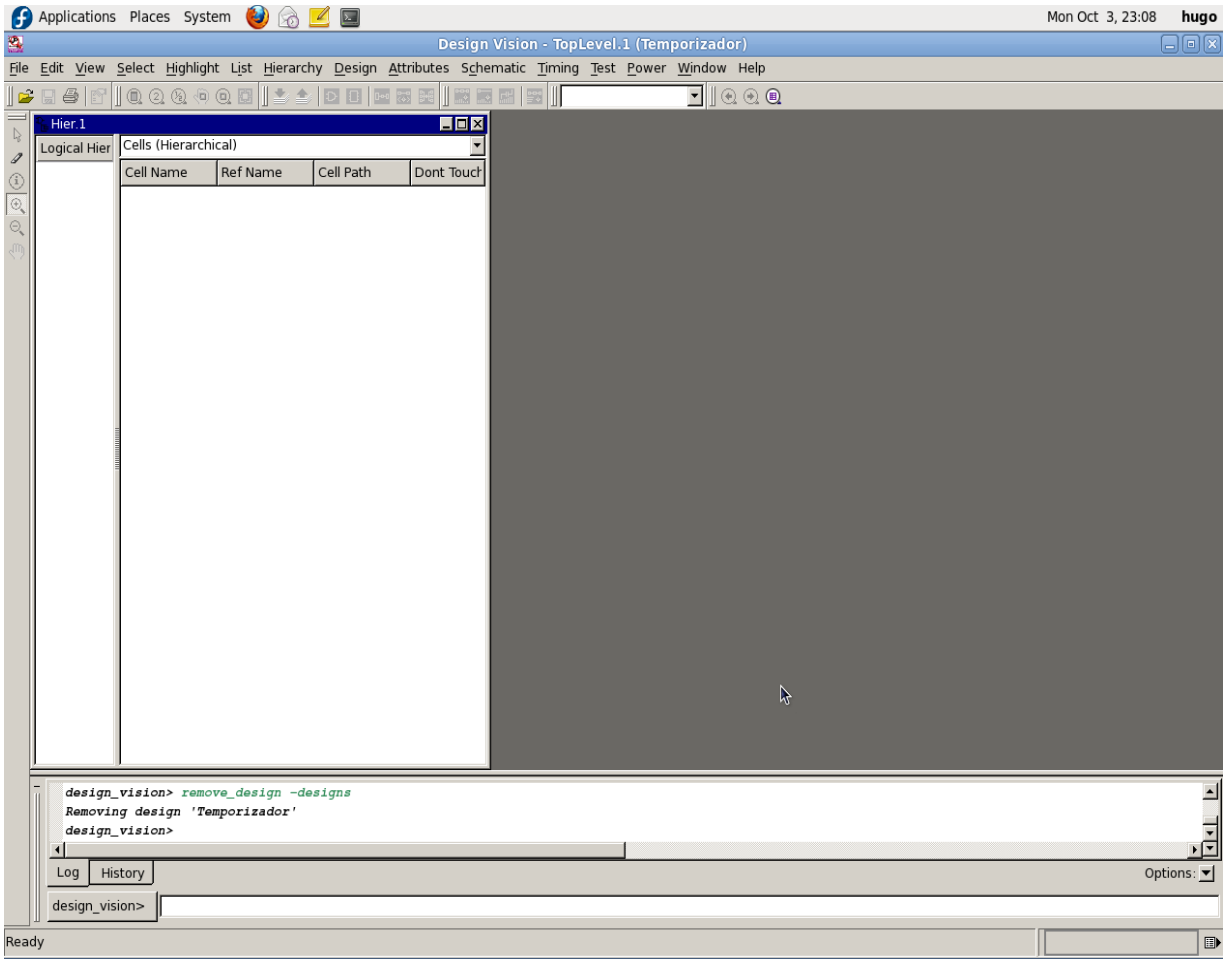


Figura 42. Cargar Programa Synopsys

Antes de cargar el circuito hay que introducir las librerías. Para ello se selección file=>Setup, posteriormente se introduce las librerías. Para este ejemplo y para el proyecto se introducen las librerías correspondientes a diseños de 90nm.

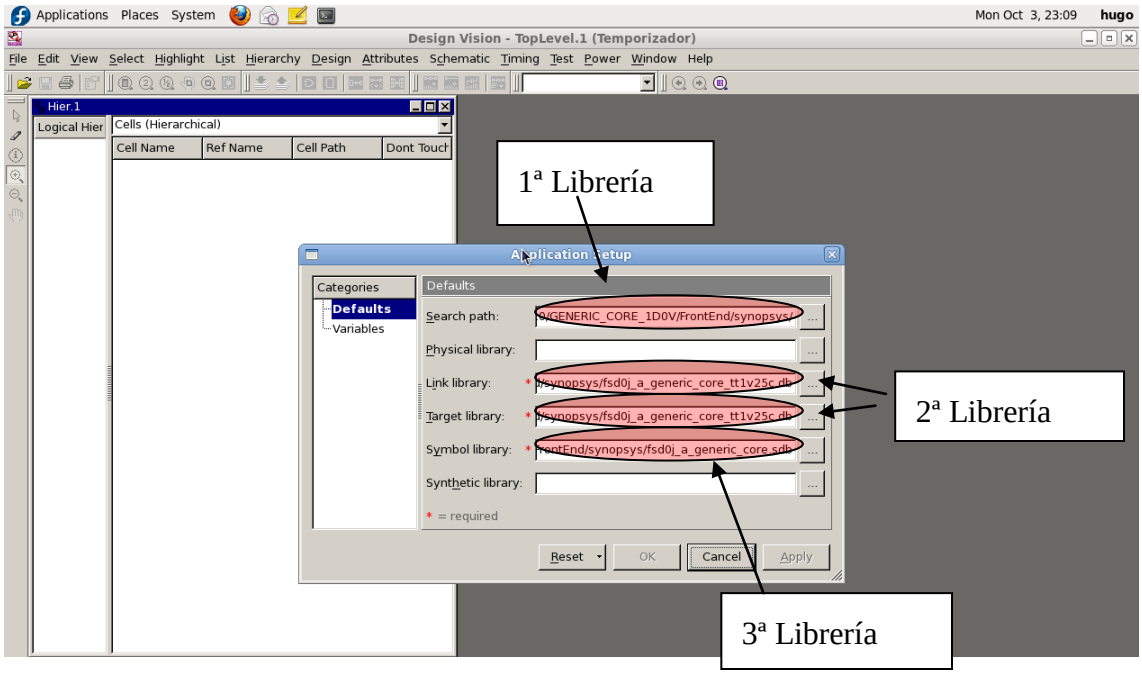


Figura 43. Librerías Synopsys

El siguiente paso es introducir el circuito el cual se quiere sintetizar.

- File=> Analyze

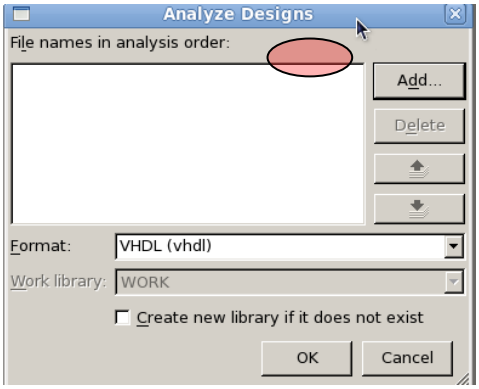


Figura 44. Analisis Synopsys

- File =>Elaborate Design

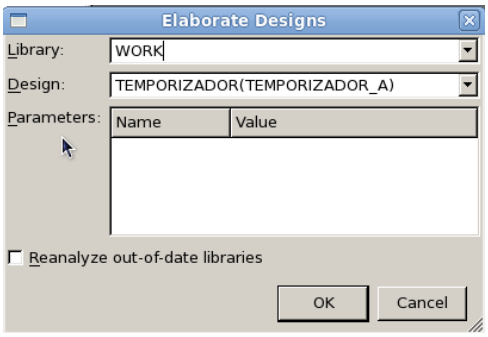


Figura 45. Diseño Synopsys

En el caso de que en el diseño se especifique un número de bits habrá que especificar el número de bits el cual se quiere realizar la síntesis.

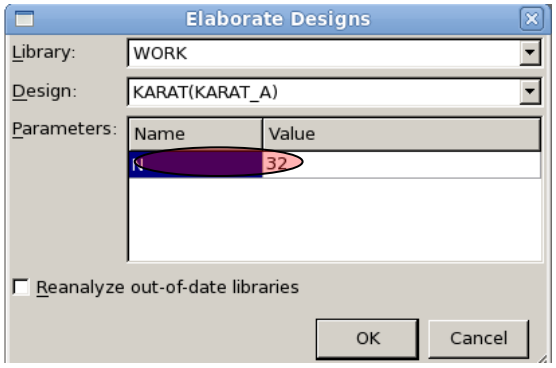


Figura 46. Selección Nº bits Synopsys

El siguiente paso es hacer la compilación del circuito.

- Design => Compila

En la compilación se puede seleccionar una síntesis del circuito medio o alto, para el proyecto se seleccionará un nivel *High*.

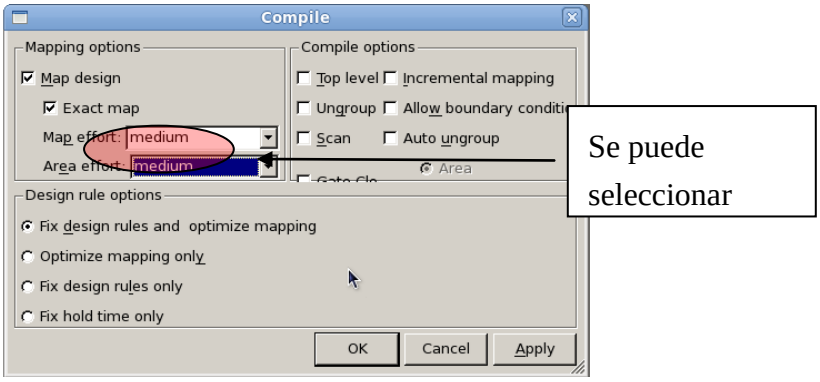


Figura 47. Compilación Synopsys

Una vez realizada la compilación se podrá observar los diferentes informes (area, potencia, etc)

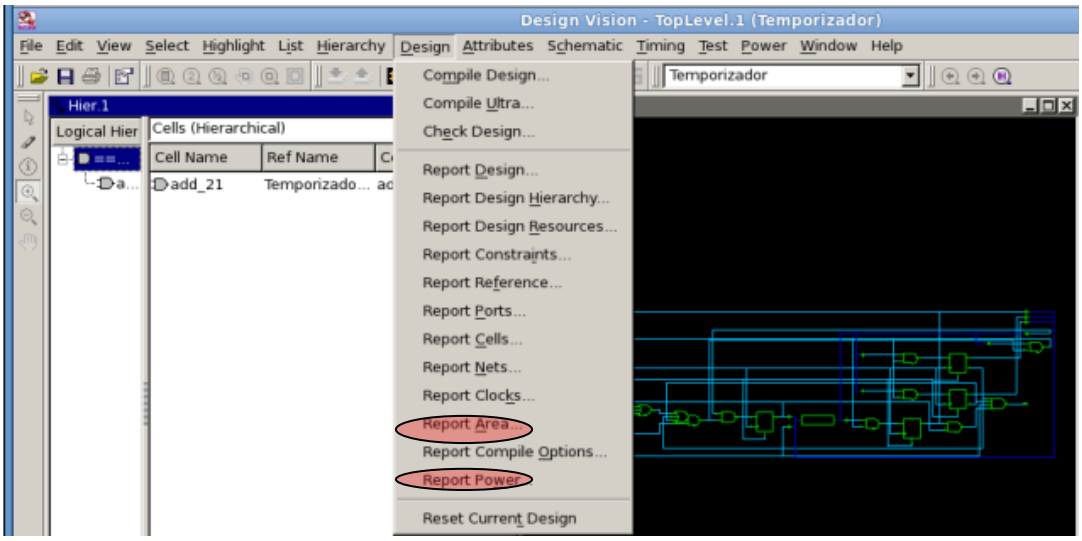


Figura 48. Informes Synopsys

En el proyecto los informes los cuales se van a obtener serán los relacionados con el área y potencia.

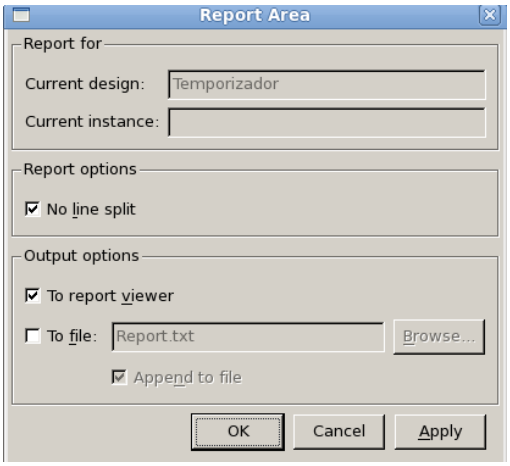


Figura 49. Informe Area Synopsys

En el informe del área se podrá obtener tanto el área combinacional como el área no combinacional, así como el área total ocupada en las cuales se mide en micrómetros.

```
****************************************
Report : area
Design : Temporizador
Version: B-2008.09
Date   : Tue Oct  4 20:42:46 2011
****************************************

Library(s) Used:

    fsd0j_a_generic_core_tt1v25c (File: /media/umc/UMC/Faraday-90nm/Des

Number of ports:                4
Number of nets:                28
Number of cells:               24
Number of references:          12

Combinational area:         83.887999
Noncombinational area:      73.696002
Net Interconnect area:      undefined  (Wire load has zero net area)

Total cell area:           157.584001
Total area:                 undefined
```

Figura 50. Resultados Área Synopsys

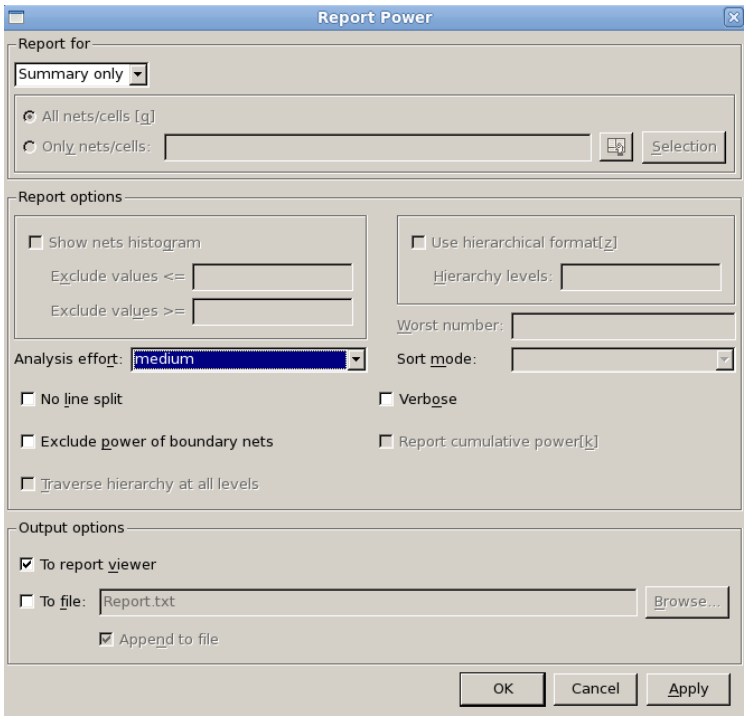


Figura 51. Informe Potencia Synopsys

```
Operating Conditions: TCCOM   Library: fsd0j_a_generic_core_tt1v25c
Wire Load Model Mode: enclosed

Design        Wire Load Model            Library
------------------------------------------------
Temporizador           enG5K             fsd0j_a_generic_core_tt1v25c


Global Operating Voltage = 1
Power-specific unit information :
    Voltage Units = 1V
    Capacitance Units = 1.000000pf
    Time Units = 1ns
    Dynamic Power Units = 1mW    (derived from V,C,T units)
    Leakage Power Units = 1pW


  Cell Internal Power  =   4.5162 nW   (81%)
  Net Switching Power  =   1.0332 nW   (19%)
                         ---------
Total Dynamic Power    =   5.5493 nW  (100%)

Cell Leakage Power     = 264.7372 pW
```

Figura 52. Resultados Potencia Synopsys

En la barra de herramientas se podrá seleccionar las diferentes funciones las cuales pueden ser útiles.

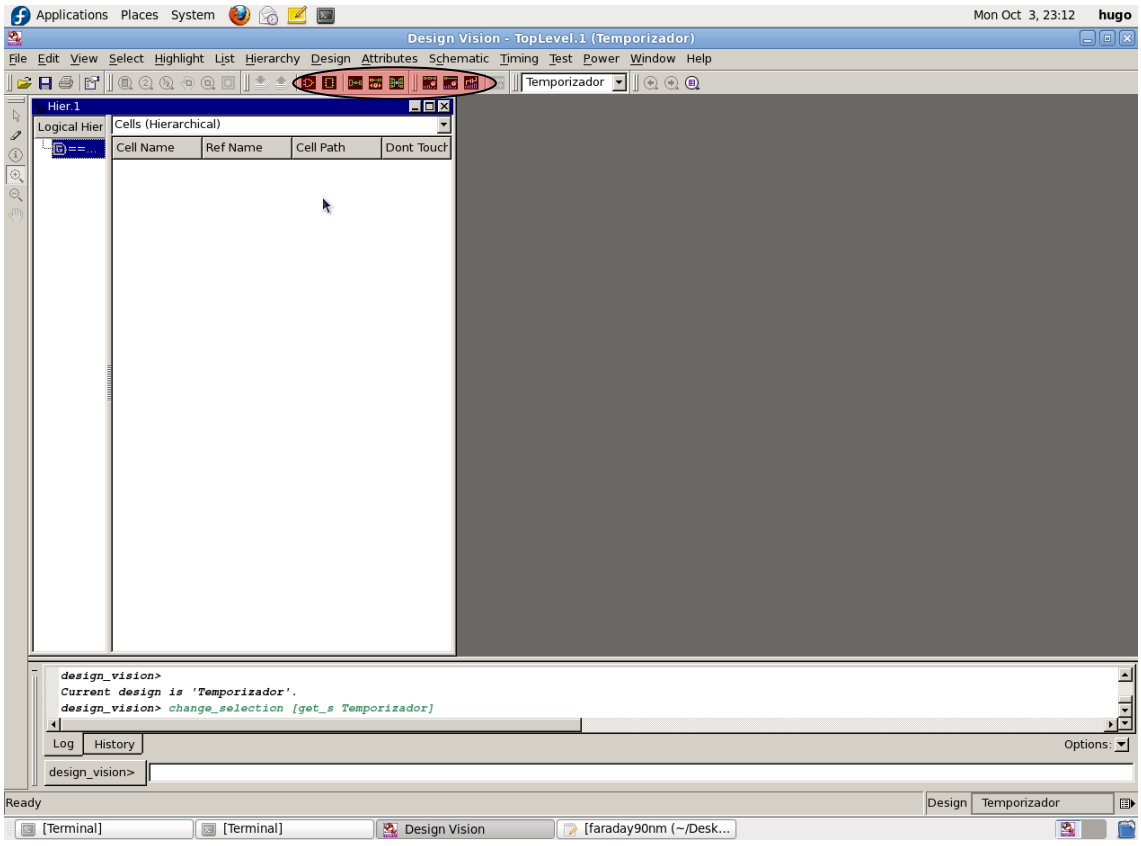


Figura 53. Opciones Synopsys

- Ver esquema del circuito. 

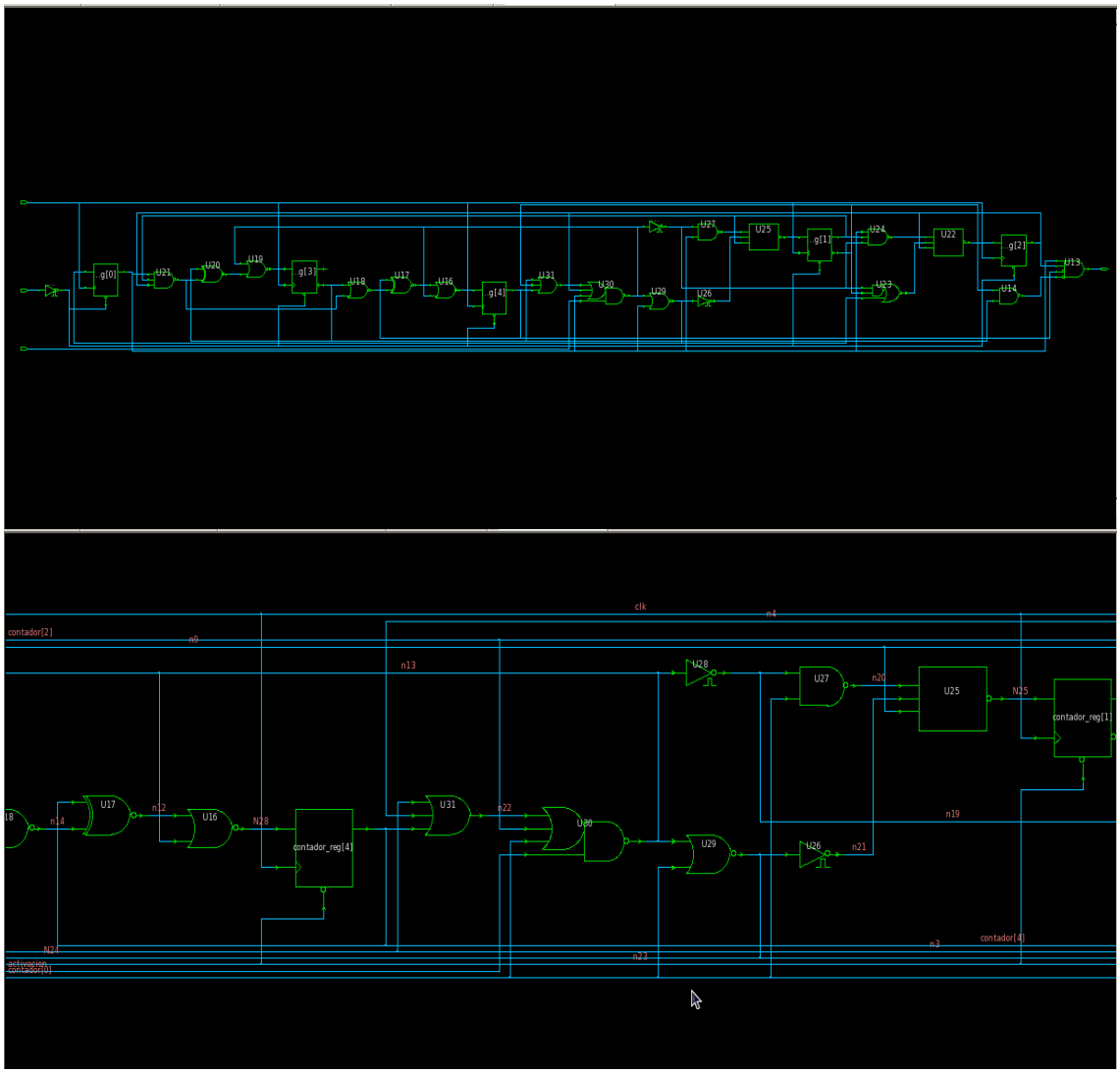


Figura 54. Hardware Synopsys

- Ver componente del circuito, en el que se visualizará simplemente el componente con los puertos de entrada y salida. 

Figura 55. Bloques Synopsys

- Ver los puertos del circuito que se han indicado en la entidad. 

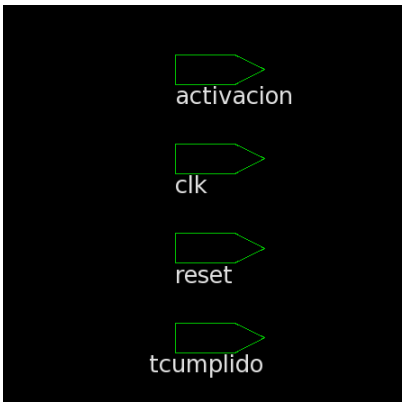


Figura 56. Puertos Synopsys

Para obtener el informe de pontecia es necesario establecer la frecuencia del reloj a la cual se supone que funcionará el circuito.
Para ello en el ciruito creado se selecciona el reloj y se establece su frecuencia en “Specify clock”

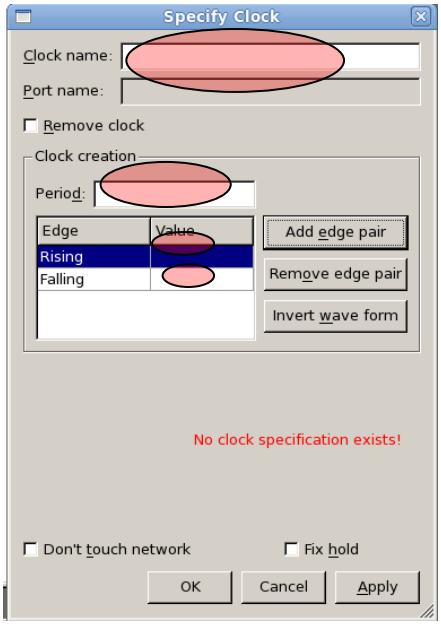


Figura 57. Reloj Synopsys

Para el caso de RFID se establece una frecuencia de 100kHz, para ello se establece la señal de la siguiente manera.

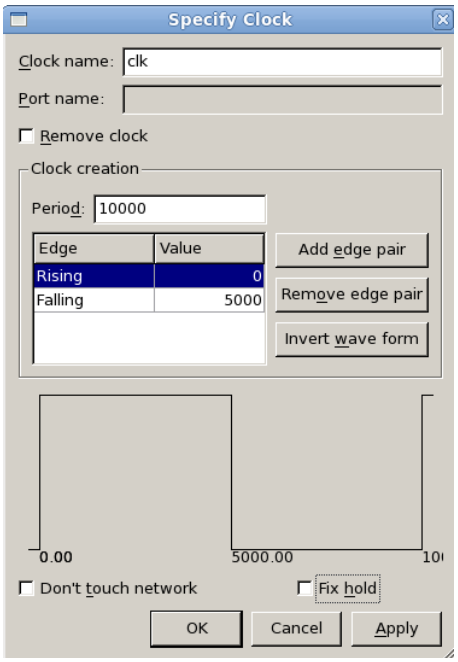


Figura 58. Configuración clk Synopsys

En este ejemplo se ha probado con un circuito, en el caso de que se tengan varios circuitos implementados juntos, se tendrán que añadir los circuitos más simples, añadiendo seguidamente los circuitos que componen a estos. El último diseño que se añade sería aquel que incluyo a todos los circuitos añadidos previamente.

CODIGOS

--PROTOCOLO TOTAL AZUMI PARA 32 BITS

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

--ENTIDAD

ENTITY Protocolo_Azumi_32 IS
GENERIC(N:INTEGER:=32);
PORT(
Dato_Reader_entrada: In Std_logic_vector (2*N-1 Downto 0);
Dato_Reader_salida: out std_logic_vector(4*N-1 downto 0);
Flag_Reader: In std_logic;
Flag_tag: out std_logic;
ok, no_ok: out std_logic;
CLK,RESET: IN STD_LOGIC
);
END Protocolo_Azumi_32;

architecture protocolo_Azumi_32_a of protocolo_Azumi_32 is
    Signal a,b,dato_xor: Std_logic_vector(N-1 downto 0);
        signal termino1,termino2,salida: std_logic_vector(N-1 downto 0);
    signal dato_random: std_logic_vector(N-1 downto 0);
    signal   dato_prgn: Std_logic_vector(N/2-1 downto 0);
    signal semilla_prgn: Std_logic_vector(N-1 downto 0);
    signal Flag_prgn,flag_random,inicio_prgn,inicio_random: Std_logic;
    signal carryin,carryout: std_logic;


Component Azumi_32
   GENERIC(N:INTEGER:=32);
PORT(
Dato_Reader_entrada: In Std_logic_vector (2*N-1 Downto 0);
Dato_Reader_salida: out std_logic_vector(4*N-1 downto 0);
Dato_xor: In Std_logic_vector(N-1 Downto 0);
Dato_prgn: In Std_logic_vector(N/2-1 Downto 0);
Dato_random: In Std_logic_vector(N-1 downto 0);
Operador_A: out Std_logic_vector((N-1) downto 0);
Operador_B: out Std_logic_vector (N-1 downto 0);
Semilla_Prgn: out Std_logic_vector (N-1 downto 0);
Flag_Prgn: in std_logic;
Flag_Random: in std_logic;
Flag_tag: out std_logic;
Inicio_prgn: out std_logic;
Inicio_random: out std_logic;
Flag_Reader: in std_logic;
ok, no_ok: out std_logic;
CLK,RESET: IN STD_LOGIC
);
   end component;

 Component funcion_xor_32
   GENERIC(N:INTEGER:=32);
   port (a : in Std_logic_vector(N-1 DOWNTO 0);
```

```
        b : in Std_logic_vector(N-1 DOWNTO 0);
        dato_xor : out Std_logic_vector(N-1 DOWNTO 0)
        );
   end component;

   Component prngc8_1
      GENERIC(N:INTEGER:=32);
      PORT(

      z0: OUT STD_LOGIC_VECTOR(((n/2)-1) DOWNTO 0);
      seed: IN STD_LOGIC_VECTOR (N-1 downto 0);
     INICIO: IN STD_LOGIC;
     FIN: OUT STD_LOGIC;
     CLK,RESET: IN STD_LOGIC;
     a1,a2: out std_logic_vector(N-1 downto 0);
     resultsuma:in std_logic_vector (N-1 downto 0)
      );
end component;

Component prngc8_4
      GENERIC(N:INTEGER:=32);
      PORT(
      z0: OUT STD_LOGIC_VECTOR(((n/2)-1) DOWNTO 0);
      seed: IN STD_LOGIC_VECTOR (N-1 downto 0);
      INICIO: IN STD_LOGIC;
      FIN: OUT STD_LOGIC;
      CLK,RESET: IN STD_LOGIC
      );
end component;

   Component sumador_2
      GENERIC (N:INTEGER:=32);
       Port(   termino1 :IN std_logic_Vector(N-1 DOWNTO 0);
     termino2 :IN std_logic_Vector(N-1 DOWNTO 0);
       salida: out std_logic_Vector(N-1 DOWNTO 0));
END component;

   Component sumador_3
       GENERIC (N:INTEGER:=32);
      Port(  termino1 : in std_logic_Vector((N/2)-1 DOWNTO 0);
        termino2 : in std_logic_Vector((N/2)-1 DOWNTO 0);
        carryin : in std_logic;
        salida : out std_logic_Vector((N/2)-1 DOWNTO 0);
        carryout : out std_logic);
END component;

  begin
       mi_maquina:azumi_32Portmap                    (dato_reader_entrada,dato_reader_salida,dato_xor,dato_prgn,
dato_random,a,b,semilla_prgn,flag_prgn,flag_random,flag_tag,inicio_prgn,inicio_random,flag_reader,ok,no_ok,clk,
reset);
       mi_funcion_xor: funcion_xor_32 Port map (a,b,dato_xor);
       mi_random:random_azumi_32Port map(dato_random,inicio_random,flag_random,clk, reset);
       mi_prgn: prngc8_1 Port map(dato_prgn,semilla_prgn,inicio_prgn,flag_prgn,
       clk,reset,termino1,termino2,salida);
     -- mi_sumador: Sumador_3 Port Map (termino1,termino2,carryin,salida,carryout);
end protocolo_azumi_32_a;
```

--MAQUINA DE ESTADOS AZUMI PARA 32 BITS

```
library ieee;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.numeric_std.all;

--ENTIDAD

ENTITY Azumi_32 IS
GENERIC(N:INTEGER:=32);
PORT(

Dato_Reader_entrada: In Std_logic_vector (2*N-1 Downto 0);
Dato_Reader_salida: out std_logic_vector(4*N-1 downto 0);
Dato_xor: In Std_logic_vector(N-1 Downto 0);
Dato_prgn: In Std_logic_vector(N/2-1 Downto 0);
Dato_random: In Std_logic_vector(N-1 downto 0);
Operador_A: out Std_logic_vector((N-1) downto 0);
Operador_B: out Std_logic_vector (N-1 downto 0);
Semilla_Prgn: out Std_logic_vector (N-1 downto 0);
Flag_Prgn: in std_logic;
Flag_Random: in std_logic;
Flag_tag: out std_logic;
Inicio_prgn: out std_logic;
Inicio_random: out std_logic;
Flag_Reader: in std_logic;
ok, no_ok: out std_logic;
CLK,RESET: IN STD_LOGIC
);
END Azumi_32;

architecture Azumi_32_a of Azumi_32 is
type estado is (espera,inicio,Random_1,obtener_random_1,random_2,
funcion_A1,funcion_A2,funcion_A3,funcion_A4,funcion_A5,funcion_A6,
Prng_A,Prng_A2,Resultado_A,
Funcion_B1,Funcion_B2,Prgn_B,Prgn_B2,Resultado_B,
Funcion_C1,Resultado_C,
Enviar_Tag,Recibir_D,
Funcion_D1,Funcion_D2,Funcion_D3,Funcion_D4,Funcion_D5,Funcion_D6,Funcion_D7,Funcion_D8,
Prng_D,Prng_D2,Resultado_D, Updating,error,comparar,
Updating_k16,Updating_N16,Envio_datos,Prng_k16,Prng_N16,
Prng_N16_2,Prng_k16_2);
signal actual,siguiente: estado;
signal h1,h2: std_logic_vector(N-1 downto 0); -- Registros señales intermedias
signal h3:std_logic_vector(N-1 downto 0):= x"00000066";--RND
signal h4: std_logic_vector(N-1 downto 0):=x"00000021";--RND'
signal Nt: std_logic_Vector(2*N-1 downto 0):= x"0000000000000022";
signal EPC16: std_logic_vector(N-1 downto 0):= x"00000033";
signal k16: std_logic_vector(N-1 downto 0):= x"00000044";
signal resultado,Resultado_aux:std_logic_vector(4*N-1 downto 0);
```

```
signal N16: std_logic_vector(N-1 downto 0):= x"00000055";
signal flag_h1: std_logic_vector(1 downto 0);
signal aux_h2,
aux_k16,aux_N16,aux_h3,aux_h4: std_logic_vector(N-1 downto 0);
Begin

 --PROCESO SECUENCIAL DE LA MAQUINA DE ESTADOS

process(clk, reset)

begin
   if reset = '1' then
      actual<=espera;
   elsif clk'event and clk='1' then
      actual<=siguiente;

end if;
end process;

-- Proceso secuencial H1
process(clk,reset)
    begin
        if reset='1' then
         H1<=(others=>'0');
         elsif clk'event and clk='1' then
         if flag_h1="01" then --Guardar el valor de ID16
           h1<=Dato_Reader_entrada(2*n-1 downto N);
         elsif flag_h1="11" then
            h1<=Dato_Xor;

     end if;
 end if;
 end process;

 -- Proceso secuencial
process(clk,reset)
    begin
        if reset='1' then
         H2<=(others=>'0');
         Resultado<=(others=>'0');
         h3<=(others=>'0');
         h4<=(others=>'0');
         elsif clk'event and clk='1' then

        h2<=aux_h2;
         k16<=aux_k16;
         N16<=aux_N16;
         h3<=aux_h3;
         h4<=aux_h4;
    Resultado<=Resultado_aux;

     end if;
 end process;

 -- Proceso Combinacional
    Process (Actual,flag_reader,flag_random,dato_xor,flag_Prgn,h1,h2,h3,Nt,EPC16,H4,
```

```
k16,Resultado,N16)
   Begin
      Case Actual Is
         When Espera=>
            Operador_A<=(others=>'0');
            Operador_B<=(others=>'0');
            Semilla_PRGN<=(others=>'0');
            Inicio_PRGN<='0';
            Inicio_Random<='0';
            ok<='0';
            no_ok<='0';
            flag_tag<='0';
            flag_h1<="00";
            aux_h2<=Dato_Reader_entrada(n-1 downto 0);
            aux_k16<=x"00000044";
            aux_N16<=x"00000055";
            aux_h3<=h3;
            aux_h4<=h4;
            Resultado_aux<=Resultado;
            Dato_Reader_Salida<=Resultado;
            if flag_reader = '1' then
               siguiente<=inicio;
             else
                siguiente<=actual;
             end if;

         When Inicio=>
            Operador_A<=(others=>'0');
            Operador_B<=(others=>'0');
            Semilla_PRGN<=(others=>'0');
            Inicio_PRGN<='0';
            Inicio_Random<='0';
            ok<='0';
            no_ok<='0';
            flag_tag<='0';
            flag_h1<="01";
            aux_h2<=Dato_Reader_entrada(n-1 downto 0);
            aux_k16<=k16;
            aux_N16<=N16;
            aux_h3<=h3;
            aux_h4<=h4;
            Resultado_aux<=Resultado;
            Dato_Reader_Salida<=Resultado;
            siguiente<=Random_1;

         when Random_1=> --Se pide calcular RND
           Operador_A<=(others=>'0');
           Operador_B<=(others=>'0');
           Semilla_PRGN<=(others=>'0');
           Inicio_PRGN<='0';
           Inicio_Random<='1';
            ok<='0';
           no_ok<='0';
          flag_tag<='0';
            flag_h1<="00";
            aux_h2<=Dato_Reader_entrada(n-1 downto 0);
```

```
    aux_k16<=k16;
    aux_N16<=N16;
    aux_h3<=h3;
    aux_h4<=h4;
    Dato_Reader_Salida<=Resultado;
    Resultado_aux<=Resultado;
    If flag_random='1' then
    siguiente<=Obtener_Random_1;
    else
    siguiente<=actual;
    end if;

when obtener_Random_1=>
    Operador_A<=(others=>'0');
   Operador_B<=(others=>'0');
    Semilla_PRGN<=(others=>'0');
    Inicio_PRGN<='0';
    Inicio_Random<='0';
    ok<='0';
    no_ok<='0';
  flag_tag<='0';
    flag_h1<="00";
    aux_h2<=Dato_Reader_entrada(n-1 downto 0);
    aux_k16<=k16;
    aux_N16<=N16;
    aux_h3<=dato_random;
    aux_h4<=h4;
    Dato_Reader_Salida<=Resultado;
    Resultado_aux<=Resultado;
    siguiente<=Random_2;

when Random_2=> --Se pide calcular RND'
    Operador_A<=(others=>'0');
    Operador_B<=(others=>'0');
    Semilla_PRGN<=(others=>'0');
    Inicio_PRGN<='0';
    Inicio_Random<='1';
    ok<='0';
    no_ok<='0';
    flag_tag<='0';
     flag_h1<="00";
    aux_h2<=Dato_Reader_entrada(n-1 downto 0);
    aux_k16<=k16;
    aux_N16<=N16;
    aux_h3<=h3;
    aux_h4<=Dato_Random;
    Dato_Reader_Salida<=Resultado;
    Resultado_aux<=Resultado;
    if Flag_random='1' then
        siguiente<=Funcion_A1;
        else
       siguiente<=actual;
    end if;
```

```
when Funcion_A1 => --XOR (rnd_reader,id16)
   Operador_A<=H1;
   Operador_B<=H2;
   Semilla_PRGN<=(others=>'0');
   Inicio_PRGN<='0';
   Inicio_Random<='0';
   ok<='0';
   no_ok<='0';
   flag_tag<='0';
   flag_h1<="11";
   aux_h2<=h2;
   aux_k16<=k16;
   aux_N16<=N16;
   aux_h3<=h3;
   aux_h4<=h4;
   Resultado_aux<=Resultado;
   Dato_Reader_Salida<=Resultado;
   siguiente<=Funcion_A2;

When Funcion_A2 => -- Guardar resultado XOR
   Operador_A<=(others=>'0');
   Operador_B<=(others=>'0');
   Semilla_PRGN<=(others=>'0');
   Inicio_PRGN<='0';
   Inicio_Random<='0';
   ok<='0';
   no_ok<='0';
   flag_tag<='0';
   flag_h1<="00";
   aux_h2<=h2;
   aux_k16<=k16;
   aux_N16<=N16;
   aux_h3<=h3;
   aux_h4<=h4;
   Resultado_aux<=Resultado;
   Dato_Reader_Salida<=Resultado;
   siguiente<=Funcion_A3;

When Funcion_A3 => -- XOR (RND,H1)
   Operador_A<=H1;
   Operador_B<=H3;
   Semilla_PRGN<=(others=>'0');
   Inicio_PRGN<='0';
   Inicio_Random<='0';
   ok<='0';
   no_ok<='0';
   flag_tag<='0';
   flag_h1<="11";
   aux_h2<=h2;
   aux_k16<=k16;
   aux_N16<=N16;
   aux_h3<=h3;
   aux_h4<=h4;
   Resultado_aux<=Resultado;
   Dato_Reader_Salida<=Resultado;
   siguiente<=Funcion_A4;
```

```
When Funcion_A4 => -- Guardar resultado XOR
   Operador_A<=(others=>'0');
   Operador_B<=(others=>'0');
   Semilla_PRGN<=(others=>'0');
   Inicio_PRGN<='0';
   Inicio_Random<='0';
   ok<='0';
   no_ok<='0';
   flag_tag<='0';
   flag_h1<="00";
   aux_h2<=h2;
   aux_k16<=k16;
   aux_N16<=N16;
   aux_h3<=h3;
   aux_h4<=h4;
   Resultado_aux<=Resultado;
   Dato_Reader_Salida<=Resultado;
   siguiente<=Funcion_A5;

When Funcion_A5 => -- XOR (Nt,H1)
   Operador_A<=H1;
   Operador_B<=Nt(N-1 downto 0);
   Semilla_PRGN<=(others=>'0');
   Inicio_PRGN<='0';
   Inicio_Random<='0';
   ok<='0';
   no_ok<='0';
   flag_tag<='0';
   flag_h1<="11";
   aux_h2<=h2;
   aux_k16<=k16;
   aux_N16<=N16;
   aux_h3<=h3;
   aux_h4<=h4;
   Resultado_aux<=Resultado;
   Dato_Reader_Salida<=Resultado;
   siguiente<=Funcion_A6;

When Funcion_A6 => -- Guardar resultado XOR
   Operador_A<=(others=>'0');
   Operador_B<=(others=>'0');
   Semilla_PRGN<=(others=>'0');
   Inicio_PRGN<='0';
   Inicio_Random<='0';
   ok<='0';
   no_ok<='0';
   flag_tag<='0';
   flag_h1<="00";
   aux_h2<=h2;
   aux_k16<=k16;
   aux_N16<=N16;
   aux_h3<=h3;
   aux_h4<=h4;
   Resultado_aux<=Resultado;
   Dato_Reader_Salida<=Resultado;
```

```
    siguiente<=PRng_A;

When Prng_A => -- Calcular el numero PRNG (low) para A
    Operador_A<=(others=>'0');
    Operador_B<=(others=>'0');
    Semilla_PRGN<= H1;
    Inicio_PRGN<='1';
    Inicio_Random<='0';
    ok<='0';
    no_ok<='0';
    flag_tag<='0';
    flag_h1<="00";
    aux_h2<=h2;
    aux_k16<=k16;
    aux_N16<=N16;
    aux_h3<=h3;
    aux_h4<=h4;
   --Resultado_aux<=Resultado(127 downto 96) & Dato_Prgn & Resultado(79 downto 0);
   Resultado_aux(95 downto 80)<=Dato_Prgn;
    Dato_Reader_Salida<=Resultado;
    if flag_Prgn='1' then
    siguiente<=Prng_A2;
    else
    siguiente<=Actual;
    end if;

When Prng_A2 => -- Calcular el numero PRNG (High) para A
    Operador_A<=(others=>'0');
    Operador_B<=(others=>'0');
    Semilla_PRGN<= Nt(2*N-1 downto N);
    Inicio_PRGN<='1';
    Inicio_Random<='0';
    ok<='0';
    no_ok<='0';
    flag_tag<='0';
    flag_h1<="00";
    aux_h2<=h2;
    aux_k16<=k16;
    aux_N16<=N16;
    aux_h3<=h3;
    aux_h4<=h4;
    Resultado_aux<=Resultado;
    Dato_Reader_Salida<=Resultado;
    if flag_Prgn='1' then
    siguiente<=Resultado_A;
    else
    siguiente<=Actual;
    end if;

when Resultado_A => --Obtener el resultado para A
    Operador_A<=(others=>'0');
    Operador_B<=(others=>'0');
    Semilla_PRGN<=(others=>'0');
    Inicio_PRGN<='0';
    Inicio_Random<='0';
    ok<='0';
```

```
    no_ok<='0';
    flag_tag<='0';
    flag_h1<="00";
    aux_h2<=h2;
    aux_k16<=k16;
    aux_N16<=N16;
    aux_h3<=h3;
    aux_h4<=h4;
   -- Resultado_aux<=Resultado(127 downto 80) & Dato_Prgn & Resultado(63 downto 0);
    Resultado_aux(79 downto 64)<=Dato_Prgn;
    Dato_Reader_Salida<=Resultado;
    siguiente<=Funcion_B1;

When Funcion_B1 => -- XOR Para B (RPC16t, RND')
    Operador_A<=EPC16;
    Operador_B<=H4;
    Semilla_PRGN<=(others=>'0');
    Inicio_PRGN<='0';
    Inicio_Random<='0';
    ok<='0';
    no_ok<='0';
    flag_tag<='0';
    flag_h1<="11";
    aux_h2<=h2;
    aux_k16<=k16;
    aux_N16<=N16;
    aux_h3<=h3;
    aux_h4<=h4;
    Resultado_aux<=Resultado;
    Dato_Reader_Salida<=Resultado;
    siguiente<=Funcion_B2;

When Funcion_B2 => -- Guardar el resultado
    Operador_A<=(others=>'0');
    Operador_B<=(others=>'0');
    Semilla_PRGN<=(others=>'0');
    Inicio_PRGN<='0';
    Inicio_Random<='0';
    ok<='0';
    no_ok<='0';
    flag_tag<='0';
    flag_h1<="00";
    aux_h2<=h2;
    aux_k16<=k16;
    aux_N16<=N16;
    aux_h3<=h3;
    aux_h4<=h4;
    Resultado_aux<=Resultado;
    Dato_Reader_Salida<=Resultado;
    siguiente<=PRgn_B;

when PRgn_B => -- Obtener el PRGn de B (low)
    Operador_A<=(others=>'0');
    Operador_B<=(others=>'0');
    Semilla_PRGN<= H1;
    Inicio_PRGN<='1';
```

```
    Inicio_Random<='0';
    ok<='0';
    no_ok<='0';
    flag_tag<='0';
    flag_h1<="00";
    aux_h2<=h2;
    aux_k16<=k16;
    aux_N16<=N16;
    aux_h3<=h3;
    aux_h4<=h4;
  -- Resultado_aux<=Resultado(127 downto 64) & Dato_Prgn & Resultado(47 downto 0);
    Resultado_aux(63 downto 48)<=Dato_Prgn;
    Dato_Reader_Salida<=Resultado;
    if flag_prgn='1' then
    siguiente<=Prgn_B2;
    else
    siguiente<=actual;
    end if;

 when PRgn_B2 => -- Obtener el PRGn de B (High)
    Operador_A<=(others=>'0');
    Operador_B<=(others=>'0');
    Semilla_PRGN<=Conv_std_logic_Vector(0,N);
    Inicio_PRGN<='1';
    Inicio_Random<='0';
    ok<='0';
    no_ok<='0';
    flag_tag<='0';
    flag_h1<="00";
    aux_h2<=h2;
    aux_k16<=k16;
    aux_N16<=N16;
    aux_h3<=h3;
    aux_h4<=h4;
    Resultado_aux<=Resultado;
    Dato_Reader_Salida<=Resultado;
    if flag_prgn='1' then
    siguiente<=Resultado_B;
    else
    siguiente<=actual;
    end if;

 when Resultado_B => -- Resultado
    Operador_A<=(others=>'0');
    Operador_B<=(others=>'0');
    Semilla_PRGN<=(others=>'0');
    Inicio_PRGN<='0';
    Inicio_Random<='0';
    ok<='0';
    no_ok<='0';
    flag_tag<='0';
    flag_h1<="00";
    aux_h2<=h2;
    aux_k16<=k16;
    aux_N16<=N16;
    aux_h3<=h3;
```

```
    aux_h4<=h4;
 -- Resultado_aux<=Resultado(127 downto 48) & Dato_Prgn & Resultado(31 downto 0);
    Resultado_aux(47 downto 32)<=Dato_Prgn;
    Dato_Reader_Salida<=Resultado;
    siguiente<=Funcion_C1;

When Funcion_C1=> --Obtener C xor Rnd',k16
    Operador_A<=H4;
    Operador_B<=k16;
    Semilla_PRGN<=(others=>'0');
    Inicio_PRGN<='0';
    Inicio_Random<='0';
    ok<='0';
    no_ok<='0';
    flag_tag<='0';
    flag_h1<="11";
    aux_h2<=h2;
    aux_k16<=k16;
    aux_N16<=N16;
    aux_h3<=h3;
    aux_h4<=h4;
   Resultado_aux(127 downto 96)<=H3;
    Resultado_aux(31 downto 0)<=Dato_xor;

    Dato_Reader_Salida<=Resultado;
    siguiente<=Resultado_C;

When Resultado_C=> --Obtener Resultado_C
    Operador_A<=(others=>'0');
    Operador_B<=(others=>'0');
    Semilla_PRGN<=(others=>'0');
    Inicio_PRGN<='0';
    Inicio_Random<='0';
    ok<='0';
    no_ok<='0';
    flag_tag<='0';
    flag_h1<="00";
    aux_h2<=h2;
    aux_k16<=k16;
    aux_N16<=N16;
    aux_h3<=h3;
    aux_h4<=h4;
    Resultado_aux<=Resultado;
    Dato_Reader_Salida<=Resultado;
    siguiente<=Enviar_Tag;

When Enviar_Tag=> --Se envia el valor de Resultado al reader
    Operador_A<=(others=>'0');
    Operador_B<=(others=>'0');
    Semilla_PRGN<=(others=>'0');
    Inicio_PRGN<='0';
    Inicio_Random<='0';
    ok<='0';
    no_ok<='0';
    flag_tag<='1';
    flag_h1<="00";
```

```
    aux_h2<=h2;
    aux_k16<=k16;
    aux_N16<=N16;
    aux_h3<=h3;
    aux_h4<=h4;
    Resultado_aux<=Resultado;
    Dato_Reader_Salida<=Resultado;
    if flag_reader='1' then
    siguiente<=Recibir_D;
    else
    siguiente<=actual;
    end if;

When Recibir_D=> -- Guardar el valor de D del lector y calcular el tag=>D
    Operador_A<=(others=>'0');
    Operador_B<=(others=>'0');
    Semilla_PRGN<=(others=>'0');
    Inicio_PRGN<='0';
    Inicio_Random<='0';
    ok<='0';
    no_ok<='0';
    flag_tag<='0';
    flag_h1<="00";
    aux_h2<=h2;
    aux_k16<=k16;
    aux_N16<=N16;
    aux_h3<=h3;
    aux_h4<=h4;
    Resultado_aux(31 downto 0)<= Dato_reader_entrada(N-1 downto 0);

    Resultado_aux(127 downto 32)<=conv_std_logic_vector(0,3*N);
    Dato_Reader_Salida<=Resultado;
    siguiente<=Funcion_D1;

when Funcion_D1=> -- Calcular el valor D desde la tag
    Operador_A<=H4;
    Operador_B<=EPC16;
    Semilla_PRGN<=(others=>'0');
    Inicio_PRGN<='0';
    Inicio_Random<='0';
    ok<='0';
    no_ok<='0';
    flag_tag<='0';
    flag_h1<="11";
    aux_h2<=h2;
    aux_k16<=k16;
    aux_N16<=N16;
    aux_h3<=h3;
    aux_h4<=h4;
    Resultado_aux<=Resultado;
    Dato_Reader_Salida<=Resultado;
    siguiente<=Funcion_D2;

when Funcion_D2 => --Obtener valor de XOR
    Operador_A<=(others=>'0');
    Operador_B<=(others=>'0');
```

```
    Semilla_PRGN<=(others=>'0');
    Inicio_PRGN<='0';
    Inicio_Random<='0';
    ok<='0';
    no_ok<='0';
    flag_tag<='0';
    flag_h1<="00";
    aux_h2<=h2;
    aux_k16<=k16;
    aux_N16<=N16;
    aux_h3<=h3;
    aux_h4<=h4;
    Resultado_aux<=Resultado;
    Dato_Reader_Salida<=Resultado;
    siguiente<=Funcion_D3;

When Funcion_D3 => --  XOR
    Operador_A<=H1;
    Operador_B<=H3;
    Semilla_PRGN<=(others=>'0');
    Inicio_PRGN<='0';
    Inicio_Random<='0';
    ok<='0';
    no_ok<='0';
    flag_tag<='0';
    flag_h1<="11";
    aux_h2<=h2;
    aux_k16<=k16;
    aux_N16<=N16;
    aux_h3<=h3;
    aux_h4<=h4;
    Resultado_aux<=Resultado;
    Dato_Reader_Salida<=Resultado;
    siguiente<=Funcion_D4;

When Funcion_D4 => -- Guardar XOR
    Operador_A<=(others=>'0');
    Operador_B<=(others=>'0');
    Semilla_PRGN<=(others=>'0');
    Inicio_PRGN<='0';
    Inicio_Random<='0';
    ok<='0';
    no_ok<='0';
    flag_tag<='0';
    flag_h1<="00";
    aux_h2<=h2;
    aux_k16<=k16;
    aux_N16<=N16;
    aux_h3<=h3;
    aux_h4<=h4;
    Resultado_aux<=Resultado;
    Dato_Reader_Salida<=Resultado;
    siguiente<=Funcion_D5;

When Funcion_D5 => --  XOR
    Operador_A<=H1;
```

```
        Operador_B<=H2;
        Semilla_PRGN<=(others=>'0');
        Inicio_PRGN<='0';
        Inicio_Random<='0';
        ok<='0';
        no_ok<='0';
        flag_tag<='0';
        flag_h1<="11";
        aux_h2<=h2;
        aux_k16<=k16;
        aux_N16<=N16;
        aux_h3<=h3;
        aux_h4<=h4;
        Resultado_aux<=Resultado;
        Dato_Reader_Salida<=Resultado;
        siguiente<=Funcion_D6;

    When Funcion_D6 => -- Guardar
        Operador_A<=(others=>'0');
        Operador_B<=(others=>'0');
        Semilla_PRGN<=(others=>'0');
        Inicio_PRGN<='0';
        Inicio_Random<='0';
        ok<='0';
        no_ok<='0';
        flag_tag<='0';
        flag_h1<="00";
        aux_h2<=h2;
        aux_k16<=k16;
        aux_N16<=N16;
        aux_h3<=h3;
        aux_h4<=h4;
        Resultado_aux<=Resultado;
        Dato_Reader_Salida<=Resultado;
        siguiente<=Funcion_D7;

    When Funcion_D7 => --  XOR
        Operador_A<=Nt(N-1 downto 0);
        Operador_B<=H1;
        Semilla_PRGN<=(others=>'0');
        Inicio_PRGN<='0';
        Inicio_Random<='0';
        ok<='0';
        no_ok<='0';
        flag_tag<='0';
        flag_h1<="11";
        aux_h2<=h2;
        aux_k16<=k16;
        aux_N16<=N16;
        aux_h3<=h3;
        aux_h4<=h4;
        Resultado_aux<=Resultado;
        Dato_Reader_Salida<=Resultado;
        siguiente<=Funcion_D8;

    When Funcion_D8 => --  Guardar el valor semilla
```

```
    Operador_A<=(others=>'0');
    Operador_B<=(others=>'0');
    Semilla_PRGN<=(others=>'0');
    Inicio_PRGN<='0';
    Inicio_Random<='0';
    ok<='0';
    no_ok<='0';
    flag_tag<='0';
    flag_h1<="00";
    aux_h2<=h2;
    aux_k16<=k16;
    aux_N16<=N16;
    aux_h3<=h3;
    aux_h4<=h4;
    Resultado_aux<=Resultado;
    Dato_Reader_Salida<=Resultado;
    siguiente<=PRng_D;

When PRng_D => --  Calcular el prgn para D (low)
    Operador_A<=(others=>'0');
    Operador_B<=(others=>'0');
    Semilla_PRGN<= h1;
    Inicio_PRGN<='1';
    Inicio_Random<='0';
    ok<='0';
    no_ok<='0';
    flag_tag<='0';
    flag_h1<="00";
    aux_h2<=h2;
    aux_k16<=k16;
    aux_N16<=N16;
    aux_h3<=h3;
    aux_h4<=h4;
   Resultado_aux(47 downto 32)<=Dato_Prgn;

    Dato_Reader_Salida<=Resultado;
    if flag_prgn='1' then
    siguiente<=Prng_D2;
    else
    siguiente<=Actual;
    end if;

When PRng_D2 => --  Calcular el prgn para D (High)
    Operador_A<=(others=>'0');
    Operador_B<=(others=>'0');
    Semilla_PRGN<=Nt(2*n-1 downto 32);
    Inicio_PRGN<='1';
    Inicio_Random<='0';
    ok<='0';
    no_ok<='0';
    flag_tag<='0';
    flag_h1<="00";
    aux_h2<=h2;
    aux_k16<=k16;
    aux_N16<=N16;
    aux_h3<=h3;
```

```
    aux_h4<=h4;
   Resultado_aux(63 downto 48)<=Dato_prgn;

    Dato_Reader_Salida<=Resultado;
    if flag_prgn='1' then
    siguiente<=Resultado_D;
    else
    siguiente<=Actual;
    end if;

When Resultado_D => -- Calcular D
    Operador_A<=(others=>'0');
    Operador_B<=(others=>'0');
    Semilla_PRGN<=(others=>'0');
    Inicio_PRGN<='0';
    Inicio_Random<='0';
    ok<='0';
    no_ok<='0';
    flag_tag<='0';
    flag_h1<="00";
    aux_h2<=h2;
    aux_k16<=k16;
    aux_N16<=N16;
    aux_h3<=h3;
    aux_h4<=h4;
    Resultado_aux<=Resultado;
    Dato_Reader_Salida<=Resultado;
    siguiente<=Comparar;

When Comparar => -- Comparar D tag con D Reader
    Operador_A<=(others=>'0');
    Operador_B<=(others=>'0');
    Semilla_PRGN<=(others=>'0');
    Inicio_PRGN<='0';
    Inicio_Random<='0';
    ok<='0';
    no_ok<='0';
    flag_tag<='0';
    flag_h1<="00";
    aux_h2<=h2;
    aux_k16<=k16;
    aux_N16<=N16;
    aux_h3<=h3;
    aux_h4<=h4;
    Resultado_aux<=Resultado;
    Dato_Reader_Salida<=Resultado;
    if resultado(63 downto 32)=Resultado(31 downto 0) then
    siguiente<=Updating;
    else
    siguiente<=error;
end if;


When Updating => -- Procolo Ok=> actualizar N16,K16
    Operador_A<=(others=>'0');
    Operador_B<=(others=>'0');
```

```
        Semilla_PRGN<=(others=>'0');
        Inicio_PRGN<='0';
        Inicio_Random<='0';
        ok<='1';
        no_ok<='0';
        flag_tag<='0';
        flag_h1<="00";
        aux_h2<=h2;
        aux_k16<=k16;
        aux_N16<=N16;
        aux_h3<=h3;
        aux_h4<=h4;
        Resultado_aux<=Resultado;
        Dato_Reader_Salida<=Resultado;
        siguiente<=Updating_N16;

    When Error => -- Abortar protocolo
        Operador_A<=(others=>'0');
        Operador_B<=(others=>'0');
        Semilla_PRGN<=(others=>'0');
        Inicio_PRGN<='0';
        Inicio_Random<='0';
        ok<='0';
        no_ok<='1';
        flag_tag<='0';
        flag_h1<="00";
        aux_h2<=h2;
        aux_k16<=k16;
        aux_N16<=N16;
        aux_h3<=h3;
        aux_h4<=h4;
        Resultado_aux<=Resultado;
        Dato_Reader_Salida<=Resultado;
        siguiente<=Error;

    When Updating_N16 => -- Xor N16,RND'
        Operador_A<=h4;
        Operador_B<=N16;
        Semilla_PRGN<=(others=>'0');
        Inicio_PRGN<='0';
        Inicio_Random<='0';
        ok<='1';
        no_ok<='0';
        flag_tag<='0';
        flag_h1<="00";
        aux_h2<=h2;
        aux_k16<=k16;
        aux_N16<=dATO_XOR;
        aux_h3<=h3;
        aux_h4<=h4;
        Dato_Reader_Salida<=Resultado;
        Resultado_aux<=Resultado;
        siguiente<=Prng_N16;

    When Prng_N16 => -- PRng N16,RND' parte low
        Operador_A<=(others=>'0');
```

```
    Operador_B<=(others=>'0');
    Semilla_PRGN<= N16;
    Inicio_PRGN<='1';
    Inicio_Random<='0';
    ok<='1';
    no_ok<='0';
    flag_tag<='0';
    flag_h1<="00";
    aux_h2<=h2;
    aux_k16<=k16;
    aux_N16(n/2-1 downto 0)<=Dato_prgn;
    aux_h3<=h3;
    aux_h4<=h4;
    Dato_Reader_Salida<=Resultado;
    Resultado_aux<=Resultado;
    if flag_prgn='1' then
    siguiente<=Prng_N16_2;
    else
    siguiente<=Prng_N16;
    end if;

When Prng_N16_2 => -- PRng N16,RND' parte high
    Operador_A<=(others=>'0');
    Operador_B<=(others=>'0');
    Semilla_PRGN<= N16;
    Inicio_PRGN<='1';
    Inicio_Random<='0';
    ok<='1';
    no_ok<='0';
    flag_tag<='0';
    flag_h1<="00";
    aux_h2<=h2;
    aux_k16<=k16;
    aux_N16(n-1 downto 16)<=Dato_prgn;
    aux_h3<=h3;
    aux_h4<=h4;
    Dato_Reader_Salida<=Resultado;
    Resultado_aux<=Resultado;
    if flag_prgn='1' then
    siguiente<=Updating_k16;
    else
    siguiente<=Prng_N16_2;
    end if;

When Updating_K16 => -- Xor k16,RND'
    Operador_A<=K16;
    Operador_B<=H4;
    Semilla_PRGN<=(others=>'0');
    Inicio_PRGN<='0';
    Inicio_Random<='0';
    ok<='1';
    no_ok<='0';
    flag_tag<='0';
    flag_h1<="00";
    aux_h2<=h2;
    aux_k16<=Dato_Xor;
```

```
    aux_N16<=N16;
    aux_h3<=h3;
    aux_h4<=h4;
    Dato_Reader_Salida<=Resultado;
    Resultado_aux<=Resultado;
    siguiente<=PRng_K16;

When Prng_k16 => -- PRng k16,RND'
    Operador_A<=(others=>'0');
    Operador_B<=(others=>'0');
    Semilla_PRGN<= k16;
    Inicio_PRGN<='1';
    Inicio_Random<='0';
    ok<='1';
    no_ok<='0';
    flag_tag<='0';
    flag_h1<="00";
    aux_h2<=h2;
    aux_k16(n/2 -1 downto 0)<=dato_prgn;
    aux_N16<=N16;
    aux_h3<=h3;
    aux_h4<=h4;
    Dato_Reader_Salida<=Resultado;
    Resultado_aux<=Resultado;
    if flag_prgn='1' then
    siguiente<=Prng_K16_2;
    else
    siguiente<=Prng_k16;
    end if;

When Prng_k16_2 => -- PRng k16,RND' parte high
    Operador_A<=(others=>'0');
    Operador_B<=(others=>'0');
    Semilla_PRGN<= N16;
    Inicio_PRGN<='1';
    Inicio_Random<='0';
    ok<='1';
    no_ok<='0';
    flag_tag<='0';
    flag_h1<="00";
    aux_h2<=h2;
    aux_k16<=k16;
    aux_N16(n-1 downto 16)<=Dato_prgn;
    aux_h3<=h3;
    aux_h4<=h4;
    Dato_Reader_Salida<=Resultado;
    Resultado_aux<=Resultado;
    if flag_prgn='1' then
    siguiente<=Envio_datos;
    else
    siguiente<=Prng_k16_2;
    end if;

When Envio_datos => -- Estado de envio de datos
    Operador_A<=(others=>'0');
    Operador_B<=(others=>'0');
```

```
                Semilla_PRGN<=(others=>'0');
                Inicio_PRGN<='0';
                Inicio_Random<='0';
                ok<='1';
                no_ok<='0';
                flag_tag<='0';
                flag_h1<="00";
                aux_h2<=h2;
                aux_k16<=k16;
                aux_N16<=N16;
                aux_h3<=h3;
                aux_h4<=h4;
                Resultado_aux<=Resultado;
                Dato_Reader_Salida<=Resultado;
                siguiente<=Espera;

            end case;
        end process;

End Azumi_32_a;
```

---FUNCION XOR

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity funcion_xor_32 is
   GENERIC(N:INTEGER:=32);
  port (a : in Std_logic_vector(N-1 DOWNTO 0);
       b : in Std_logic_vector(N-1 DOWNTO 0);

       dato_xor : out Std_logic_vector(N-1 DOWNTO 0)
       );
end funcion_xor_32;

-- implementación de comportamiendo del xor
architecture funcion_xor_32_a of funcion_xor_32 is
begin
  dato_xor<=a + b;
end funcion_xor_32_a;
```

--- SUMADOR DE LOS GENERADORES PRNG

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
   --ENTIDAD

ENTITY sumador_2 IS
   GENERIC (N:INTEGER:=32);
Port(   termino1 :IN std_logic_Vector(N-1 DOWNTO 0);
       termino2 :IN std_logic_Vector(N-1 DOWNTO 0);
       salida: out std_logic_Vector(N-1 DOWNTO 0));
```

```
END sumador_2;
ARCHITECTURE f_2 OF sumador_2 is
   begin
                    process(termino1,termino2)
       begin
      salida<=termino1 + termino2;

                    end process;
   end f_2;


-- ALGORITMO A 8 BITS
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
   --ENTIDAD


---GENERADOR PRNG-1, VARIOS SUMADORES
ENTITY prngC8_1 IS
GENERIC(N:INTEGER:=32);
PORT(

z0: OUT STD_LOGIC_VECTOR(((n/2)-1) DOWNTO 0);
seed: IN STD_LOGIC_VECTOR (N-1 downto 0);
INICIO: IN STD_LOGIC;
     FIN: OUT STD_LOGIC;
     CLK,RESET: IN STD_LOGIC;
     a1,a2: out std_logic_vector(N-1 downto 0);
     resultsuma:in std_logic_vector (N-1 downto 0)
);
END prngC8_1;
architecture opcionA_1 of prngC8_1 is
--DECLARACION DE SEÑALES
   SIGNAL z,x1,a_z,a_x1: std_logic_Vector((N-1) DOWNTO 0);--z en el reset obtendrá el valor de x0
   SIGNAL contador_i: INTEGER RANGE 0 TO 64;
   SIGNAL start:STD_LOGIC;

   TYPE ESTADO IS (Reposo,IterandoZ,iterandoy,final);
   SIGNAL ACTUAL,SIGUIENTE:ESTADO;

   BEGIN

-- PROCESO PARA LA MAQUINA DE ESTADOS
      PROCESS(CLK,RESET)
               BEGIN
                  IF reset='1' THEN
                     actual<=reposo;
                   elsif clk 'EVENT AND clk='1' THEN
                      actual<=siguiente;
                   END IF;
                END PROCESS;

--MAQUINA DE ESTADOS-------------------------------------------------------------------------

PROCESS (ACTUAL,Inicio,contador_i,z,x1,seed)
```

```
 variable aux: STD_LOGIC_VECTOR (N-1 downto 0);
  BEGIN
     a_z<=z;-- el registro mantiene estado...
     a_x1<=x1;
        start<='0';
    CASE ACTUAL IS

       WHEN Reposo=>
          if inicio ='1' then
            a_z<=seed;-- valores semillas entrantes, si siempre son los mismos comentar...
            a_x1<=x"00000021";--valores semillas precalculados
              siguiente<=iterandoz;
          else
             siguiente<=reposo;
          end if;

       WHEN IterandoZ=>
            start<='1';
            aux:=z+x"00000022";---constante 22
            a_z<=(z(n-2 downto 0)& z(n-1))+(aux(0)& aux(n-1 downto 1));
            IF contador_i=31 THEN
              start<='1';
                 siguiente<=iterandoy;
            ELSE
                 siguiente<=IterandoZ;
            END IF;
       WHEN iterandoy=>
            aux:=(z(n-2 downto 0)& z(n-1))+ (z(0)& z(n-1 downto 1))+z+ x1;
            a_z<=aux XOR z;
            siguiente<=final;
        WHEN final=>
                 siguiente<=reposo;
       END CASE;
  END PROCESS;


 --CONTADOR
PROCESS(reset,clk)
  BEGIN
    IF reset='1' THEN
      contador_i <= 0;

   elsif clk'EVENT AND CLK='1' THEN

     if start='1' then

         contador_i<=contador_i+1;
    else
        contador_i<=0;

     end if;
   end if;
end process;

fin<='1' When(actual=final) ELSE '0';
z0<=z((n/2)-1 downto 0);
```

```
 PROCESS(reset,clk)
   BEGIN
     IF reset='1' THEN
        z <= x"00000023";--valores semillas precalculados
        x1 <= x"00000032"; --valores semillas precalculados
    ELSIF clk'EVENT AND CLK='1' THEN
        z<=a_z;
        x1<=a_x1;
   END IF;
 END PROCESS;
END opcionA_1;
```

-- GENERADOR PRNG-2

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
   --ENTIDAD

ENTITY prngC8_2 IS
GENERIC(N:INTEGER:=32);
PORT(

z0: OUT STD_LOGIC_VECTOR(((n/2)-1) DOWNTO 0);
seed: IN STD_LOGIC_VECTOR (N-1 downto 0);
INICIO: IN STD_LOGIC;
FIN: OUT STD_LOGIC;
CLK,RESET: IN STD_LOGIC;
a1,a2: out std_logic_vector(N-1 downto 0);
resultsuma:in std_logic_vector (N-1 downto 0)
);
END prngC8_2;

--ARQUITECTURA

architecture opcionc8_2 of prngC8_2 is

       ---Sumador
   --      component sumador_2
   --        Port(termino1 :IN std_logic_Vector(N-1 DOWNTO 0);termino2 :IN std_logic_Vector(N-1 DOWNTO
0);salida: out std_logic_Vector(N-1 DOWNTO 0));
   -- END component;

--DECLARACION DE SEÑALES
    SIGNAL z,x1,a_z,a_x1,a_aux,aux: std_logic_Vector((N-1) DOWNTO 0);--z en el reset obtendrá el valor de x0
    SIGNAL contador_i: INTEGER RANGE 0 TO 31;
    SIGNAL start:STD_LOGIC_vector (1 downto 0);

    TYPE ESTADO IS (Reposo,Iterandoz1,final,Iterandoz2,Iterandoy,iterandoz0,iterandoy2,iterandoy3,iterandoy4);
    SIGNAL ACTUAL,SIGUIENTE:ESTADO;

    BEGIN
----MAP------------
   --    sum:sumador_2 PORT MAP(a1,a2,resultsuma);
-- PROCESO PARA LA MAQUINA DE ESTADOS
```

```
      PROCESS(CLK,RESET)
            BEGIN

                IF reset='1' THEN
                   actual<=reposo;

                 elsif clk 'EVENT AND clk='1' THEN
                    actual<=siguiente;

                 END IF;
               END PROCESS;



--MAQUINA DE ESTADOS-------------------------------------------------------------------------------

PROCESS (ACTUAL,Inicio,contador_i,z,x1,resultsuma,aux)
  BEGIN
     a_z<=z;-- el registro mantiene estado...
     a_x1<=x1;
     a_aux<=aux;
                    a1<=(others=>'0');
                    a2<=(others=>'0');
       start<="00";
    CASE ACTUAL IS

       WHEN Reposo=>
          if inicio ='1' then
            a_z<=seed;-- asignamos a z X0
            a_x1<=x"00000021";

             siguiente<=iterandoz0;
          else
             siguiente<=reposo;
          end if;
        WHEN Iterandoz0=>
          start<="11";
          a1<=z;
          a2<=x"00000022";--constante...
          a_aux<=resultsuma;
          siguiente<=iterandoz1;
       WHEN Iterandoz1=>
         a_aux<=resultsuma;
         a1<=(aux(0)& aux(n-1 downto 1));
         a2<=(z(n-2 downto 0)& z(n-1));
         start<="11";
         siguiente<=Iterandoz2;
        a_z<=resultsuma;
        WHEN Iterandoz2=>
           start<="01";
           a1<=aux;
           a2<=x"00000022";
           a_aux<=resultsuma;
           IF contador_i=31 THEN
             start<="00";
               siguiente<=iterandoy;
```

```
            ELSE
                siguiente<=Iterandoz1;
            END IF;
           WHEN iterandoy=>
            a1<=z(0)&z(n-1 downto 1);
            a2<=z(n-2 downto 0)& z(n-1);
            a_aux<=resultsuma;
            siguiente<=Iterandoy2;
           WHEN iterandoy2=>
            a1<=aux;
            a2<=x1;
            a_aux<=resultsuma;
            siguiente<=Iterandoy3;
          WHEN iterandoy3=>
            a1<=aux;
            a2<=z;
            a_aux<=resultsuma;
            siguiente<=Iterandoy4;
          WHEN iterandoy4=>
            a_z<=aux XOR z;
            siguiente<=final;
        WHEN final=>
              siguiente<=reposo;
        END CASE;
   END PROCESS;


  --CONTADOR
PROCESS(reset,clk)
  BEGIN
    IF reset='1' THEN
      contador_i <= 0;

    elsif clk'EVENT AND CLK='1' THEN

      if start="01" then

          contador_i<=contador_i+1;
      elsif start="11" then
        contador_i<=contador_i;
      else
          contador_i<=0;

      end if;
    end if;
end process;
 fin<='1' When(actual=final) ELSE '0';
 z0<=z((n/2)-1 downto 0);
 PROCESS(reset,clk)
  BEGIN
    IF reset='1' THEN
      z <= x"00000034";-- asignación de X0 inicial
      x1 <= x"00000021";
      aux<=(others=>'0');
    ELSIF clk'EVENT AND CLK='1' THEN
      z<=a_z;
```

```
        x1<=a_x1;
        aux<=a_aux;
   END IF;
 END PROCESS;
END opcionc8_2;
```

-- GENERADOR PRNG-3

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
   --ENTIDAD

ENTITY prngC8_3 IS
GENERIC(N:INTEGER:=32);
PORT(

z0: OUT STD_LOGIC_VECTOR(((n/2)-1) DOWNTO 0);
seed: IN STD_LOGIC_VECTOR (N-1 downto 0);
INICIO: IN STD_LOGIC;
     FIN: OUT STD_LOGIC;
     CLK,RESET: IN STD_LOGIC


);
end prngC8_3;

--ARQUITECTURA

architecture opcionc8_3 of prngC8_3 is
--Sumador
        component sumador_3
     Port(carryin: IN std_logic; termino1 :IN std_logic_Vector((N/2)-1 DOWNTO 0);termino2 :IN
std_logic_Vector((N/2)-1 DOWNTO 0);carryout: OUT std_logic; salida: out std_logic_Vector((N/2)-1 DOWNTO
0));

END component;

--DECLARACION DE SEÑALES
   SIGNAL z,x1,a_z,a_x1: std_logic_Vector((N-1) DOWNTO 0);--z en el reset obtendrá el valor de x0
   SIGNAL a11,a22,resultsuma1,a_aux1,aux1,a_aux2,aux2:std_logic_Vector((N/2)-1 DOWNTO 0);
   SIGNAL contador_i: INTEGER RANGE 0 TO 31;
   SIGNAL start:STD_LOGIC_vector (1 downto 0);
   SIGNAL cin,cout,bitt,a_bitt,a_aux, aux: std_logic;
   TYPE                                                ESTADO                                                IS
(Reposo,Iterandoz1,final,Iterandoz2,Iterandoz3,iterandoz0,iterandoz4,iterandoy,iterandoy1,iterandoy2,iterandoy3,ite
randoy4,iterandoy5,iterandoy6);
   SIGNAL ACTUAL,SIGUIENTE:ESTADO;

   BEGIN
----MAP-------------
      sum:sumador_3 PORT MAP(cin,a11,a22,cout,resultsuma1);
-- PROCESO PARA LA MAQUINA DE ESTADOS

       PROCESS(CLK,RESET)
              BEGIN
```

```
                IF reset='1' THEN
                   actual<=reposo;

                 elsif clk 'EVENT AND clk='1' THEN
                    actual<=siguiente;

                 END IF;
               END PROCESS;


--MAQUINA DE ESTADOS-------------------------------------------------------------------------------

PROCESS (ACTUAL,Inicio,contador_i,z,x1,resultsuma1,aux1,aux2,aux,cout,bitt)
  BEGIN
     a_z<=z;-- el registro mantiene estado...
     a_x1<=x1;
     a_aux1<=aux1;
     a_aux2<=aux2;
     a_aux<=aux;
     cin<=bitt;
     a_bitt<=bitt;
                    a11<=(others=>'0');
                    a22<=(others=>'0');
       start<="00";
    CASE ACTUAL IS

       WHEN Reposo=>
          if inicio ='1' then
            a_z<=seed;-- asignamos a z el valor semilla
            a_bitt<='0';
              siguiente<=iterandoz0;
          else
             siguiente<=reposo;
          end if;
        WHEN Iterandoz0=>
          start<="11";
          a_bitt<=cout;
          a11<=z((n/2-1) downto 0);
          a22<=x"0002";
          a_aux1<=resultsuma1;
          siguiente<=iterandoz1;
       WHEN Iterandoz1=>
         a_bitt<='0';
         a_aux2<=resultsuma1;
         a11<=x"0002";
         a22<=z(n-1 downto (n/2));
         start<="11";
         siguiente<=Iterandoz2;
       WHEN Iterandoz2=>
         a_aux<=aux1(0);
           start<="11";
         a_bitt<=cout;
         a11<=aux2(0)& aux1(N/2-1 downto 1);
         a22<=z((n/2)-2 downto 0)& z(n-1);
```

```
      a_aux1<=resultsuma1;
       siguiente<=Iterandoz3;
    WHEN Iterandoz3=>
     a_bitt<='0';
     a_aux2<=resultsuma1;
     a11<=aux & aux2(N/2-1 downto 1);
     a22<=z(n-2 downto (n/2)-1);
     start<="11";

     siguiente<=Iterandoz4;
     WHEN Iterandoz4=>
      start<="01";
     a_z<=aux2 & aux1;

      a_bitt<=cout;
      a11<=aux1;
      a22<=x"0002";
      a_aux1<=resultsuma1;

        IF contador_i=31 THEN
          start<="00";
          a_bitt<='0';
             siguiente<=iterandoy;
        ELSE
        a_bitt<=cout;
             siguiente<=Iterandoz1;
        END IF;
    WHEN Iterandoy=>
      a_bitt<=cout;
      a11<=z((n/2-1) downto 0);
      a22<=x1((n/2-1) downto 0);
      a_aux1<=resultsuma1;
      siguiente<=iterandoy1;
   WHEN Iterandoy1=>
     a_bitt<='0';
     a_aux2<=resultsuma1;
     a11<=z(n-1 downto n/2);
     a22<=x1(n-1 downto n/2);
     start<="11";
     siguiente<=Iterandoy2;
   WHEN Iterandoy2=>
       a_bitt<=cout;
       a11<=aux1;
       a22<=z((n/2-2) downto 0)& z(n-1);
       a_aux1<=resultsuma1;
       siguiente<=iterandoy3;
  WHEN Iterandoy3=>
     a_bitt<='0';
     a_aux2<=resultsuma1;
     a11<=z(n-2 downto (n/2-1));
     a22<=aux2;
     start<="11";
     siguiente<=Iterandoy4;
  WHEN Iterandoy4=>
       a_bitt<=cout;
       a11<=aux1;
```

```
            a22<=z((n/2) downto 1);
            a_aux1<=resultsuma1;
            siguiente<=iterandoy5;
        WHEN Iterandoy5=>
            a_bitt<='0';
            a_aux2<=resultsuma1;
            a11<=z(0)& z(n-1 downto (n/2+1));
            a22<=aux2;
            start<="11";
            siguiente<=Iterandoy6;
          WHEN Iterandoy6=>
            a_z<= z XOR (aux2 & aux1);
            siguiente<=final;

          WHEN final=>
                siguiente<=reposo;
          END CASE;
    END PROCESS;

  --CONTADOR
PROCESS(reset,clk)
   BEGIN
      IF reset='1' THEN
        contador_i <= 0;

     elsif clk'EVENT AND CLK='1' THEN

       if start="01" then

           contador_i<=contador_i+1;
       elsif start="11" then
          contador_i<=contador_i;
       else
            contador_i<=0;

       end if;
     end if;
end process;
 fin<='1' When(actual=final) ELSE '0';
 z0<=z((n/2)-1 downto 0);
 PROCESS(reset,clk)
   BEGIN
      IF reset='1' THEN
        z <= x"00000034";-- asignación de X0 inicial
        x1 <= x"00000021";
        bitt<='0';
        aux<='0';
        aux1<=(others=>'0');
        aux2<=(others=>'0');
     ELSIF clk'EVENT AND CLK='1' THEN
        z<=a_z;
        x1<=a_x1;
        bitt<=a_bitt;
        aux<=a_aux;
        aux1<=a_aux1;
        aux2<=a_aux2;
```

```
   END IF;
 END PROCESS;
END opcionc8_3;
```

-- GENERADOR PRNG-4

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
   --ENTIDAD

ENTITY prngC8_4 IS
GENERIC(N:INTEGER:=32);
PORT(

z0: OUT STD_LOGIC_VECTOR(((n/2)-1) DOWNTO 0);
seed: IN STD_LOGIC_VECTOR (N-1 downto 0);
INICIO: IN STD_LOGIC;
FIN: OUT STD_LOGIC;
CLK,RESET: IN STD_LOGIC
);
END prngC8_4;

--ARQUITECTURA

architecture opcionc8_4 of prngC8_4 is
---Sumador
        component sumador_4
      Port(carryin: IN std_logic; termino1 :IN std_logic_Vector((N/4)-1 DOWNTO 0);termino2 :IN
std_logic_Vector((N/4)-1 DOWNTO 0);carryout: OUT std_logic; salida: out std_logic_Vector((N/4)-1 DOWNTO
0));
   END component;

--DECLARACION DE SEÑALES
   SIGNAL z,x1,a_z,a_x1: std_logic_Vector((N-1) DOWNTO 0);--z en el reset obtendrá el valor de x0
   SIGNAL          a1,a2,resultsuma,a_aux1,aux1,a_aux2,aux2,a_aux3,aux3,a_aux4,aux4:std_logic_Vector((N/4)-1
DOWNTO 0);
   SIGNAL contador_i: INTEGER RANGE 0 TO 31;
   SIGNAL start:STD_LOGIC_vector (1 downto 0);
   SIGNAL cin,cout,bitt,a_bitt,bitaux,a_bitaux: std_logic;
   TYPE                                    ESTADO                                    IS
(Reposo,Iterandoz1,final,Iterandoz2,Iterandoz3,iterandoz4,iterandoz5,iterandoz6,iterandoz7,iterandoz8,iterandoz0,it
erandoy1,iterandoy2,iterandoy3,iterandoy4,iterandoy5,iterandoy6,iterandoy7,iterandoy8,iterandoy9,iterandoy10,iter
andoy11,iterandoy12,iterandoy13);
   SIGNAL ACTUAL,SIGUIENTE:ESTADO;

   BEGIN
----MAP-------------
     sum:sumador_4 PORT MAP(cin,a1,a2,cout,resultsuma);
-- PROCESO PARA LA MAQUINA DE ESTADOS

      PROCESS(CLK,RESET)
            BEGIN

               IF reset='1' THEN
                  actual<=reposo;
```

```
                elsif clk 'EVENT AND clk='1' THEN
                   actual<=siguiente;

                END IF;
              END PROCESS;



--MAQUINA DE ESTADOS-------------------------------------------------------------------------

PROCESS (ACTUAL,Inicio,contador_i,z,x1,resultsuma,aux1,aux2,aux3,aux4,cout,bitt,bitaux)
  BEGIN
    a_z<=z;-- el registro mantiene estado...
    a_x1<=x1;
    a_aux1<=aux1;
    a_aux2<=aux2;
    a_aux3<=aux3;
    a_aux4<=aux4;
    a_bitaux<=bitaux;
    cin<=bitt;
    a_bitt<=bitt;
                    a1<=(others=>'0');
                    a2<=(others=>'0');
      start<="00";
    CASE ACTUAL IS

      WHEN Reposo=>
        if inicio ='1' then
          a_z<=seed;-- asignamos a z X0
          a_bitt<='0';
            siguiente<=iterandoz0;
        else
           siguiente<=reposo;
        end if;
       WHEN Iterandoz0=>
         start<="11";
         a_bitt<=cout;
         a1<=x"10";--constante
         a2<=z((n/4)-1 downto 0);
         a_aux1<=resultsuma;
         a_bitaux<=resultsuma(0);
         siguiente<=iterandoz1;
      WHEN Iterandoz1=>
        a_bitt<=cout;
        a_aux2<=resultsuma;
        a1<=x"00";
        a2<=z((n/2)-1 downto (n/4));
        start<="11";
        siguiente<=Iterandoz2;
      WHEN Iterandoz2=>
          start<="11";
        a_bitt<=cout;
        a1<=x"10";
        a2<=z((n/2)+(N/4)-1 downto (n/2));
        a_aux3<=resultsuma;
         siguiente<=Iterandoz3;
```

```
WHEN Iterandoz3=>
 a_bitt<='0';
 a_aux4<=resultsuma;
 a1<=x"00";
 a2<= z(n-1 downto ((n/2)+(N/4)));
 start<="11";
 siguiente<=Iterandoz4;

WHEN Iterandoz4=>
 a_bitt<=cout;
 a_aux1<=resultsuma;
 a1<= aux2(0)& aux1((n/4)-1 downto 1);
 a2<= z((n/4)-2 downto 0)& z(n-1);
 start<="11";
 siguiente<=Iterandoz5;

WHEN Iterandoz5=>
 a_bitt<=cout;
 a_aux2<=resultsuma;
 a1<= aux3(0)& aux2((n/4)-1 downto 1);
 a2<= z((n/2)-2 downto (n/4)-1);
 start<="11";
 siguiente<=Iterandoz6;

WHEN Iterandoz6=>
 a_bitt<=cout;
 a_aux3<=resultsuma;
 a1<= aux4(0)& aux3((n/4)-1 downto 1);
 a2<= z((n/2)+(n/4)-2 downto (n/2)-1);
 start<="11";
 siguiente<=Iterandoz7;

WHEN Iterandoz7=>
 a_bitt<='0';
 a_aux4<=resultsuma;
 a1<= bitaux & aux4((n/4)-1 downto 1);
 a2<= z(n-2 downto (n/2)+(n/4)-1);
 start<="11";
 siguiente<=Iterandoz8;

 WHEN Iterandoz8=>
   start<="01";
   a_z<= aux4 & aux3 & aux2 & aux1;
   a_bitaux<=resultsuma(0);
   a1<=x"10";--constante
   a2<=aux1;
   a_aux1<=resultsuma;

     IF contador_i=31 THEN
       start<="00";
       a_bitt<='0';
          siguiente<=Iterandoy1;
     ELSE
     a_bitt<=cout;
     siguiente<=iterandoz1;
```

```
      END IF;

    WHEN iterandoy1=>
    start<="11";
    a_bitt<=cout;
    a1<=x1((n/4)-1 downto 0);
    a2<=z((n/4)-1 downto 0);
    a_aux1<=resultsuma;
    siguiente<=iterandoy2;

   WHEN Iterandoy2=>
    a_bitt<=cout;
    a_aux2<=resultsuma;
    a1<=x1(n/2-1 downto (n/4));
    a2<=z(n/2-1 downto (n/4));
    start<="11";
    siguiente<=Iterandoy3;

  WHEN Iterandoy3=>
      start<="11";
    a_bitt<=cout;
    a1<=x1((n/2)+(N/4)-1 downto (n/2));
    a2<=z((n/2)+(N/4)-1 downto (n/2));
    a_aux3<=resultsuma;
    siguiente<=Iterandoy4;

  WHEN Iterandoy4=>
   a_bitt<='0';
   start<="11";
   a_aux4<=resultsuma;
   a1<=x1(n-1 downto ((n/2)+(N/4)));
   a2<= z(n-1 downto ((n/2)+(N/4)));
   start<="11";
   siguiente<=Iterandoy5;

   WHEN Iterandoy5=>
    start<="11";
    a_bitt<=cout;
    a1<=aux1;
    a2<=z((n/4)-2 downto 0)& z(n-1);
    a_aux1<=resultsuma;
     siguiente<=Iterandoy6;

   WHEN Iterandoy6=>
    a_bitt<=cout;
    a_aux2<=resultsuma;
    a1<=aux2;
    a2<=z((n/2)-2 downto (n/4)-1);
    start<="11";
    siguiente<=Iterandoy7;

   WHEN Iterandoy7=>
    a_bitt<=cout;
    a_aux3<=resultsuma;
    a1<=aux3;
    a2<=z((n/2)+(n/4)-2 downto (n/2)-1);
```

```
            start<="11";
            siguiente<=Iterandoy8;

          WHEN Iterandoy8=>
            a_bitt<='0';
            a_aux4<=resultsuma;
            a1<=aux4;
            a2<=z(n-2 downto (n/2)+(n/4)-1);
            start<="11";
            siguiente<=Iterandoy9;
          WHEN Iterandoy9=>
            a_bitt<=cout;
            a_aux1<=resultsuma;
            a1<=aux1;
            a2<=z((n/4) downto 1);
            start<="11";
            siguiente<=Iterandoy10;
          WHEN Iterandoy10=>
            a_bitt<=cout;
            a_aux2<=resultsuma;
            a1<=aux2;
            a2<=z((n/2) downto (n/4)+1);
            start<="11";
            siguiente<=Iterandoy11;
          WHEN Iterandoy11=>
            a_bitt<=cout;
            a_aux3<=resultsuma;
            a1<=aux3;
            a2<=z((n/2)+(n/4) downto (n/2)+1);
            start<="11";
            siguiente<=Iterandoy12;
          WHEN Iterandoy12=>
            a_bitt<='0';
            a_aux4<=resultsuma;
            a1<=aux4;
            a2<=z(0)& z(n-1 downto (n/2)+(n/4)+1);
            start<="11";
            siguiente<=Iterandoy13;

           WHEN Iterandoy13=>
            start<="01";
            a_z<=z XOR (aux4 & aux3 & aux2 & aux1);

              siguiente<=final;

          WHEN final=>
                siguiente<=reposo;
          END CASE;
    END PROCESS;


  --CONTADOR
PROCESS(reset,clk)
   BEGIN
      IF reset='1' THEN
        contador_i <= 0;
```

```
    elsif clk'EVENT AND CLK='1' THEN

      if start="01" then

          contador_i<=contador_i+1;
      elsif start="11" then
        contador_i<=contador_i;
      else
           contador_i<=0;

      end if;
    end if;
end process;
 fin<='1' When(actual=final) ELSE '0';
 z0<=z((n/2)-1 downto 0);
 PROCESS(reset,clk)
   BEGIN
     IF reset='1' THEN
        z <= x"00000023";-- asignación de X0 inicial
        x1 <= x"00000021";
        bitt<='0';
        bitaux<='0';
        aux1<=(others=>'0');
        aux2<=(others=>'0');
        aux3<=(others=>'0');
        aux4<=(others=>'0');
    ELSIF clk'EVENT AND CLK='1' THEN
        z<=a_z;
        x1<=a_x1;
        bitt<=a_bitt;
        bitaux<=a_bitaux;
        aux1<=a_aux1;
        aux2<=a_aux2;
        aux3<=a_aux3;
        aux4<=a_aux4;
   END IF;
 END PROCESS;
END opcionc8_4;


Xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

---PROTOCOLO HMAC

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.numeric_std.all;

entity protocolo_hmac3 is
   Port ( clk : in STD_LOGIC;
   reset : in STD_LOGIC;
   recibido_lector : in STD_LOGIC;
   Dato_lector: in STD_LOGIC_VECTOR (31 DOWNTO 0);
   Dato_tag: out STD_LOGIC_VECTOR(31 DOWNTO 0);
   enviado_lector: out STD_LOGIC;
   OK: out STD_LOGIC;
   NO_OK: OUT STD_LOGIC);
end protocolo_hmac3;

architecture protocolo_hmac_a3 of protocolo_hmac3 is

    Signal Resultado_Alu,Dato_A,Dato_B,
      dato_hash,h1_hash,h2_hash: std_logic_vector(31 downto 0);
    Signal inicio_hash,fin_hash,inicio_xor,
    inicio_suma,fin_alu: std_logic;


Component Maquina_estados3
   Port ( clk : in STD_LOGIC;
   reset : in STD_LOGIC;
   recibido_lector : in STD_LOGIC;
   Dato_lector: in STD_LOGIC_VECTOR (31 DOWNTO 0);
   Dato_tag: out STD_LOGIC_VECTOR(31 DOWNTO 0);
   enviado_lector: out STD_LOGIC;
   OK: out STD_LOGIC;
   NO_OK: OUT STD_LOGIC;
   fin_hash:in std_logic;
   inicio_hash:out std_logic;
   Dato_hash:in std_logic_vector (31 Downto 0);
   h1_hash:out STD_LOGIC_VECTOR (31 DOWNTO 0);
   h2_hash:out STD_LOGIC_VECTOR (31 DOWNTO 0));
   end component;
```

```
  Component hash3
    -- Generic (n: integer:=32);
     Port ( clk : in STD_LOGIC;
     reset : in STD_LOGIC;
     inicio_hash : in STD_LOGIC;
     fin_hash: out STD_LOGIC;
     inicio_suma: out std_logic;
     inicio_xor: out std_logic;

     Nr: in STD_LOGIC_VECTOR (31 DOWNTO 0);
     Nt: in STD_LOGIC_VECTOR(31 DOWNTO 0);
     A: out STD_LOGIC_VECTOR(31 DOWNTO 0);
     B: out STD_LOGIC_VECTOR(31 DOWNTO 0);
     Resultado_alu:in STD_LOGIC_VECTOR(31 DOWNTO 0);
     a0: out STD_LOGIC_VECTOR(31 DOWNTO 0));
   end component;

  Component Alu3
    --GENERIC(N:INTEGER:=32);
    PORT( Resultado:OUT std_logic_vector(31 DOWNTO 0);
    INICIO_suma: IN STD_LOGIC;
    DATO_A: IN STD_LOGIC_VECTOR(31 DOWNTO 0);
    DATO_B: IN STD_LOGIC_VECTOR(31 DOWNTO 0);
    Inicio_Xor:in std_LOGIC;

    CLK,RESET: IN STD_LOGIC);
  end component;

  begin
     mi_maquina: maquina_estados3 Port map (clk,reset,recibido_lector,
     dato_lector,dato_tag,enviado_lector,ok,no_ok,
     fin_hash,inicio_hash,dato_hash,h1_hash,h2_hash);


     mi_hash: hash3 port map (clk,reset,inicio_hash,fin_hash,inicio_suma,
     inicio_xor,h1_hash,
     h2_hash,Dato_A,Dato_B,Resultado_Alu,dato_hash);

     mi_Alu: Alu3 port map (Resultado_Alu,inicio_Suma,Dato_A,Dato_B,inicio_XOR,
     clk,reset);
end protocolo_hmac_a3;
```

-- Alu para realizar la suma y XOR

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
use ieee.numeric_std.all;

--ENTIDAD
ENTITY ALu3 IS
  GENERIC(N:INTEGER:=32);
  PORT( Resultado:OUT std_logic_vector(n-1 DOWNTO 0);
  INICIO_suma: IN STD_LOGIC; --ESTE VALOR SI ESTA A 0 SUMA SI ESTA A 1 XOR
  DATO_A: IN STD_LOGIC_VECTOR(N-1 DOWNTO 0);
```

```
  DATO_B: IN STD_LOGIC_VECTOR(N-1 DOWNTO 0);
  Inicio_Xor:in std_LOGIC;
  CLK,RESET: IN STD_LOGIC);
  END Alu3;

--ARQUITECTURA
architecture alu_a3 of alu3 is
--DECLARACION DE SEÑALES
BEGIN
-- PROCESO PARA LAS SEÑALES
PROCESS(CLK,RESET)
BEGIN
  if inicio_suma='0' then
     Resultado<=Dato_A+Dato_B;
   else
      Resultado<= Dato_A xor Dato_B;
END IF;
END PROCESS;
   end alu_a3;
```

Maquina de estados 3

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.numeric_std.all;

-- ENTIDAD
entity Maquina_estados3 is
   Generic (n: integer:=32);
  Port ( clk : in STD_LOGIC;
  reset : in STD_LOGIC;
  recibido_lector : in STD_LOGIC;
  Dato_lector: in STD_LOGIC_VECTOR (N-1 DOWNTO 0);
  Dato_tag: out STD_LOGIC_VECTOR(N-1 DOWNTO 0);
  enviado_lector: out STD_LOGIC;
  OK: out STD_LOGIC;
  NO_OK: OUT STD_LOGIC;
  fin_hash:in std_logic;
  inicio_hash:out Std_logic;
  Dato_hash:in std_logic_vector (N-1 Downto 0);
```

```
  h1_hash:out STD_LOGIC_VECTOR (N-1 DOWNTO 0);
  h2_hash:out STD_LOGIC_VECTOR (N-1 DOWNTO 0));
end Maquina_estados3;



--ARQUITECTURA
architecture Maquina_estados_a3 of Maquina_estados3 is

type estado is (espera,leer_Nr,generar_Hash1,enviar_Nt,enviar_Hash1,
leer_H2,comparar,comprobar, estado_ok, error,generar_Hash2);
signal actual,siguiente:estado;
signal ID: std_logic_vector (N-1 downto 0);

--signal registro1,aux_registro1: std_logic_vector (N-1 downto 0);

--PROCESO SECUENCIAL DE LA MAQUINA DE ESTADOS
begin
process(clk, reset)

begin
  if reset = '1' then
     actual<=espera;
  elsif clk'event and clk='1' then
     actual<=siguiente;
end if;
end process;

--PROCESO COMBINACIONAL DE LA MAQUINA DE ESTADOS
process(recibido_lector,actual,fin_hash)
variable registro2: std_logic_vector (N-1 downto 0);
variable registro1: std_logic_vector (N-1 downto 0);
   begin
      case actual IS

         when espera=>
            h1_hash<=conv_std_logic_vector(0,N);
```

```
    h2_hash<=conv_std_logic_vector(0,N);
    Dato_tag<=conv_std_logic_vector(0,N);

    enviado_lector<='0';
    ok<='0';
    no_ok<='0';
    inicio_hash<='0';
                    registro1:=conv_std_logic_vector(0,N);
                    registro2:=conv_std_logic_vector(0,N);
    if recibido_lector='1' then
       siguiente<=leer_Nr;
    else
       siguiente<=actual;
    end if;

when leer_Nr=>  -- EN ESTE ESTADO ADEMAS DE LEER_NR SE
              --ESPERA A CALCULA NUMERO ALEATORIO

    h1_hash<=conv_std_logic_vector(0,N);
    h2_hash<=conv_std_logic_vector(0,N);
    Dato_tag<=conv_std_logic_vector(0,N);

    enviado_lector<='0';
    ok<='0';
    no_ok<='0';
    inicio_hash<='0';
              registro1:=Dato_lector;
              registro2:=conv_std_logic_vector(4,N);

       siguiente<=generar_hash1;

        when generar_Hash1=>
    H1_hash<=registro1;
    H2_hash<=registro2;
    Dato_tag<=conv_std_logic_vector(0,N);
    inicio_hash<='1';
```

```
    enviado_lector<='0';
    ok<='0';
    no_ok<='0';
                    Registro1:=Dato_hash;
                    registro2:=conv_std_logic_vector(4,N);
    if fin_hash='1' then
       siguiente<=enviar_hash1;
    else
       siguiente<=generar_hash1;
    end if;

when enviar_hash1=>
    h1_hash<=conv_std_logic_vector(0,N);
    h2_hash<=conv_std_logic_vector(0,N);
    Dato_tag<=conv_std_logic_vector(0,N);
    Dato_tag<=registro1;
    inicio_hash<='0';

    enviado_lector<='1';
    ok<='0';
    no_ok<='0';
                    registro1:=Dato_hash;
                    registro2:=conv_std_logic_vector(4,N);
    siguiente<=enviar_Nt;

when enviar_Nt=>
    h1_hash<=conv_std_logic_vector(0,N);
    h2_hash<=conv_std_logic_vector(0,N);
    Dato_tag<=registro2;
    inicio_hash<='0';

    enviado_lector<='1';
    ok<='0';
    no_ok<='0';
                    registro1:=Dato_hash;
```

```
                    registro2:=conv_std_logic_vector(4,N);
    if recibido_lector='1' then
       siguiente<=leer_h2;
    else
       siguiente<=actual;
    end if;

when leer_h2=>
    h1_hash<=conv_std_logic_vector(0,N);
    h2_hash<=conv_std_logic_vector(0,N);
    Dato_tag<=conv_std_logic_vector(0,N);
    inicio_hash<='1';

    enviado_lector<='0';
    ok<='0';
    no_ok<='0';
                    registro1:=Dato_lector;
   registro2:=conv_std_logic_vector(4,N);
    siguiente<=Generar_hash2;

 when Generar_hash2=>        --Volver a calcular valor de hash
    H1_hash<=registro1;
    H2_hash<=registro2;
    Dato_tag<=conv_std_logic_vector(0,N);
    inicio_hash<='1';

    enviado_lector<='0';
    ok<='0';
    no_ok<='0';
                    registro1:=Dato_lector;
                    registro2:=dato_hash;
    if fin_hash='1' then
    siguiente<=comparar;
    else
    siguiente<=Generar_hash2;
end if;
```

```
when comparar=>
   h1_hash<=conv_std_logic_vector(0,N);
   h2_hash<=conv_std_logic_vector(0,N);
   Dato_tag<=conv_std_logic_vector(0,N);
   inicio_hash<='0';

   enviado_lector<='0';
   ok<='0';
   no_ok<='0';
                  registro1:=Dato_lector;
                  registro2:=dato_hash;

   siguiente<=comprobar;

when comprobar=>
   h1_hash<=conv_std_logic_vector(0,N);
   h2_hash<=conv_std_logic_vector(0,N);
   Dato_tag<=conv_std_logic_vector(0,N);
   inicio_hash<='0';

   enviado_lector<='0';
   ok<='0';
   no_ok<='0';
   registro1:=Dato_lector;
                   registro2:=dato_hash;


  if registro1=registro2 then
     siguiente<=estado_ok;
   else
      siguiente<=error;
   end if;

when estado_ok=>
   h1_hash<=conv_std_logic_vector(0,N);
```

```
            h2_hash<=conv_std_logic_vector(0,N);
            Dato_tag<=conv_std_logic_vector(0,N);
            inicio_hash<='0';

            enviado_lector<='0';
            ok<='1';
            no_ok<='0';
                        registro1:=conv_std_logic_vector(0,N);
                        registro2:=conv_std_logic_vector(4,N);

            siguiente<=espera;

          when error=>
            h1_hash<=conv_std_logic_vector(0,N);
            h2_hash<=conv_std_logic_vector(0,N);
            Dato_tag<=conv_std_logic_vector(0,N);
            inicio_hash<='0';

            enviado_lector<='0';
            ok<='0';
            no_ok<='1';
                        registro1:=conv_std_logic_vector(0,N);
                        registro2:=conv_std_logic_vector(4,N);

            siguiente<=espera;
          end case;
       end process;

end maquina_estados_a3;
```

Funcion_hash

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.numeric_std.all;

-- ENTIDAD
```

```
entity hash3 is
  Generic (n: integer:=32);
   Port ( clk : in STD_LOGIC;
   reset : in STD_LOGIC;
   inicio_hash : in STD_LOGIC;
   fin_hash: out STD_LOGIC;
   inicio_suma: out std_logic;
   inicio_xor: out std_logic;
   fin_alu: in std_logic;
   Nr: in STD_LOGIC_VECTOR (N-1 DOWNTO 0);
   Nt: in STD_LOGIC_VECTOR(N-1 DOWNTO 0);
   A: out STD_LOGIC_VECTOR(N-1 DOWNTO 0);
   B: out STD_LOGIC_VECTOR(N-1 DOWNTO 0);
   Resultado_alu:in STD_LOGIC_VECTOR(N-1 DOWNTO 0);
   a0: out STD_LOGIC_VECTOR(N-1 DOWNTO 0));
end hash3;

architecture hash_a3 of hash3 is

    signal r1,i,j: integer range 0 to 32;
    signal nstate: integer range 0 to 5;
    signal r2:integer range 0 to 8;
    signal h3,h3_1
    ,h4: std_logic_vector(N-1 Downto 0);
    signal h3_2: std_logic_vector(127 downto 0);
    TYPE ESTADO IS (inicio,Funcion_A1,Funcion_A2,Funcion_B, Inicio_CD,Funcion_C1,
    Funcion_C2,Funcion_C3,Funcion_C4,Funcion_C5,Funcion_C6,
    Funcion_D1,Funcion_D2,Funcion_D3,Funcion_D4,
    Funcion_D5,Funcion_D6,Funcion_D7,Funcion_Final,Suma_Funcion_Final,
    Suma_funcion_A,Suma_funcion_A2,
    Suma_funcion_B,Suma_funcion_B2,Suma_funcion_B3,Suma_funcion_B4,
    Suma_funcion_C1,XOR_funcion_C2,Suma_Funcion_C3,Suma_Funcion_C4,
    XOR_Funcion_C5,Suma_Funcion_C6,
    Xor_Funcion_D1,Xor_Funcion_D2,Suma_Funcion_d3,Suma_Funcion_D4,
    Suma_Funcion_D5,
    Mod_estado0,estado0,Xor_estado0,resultado_estado0,
    Mod_estado1,estado1,Xor_estado1,resultado_estado1,
    Mod_estado2,estado2,Xor_estado2,resultado_estado2,
    Mod_estado3,estado3,Xor_estado3,resultado_estado3,
    inicio_2);
    SIGNAL ACTUAL,SIGUIENTE:ESTADO;
    signal start_i,start_j: std_logic_vector(1 downto 0);
    Begin
    -- Proceso secuencial maquina de estados
    PROCESS(CLK,RESET)
    BEGIN
    IF reset='1' THEN
       actual<=inicio;
       elsif clk 'EVENT AND clk='1' THEN
       actual<=siguiente;
     END IF;
     END PROCESS;

     -- Proceso Inicio
    PROCESS(CLK,RESET)
    BEGIN
```

```
IF reset='1' THEN
elsif clk 'EVENT AND clk='1' THEN
  if actual=Inicio then

   r1<=32;
   r2<=8;
   nstate<=4;


   Else
   end if;
 END IF;
 END PROCESS;

-- Proceso Contador_i (0,0) & (1,1) =>0
                    -- (01) =>contar
                    -- (10) =>parar
PROCESS(CLK,RESET)
BEGIN
IF reset='1' THEN
elsif clk 'EVENT AND clk='1' THEN
   if start_i="01" then
         i<=i+1;
     elsif start_i="10" then
        i<=i;
     else
          i<=0;
       end if;


 END IF;
 END PROCESS;

 -- Proceso Contador_j
PROCESS(CLK,RESET)
BEGIN
IF reset='1' THEN
elsif clk 'EVENT AND clk='1' THEN

   if start_j="01" then
        j<=j+1;
       elsif start_j="10" then
        j<=j;
          else
          j<=0;
      end if;


 END IF;
 END PROCESS;




  -- Proceso Funcion_A
```

```
  PROCESS(CLK,RESET)
  BEGIN
  IF reset='1' THEN
--      h0 <=(others=>'1');
  elsif clk 'EVENT AND clk='1' THEN
     if actual=Inicio_2 then
   --      h0<=Nr;

      elsif Actual=Funcion_A2 then
       --      h0<=Resultado_Alu;
         end if;

   END IF;
   END PROCESS;

     -- Proceso Funcion_B
  PROCESS(CLK,RESET)
  BEGIN
  IF reset='1' THEN
  --    h1 <=(others=>'0');
  elsif clk 'EVENT AND clk='1' THEN
     if actual=Inicio_2 then
     --    h1<=Nr;

     elsif Actual=Suma_Funcion_B3 then
        H1<=Resultado_Alu;
     elsif Actual=Funcion_B then
          --h1<=('0' & h1(N-1 downto 1)) + (h1(N-2 downto 0) & '0')+h1+Nt;
       --    h1<=Resultado_Alu;
         end if;

   END IF;
   END PROCESS;

  -- FUNCONES C Y D
  PROCESS(CLK,RESET)
  BEGIN
  IF reset='1' THEN
      h3 <=(others=>'0');
      h4<=(others=>'0');
      h3_1<=(others=>'0');
      h3_2<=(others=>'0');
     -- h3_3<=(others=>'0');

  elsif clk 'EVENT AND clk='1' THEN
     if actual=Inicio_CD then
       h3<=conv_std_logic_vector(3,N); --sustituir estas dos por las
       h4<=conv_std_logic_vector(4,N); --otras dos
        --  h3<=h0;
        --  h4<=h1;
    elsif Actual=Funcion_A1 then-- Estos dos valores valen para la funcion A
        -- h3_1<=(h0(N-2 downto 0) & '0');
         h3_1<=(Resultado_ALu);

  --  elsif Actual=Suma_funcion_B then     -- Aqui se van guardando los valores de F(B)
      --   h3_1<=('0' & h1(N-1 downto 1));
```

```
   --   h3_2<= (h1(N-2 downto 0) & '0');

  elsif Actual=Suma_funcion_B2 then
      h3_1<=Resultado_Alu;



  elsif Actual=Funcion_C1 then
        h3_1<=Resultado_Alu;
    --    h3<=h3+H4; -- en vez del 4 cambiar por h4;
 --  elsif Actual=Funcion_C2 then
       -- h3<="000" & h3(N-1 downto 3);
   elsif Actual=Funcion_C2 then
        h3<=Resultado_Alu;
   elsif Actual=Funcion_C3 then
      --  H3_1<=("00" & h3(N-1 downto 2))+h3;
       H3_1<=("00" & Resultado_Alu(N-1 downto 2));
      -- h3_2<=h3(N-4 downto 0) & "000";
      -- h3_3<=h3(N-2 downto 0) & '0';
    elsif Actual=Funcion_C4 then
         h3_1<=Resultado_Alu;
    --    h3<=("00" & h3_1(N-1 downto 2))+h3_2
    --    +(h3_3 XOR conv_std_logic_vector(125,N)); -- 15 es un valor cualquiera
   --     H4<=CONV_STD_LOGIC_VECTOR(27,N); --ELIMINAR
    elsif Actual=Funcion_C5 then
         h3<=Resultado_Alu;
    elsif Actual=Funcion_C6 then
         h3<=Resultado_Alu;
    elsif Actual=Funcion_D1 then
         h3_1<='0' & Resultado_Alu(N-1 downto 1);
     --   h3_1<=H4;
    --    H3<=CONV_STD_LOGIC_VECTOR(3,N); --ELIMINAR
      --  h4<=H3 XOR h4; --cambiarlo por h3
    elsif Actual=Funcion_D2 then
       --h4<='0' & h4(N-1 downto 1);
       h4<=Resultado_Alu;
    elsif Actual=Funcion_D3 then
       --h4<=h3_1 Xor h4;
       h3_1<=Resultado_Alu;
    elsif Actual=Funcion_D4 then
       h4<=Resultado_Alu;
       --h4<=("0000" & h4(N-1 downto 4)) + ("000" & h4(N-1 downto 3))
       --+ (h4(N-4 downto 0) & "000") + h4;
     elsif Actual=Funcion_D5 then
        h4<=Resultado_Alu;
     elsif Actual=Estado0 then
        h3_2(31 downto 0)<=Resultado_Alu;
     elsif Actual=Resultado_estado0 then
        h3_2(31 downto 0)<=Resultado_Alu;
    elsif Actual=Estado1 then
        h3_2(63 downto 32)<=Resultado_Alu;
     elsif Actual=Resultado_estado1 then
        h3_2(63 downto 32)<=Resultado_Alu;
     elsif Actual=Estado2 then
        h3_2(95 downto 64)<=Resultado_Alu;
     elsif Actual=Resultado_estado2 then
```

```
        h3_2(95 downto 64)<=Resultado_Alu;
      elsif Actual=Estado3 then
        h3_2(127 downto 96)<=Resultado_Alu;
      elsif Actual=Resultado_estado3 then
        h3_2(127 downto 96)<=Resultado_Alu;
       end if;

 END IF;
 END PROCESS;

-- Proceso de la funcion final
PROCESS(CLK,RESET)
BEGIN
IF reset='1' THEN
    a0 <=(others=>'0');
elsif clk 'EVENT AND clk='1' THEN
   if actual=Funcion_Final then
      --a0<=H4+h3;
      a0<=Resultado_Alu;
      end if;
 END IF;
 END PROCESS;


-- Proceso Combinacional
Process (Actual,i,j,inicio_hash,fin_alu)
    Begin
       Case Actual Is
          When Inicio=>
             start_i<="00";
             start_j<="00";
             fin_hash<='0';
             inicio_suma<='0';
             inicio_xor<='0';
             A <=(others=>'0');
             B <=(others=>'0');
             if inicio_hash='1' then
             siguiente<=Inicio_2;
             else
             siguiente<=Inicio;
             end if;
          When Inicio_2=>
             start_i<="00";
             start_j<="00";
             fin_hash<='0';
             inicio_suma<='0';
             inicio_xor<='0';
             A <=(others=>'0');
             B <=(others=>'0');
             siguiente<=SUMA_FUNCION_A;

          When Suma_funcion_A=>  --SE OBTIENE LA SUMA HO+NT
             start_i<="10";
             start_j<="00";
             fin_hash<='0';
             inicio_suma<='0';
```

```
    inicio_xor<='0';
    A<=h3;
    B<=Nt;
  --  if fin_alu='1' then
        siguiente<=Funcion_A1;
    --else
     --   siguiente<=Suma_funcion_A;
    --end if;
When Funcion_A1=>
    start_i<="01";
    start_j<="00";
    fin_hash<='0';
     inicio_suma<='0';
     inicio_xor<='0';
     A <=(others=>'0');
    B <=(others=>'0');
    siguiente<=Suma_funcion_A2;
 When Suma_funcion_A2=>  --SE HACE LA SUMA TOTAL DE LA FUNCION
    start_i<="10";
    start_j<="00";
    fin_hash<='0';
    inicio_suma<='0';
    inicio_xor<='0';
    A<=(h3(N-2 downto 0) & '0');
    B<=('0' & h3_1(N-1 downto 1));
   -- if fin_alu='1' then
        siguiente<=Funcion_A2;
    --else
      --  siguiente<=Suma_funcion_A2;
    --end if;
when Funcion_A2=>
    start_i<="10";
    start_j<="00";
    fin_hash<='0';
     inicio_suma<='0';
     inicio_xor<='0';
     A <=(others=>'0');
    B <=(others=>'0');
    if i=r1 then
        siguiente<=Suma_Funcion_B;
    else
        siguiente<=Suma_Funcion_A;
    end if;

When Suma_funcion_B =>
    start_i<="00";
    start_j<="10";
    fin_hash<='0';
     inicio_suma<='0';
     inicio_xor<='0';
     A<=h4;
     B<=Nt;
     --if fin_alu='1' then
        siguiente<=Suma_Funcion_B2;
    --else
     --   siguiente<=Suma_funcion_B;
```

```
    --end if;

When Suma_funcion_B2 =>
    start_i<="00";
    start_j<="10";
    fin_hash<='0';
     inicio_suma<='0';
     inicio_xor<='0';
     A<=('0' & h4(N-1 downto 1));
     B<=(h4(N-2 downto 0) & '0');
     --if fin_alu='1' then
        siguiente<=Suma_Funcion_B3;
    --else
    --    siguiente<=Suma_funcion_B2;
    --end if;

When Suma_funcion_B3=>
    start_i<="00";
    start_j<="10";
    fin_hash<='0';
     inicio_suma<='0';
     inicio_xor<='0';
     A <=(others=>'0');
    B <=(others=>'0');
    siguiente<=Suma_funcion_B4;

When Suma_funcion_B4=>
    start_i<="00";
    start_j<="10";
    fin_hash<='0';
     inicio_suma<='0';
     inicio_xor<='0';
     A<=H3_1;
     B<=H4;
    -- if fin_alu='1' then
        siguiente<=Funcion_B;
    --else
     --   siguiente<=Suma_funcion_B4;
    --end if;

when Funcion_B=>
    start_i<="00";
    start_j<="01";
    fin_hash<='0';
     inicio_suma<='0';
     inicio_xor<='0';
     A <=(others=>'0');
    B <=(others=>'0');
    if j=r1-1 then
        siguiente<=Inicio_CD;
    else
        siguiente<=Suma_funcion_B;
    end if;
When Inicio_CD=>
    start_i<="00";
    start_j<="00";
```

```
    fin_hash<='0';
     inicio_suma<='0';
     inicio_xor<='0';
     A <=(others=>'0');
    B <=(others=>'0');
    siguiente<=Suma_Funcion_C1;

When Suma_Funcion_C1=>
    start_i<="10";
    start_j<="10";
    fin_hash<='0';
     inicio_suma<='0';
     inicio_xor<='0';
     A<=H3;
     B<=H4;
    --if fin_alu='1' then
        siguiente<=Funcion_C1;
   -- else
    --    siguiente<=Suma_funcion_C1;
    --end if;

When Funcion_C1=>
    start_i<="10";
    start_j<="10";
    fin_hash<='0';
     inicio_suma<='0';
     inicio_xor<='0';
     A <=(others=>'0');
    B <=(others=>'0');
    siguiente<=XOR_Funcion_C2;

 When XOR_Funcion_C2=>
    start_i<="10";
    start_j<="10";
    fin_hash<='0';
     inicio_suma<='1';
     inicio_xor<='1';
      A<=H3;
     B<="000" & h3_1(N-1 downto 3);
   -- if fin_alu='1' then
        siguiente<=Funcion_C2;
    --else
    --    siguiente<=XOR_funcion_C2;
    --end if;

 When Funcion_C2=>
    start_i<="10";
    start_j<="10";
    fin_hash<='0';
     inicio_suma<='0';
     inicio_xor<='0';
     A <=(others=>'0');
    B <=(others=>'0');
    siguiente<=Suma_Funcion_C3;

When Suma_Funcion_C3=>
```

```
    start_i<="10";
    start_j<="10";
    fin_hash<='0';
     inicio_suma<='0';
     inicio_xor<='0';
     A<=("00" & h3(N-1 downto 2));
     B<=h3;
    --if fin_alu='1' then
        siguiente<=Funcion_C3;
    --else
     --   siguiente<=Suma_funcion_C3;
    --end if;

When Funcion_C3=>
    start_i<="10";
    start_j<="10";
    fin_hash<='0';
     inicio_suma<='0';
     inicio_xor<='0';
     A <=(others=>'0');
     B <=(others=>'0');
    siguiente<=Suma_Funcion_C4;

When Suma_Funcion_C4=>
    start_i<="10";
    start_j<="10";
    fin_hash<='0';
     inicio_suma<='0';
     inicio_xor<='0';
      A<=H3_1;
     B<=h3(N-4 downto 0) & "000";
    --if fin_alu='1' then
        siguiente<=Funcion_C4;
    --else
     --   siguiente<=Suma_funcion_C4;
    --end if;

When Funcion_C4=>
    start_i<="10";
    start_j<="10";
    fin_hash<='0';
     inicio_suma<='0';
     inicio_xor<='0';
     A <=(others=>'0');
     B <=(others=>'0');
    siguiente<=XOR_Funcion_C5;

When XOR_Funcion_C5=>
    start_i<="10";
    start_j<="10";
    fin_hash<='0';
     inicio_suma<='1';
     inicio_xor<='1';
     A<=h3(N-2 downto 0) & '0';
     B<=conv_std_logic_vector(125,N);
   -- if fin_alu='1' then
```

```
        siguiente<=Funcion_C5;
    --else
     --   siguiente<=XOR_funcion_C5;
    --end if;

When Funcion_C5=>
    start_i<="10";
    start_j<="10";
    fin_hash<='0';
     inicio_suma<='0';
     inicio_xor<='0';
     A <=(others=>'0');
     B <=(others=>'0');
    siguiente<=Suma_Funcion_c6;

When Suma_Funcion_C6=>
    start_i<="10";
    start_j<="10";
    fin_hash<='0';
     inicio_suma<='0';
     inicio_xor<='0';
       A<=H3_1;
     B<=H3;
    --if fin_alu='1' then
        siguiente<=Funcion_C6;
    --else
     --   siguiente<=Suma_funcion_C6;
    --end if;

When Funcion_C6=>
    start_i<="10";
    start_j<="10";
    fin_hash<='0';
     inicio_suma<='0';
     inicio_xor<='0';
     A <=(others=>'0');
     B <=(others=>'0');
    siguiente<=XOR_Funcion_D1;

When Xor_Funcion_D1=>
    start_i<="10";
    start_j<="10";
    fin_hash<='0';
     inicio_suma<='1';
     inicio_xor<='1';
      A<=H3;
     B<=h4;
    --if fin_alu='1' then
        siguiente<=Funcion_D1;
    --else
    --    siguiente<=XOR_funcion_D1;
    --end if;

When Funcion_D1=>
    start_i<="10";
    start_j<="10";
```

```
    fin_hash<='0';
     inicio_suma<='0';
     inicio_xor<='0';
     A <=(others=>'0');
     B <=(others=>'0');
    siguiente<=Xor_Funcion_D2;

When XOR_Funcion_D2=>
    start_i<="10";
    start_j<="10";
    fin_hash<='0';
     inicio_suma<='1';
     inicio_xor<='1';
       A<=H3_1;
      B<=H4;
    --if fin_alu='1' then
        siguiente<=Funcion_D2;
    --else
     --   siguiente<=Xor_funcion_d2;
    --end if;

When Funcion_D2=>
    start_i<="10";
    start_j<="10";
    fin_hash<='0';
     inicio_suma<='0';
     inicio_xor<='0';
     A <=(others=>'0');
     B <=(others=>'0');
    siguiente<=Suma_Funcion_D3;

When Suma_Funcion_D3=>
    start_i<="10";
    start_j<="10";
    fin_hash<='0';
     inicio_suma<='0';
     inicio_xor<='0';
     A<=("0000" & h4(N-1 downto 4));
     B<=("000" & h4(N-1 downto 3));
    --if fin_alu='1' then
        siguiente<=Funcion_D3;
    --else
     --   siguiente<=Suma_funcion_d3;
    --end if;

When Funcion_D3=>
    start_i<="10";
    start_j<="10";
    fin_hash<='0';
     inicio_suma<='0';
     inicio_xor<='0';
     A <=(others=>'0');
     B <=(others=>'0');
    siguiente<=Suma_Funcion_D4;

When Suma_Funcion_D4=>
```

```
    start_i<="10";
    start_j<="10";
    fin_hash<='0';
     inicio_suma<='0';
     inicio_xor<='0';
    A<=(h4(N-4 downto 0) & "000");
    B<=h4;
   -- if fin_alu='1' then
        siguiente<=Funcion_D4;
   -- else
    --    siguiente<=Suma_funcion_d4;
    --end if;

When Funcion_D4=>
    start_i<="01";
    start_j<="10";
    fin_hash<='0';
     inicio_suma<='0';
     inicio_xor<='0';
     A <=(others=>'0');
     B <=(others=>'0');
    siguiente<=Suma_Funcion_D5;

When Suma_Funcion_D5=>
    start_i<="10";
    start_j<="10";
    fin_hash<='0';
     inicio_suma<='0';
     inicio_xor<='0';
    A<=H3_1;
    B<=H4;
    --if fin_alu='1' then
        siguiente<=Funcion_D5;
    --else
     --   siguiente<=Suma_funcion_d5;
    --end if;

When Funcion_D5=>
    start_i<="10";
    start_j<="10";
    fin_hash<='0';
     inicio_suma<='0';
     inicio_xor<='0';
     A <=(others=>'0');
     B <=(others=>'0');
    if i=r2 then
    siguiente<=Funcion_D6;
    else
    siguiente<=Suma_Funcion_C1;
   end if;

when Funcion_D6 =>
    start_i<="00";
    start_j<="01";
    fin_hash<='0';
     inicio_suma<='0';
```

```
                inicio_xor<='0';
                A <=(others=>'0');
                B <=(others=>'0');
               siguiente<=Funcion_D7;

            when Funcion_D7 =>
               start_i<="00";
               start_j<="10";
               fin_hash<='0';
                inicio_suma<='0';
                inicio_xor<='0';
                A <=(others=>'0');
                B <=(others=>'0');
               if j=(nstate) then
               siguiente<=Mod_estado3;
               elsif j=1 then
               siguiente<=MOd_Estado0;
               elsif j=2 then
               siguiente<=Mod_estado1;
               elsif j=3 then
               siguiente<=Mod_estado2;
               else
               siguiente<=inicio;
               --else
               --siguiente<=Suma_Funcion_C1;
             end if;

            when Mod_estado0=> --Primera suma para el estado0
               start_i<="00";
               start_j<="10";
               fin_hash<='0';
                inicio_suma<='0';
                inicio_xor<='0';
              A<=h3;
             B<=h4;
               siguiente<=Estado0;

            when Estado0=> -- Paso intermedio guardar en h3_2
               start_i<="00";
               start_j<="10";
               fin_hash<='0';
                inicio_suma<='0';
                inicio_xor<='0';
               A <=(others=>'0');
               B <=(others=>'0');
               siguiente<=Xor_estado0;

            when Xor_estado0=>
               start_i<="00";
               start_j<="10";
               fin_hash<='0';
                inicio_suma<='1';
                inicio_xor<='0';
--               A <=h4;
  --             B <=h4;
               siguiente<=Resultado_estado0;
```

```
            when Resultado_estado0=> --Guardar en h3_2
               start_i<="00";
               start_j<="10";
               fin_hash<='0';
                inicio_suma<='0';
                inicio_xor<='0';
               A <=(others=>'0');
               B <=(others=>'0');
               siguiente<=Suma_Funcion_C1;

            when Mod_estado1=> --Primera suma para el estado0
               start_i<="00";
               start_j<="10";
               fin_hash<='0';
                inicio_suma<='0';
                inicio_xor<='0';
--               A<=h3;
 --              B<=h4;
               siguiente<=Estado1;

            when Estado1=> -- Paso intermedio guardar en h3_2
               start_i<="00";
               start_j<="10";
               fin_hash<='0';
                inicio_suma<='0';
                inicio_xor<='0';
               A <=(others=>'0');
               B <=(others=>'0');
               siguiente<=Xor_estado1;

            when Xor_estado1=>
               start_i<="00";
               start_j<="10";
               fin_hash<='0';
                inicio_suma<='1';
                inicio_xor<='0';
               A <=h4;
               B <=h4;
               siguiente<=Resultado_estado1;

            when Resultado_estado1=> --Guardar en h3_2
               start_i<="00";
               start_j<="10";
               fin_hash<='0';
                inicio_suma<='0';
                inicio_xor<='0';
               A <=(others=>'0');
               B <=(others=>'0');
               siguiente<=Suma_Funcion_C1;

            when Mod_estado2=> --Primera suma para el estado0
               start_i<="00";
               start_j<="10";
               fin_hash<='0';
                inicio_suma<='0';
```

```
     inicio_xor<='0';
    A<=h3;
    B<=h4;
    siguiente<=Estado2;

when Estado2=> -- Paso intermedio guardar en h3_2
    start_i<="00";
    start_j<="10";
    fin_hash<='0';
     inicio_suma<='0';
     inicio_xor<='0';
    A <=(others=>'0');
    B <=(others=>'0');
    siguiente<=Xor_estado2;

when Xor_estado2=>
    start_i<="00";
    start_j<="10";
    fin_hash<='0';
     inicio_suma<='1';
     inicio_xor<='0';
    A <=h4;
    B <=h4;
    siguiente<=Resultado_estado2;

when Resultado_estado2=> --Guardar en h3_2
    start_i<="00";
    start_j<="10";
    fin_hash<='0';
     inicio_suma<='0';
     inicio_xor<='0';
    A <=(others=>'0');
    B <=(others=>'0');
    siguiente<=Suma_Funcion_C1;

when Mod_estado3=> --Primera suma para el estado0
    start_i<="00";
    start_j<="10";
    fin_hash<='0';
     inicio_suma<='0';
     inicio_xor<='0';
    A<=h3;
    B<=h4;
    siguiente<=Estado3;

when Estado3=> -- Paso intermedio guardar en h3_2
    start_i<="00";
    start_j<="10";
    fin_hash<='0';
     inicio_suma<='0';
     inicio_xor<='0';
    A <=(others=>'0');
    B <=(others=>'0');
    siguiente<=Xor_estado3;

when Xor_estado3=>
```

```
            start_i<="00";
            start_j<="10";
            fin_hash<='0';
             inicio_suma<='1';
             inicio_xor<='0';
            A <=h4;
            B <=h4;
            siguiente<=Resultado_estado3;

         when Resultado_estado3=> --Guardar en h3_2
            start_i<="00";
            start_j<="10";
            fin_hash<='0';
             inicio_suma<='0';
             inicio_xor<='0';
            A <=(others=>'0');
            B <=(others=>'0');
            siguiente<=Suma_Funcion_Final;

         when Suma_Funcion_Final =>
            start_i<="00";
            start_j<="00";
            fin_hash<='0';
             inicio_suma<='0';
             inicio_xor<='0';
            A<=h3;
            B<=h4;
            siguiente<=Funcion_Final;

         when Funcion_Final =>
            start_i<="00";
            start_j<="00";
            fin_hash<='1';
             inicio_suma<='0';
             inicio_xor<='0';
             A <=(others=>'0');
             B <=(others=>'0');
            siguiente<=Inicio;
         end case;
      end process;

end hash_a3;
```