UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

DEPARTAMENTO DE TECNOLOGÍA ELECTRÓNICA

INGENIERÍA INDUSTRIAL

PROYECTO FIN DE CARRERA

"IMPLEMENTACIÓN Y ANÁLISIS DE DOS PROTOCOLOS DE AUTENTICACIÓN SEGUROS Y LIGEROS PARA RFID"

AUTORA: **ELENA DE LA IGLESIA TORIBIOS**

TUTORES: **MARTA PORTELA**

ENRIQUE SAN MILLÁN

JULIO 2010

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

CAPÍTULO 1
INTRODUCCIÓN

1.1. ANTECEDENTES

Existen multitud de tecnologías basadas en la detección e identificación de objetos de forma remota. Aunque durante los últimos 25 años la más conocida y empleada ha sido el código de barras, esta situación ha empezado a cambiar con el desarrollo de los sistemas RFID (*Radio Frequency IDentification*).

La identificación por radio frecuencia, a la que haremos referencia a partir de ahora por sus siglas RFID, no es una tecnología nueva, ya que su origen se remonta a la segunda Guerra Mundial, pero es actualmente cuando su desarrollo está cobrando mayor protagonismo. Debido a la evolución tecnológica en este campo, que ha permitido reducir los costes de fabricación y aplicación, su uso se está generalizando entre las empresas. Es por ello que la tecnología RFID se postula como la futura identificación unitaria, sustituyendo a la actual tecnología del código de barras.

Una de las múltiples ventajas que presenta la tecnología RFID es la mayor capacidad de almacenamiento de datos asociados al objeto, que es identificado por el *tag* o etiqueta RFID. Pero esto puede suponer un problema para las empresas, que ven aumentar la vulnerabilidad de sus sistemas de información. Esto se debe a la relativa facilidad con la que terceras personas pueden tener acceso, leer o modificar los datos, poniendo en peligro la integridad del sistema. Con el fin de solucionar estos problemas se emplean protocolos de autenticación, que permiten realizar una identificación segura de los agentes implicados en la comunicación RFID (*tag* y lector), antes de comenzar, por ejemplo, el intercambio de datos.

1.2. OBJETIVOS DEL PROYECTO

El objetivo de este proyecto es la implementación de dos protocolos de autenticación seguros y ligeros para sistemas RFID. Dichos protocolos han de tener un nivel de seguridad suficiente como para asegurar la integridad del sistema, pero a su vez

han de ser lo suficientemente pequeños como para cumplir las restricciones de área que presentan los *tags* RFID.

La implementación de dichos protocolos se va a realizar mediante un ASIC (Circuito Integrado para Aplicaciones Específicas), es decir, un circuito hecho a medida para un uso particular. El lenguaje que se emplea para la implementación del protocolo es VHDL (*Very High Speed Integrated Circuit Hardware Description Language*). Este lenguaje creado por el Departamento de Defensa de los EE.UU., se aplica en el diseño de circuitos digitales y se encuentra estandarizado por el IEEE (*Institute of Electricals and Electronics Engineers*).

Para la implementación de los circuitos desarrollados se va a emplear la herramienta de síntesis de Synopsys que permite obtener estimaciones precisas del área necesaria para la implementación y consumo de potencia. Además de éste se van a utilizar otras herramientas como el entorno de diseño ISE de Xilinx o el programa ModelSim para la simulación de los circuitos.

El sistema de autenticación que se va a desarrollar consta de un protocolo basado en un generador de números pseudoaleatorios (PRNG). Por ello se realizarán varios diseños, empleando tanto distintos protocolos como diferentes PRNGs, con el fin de obtener el sistema más adecuado para los objetivos que se persiguen en este proyecto.

Una vez implementados los protocolos y realizadas las pruebas y simulaciones necesarias, se extraerán las conclusiones a cerca de las diferentes opciones presentadas.

1.3. ESTRUCTURA DEL CONTENIDO

El proyecto está distribuido en ocho capítulos. En el primero de ellos se realiza una breve introducción sobre el tema, se indican los objetivos del proyecto, así como la organización del mismo.

En el segundo capítulo se desarrollan los aspectos más importantes de la tecnología RFID, como el origen, método y frecuencias de funcionamiento, aplicaciones, mecanismos de control y seguridad en la comunicación RFID, como son los protocolos de comunicación, los protocolos anticolisión o los de seguridad. Siendo este último tema elemental para la comprensión del desarrollo y objetivos del proyecto.

En el tercer capítulo se describe la metodología seguida para el desarrollo de este proyecto, así como las herramientas y el lenguaje empleados a lo largo del mismo.

En el cuarto capítulo se muestra tanto el funcionamiento como la arquitectura de los protocolos de autenticación RFID implementados. Se incluye además una descripción de los generadores de números pseudoaleatorios de los que se disponía y que se han empleado para la realización de los protocolos.

El quinto capítulo contiene las pruebas y simulaciones realizadas para la comprobación del funcionamiento de los protocolos de autenticación. Se muestran también los resultados de la síntesis de los diferentes diseños implementados con cada protocolo.

En el sexto capítulo se muestran las conclusiones del proyecto, así como las posibles líneas futuras que podrían seguirse para mejorar los protocolos de autenticación presentados.

En el capítulo siete se muestra el presupuesto del proyecto realizado. La bibliografía está detallada en el capítulo ocho. Por último se encuentran, incluidos como anexo, los códigos creados para la implementación de los protocolos que se desarrollan en este proyecto.

CAPÍTULO 2
LA TECNOLOGÍA RFID

2.1. INTRODUCCIÓN A RFID

RFID (*Radio Frequency Identification* o *Identificación por radiofrecuencia*) es una tecnología que permite el almacenamiento y recuperación de datos, mediante ondas de radio con el fin de identificar un objeto, empleando etiquetas o *tags* RFID. Estas etiquetas son dispositivos pequeños, similares a una pegatina, que pueden ser adheridos a un objeto, persona o animal.

2.2. ORIGEN Y EVOLUCIÓN

El origen de la tecnología RFID se encuentra en la II Guerra Mundial. En esa época los sistemas de radar permitían la detección de aviones a kilómetros de distancia, pero no su identificación. Es por ello que durante las décadas de los 50 y los 60 se puso un gran empeño en el desarrollo de dichos dispositivos. Pero fue a partir de los años 70 cuando comenzaron a solicitarse las primeras patentes para *tags* (Charles Walton, dispositivo para la apertura de puertas sin llave o Mario W. Cardullo, etiqueta activa con memoria reescribible).

Actualmente el principal organismo responsable de la implantación y desarrollo de la tecnología RFID es Auto-ID Center, constituida en 1999 por empresas, universidades y centros de investigación, como el MIT (*Massachussets Institute of Technology*), la universidad de Cambrige, o la de Keio en Japón. Auto-ID Center es conocida desde 2003 como EPCglobal.

Como toda tecnología, ha sufrido importantes mejoras desde sus orígenes, tanto en la capacidad de emisión y recepción, como en la distancia, el precio, el tamaño, etc, lo que está haciendo posible la extensión de su uso en todos los ámbitos, principalmente en la industria, convirtiéndose poco a poco en el sustituto más probable del conocido código de barras.

Nota: Para más información sobre la tecnología RFID consultar referencias [1] a [12].

2.3. ELEMENTOS DE UN SISTEMA RFID

El funcionamiento de un sistema RFID se basa en una etiqueta que contiene los datos de identificación y que emite una señal de radiofrecuencia, si ha sido requerida. Esta señal es captada por un lector, encargado de leer la información y pasarla a un sistema de procesamiento de datos. Los sistemas RFID constan, por tanto, de tres componentes básicos: *tags*, lector y subsistema de procesamiento.

2.3.1. Etiqueta RFID o transpondedor

La etiqueta RFID está compuesta por una antena, un transductor de radio y un chip, que contiene la información, la cual se encuentra encriptada. Éste posee una memoria interna cuya capacidad varía según el modelo, desde una decena a millares de bytes. La clasificación de estos dispositivos puede hacerse según el tipo de memoria y según la fuente de alimentación.

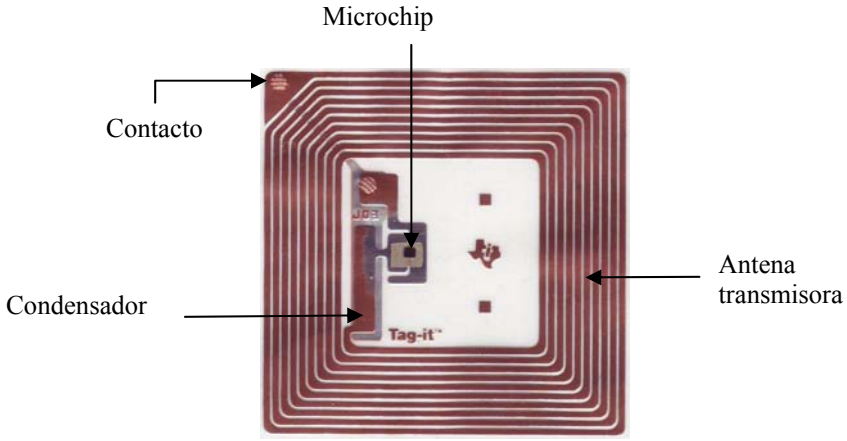

Figura 1: Tag RFID

- Según el tipo de memoria:

  - Sólo lectura: contiene un código de identificación que no puede reescribirse y que suele establecerse durante la fabricación.
  - Múltiple lectura y única escritura: este tipo de *tag* carece de código de identificación a la salida de fábrica y es el usuario del mismo el que debe establecerlo, pero no podrá ser modificado.
  - Múltiple lectura y múltiple escritura: la memoria del *tag* puede ser escrita y leída un número indefinido de veces.

- Según la fuente de alimentación:

  - *Tags* pasivos: su principal característica es que carecen de alimentación integrada, por lo que son más baratos de fabricar y más duraderos. Su funcionamiento se basa en la señal que le llega de los lectores. Ésta les induce una pequeña corriente eléctrica, suficiente para el funcionamiento del circuito integrado CMOS del *tag* y la transmisión de información al lector. La distancia de aplicación de estos *tags* va desde los 10cm hasta los pocos metros. Debido a la importancia de la energía, la respuesta de las etiquetas RFID pasivas ha de ser breve, normalmente poco más que un número de identificación. La posición u orientación de los *tags* puede afectar al funcionamiento óptimo de los mismos, según la Potencia Radiada Equivalente (ERP), decisiva en el caso de este tipo de *tags*. Las ubicaciones de las etiquetas pasivas dependiendo de la ERP se denominan zona de resonancia (R, *resonance spot*) donde habrá interferencias, zona viva (L, *live spot*) en la que se realizarán las comunicaciones óptimas y zona muerta (D, *dead spot*) donde se no podrá establecer comunicación [9].
  - *Tags* activos: este tipo de *tags* posee fuente de alimentación propia que emplean para alimentar sus circuitos integrados y propagar su señal al lector. Son más fiables y poseen mayor alcance que los pasivos (de 10 hasta 100 m) debido a que pueden emitir señales más potentes gracias a su fuente de alimentación propia, lo que les permite además generar señales a partir de recepciones de señal débiles. En este caso el *tag* tiene la posibilidad de ser el primero en comunicarse, al contrario que los pasivos, que precisan de la señal del lector para transmitir sus datos.
  - *Tags* semipasivos: los semipasivos poseen también fuente de alimentación propia, pero únicamente con la finalidad de alimentar sus circuitos integrados, no para emitir señales. Son más rápidos y fiables que los pasivos, pero por el contrario tienen menor alcance que los activos (menos de 10m).

2.3.2. Lector o transceptor

Está compuesto por una antena, un transceptor y un decodificador. Emite periódicamente señales con el fin de detectar la presencia de alguna etiqueta en sus inmediaciones.

Una vez localizada, extrae la información que ésta pueda contener y se la pasa al subsistema de procesamiento. Los lectores pueden ser fijos o móviles, de lo que dependerá tanto el tipo de comunicación con el subsistema como el tipo de antena necesaria.

2.3.3. Subsistema de procesamiento de datos

También conocido como *middleware* RFID, es la plataforma existente entre los lectores de *tags* y los sistemas de gestión empresariales para trabajar con los datos captados por el hardware RFID, tratándolos según las necesidades de la aplicación final. Se parte de la idea de que la información almacenada en un *tag* se limita a poco más que un número de identificación, por lo que se hace necesario un sistema que permita relacionar dicha clave con la información asociada al artículo. Ha de tenerse en cuenta que la necesidad de la presencia del *middleware* dependerá tanto de la dimensión como de la complejidad de la aplicación RFID.

Éste y el resto de elementos de un sistema RFID descritos se muestran en la Figura 2 [10].

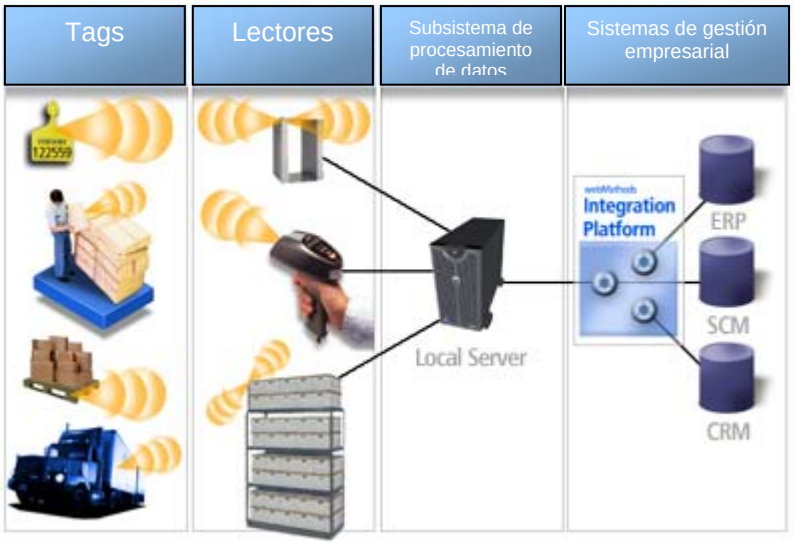

Figura 2: Elementos de un sistema RFID

2.4. FUCIONAMIENTO BÁSICO DE UN SISTEMA RFID

Un sistema RFID actúa de la siguiente manera. El lector envía una señal de requerimiento a un conjunto de *tags*, a la que éstos responden enviando cada uno su número de identificación único.

Cuando el lector recibe dicho código lo transmite a la base de datos, en la que previamente se ha almacenado la información correspondiente a cada *tag*. De este modo se puede consultar la identidad del mismo.

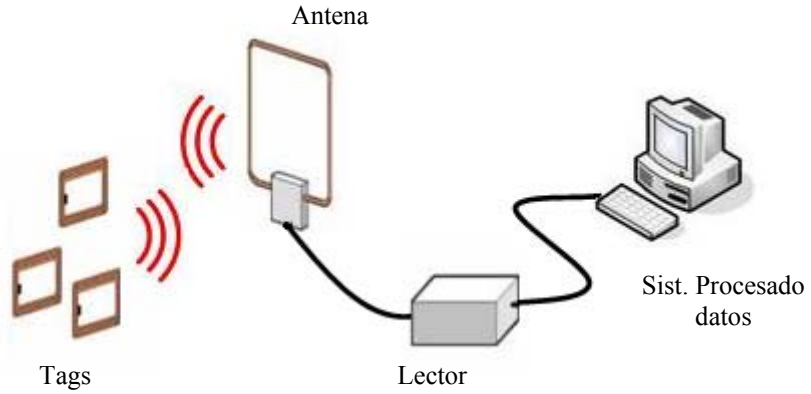

Figura 3: Esquema de funcionamiento de un sistema RFID

2.4.1. Métodos de propagación de energía

La comunicación entre un *tag* pasivo o semipasivo y un lector consiste tanto en transferencia de energía como en transferencia de datos. La transferencia de energía se realiza mediante el acoplamiento a través de campos magnéticos. Las etiquetas RFID emplean ya sea el campo magnético, el campo eléctrico o ambos para recibir la energía del lector. Existen varios métodos de transferencia, tal y como se muestra a continuación [11][12].

- ***Acoplamiento inductivo:*** cuando un conductor se expone a un campo magnético, éste produce un flujo de corriente en el mismo. Este principio es empleado para la comunicación en los sistemas RFID. La antena del lector genera un campo magnético, que induce una corriente en la antena del *tag*, que alimenta los circuitos del mismo.
- ***Acoplamiento electromagnético:*** puesto que con la distancia el campo magnético se debilita, se emplea el campo eléctrico para resonar la antena del *tag*, de esta forma se pueden realizar comunicaciones a mayores distancias. Este método de funcionamiento también se denomina acoplamiento *backscatter*. Suelen emplearse para sistemas que trabajan en alta frecuencia (UHF o microondas), lo que permite el uso de antenas más pequeñas de gran eficiencia. El lector posee un sistema que le permite separar la señal emitida de la señal

recibida del *tag*, mucho más débil que ésta, y más a medida que se aumenta la distancia.

- ***Close coupling:*** se emplean para distancias de detección muy pequeñas. Para ser detectado el *tag* debe encontrarse en el interior de la bobina del lector (con forma de U o de aro). Una corriente que recorra las espiras del lector generará un campo magnético que proporcionará alimentación al circuito del *tag*. Debido a que las pérdidas de energía durante la transmisión son mínimas, este método suele ser empleado en sistemas que precisan de chips potentes con un alto consumo de energía.

2.4.2. Formato de transmisión

Para leer o escribir en los sistemas RFID se realiza una transferencia de información entre el lector y el *tag*. Dicha transferencia puede seguir uno de los siguientes procesos.

- **Half-duplex**: el canal de comunicaciones permite la transmisión de datos en ambas direcciones pero no de forma simultánea.
- **Full-duplex**: el canal de comunicaciones permite la transmisión de datos en ambas direcciones al mismo tiempo.

2.4.3. Codificación y modulación de la señal

La codificación y modulación de señal toma el mensaje a transmitir y su representación en forma de señal y la adecua óptimamente a las características del canal de transmisión. Este proceso implica proveer al mensaje con un grado de protección contra interferencias o colisiones y contra modificaciones intencionadas de la señal.

- Codificación de la señal

La codificación de la señal permite transmitir señales digitales a través de diversos medios de transmisión. Los esquemas de codificación más comunes en los sistemas RFID son los siguientes, representados en la Figura 4 [11].

- **NRZ (*No Return to Zero*)**: un '1' binario es representado por una señal "alta" y un '0' por una señal "baja".
- **Manchester**: un '1' binario es representado por una transición negativa en la mitad del periodo del bit y un '0' por una positiva.
- **DBP**: un '1' binario es representado por ausencia de transición y un '0' por una transición en mitad del periodo del bit.
- **Miller**: un '1' binario es representado por una transición de cualquier tipo en la mitad del periodo del bit, mientras que el '0' es representado por la continuidad del nivel de la señal hasta el próximo periodo del bit.

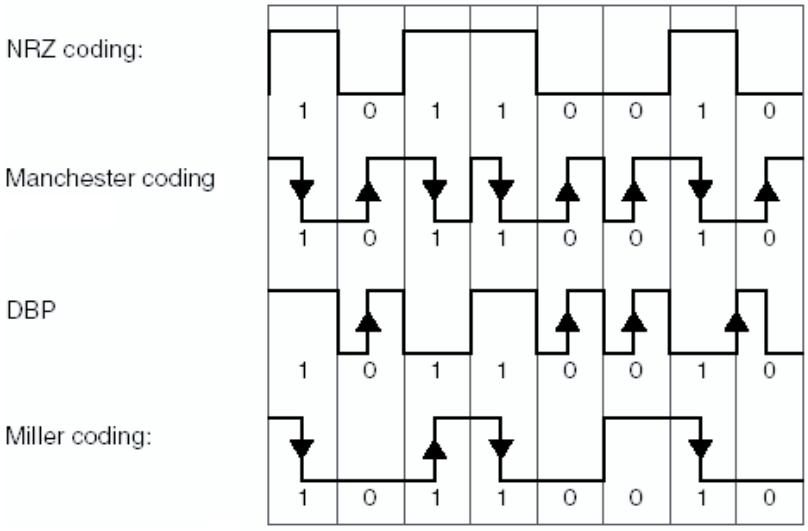

Figura 4: Representación gráfica de las principales codificaciones [11].

- Modulación de la señal

Este proceso consiste en modificar alguna de las variables de dicha señal con el fin de adecuarla a las características tanto del canal de comunicación como de los agentes que intervienen en la misma. Típicamente, las variables que se modifican son la amplitud, la frecuencia o la fase. En la Figura 5 puede observarse una representación de las siguientes modulaciones.

- **Modulación ASK (*Amplitude Shift Keying*)**: es una modulación de amplitud. Los datos digitales son enviados sobre una portadora analógica. El '1' binario se representará mediante la presencia de portadora y el '0' con la ausencia de dicha portadora.

- **Modulación FSK (*Frequency Shift Keying*)**: es una modulación en frecuencia. Los dos valores binarios se representan con dos frecuencias diferentes, próximas a la de la señal portadora.
- **Modulación PSK (*Phase Shift Keying*)**: es una modulación de fase. Consiste en hacer variar la fase de la señal portadora.

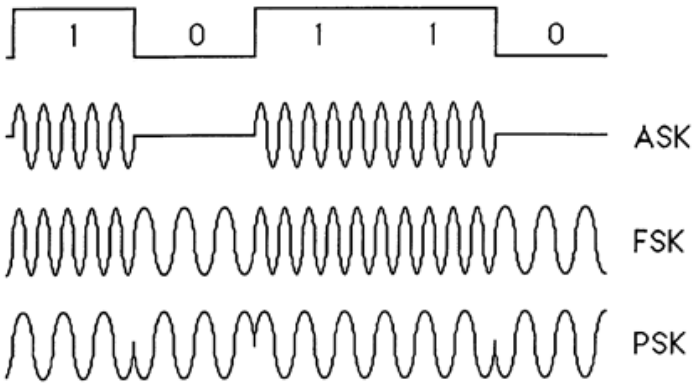

Figura 5: Representación de los principales tipos de modulación.

2.5. RANGOS DE FRECUENCIAS

Los sistemas RFID pueden trabajar a diferentes frecuencias, lo que influye en el alcance de los dispositivos y en los entornos en los que puede operar un *tag*. A medida que han ido surgiendo necesidades se han ido asignando las diferentes bandas de frecuencia (baja frecuencia para identificación de animales, UHF para cadenas de suministro, etc). A continuación se muestran las diferentes bandas de funcionamiento (Figura 6 [13]), así como sus principales características.

3. Sistemas de baja frecuencia (LF): Son de baja velocidad, cortas distancias e identificación unitaria. Rango de frecuencias 125-134 KHz.
4. Sistemas de alta frecuencia (HF): Velocidad media y distancias inferiores a 2 metros. Rango de frecuencias 13,56 MHz.
5. Sistemas de ultra alta frecuencia (UHF): Lecturas masivas a alta velocidad y largas distancias. Rango de frecuencias 860-960 MHZ.
6. Sistemas de microondas: Grandes alcances. Rango de frecuencias 2,45 GHz.

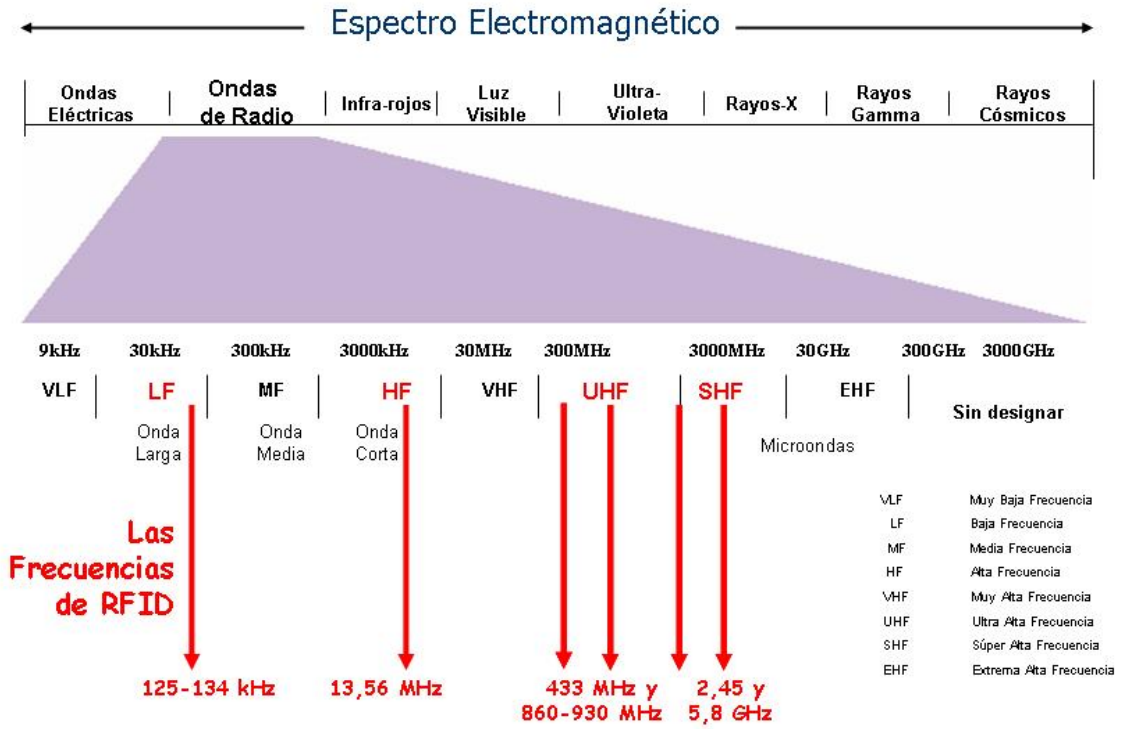

Figura 6: Espectro electromagnético [13].

Para que un sistema RFID funcione correctamente, los *tags* y los lectores deben estar funcionando a la misma frecuencia. En el caso de una aplicación a nivel interno, como en un almacén, no existe mayor problema. Las complicaciones aparecen cuando se desea aplicar en procesos logísticos completos por ejemplo, implicando a diversas empresas, incluso de forma internacional.

Uno de los principales problemas es que no existe actualmente ninguna corporación pública global que regule el uso de frecuencias para RFID, ya que el espectro radioeléctrico es patrimonio de cada estado, siendo éste el encargado de distribuir las licencias, liberar las bandas y regular la ordenación de las diferentes tecnologías que requieran su utilización.

Con el fin de establecer una regulación global de las frecuencias, se creó en 2003 GS1, una *joint venture* (acuerdo comercial temporal de empresas u organizaciones) entre EAN (European Article Number), la organización europea de numeración de artículos, y el UCC (Uniform Code Council), que es la entidad responsable de la gestión de los códigos universal de producto (UPC) en Estados Unidos. El sistema GS1 se compone de varias áreas de actuación entre las que se encuentra la organización EPCglobal, de la que se hablará más adelante.

2.6. VENTAJAS DE LA TECNOLOGÍA RFID

En la actualidad, la reducción de costes que se está produciendo en los dispositivos RFID, están situándolos como la futura forma de identificación unitaria e unívoca de productos. Esto le convierte en el sustituto natural del código de barras, debido a las numerosas ventajas que presenta frente a éste:

- Los *tags* no precisan estar en visión directa con el lector que les identifica.
- Los *tags* pueden identificar cada producto individualmente, mientras que los códigos de barras únicamente identifican el tipo de producto.
- Es posible la lectura simultánea de varios *tags* empleando protocolos anti-colisión, de los que se hablará más adelante.
- Permite almacenar mucha más información.
- Los *tags* RFID pueden ser de lectura y escritura, mientras que los códigos de barras sólo se pueden leer.
- La información que almacenan se encuentra encriptada mediante sistemas criptográficos y de autenticación, dificultando su lectura y las falsificaciones.
- Son más resistentes y duraderos.

Se muestra a continuación la Tabla 1 [6] con una pequeña comparativa entre el código de barras y la tecnología RFID, tanto activa como pasiva.

	RFID pasivo	RFID activo	Código de barras
Modificación de datos	Modificable	Modificable	No modificable
Seguridad de datos	De baja a alta	Alta	Mínima
Almacenamiento	Alrededor de 64 KB	Alrededor de 8 MB	De 8 a 30 caracteres
Coste	Medio	Alto	Bajo
Estándares	En fase de implantación	Propietario y estándares abiertos	Estable e implantado
Tiempo de vida	Indefinido	De 3-5 años de vida de batería	Bajo por deterioro
Distancia de lectura	Del orden de 1m	Del orden de 100m	Pocos centímetros

Tabla 1: Comparativa entre tecnologías de identificación de objeto.

2.7. APLICACIONES DE RFID

Debido a su versatilidad, la tecnología RFID puede ser aplicada en multitud de sectores. Dependiendo de la frecuencia, el coste y el alcance, las aplicaciones son diferentes. Algunas de las áreas en las que se emplean sistemas RFID son las siguientes:

- **Sector textil**: Los *tags* RFID en este ámbito suelen estar encapsulados en resina epoxi. Van insertados en las prendas de forma discreta, dentro de dobladillos, termosellados o simplemente cosidos. La posición es muy importante, han de estar situados en un entorno no metálico o debidamente aislados. Gracias a su aplicación en el sector textil se consigue la optimización de recursos humanos, así como una reducción considerable de pérdidas, extravíos o robos de prendas.

- **Sector sanitario**: Se emplean para la identificación y localización de activos, control de medicamentos, trazabilidad y control de temperaturas de productos [14]. También se aplica para la gestión hospitalaria, realizando una identificación única del paciente así como su seguimiento, tanto médico como físico por las diferentes áreas del centro.

- **Logística**: Actualmente es la aplicación a la que mayor valor añadido aportan los sistemas RFID. El uso de esta tecnología permite tener localizado cualquier producto dentro de la cadena de suministro. Su aplicación se está imponiendo como tecnología básica para el desarrollo de soluciones globales para centros de almacenamiento y distribución [15]-[19].

  Las claves de esta adopción generalizada son los beneficios derivados de la automatización de procesos, consiguiendo una reducción de tiempo y errores, así como la mejor visibilidad de los productos a lo largo de la cadena de suministro. El nivel de integración de estos sistemas en el proceso es muy variado, adaptándose a las necesidades de cada empresa. Dentro de la logística, algunos de los ámbitos en los que se aplica son:

  - Gestión de almacén: Identificación unívoca de las posiciones en el almacén, gestión inteligente de huecos, guiado de carretillas elevadoras, gestión de inventarios, información al operario para la recogida y actualización de datos, gestión de stock, control de productos perecederos, recepción de mercancías.

- Gestión de trazabilidad: visibilidad de productos a nivel de cadena de suministro, posibilidad de compartir información entre empresas asociadas.
- Localización de bienes y personas: localización en tiempo real, seguridad del personal, alerta frente a hurtos de productos.
- Gestión de expediciones: comprobación y validación de mercancías expedidas, control de carga de vehículos con posibilidad de integración con sistemas de gestión de flotas de camiones.

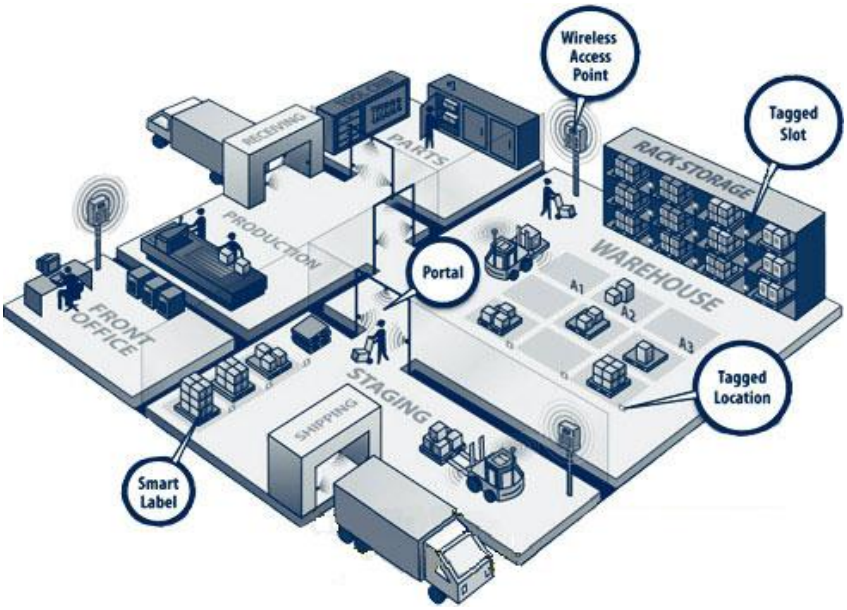

Figura 7: Aplicaciones RFID en la logística.

- **Implantes humanos**: La implantación de estos sistemas en seres humanos podrían suponer una solución a la usurpación de la identidad, permitir el acceso seguro a edificios, ordenadores, expedientes médicos, etc, pero por otro lado presenta un gran número de inconvenientes, entre ellos el rechazo de esta tecnología por parte del sujeto debido a los problemas de privacidad que representa.

Figura 8: RFID encapsulado, para implantación en humanos

- **Tráfico y posicionamiento**: aplicaciones como señales de tráfico inteligentes, posicionamiento en el interior de túneles, guiado de personas invidentes o seguimiento de vehículos, están comenzando a desarrollarse con esta tecnología.
- **Seguridad, pasaportes y carnets.**
- **Identificación de animales**: Se emplea un microchip en una cápsula de cristal que se introduce bajo la piel del animal. Sirve para identificarlo y almacenar información relevante, como quién es su dueño, vacunas, edad, etc.

2.8. ESTÁNDARES

RFID es una tecnología heterogénea con un número significativo de estándares. La estandarización de esta tecnología se debe principalmente a dos organizaciones, ISO (*International Organization for Standardization*) y EPCGlobal. Los estándares más relevantes describen las capas física y de enlace de datos, abarcando la interfaz aérea, anticolisión, protocolos de comunicación y funciones de seguridad. Los estándares RFID pretenden abarcar diferentes áreas:

- Protocolo en la interfaz aérea: especifica la manera en la que las etiquetas RFID y los lectores se comunican mediante radiofrecuencia.
- Contenido de los datos: especifica el formato de los datos que se comunican.
- Certificación: garantías de que los productos cumplen los estándares para poder interoperar entre distintos fabricantes.
- Aplicaciones: usos de los sistemas RFID.

2.8.1. Estándar ISO

Son muchas las normas ISO existentes para regular la radio frecuencia, pero con el fin de normalizar las diferentes frecuencias empleadas por los sistemas RFID, ISO las ha englobado bajo la serie 18000 (Tabla 2). Existen distintas versiones según el rango de frecuencias, debido a las diferencias básicas que existen en las leyes físicas que las gobiernan. Estos estándares se centran en la identificación automática y la gestión de objetos.

Rango de frecuencias	Versión serie 18000
< 135 KHz	ISO 18000-2
HF a 13,56 MHz	ISO 18000-3
UHF a 2,45 GHz	ISO 18000-4
UHF entre 860 y 960 MHz	ISO 18000-6 (A, B o C)
RFID UHF Gen2	ISO 18000-6 (C)
UHF a 433 MHz	ISO 18000-7

Tabla 2: Correspondencia estándar ISO – Frecuencia.

2.8.2. Estándar EPCGlobal

EPCglobal es una organización encargada de registrar todos los Códigos Electrónicos de Producto (EPC), administrar estándares, además de impulsar la adopción e implementación mundial de la Red EPCglobal en todos los sectores.

La red EPCglobal es una estructura que combina la tecnología de identificación de radio frecuencia de bajo costo, una infraestructura de redes de comunicación ya existente como es Internet y el código electrónico de producto para crear información precisa y efectiva que permita la identificación automática y el seguimiento de objetos físicos en tiempo real. La investigación de la organización EPCglobal se ha centrado en el desarrollo de estos cinco elementos esenciales para la red EPCglogal [20][21]:

- Código electrónico de producto (EPC): es un esquema para la identificación universal de objetos físicos a través de etiquetas RFID y otros medios. Consiste en un número de identificación del fabricante o administrador, un identificador de clase de objeto y un número de serie utilizado para identificar de forma única la identidad del objeto.
- Sistema de ID: lectores y *tags* de radio frecuencia.
- Servicio de nominación de objeto (ONS): es un directorio universal basado en el Sistema de Nombres de Dominio (DNS). ONS no contiene los datos reales sobre el EPC, sólo contiene la dirección de red donde residen los datos.
- Servicios de información EPC (EPCIS): es un estándar para el intercambio de datos basado en el código electrónico de producto. Define un esquema para los datos e interfaces para la captura y consulta de dichos datos.

- Software EPC middleware (Savant): es una tecnología de software diseñada para gestionar y mover la información en una forma que no sobrecargue las redes corporativas y públicas existentes, actúa como el *middleware* de la nueva Red EPCglobal, administrando el flujo de información.

2.8.2.1. Código Electrónico de Producto (EPC)

El EPC es considerado la siguiente generación de identificación de producto desde el código de barras. Es capaz de identificar de forma inequívoca un artículo individual, sin necesidad de línea de visión directa para su lectura, puesto que no se encuentra impreso sino insertado en el *tag*. Lo ideal es que el EPC sea la única información almacenada en el microchip del *tag*. Una cantidad infinita de datos dinámicos pueden asociarse al número EPC, almacenándolos en una base de datos. El formato de un paquete EPC es el siguiente:

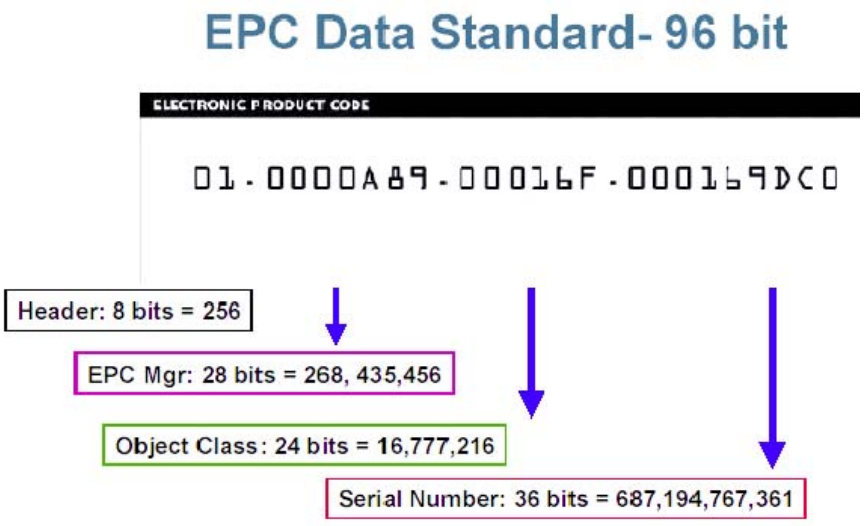

Figura 9: Formato del paquete EPC [6].

- Cabecera o número de versión (0-7 bits).
- Número de administración de dominio o fabricante (8-35 bits).
- Identificador del objeto o producto (36-59 bits).
- Número de serie (60-95 bits).

2.8.2.2. Sistema de ID

Las especificaciones del EPCglobal referentes a los elementos que se emplean en la comunicación se pueden dividir en las especificaciones para las etiquetas, referentes a los datos almacenados en ellas, a los protocolos de comunicación con el lector, etc y a las especificaciones para los lectores, referentes al interfaz aéreo y a las comunicaciones con las etiquetas.

Según su funcionalidad, el EPCglobal clasifica los *tags* teniendo en cuenta los diferentes niveles de sofisticación y capacidad, tal y como se muestra en la Figura 10 [11].

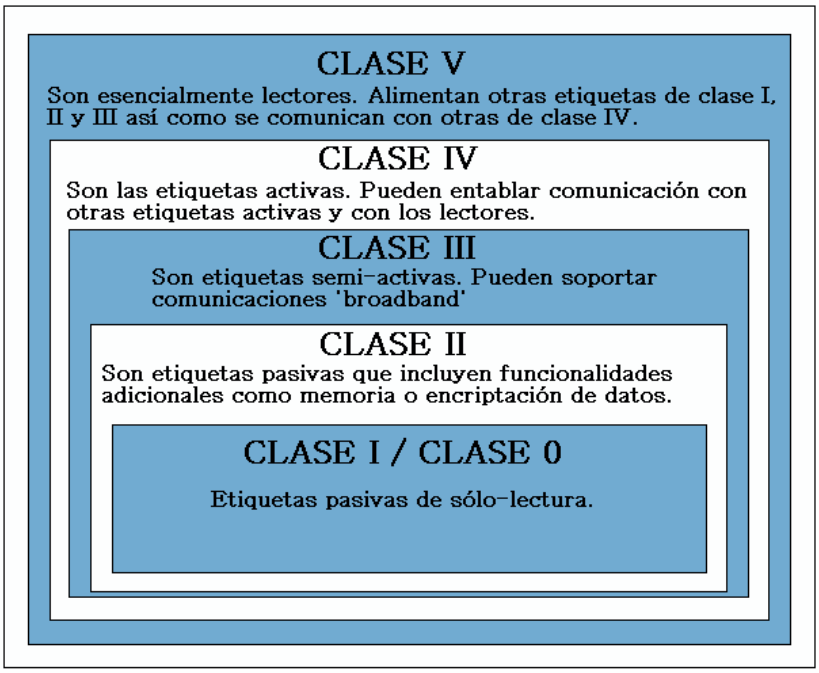

Figura 10: Tipos de etiquetas definidos por EPCglobal [11].

- Clase 0/ Clase 1: *tags* pasivos de solo lectura.
- Clase 2: *tags* pasivos con funcionalidad adicional como memoria o encriptación.
- Clase 3: *tags* semi-pasivos que pueden soportar comunicaciones de banda ancha.
- Clase 4: *tags* activos que pueden mantener comunicaciones de banda ancha tanto con lectores como con otros *tags*.

Las especificaciones de la capa física para el lector establecen el tipo de comunicación, de modulación y de codificación que se han de emplear según el tipo de *tag* con el que se vaya a realizar la comunicación y el estándar que se emplee.

2.8.2.2. Estándar de comunicación EPCglobal Class1 Gen2

La norma de comunicaciones EPCGlobal UHF Class-1 Gen-2, es el nuevo estándar de EPCglobal y supone una mejora de las versiones anteriores. Define las especificaciones físicas y lógicas para sistemas RFID activos y pasivos que trabajan a una frecuencia de 860 a 960 MHz [6][20]. Es un protocolo basado en Aloha Ranurado, caracterizado por:

- Simplicidad y robustez, debido a que la complejidad del algoritmo recae sobre el lector.
- No supone un coste hardware añadido, ya que los *tags* actuales en el mercado cumplen los requisitos para su implementación.
- Adecuado para los sistemas en los que el lector no conoce de antemano el número de *tags* que se encuentran en su zona de cobertura.
- Se obtienen mejores resultados en tiempo de identificación, utilización del canal y probabilidad de colisión.

El protocolo define dos capas: la capa física y la capa de identificación del *tag*. En la capa física se definen los requisitos hardware de un sistema RFID (Lector, antenas, *tags*, etc.). Conforme a la especificación de EPCGlobal, el *tag* cumplirá los requisitos de comunicación, implementará los comandos obligatorios definidos y modulará la señal *backscatter* [6] solo después de recibir los comandos requeridos procedentes del lector y conforme a la regulación local de radiofrecuencia.

El tipo de comunicación basado en modulación *backscatter* es empleado tanto por los *tags* pasivos como los activos. El lector envía una señal de radiofrecuencia continua que contiene corriente alterna y el reloj de la señal a la frecuencia a la que trabaja el *tag*. Éste obtiene la energía transformándola a corriente continua y alimentando los sistemas.

La información que envía el *tag* al lector se puede desglosar en paquetes de 96 bits para el número de identificación único (EPC), 16 bits para el código de control de errores (CRC) y 24 bits de código de habilitación/deshabilitación.

- *Paquetes EPC*: Los números EPC siguen el formato especificado en la norma general tal y como se ha visto en el capítulo anterior, haciendo posible la verificación de los datos leídos y escritos en el *tag*. La comprobación se realiza en el

interfaz, al contrario de lo que ocurría en protocolos anteriores, lo que permite que la ejecución sea más rápida.

- *Control de errores*: El control de redundancia cíclica le permite saber al lector si ha recibido correctamente el paquete.
- *Seguridad en EPC*: El código de habilitación/deshabilitación (*kill code*) permite deshabilitar el *tag* frente a un ataque externo, evitando que el usurpador acceda a la información que contiene.

Un lector puede realizar básicamente tres operaciones sobre una población de *tags*: selección (elige un *tag* de toda la población), inventario (operación de identificación de todos los *tags*) y acceso (operación de lectura/escritura en el *tag*). Para ello se ha de seguir un protocolo de comunicación establecido por el estándar EPCGen2 que consta de cuatro pasos:

- Una solicitud (*query*).
- La generación de un número aleatorio (r) de 16bits (*RN16*).
- Un reconocimiento (*ACK16bits*).
- El dato de identificación del *tag* (*EPC-data*).

Antes de comenzar el ciclo de identificación, el lector envía un paquete *broadcast* a todos los *tags* que se encuentren en su radio de acción, señal a la que únicamente contestarán, enviando su ID, aquellos *tags* a los que les corresponda según el paquete recibido del lector. Una vez hecho esto comienza el ciclo de identificación.

Primero el lector manda un paquete de solicitud de 4bits (Q=[0,15]) al *tag*. Con esto, el *tag* genera un número aleatorio r ($r=[0, 2^{(Q-1)}]$). Este número será un contador que al llegar a cero hace que el *tag* envíe su ID al lector. Si éste lo recibe sin problemas, es decir, sin colisiones entre el envío de los ID de otros *tags*, manda un paquete ACK, que indica al *tag* que ha de enviar sus datos almacenados (PC, EPC y CRC).

Una vez que el lector recibe los datos del tag y los verifica, pueden ocurrir diferentes cosas. Si son correctos, el lector envía un paquete QueryRep, que indica a los *tags* que aun no han sido identificados que comienza un nuevo ciclo de identificación, por lo que han de decrementar el valor de su r. Si los datos son incorrectos, enviará un paquete NACK (lo que indica al *tag* que pretendía identificarse que no lo ha hecho de forma correcta, quedando fuera de la población de *tags* a analizar), seguido de un paquete QueryRep, para continuar identificando al resto de *tags*.

2.9. MECANISMOS DE CONTROL Y SEGURIDAD EN LA COMUNICACIÓN RFID

En el proceso de comunicación de los sistemas RFID, es necesario tener en cuenta aspectos como las interacciones que pueden producirse entre varios *tags* RFID al responder a una llamada del lector, la seguridad y robustez del sistema frente a los posibles ataques externos que puedan producirse o los protocolos de comunicación más comunes, como es el caso del EPC Gen2 del que se habla más adelante.

2.9.1. Protocolos anticolisión

Los lectores pueden requerir la exploración de los *tags* de su entorno o la selección de uno en concreto de entre muchos candidatos posibles. Al intentar trabajar con un conjunto de *tags* surge el problema de las colisiones en el acceso al medio. Este conflicto aparece cuando varios *tags* intentan identificarse en el mismo instante temporal, por lo que se hace necesario el uso de protocolos anticolisión. [22]

- Protocolos deterministas:

En este tipo de protocolos se emplean algoritmos de "búsqueda en árbol", consistentes en ir descomponiendo la población de *tags* bajo su cobertura en pequeños grupos, empleando técnicas de segmentación o *splitting*. El lector envía un bit, al que responderán todos aquellos *tags* cuyo número de identificación comience por ese valor. Si en dicha respuesta se detectan colisiones, el lector segmentará la población de *tags* en dos subconjuntos enviando un segundo bit, al que contestarán aquellos *tags* cuyo identificador comience por esos dos bits enviados. Así se continuará sucesivamente hasta que el lector no detecte colisiones en un subgrupo, lo que indicará que el lector ha identificado con éxito todos los *tags* de ese subconjunto. Una vez que se ha finalizado la identificación de un subgrupo se continúa con los restantes.

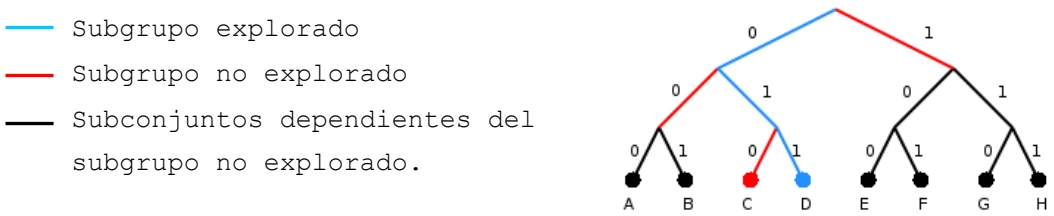

Figura 11: Ejemplo de búsqueda en árbol.

- Protocolos probabilísticos:

Este tipo de protocolos basan su resultado en decisiones tomadas al azar, de manera, que por promedio, se obtienen soluciones adecuadas. La gran mayoría de las veces los sistemas RFID operan en entornos en los que desconocen el número de *tags* que van a competir por identificarse, situaciones en las que es más adecuado emplear este tipo de protocolos. Existen varios tipos, todos ellos basados en el protocolo Aloha.

- **Aloha**: La idea básica es que en el momento en el que el lector excita a los *tags* que se encuentran en su entorno, éstos envían su código de identificación. Si la respuesta de dos o más *tags* se solapa en el tiempo se produce un colisión, la recepción de información no se realiza con éxito, simplemente se pierde.
- **Aloha ranurado (SA)**: Se basa en la emisión periódica de referencias temporales, que hacen que los *tags* esperen al comienzo de una "ranura" temporal para emitir su código, de forma que se eviten o reduzcan las colisiones entre *tags*.
- **Aloha ranurado por trama (FSA)**: Variante del anterior, donde los *spots* se agrupan en tramas consecutivas.
- **CSMA/CD**: Evolución del protocolo Aloha, que pretende mejorar las prestaciones de éste, ya que el *tag* escucha a la vez que transmite, por lo que si detecta colisión no emite señal, produciéndose una ganancia en tiempo y consumo.

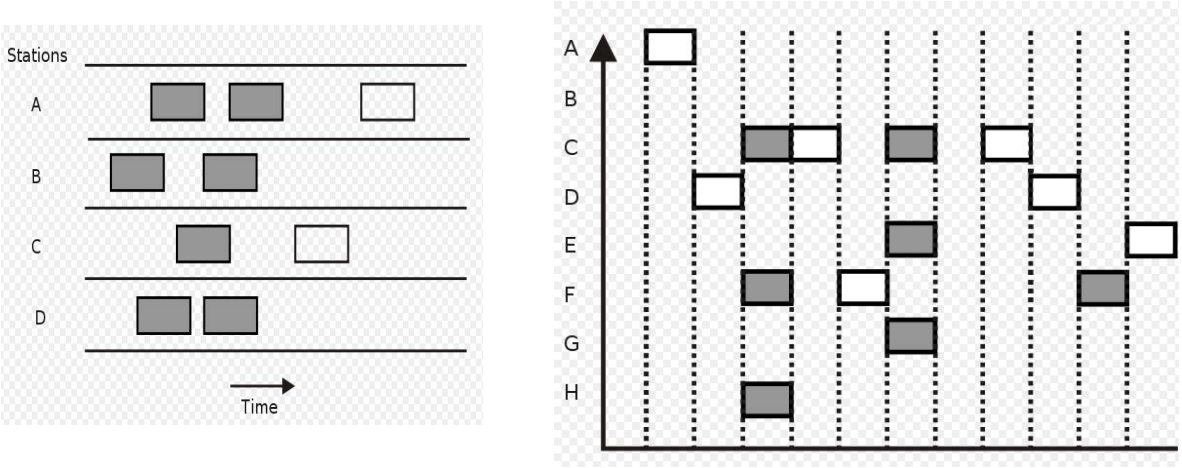

Figura 12: Aloha puro y aloha ranurado (los spots sombreados indican colisión)

2.9.2. La seguridad y los sistemas RFID

2.9.2.1. Ataques y amenazas

Uno de los problemas que pueden presentar los sistemas RFID es el de la seguridad, ya que en muchas ocasiones, con emplear un lector a la frecuencia adecuada se puede extraer la información de cualquier *tag* RFID o incluso llegar a modificarla.

Es importante por ello ser consciente de los ataques que pueden sufrir este tipo de dispositivos con el fin de hacerles frente mediante protocolos de seguridad adecuados. Algunas de las amenazas más comunes se muestran a continuación.

- **Desactivación del *tag***: Es un ataque a la disponibilidad del *tag*.
- **Clonación del *tag***: Es un ataque a la integridad del *tag*, debido a que el atacante captura la información de identificación de éste.
- **Seguimiento del *tag***: Es un ataque a la privacidad, ya que el atacante es capaz trazar el flujo del *tag*.
- **Ataques de *replay***: Es un ataque a la integridad del *tag*, se envían tramas que se han observado en una comunicación anterior.
- **MitM (*Man in the middle*)**: Es un ataque a la integridad del *tag*, el atacante se interpone entre el *tag* y el terminal, siendo capaz de añadir, modificar o eliminar tramas.

2.9.2.2. Requisitos de seguridad

Los requisitos básicos de una tecnología como la de los sistemas RFID son los siguientes:

- **Confidencialidad**: ha de impedirse el acceso a toda aquella persona no autorizada tanto al canal de comunicación como a la información contenida en el *tag* RFID.
- **Integridad**: ha de impedirse la manipulación de los datos, tanto de los almacenados como durante el tránsito de los mismos.
- **Disponibilidad**: ha de impedirse la interrupción o interferencia en la comunicación entre los *tags* y el lector.

Nota: Para más información sobre la seguridad y las amenazas consultar referencias [23] a [25].

- **Autenticidad**: ha de impedirse la falsificación de los datos.
- **Anonimato o privacidad**: ha de impedirse el rastreo en tiempo y espacio del portador del *tag*.

El trabajo realizado en este proyecto fin de carrera se centra en el desarrollo y análisis de protocolos de autenticación. En este capítulo se hace una introducción a los conceptos básicos mientras que una descripción más detallada se realiza en el capítulo 4. Los procesos de autenticación pueden ser de diversos tipos, es decir, de autenticación de datos, del *tag*, del usuario o del lector. La autenticación de *tag* también es conocida como autenticación interna (*client authentication* o *internal authentication* en inglés*)* y es un proceso en el cual una parte del sistema, el lector, pretende confirmar la identidad de otra parte, el *tag*, mediante la adquisición de evidencias que lo corroboren. En este caso se va a asumir que el lector está implícitamente autenticado.

A la hora de crear un protocolo de autenticación es importante tener en cuenta que las sesiones de comunicación a un mismo *tag* han de ser independientes, es decir, que el conocimiento por parte de un adversario de la información transmitida en un proceso de comunicación, no puede permitir la deducción o el descifrado de comunicaciones anteriores o posteriores:

- Seguridad hacia delante: se refiere a la seguridad de comunicaciones futuras. Es decir que un adversario no sea capaz de predecir comunicaciones futuras en el caso en el que logre acceder a la comunicación o clave presente.
- Seguridad hacia atrás: se refiere a la seguridad de comunicaciones pasadas. Es decir que un adversario no sea capaz de descifrar comunicaciones pasadas en el caso en el que logre acceder a la comunicación o clave presente.

Con respecto a estos dos últimos términos existe cierta ambigüedad, ya que hay controversia dentro de la comunidad criptográfica a la hora de definirlos [26], intercambiando la definición del término entre ambos en muchas ocasiones [27][28]. En este proyecto se ha tomado la notación considerada más intuitiva [28].

2.9.2.3. Clasificación de los protocolos de autenticación RFID

Utilizando la clasificación de los protocolos de autenticación según el nivel de complejidad de las operaciones que éste lleve a cabo [29], pueden dividirse en cuatro grupos diferentes, tal y como se muestra a continuación:

- ***Full-fledged***: permiten de la aplicación de las funciones criptográficas clásicas, como el cifrado simétrico o la clave pública.
- **Simples**: admiten la generación de número aleatorios y funciones *hash*.
- **Ligeros**: soportan la generación de números aleatorios, funciones simples como el Código de Redundancia Cíclica y operaciones sencillas bit a bit.
- **Ultraligeros**: admiten únicamente operaciones sencillas bit a bit (XOR, AND, OR…).

CAPÍTULO 3
METODOLOGÍA

En este capítulo se describen brevemente los pasos que se han llevado a cabo en la realización del proyecto, así como el lenguaje y las herramientas que han sido necesarias para su desarrollo.

3.1. FLUJO DE TRABAJO

Para la realización de este proyecto se ha seguido el siguiente flujo de trabajo. Pueden diferenciarse dos bloques de trabajo principales para los distintos protocolos basados en generadores de números pseudoaleatorios que se van a probar, el primero en el que se ha implementado el protocolo de autenticación y otro en el que se han realizado las pruebas para el protocolo, empleando diferentes generadores de números.

Inicialmente se ha realizado una búsqueda de algoritmos de autenticación (SLAP basado en exponenciación modular, protocolo Gossamer, protocolo BHKY, protocolos ultraligeros, etc) con el fin de determinar los que se van a implementar para alcanzar los objetivos que persigue este proyecto. Una vez seleccionados los protocolos de autenticación que se desarrollarán en este trabajo, se lleva a cabo el diseño de los mismos en un lenguaje de descripción hardware. Se realizarán las simulaciones necesarias (mediante Modelsim) hasta la comprobación del correcto funcionamiento de los diseños, así como una primera síntesis orientativa de los protocolos implementados mediante el programa ISE.

Una vez terminada esta fase, han de seleccionarse los generadores de números pseudoaleatorios que van a implementarse junto con el protocolo de autenticación para que éste funcione de manera adecuada, ya que hasta dicho momento las pruebas realizadas se habían llevado a cabo con un bloque denominado “generador”, cuya función era únicamente recibir y producir las señales oportunas para poder comprobar el diseño del protocolo.

Cuando tengamos seleccionados los generadores que se desean, se integrarán en el protocolo de manera oportuna, previamente modificados para adaptarlos a las necesidades de nuestro sistema, y uno a uno cada diseño completo será sintetizado empleando la herramienta de síntesis de Synopsys. Esto nos permitirá realizar una

comparación entre los recursos necesarios de área y consumo para cada uno de los sistemas, con el fin de determinar cual se adapta mejor a las necesidades y limitaciones que presentan los *tags* RFID.

A continuación se muestra de forma gráfica el flujo de trabajo anteriormente descrito.

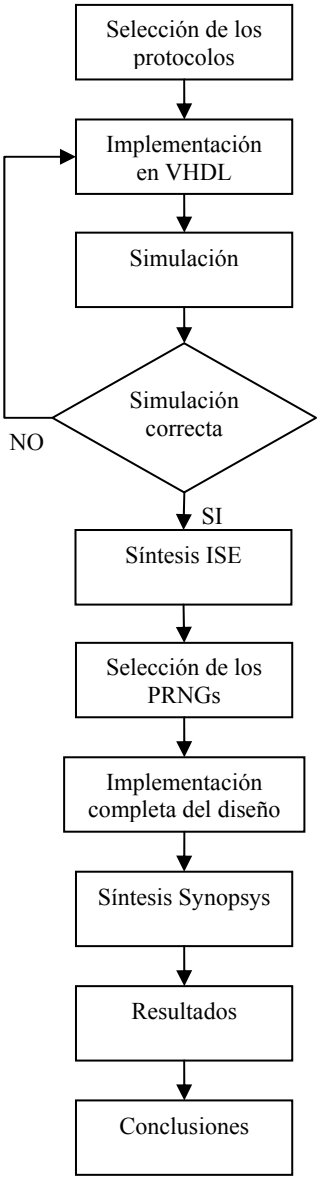

Figura 13: Diagrama de flujo de la implementación de los protocolos.

3.2. LENGUAJE VHDL

En este proyecto fin de carrera, se ha empleado el lenguaje VHDL para realizar la implementación del protocolo. A continuación se realiza una breve introducción a dicho lenguaje.

VHDL es el acrónimo resultante de la combinación de VHSIC (*Very High Speed Integrated Circuit*) y HDL (*Hardware Description Language)*. Es un lenguaje estandarizado por el IEEE (*Institute of Electrical and Electronics Engineers*), que se emplea para el diseño de circuitos digitales. Puede ser empleado para describir circuitos digitales en cualquier nivel de abstracción independientemente de si su implementación final es en un PLD (*Programable Logic Device*), FPGA (*Field Programmable Gate Array*) o ASIC (*Circuito Integrado de Aplicaciones Espec*íficas).

Es en la década de los setenta cuando se produce una gran evolución en los procesos de fabricación de los circuitos integrados surgiendo la tecnología MOS (*Metal Oxide Semiconductor*). El proceso de diseño era altamente manual, únicamente se empleaban herramientas como PSPICE para simular pequeños esquemas eléctricos, de modo que a medida que los procesos tecnológicos se hacían más complejos, estos métodos de diseño iban quedando obsoletos.

Con el fin de subsanar el desfase existente entre tecnología y diseño, empiezan a aparecer los primeros lenguajes de descripción hardware (HDL) en los años ochenta. Es entonces cuando el Departamento de Defensa de EE.UU. desarrolla un proyecto llamado VHSIC con el objeto de crear un lenguaje de descripción hardware. En el año 1982 las compañías IBM, Texas Instrument e Intermetrics obtuvieron la concesión para el desarrollo del lenguaje y de un conjunto de herramientas para su aplicación. En el año 1987 el lenguaje VHDL se convierte en el estándar IEEE 1076, para posteriormente ser actualizado en 1993 con la norma 1164, que incorpora cambios menores para facilitar su uso. Desde 1994, el lenguaje VHDL ha sido ampliamente aceptado, estando incluido en todas las herramientas de EDA (*Electronic Design Automation*) como las de Synopsys, Cadence, Xilinx, Mentor, etc, que soportan la especificación, diseño, síntesis y simulación con VHDL.

3.3. HERRAMIENTAS UTILIZADAS

Los programas que se han empleado para realizar este proyecto fin de carrera son ISE de Xilinx, ModelSim de Menthors Graphics, VMware Player de VMware Inc. y el sintetizador de Synopsys Inc.

El primero de ellos, ISE (*Integrate Software Environment*) [30][31][32], se ha utilizado para el diseño del código en el lenguaje de descripción hardware, así como una primera síntesis orientativa acerca del consumo de recursos. Este entorno de diseño consiste en una herramienta muy eficaz que permite realizar un diseño completo y su implementación en lógica programable. Las etapas principales son las de diseño, síntesis y simulación, tal y como se ve en la Figura 14.

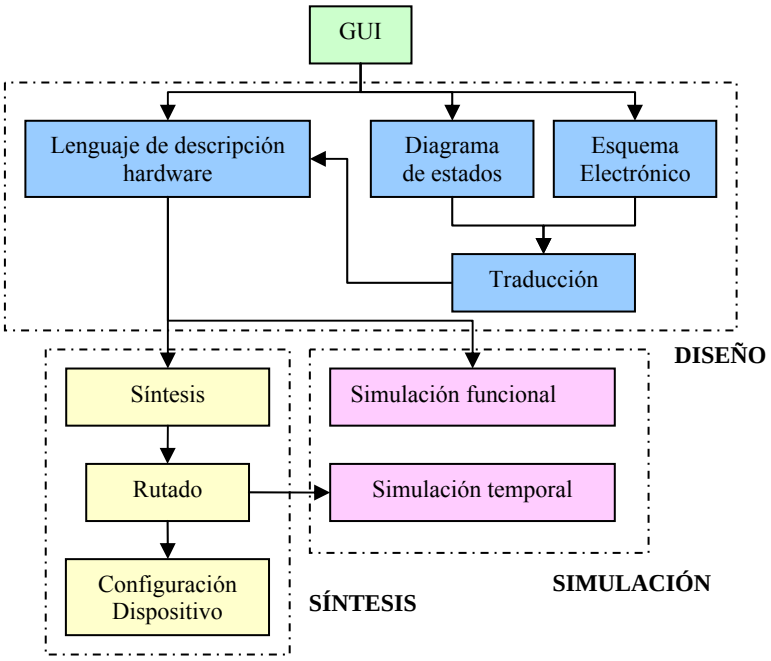

Figura 14: Etapas de diseño basado en lógica programable.

Una vez realizado el diseño, se ha comprobado el correcto funcionamiento del mismo empleando ModelSim. Este programa es una herramienta eficiente de simulación para diseños de circuitos digitales.

Posteriormente, se ha empleado el sintetizador de Synopsys, que trabaja sobre Linux, motivo por el cual se ha utilizado la máquina virtual que se describe más adelante. Este programa se ha empleado para la síntesis de los algoritmos diseñados, permitiendo realizar el estudio de área y potencia consumidas.

El hecho de emplear Synopsys [33][34][35] se debe a que el algoritmo que se está diseñando tiene como destino un *tag* RFID, el cual carece de una FPGA, por lo que éste ha de ser implementado en ASIC's, es decir en circuitos integrados hechos a medida para su uso en particular, que están basados en celdas estándar.

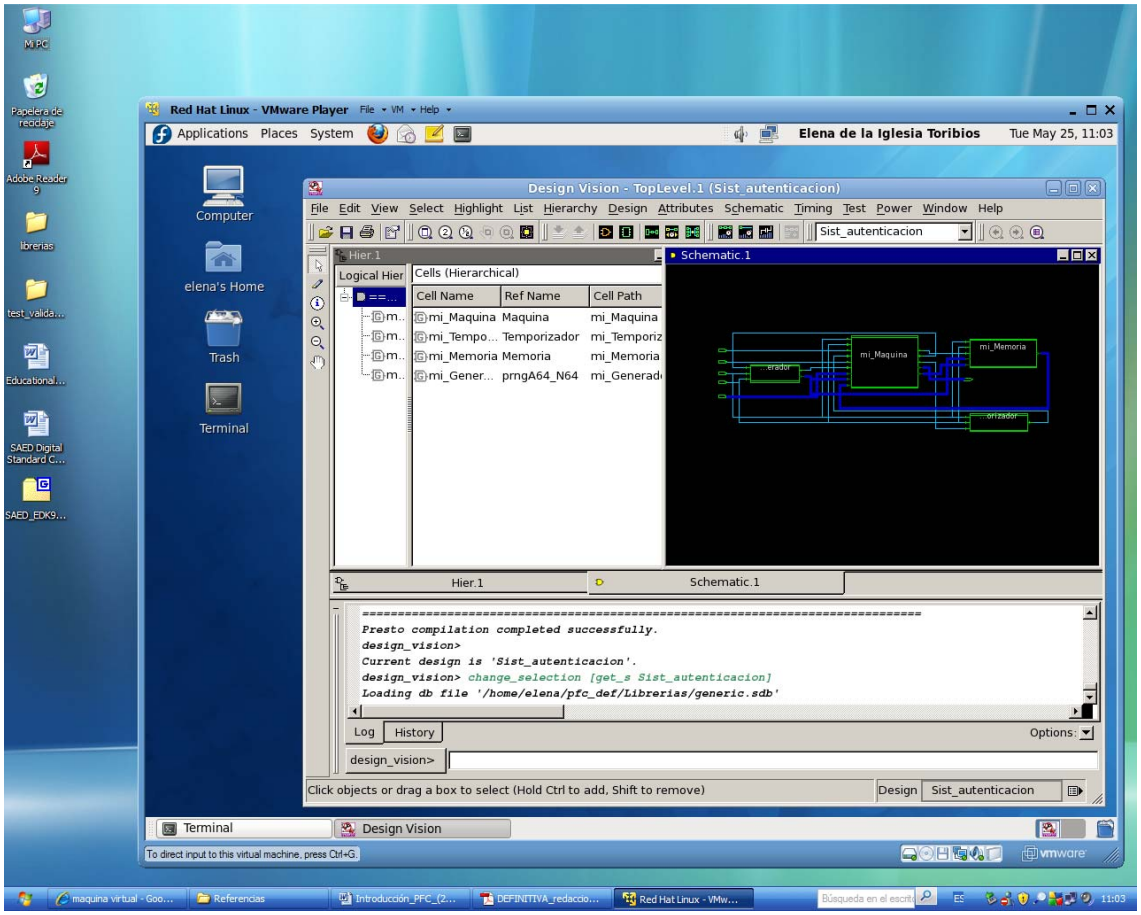

Figura 15: Programa Synopsys operando en una máquina virtual con Linux trabajando en un ordenador con Windows

El programa VMware Player se ha empleado para la utilización de una máquina virtual del sistema operativo Red Hat Linux Fedora. El objetivo de trabajar con una máquina virtual, en vez de instalar Linux en un ordenador, obedece a los beneficios que aportar la posibilidad de trabajar desde cualquier ordenador, sin necesidad de que éste trabaje con dicho sistema operativo o de que posea particiones para varios sistemas.

Una máquina virtual es un software que emula a una computadora y puede ejecutar programas como si fuese una real. Una de las características esenciales de las máquinas virtuales es que los procesos que ejecutan están limitados por los recursos y abstracciones proporcionados por ellas.

En este caso se ha empleado el software de virtualización proporcionado por VMware Inc. VMware Player es un producto gratuito que permite correr máquinas virtuales creadas con productos de VMware. Las máquinas virtuales se pueden crear con productos más avanzados como VMware Workstation.

3.4. BANCO DE PRUEBAS

Para la realización de las simulaciones de las diferentes implementaciones se ha utilizado un banco de pruebas, que estimula las señales de entrada del sistema de forma adecuada para comprobar su correcto funcionamiento.

Las señales estimuladas a través del banco de pruebas son las del reloj y el reset, como señales generales, y las señales de solicitud, de aceptación y de dato de entrada, las cuales simulan las respuestas que daría el lector que se comunique con el *tag*, ya que este proyecto fin de carrera está centrado en el funcionamiento del *tag*.

En los casos en los que ha sido necesario establecer el valor de identificadores del *tag* o de las semillas que son empleadas en el funcionamiento de los generadores de números pseudoaleatorios, se han utilizado secuencias extraídas al azar de los decimales del número pi, obteniendo así números aleatorios [36].

CAPÍTULO 4
PROTOCOLOS DE AUTENTICACIÓN

En el presente capítulo se van a mencionar diferentes protocolos de autenticación para sistemas RFID, así como sus posibles desventajas con respecto a la seguridad que aportan al sistema. Posteriormente se desarrollarán en profundidad los dos protocolos elegidos para ser implementados en este proyecto.

4.1. PROTOCOLOS DE AUTENTICACIÓN

Actualmente existen protocolos que se adaptan a los requisitos del estándar EPCGen2, pero muchos de ellos muestran debilidades frente a las amenazas de seguridad o son demasiado complejos. A continuación se mencionan algunos de ellos. [37][38][40] y sus limitaciones más importantes:

- **Protocolo de Chen-Deng**: Puede ser víctima de ataques de clonación del tag, ya que los datos del mismo se pueden obtener fácilmente mediante escucha.
- **Protocolo de Sun-Ting**: Vulnerable a ataques de *replay*, debido a que en este caso únicamente es el *tag* el que presenta aleatoriedad.
- **Protocolo de Qingling-Yiju-Yonghua**: En este protocolo la información puede ser fácilmente manipulada o el *tag* clonado, debido a que emplea CRC16 como cifrador de datos.
- **Protocolo de Seo-Baek**: Propone dos protocolos, ambos vulnerables a ataques de réplica.
- **Protocolo de Choi-Lim**: Presenta dos debilidades, el lector puede ser suplantado y puede ser objeto de ataques relacionados con las claves.
- **Protocolo de Li et al.'s**: Presenta vulnerabilidad frente a ataques de *replay*, así como debilidad frente a la revelación de las claves.
- **SLAP**: Basado en exponenciación modular requiere un uso de recursos demasiado elevado para su aplicación en RFID, donde el área disponible en el *tag* es muy limitada.
- **Protocolo Gossamer**: Vulnerable frente ataques de replay y de desincronización, puesto que el *tag* carece de un generador de números aleatorios, siendo únicamente el lector el encargado de su generación.

Observando las limitaciones que presentan estos algoritmos de autenticación se han elegido para este proyecto, los dos protocolos que se describen en los siguientes apartados. Esta elección se ha debido a las características de seguridad que presentan, siendo aun así suficientemente ligeros para su implementación en *tags* RFID de bajo coste.

Ambos protocolos se basan en generadores de números pseudoaleatorios. La aleatoriedad que estos proporcionan a los datos, produce que un adversario tenga mucho más difícil predecir la información transferida durante la comunicación y por tanto se dificulta la deducción de las claves del *tag*.

Por otro lado, los algoritmos de cada uno de estos dos protocolos presentan características que les hacen robustos frente a ataques de replay, de desactivación e incluso frente a ciertos tipos de ataques activos como son los *MitM*.

4.2. PROTOCOLO LIGERO DE AUTENTICACIÓN RFID 1

El primer protocolo [39] implementado en este proyecto fin de carrera pretende cumplir los siguientes requisitos: adaptarse a los estándares EPCGen2, ser robusto frente a las amenazas externas y ser lo suficientemente ligero como para permitir su implementación en los limitados recursos que presentan los *tags* RFID pasivos.

4.2.1. Descripción del protocolo

Se parte de la base de que cada *tag* (T) comparte con el servidor (S) un generador de números aleatorios sincronizado (RNG), es decir, mismo algoritmo, misma clave y mismas semillas. T y S se autentican mutuamente mediante el intercambio de entre 3 (en el caso más optimista) y 5 números.

La seguridad del protocolo se basa en que los números creados por el generador de números aleatorios no pueden ser predeterminados por el adversario y en que T y S están sincronizados en todo momento. El protocolo soporta la autenticación mutua, resistencia frente a ataques de replay, así como seguridad hacia delante y hacia detrás.

Cada *tag* T, almacenará en memoria no volátil los siguientes datos:

- Dos números pseudoaleatorios: RN_1 y RN_2.
- Su identificador: ID_{tag}.

- El estado actual: g_{tag}.
- Un *flag* de 1bit que se activa al comenzar la comunicación: cnt.

Por otro lado el servidor almacenará en su base de datos para cada *tag*:

- Seis números pseudoaleatorios: RN_1^{cur}, RN_1^{next}, RN_2, RN_3, RN_4, RN_5.
- El identificador del *tag*: ID_{tag}.
- El estado actual del tag: g_{tag}.
- Un *flag* de 1bit que se activa al comenzar la comunicación: cnt'.

La lista de la base de datos estará doblemente indexada, mediante el valor actual del primer número pseudoaleatorio RN_1^{cur} y su siguiente valor RN_1^{next}, de forma que se pueda localizar el *tag* correspondiente mediante ambos números. Para inicializar las variables del *tag*, se generan dos números aleatorios sucesivos que serán RN_1 y RN_2 y el *flag cnt* se pone a cero. Lo mismo hará el servidor, poniendo *cnt'* a cero y generando seis números aleatorios sucesivos que se asignaran a las variables correspondientes. Esto debe realizarse de tal forma que en todo momento el *tag* y el servidor compartan un número, $RN_1=RN_1^{cur}$ o bien $RN_1=RN_1^{next}$. Cuando se deban actualizar los valores de las variables, se asignará RN_1^{next} a RN_1^{cur} y se generarán cinco nuevos números aleatorios para RN_2, RN_3, RN_4, RN_5, RN_1^{next}.

Aunque en este proyecto fin de carrera se va a desarrollar la parte del protocolo correspondiente al *tag*, ya que es la que presenta limitaciones de recursos, a continuación se va a mostrar el protocolo de ambas partes, *tag* y servidor.

Protocolo:

1. **El lector (R) envía la señal de solicitud Query al *tag* (T).**
   Entonces el *tag* asigna a la señal alarma el valor de la variable *cnt*, que inicialmente vale 0. Posteriormente *cnt* se actualiza a 1. El *tag* envía RN_1 al lector.
2. **El lector recibe RN_1 por parte del *tag* y lo compara con el valor almacenado en la base de datos.**

- Si $RN_1=RN_1^{cur}$ para un *tag* de la base de datos, entonces el lector asigna a la señal *alarm'* el valor de la variable *cnt'*, poniendo posteriormente *cnt'* a 1. El lector envía RN_2 al *tag*.

- Si RN_1=RN_1^{next} para un *tag* de la base de datos, entonces la señal *alarm'* se pone a 0, se actualizan los valores de los datos y se envía RN_2 al *tag*.
- Si no, se aborta el proceso.

3. El *tag* recibe RN_2 por parte del lector y lo compara con el valor almacenado.

- Si RN_2 es correcto entonces genera cinco números sucesivos, se asignan a las variables y se pone la variable *cnt* a 0.
    - Si alarma=0 entonces se envía RN_3.
    - Si no, se envía RN_4.
- Si el valor RN_2 recibido no coincide con el valor almacenado para este número en el *tag*, se aborta el proceso.

4. El lector recibe RNi por parte del *tag* y lo compara con el valor almacenado en la base de datos. Donde 'i' indica el número enviado en cada caso.

- Si RNi=RN_3 y alarma'=0 entonces actualizar los valores de los datos y aceptar el *tag* como *tag* autorizado.
- Si RNi=RN_4 entonces, enviar RN_3 y actualizar los valores de los datos.
- Si no, se aborta el proceso.

5. El *tag* recibe RN_3 por parte del lector y lo compara con el valor almacenado.

- Si es válido, se envía RN_5.
- Si no, se aborta el proceso.

6. El lector recibe RN_5 por parte del *tag* y lo compara con el valor almacenado.

- Si es válido entonces aceptar el *tag* como *tag* autorizado.
- Si no, se aborta el proceso.

Con el fin de simplificar la descripción del flujo del protocolo, no se han incluido las temporizaciones utilizadas. La finalidad de estas es poder abortar el proceso de comunicación, si pasado determinado periodo de tiempo t_h, no se ha recibido ninguna señal de comunicación.

Como se ha dicho anteriormente, el protocolo EPCGen2 de EPCglobal tiene cuatro pasos para la identificación (Q, RN16, ACK y EPCdata). Con el fin de que el presente protocolo cumpla con dicho estándar, se sustituyen RN16 por RN_1, ACK por RN_2, EPCdata por RN_3 (en el caso más optimista).

En la Figura 16 se muestra el diagrama del protocolo, incluyendo las temporizaciones, representadas con la variable t_h.

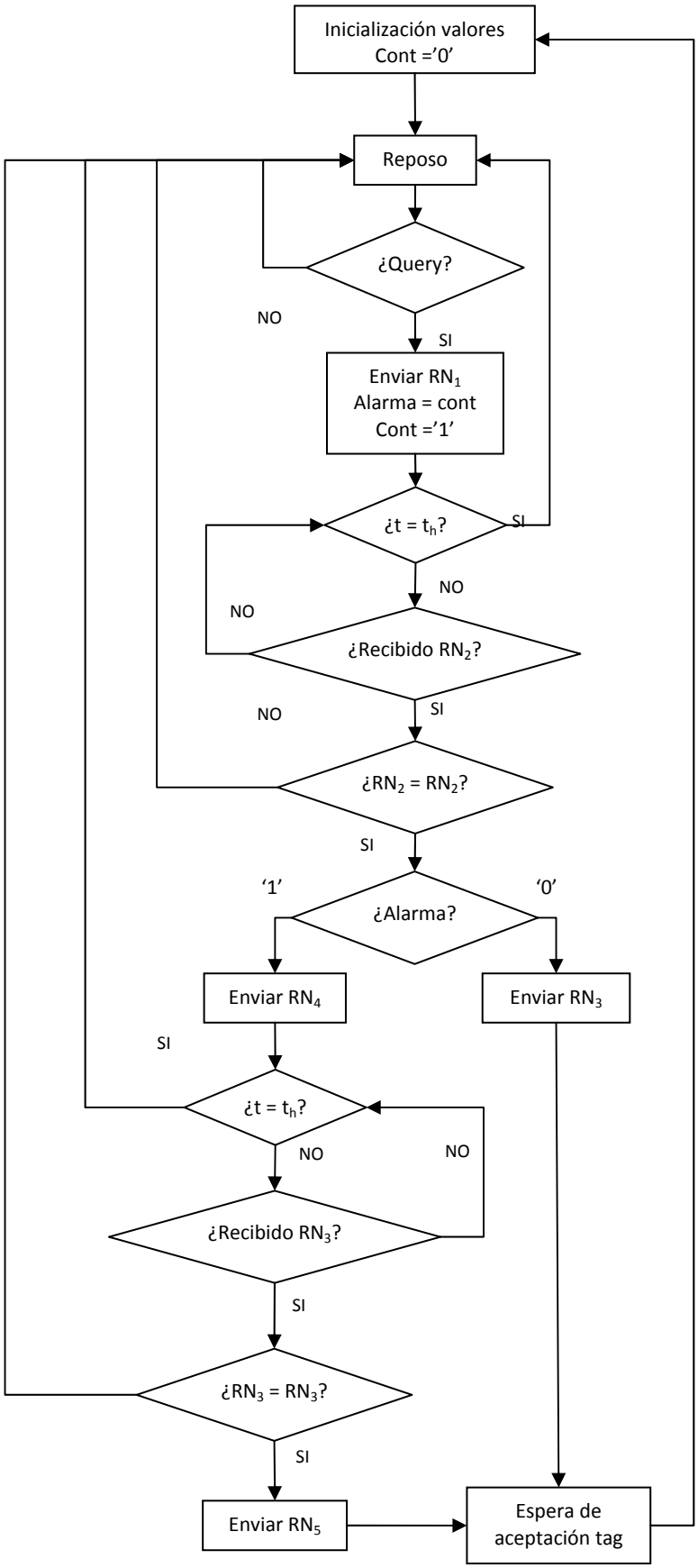

Figura 16: Diagrama de flujo del primer protocolo ligero implementado.

4.2.2. Implementación del protocolo

La arquitectura que se ha planteado para implementar este protocolo consta de los siguientes elementos. Unos registros para almacenar las variables RN_1, RN_2 y RN_3, el resto de valores RN_4 y RN_5 no es necesario almacenarlos, ya que según sean generados se enviarán al lector. Un generador de números aleatorios, acerca del cual se hablará más adelante. Un temporizador que establezca el tiempo que el *tag* espera a recibir una señal de contestación antes de abortar la comunicación. Y una máquina de estados que controle el flujo del protocolo.

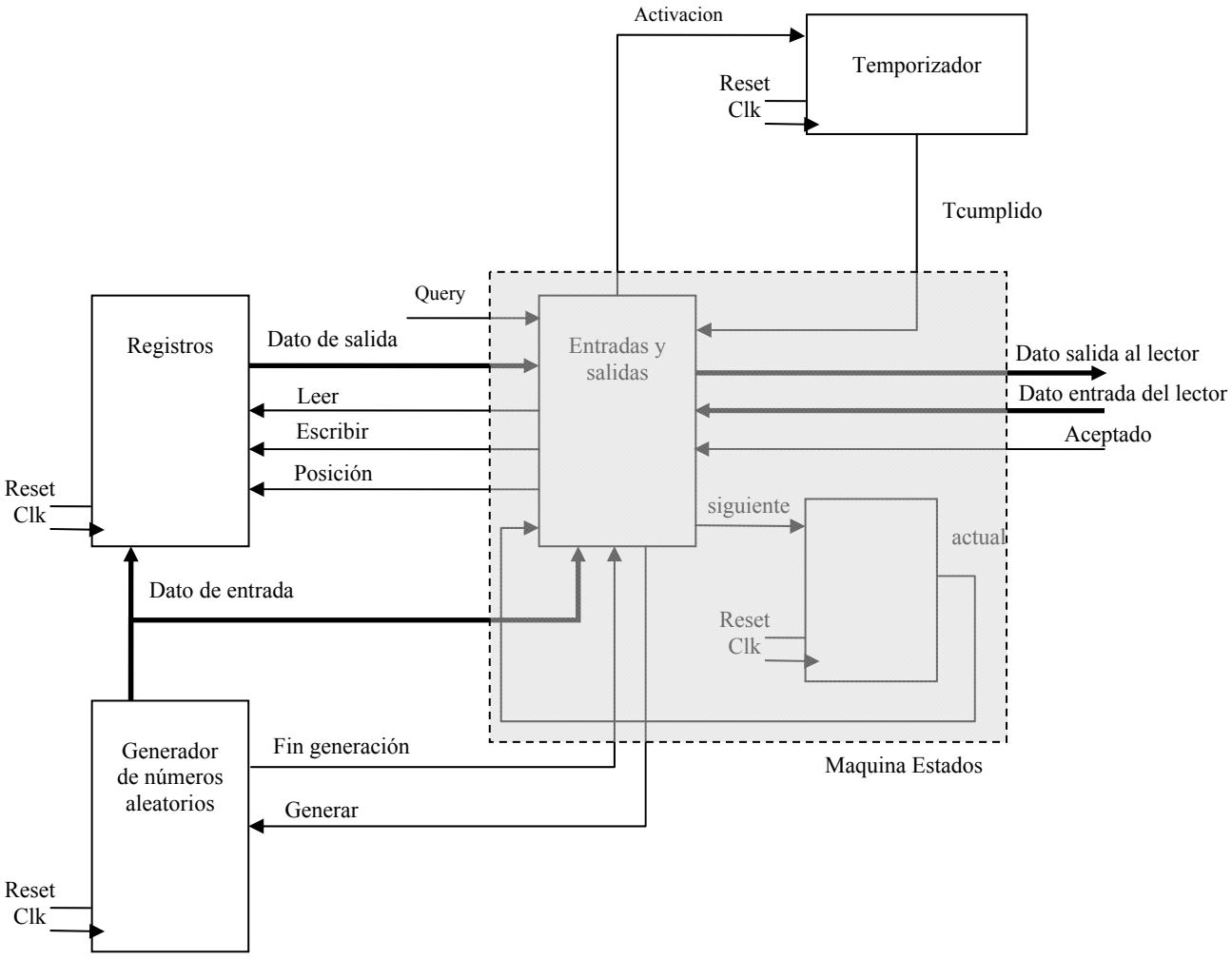

Figura 17: Arquitectura del primer protocolo.

La máquina de estados que ha sido necesario implementar para controlar el funcionamiento del protocolo consta de los siguientes estados. Se parte de un estado de inicialización en el que se asignan valores a RN_1 y RN_2, pasando seguidamente al estado de reposo, a la espera de una solicitud de comunicación por parte del lector.

Una vez que esto ocurre el *tag* envía RN_1 al lector, quedando a la espera de recibir su contestación, RN_2. Si ésta se recibe antes de que se alcance el t_h, se comparará con el valor que tiene almacenado el *tag*. En el caso de que dicha comparación sea afirmativa, se deberá valorar si alarma está o no activada, lo que indicará si se ha producido o no alguna incidencia durante la comunicación.

Si no está activada, pasará a enviarse RN_3, quedando de nuevo a la espera de la contestación por parte del lector, que esta vez será de aceptación o de rechazo del *tag*. Si está activada, RN_3 será memorizada por el *tag* y el valor que se enviará será RN_4, quedando a la espera de contestación. Una vez recibida la respuesta del lector, RN_3, se comparará con el valor almacenado. En caso de que la comparación sea positiva, se enviará RN_5 y el *tag* quedará en espera de la aceptación por parte del lector. En el momento en que el *tag* sea aceptado, se deberán actualizar los valores de RN_1 y RN_2.

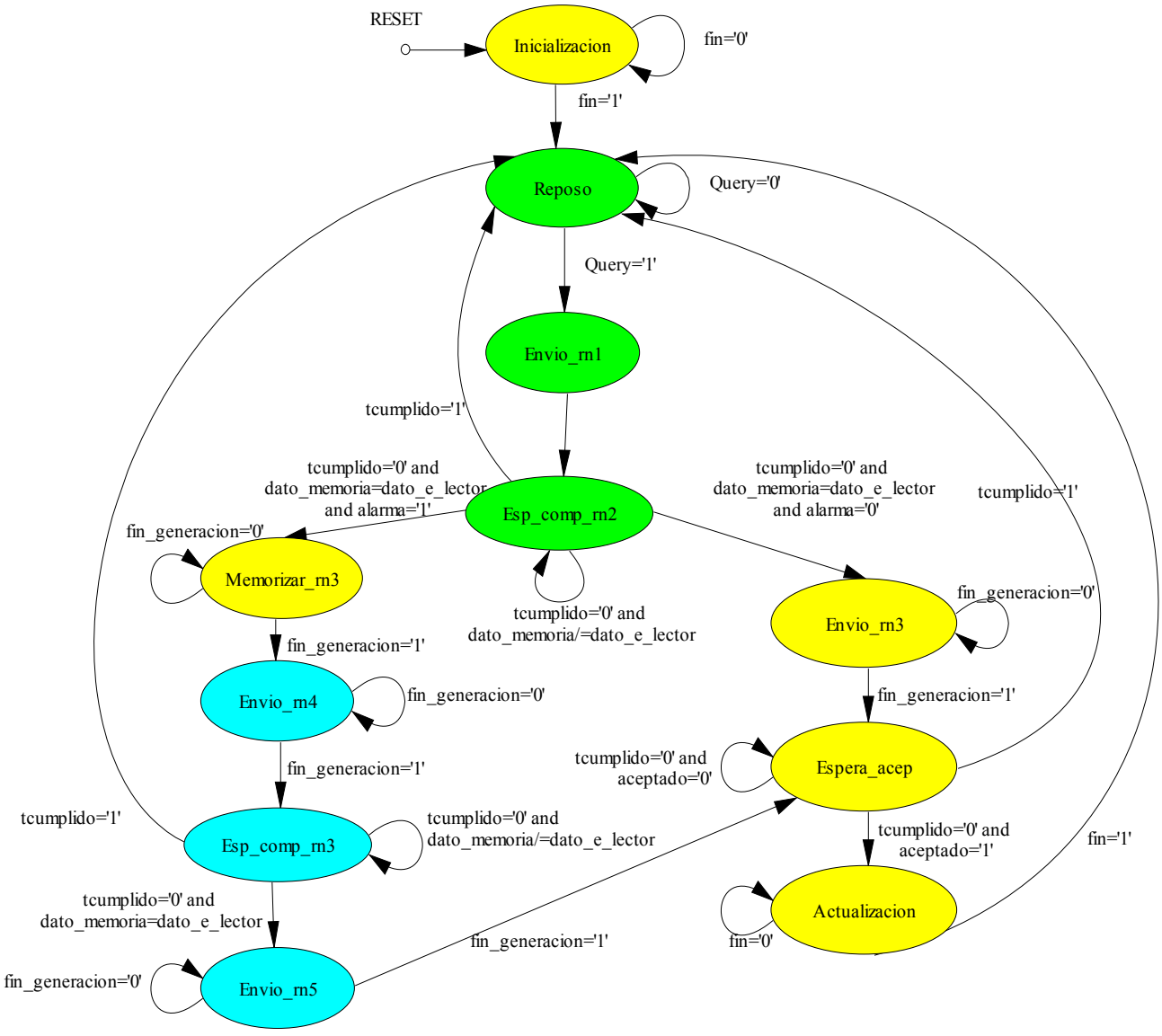

Figura 18: Máquina de estados implementada para el primer protocolo.

4.3. PROTOCOLO LIGERO DE AUTENTICACIÓN RFID 2

El segundo protocolo [40] implementado en este proyecto fin de carrera es, al igual que el primero, robusto frente a las amenazas externas, además de suficientemente ligero como para ser implementado en los tags RFID, cuyos recursos son limitados, pero las operaciones que se realizan en este protocolo de autenticación son más complejas que las del caso anterior.

4.3.1. Descripción del protocolo

Al igual que en el anterior protocolo, se parte de la base de que cada *tag* (T) comparte con el servidor (S) un generador de números aleatorios sincronizado. Los números calculados mediante dichos generadores no pueden ser determinados por el adversario. El protocolo soporta la autenticación mutua, así como seguridad frente a ataques de réplica, de trazabilidad y ataques *Denial-of-Service* (DOS).

Cada *tag* T, almacenará los siguientes datos:

- Una clave secreta x de *l*-bits.
- Un identificador de seguridad SID de *l*-bits.
- Un pseudónimo de identificación IDS de *l*-bits.

Por otro lado, el servidor almacenará en su base de datos para cada *tag*:

- Un identificador de seguridad SID de *l*-bits.
- El pseudónimo de identificación actual IDS_{new} de *l*-bits.
- El último pseudónimo de identificación IDS_{old} de *l*-bits.
- La clave secreta actual x_{new} de *l*-bits.
- La última clave secreta x_{old} de *l*-bits.
- Un *flag* de un bit, empleado para comprobar que el *tag* y la base de datos están sincronizados.

La siguiente notación es necesaria para la correcta comprensión del funcionamiento del protocolo que se describe más adelante.

- ***g(z)***: *g()* es un generador de número pseudoaleatorios y *z* es un número de entrada, es decir la semilla.
- ***g'***: es la salida del generador de números.

- ***rotate(p,w)***: el operando *p* se rota hacia la izquierda *w* posiciones.
- ***Left(s)***: selecciona la mitad izquierda de *s*.
- ***Right(s)***: selecciona la mitad derecha de *s*.

Tal y como se comentó en el protocolo anterior, en este proyecto fin de carrera se va a desarrollar la parte del protocolo correspondiente al *tag*, ya que es el que presenta limitaciones de recursos. Con el fin de comprender correctamente el funcionamiento del protocolo, a continuación se va a mostrar el protocolo de ambas partes, *tag* y lector-servidor.

Protocolo:

Este protocolo se encuentra dividido en cuatro etapas principales, dentro de las cuales se realizan las siguientes operaciones:

1. **Etapa 1:**

- El lector genera un número aleatorio R1 y lo envía al *tag* concatenado junto con la señal de solicitud Q.
- Una vez recibido esto, el *tag* generará otro número aleatorio R2, computa g'=*g*(R1||R2||x) y realiza SID'=*rotate*(SID, g').
- A continuación calcula R'=*Left*(SID' xor g')
- El *tag* envía al lector el identificador IDS y la concatenación de R2 y R'.

2. **Etapa 2:**

- El lector envía R1, R2, R' e IDS al servidor.
- El servidor busca si algún IDS almacenado coincide con el recibido.
    - Si IDS=IDS_{old}, se pone el *flag*=0 y x=x_{old}.
    - Si IDS=IDS_{new}, se pone el *flag*=1 y x=x_{new}.
- A continuación se calcula g'=*g*(R1||R2||x), SID'=*rotate*(SID,g').
- Entonces se comprueba si el valor recibido R'=*Left*(SID' xor g').
    - Si la coincidencia es positiva, se calcula R''=*Right*(SID' xor g') y se envía al *tag*.
    - En caso contrario, se cancela la comunicación.
- Si el *flag*=1, se asignan los valores IDS_{old}=IDS_{new} y x_{old}=x_{new}.

- Si el *flag*=0, se actualizan los valores $IDS_{new}=g(IDS||SID')$ y $x_{new}=g(x||g')$.

3. Etapa 3:

- El lector manda R'' al *tag*, el cual comprueba que este valor sea igual al calculado mediante *Right*(SID' xor g').
- Si el lector es autenticado correctamente el *tag* envía la señal *ok* al lector y actualiza los valores de IDS y x mediante $IDS=g(IDS||SID')$ y $x=g(x||g')$.
- En caso contrario, la autenticación no habrá sido correcta y se envía la señal *no*.

4. Etapa 4:

- Si el lector recibe la señal ok, la manda al servidor y éste le devuelve SID.
- En caso contrario el lector detiene el protocolo.

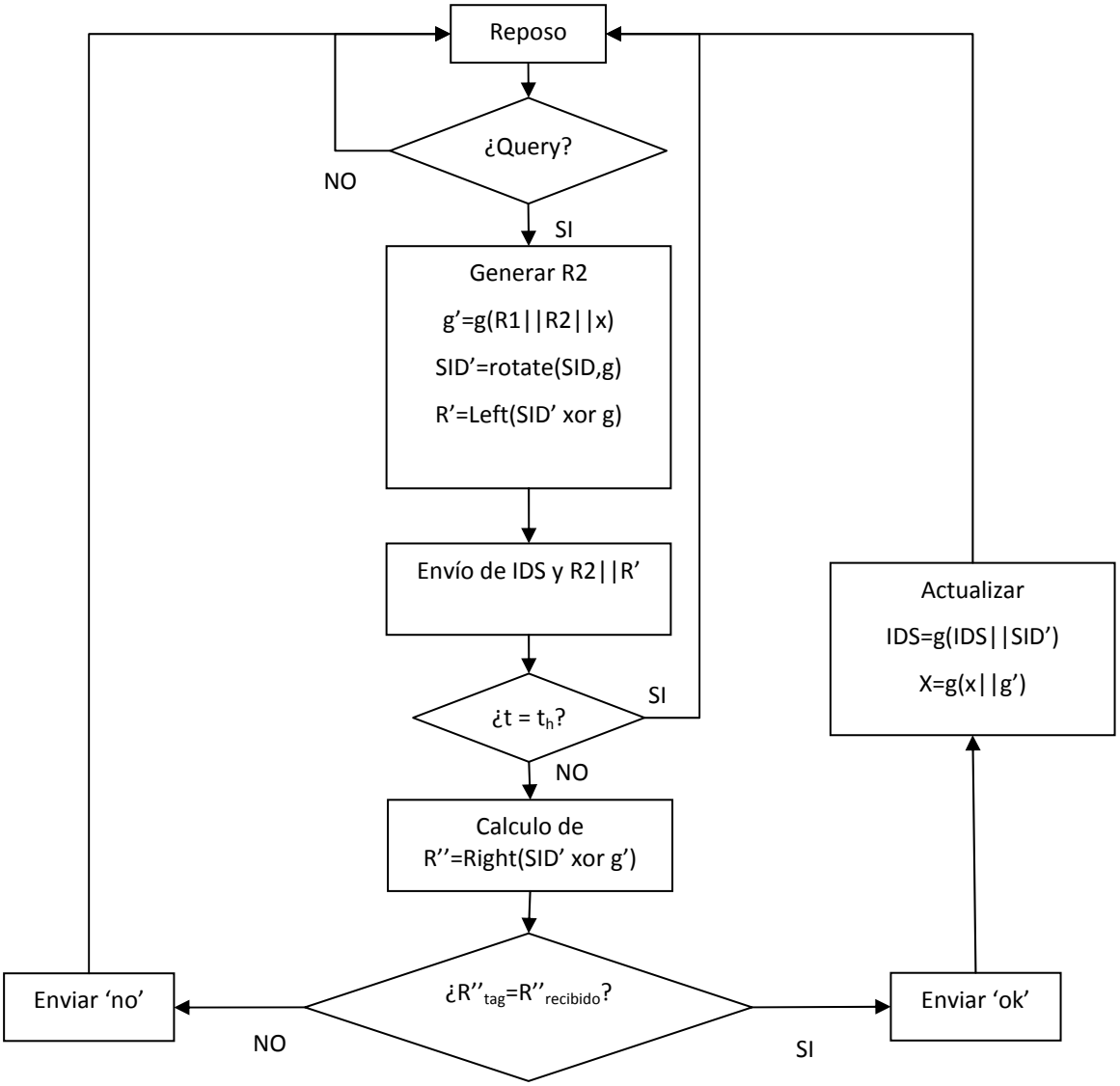

Figura 19: Diagrama de flujo del segundo protocolo ligero implementado.

4.3.2. Implementación del protocolo

La arquitectura que se ha planteado para implementar este protocolo consta de los siguientes elementos. Unos registros para almacenar tanto los datos iniciales del *tag*, como los calculados por el generador, así como aquellos enviados por parte del lector y que es necesario almacenar para operar con ellos. Un generador de números aleatorios, acerca del cual se hablará más adelante. Un temporizador que establezca el tiempo que el *tag* espera a recibir una señal de contestación antes de abortar la comunicación. Y una máquina de estados que controle el flujo del protocolo.

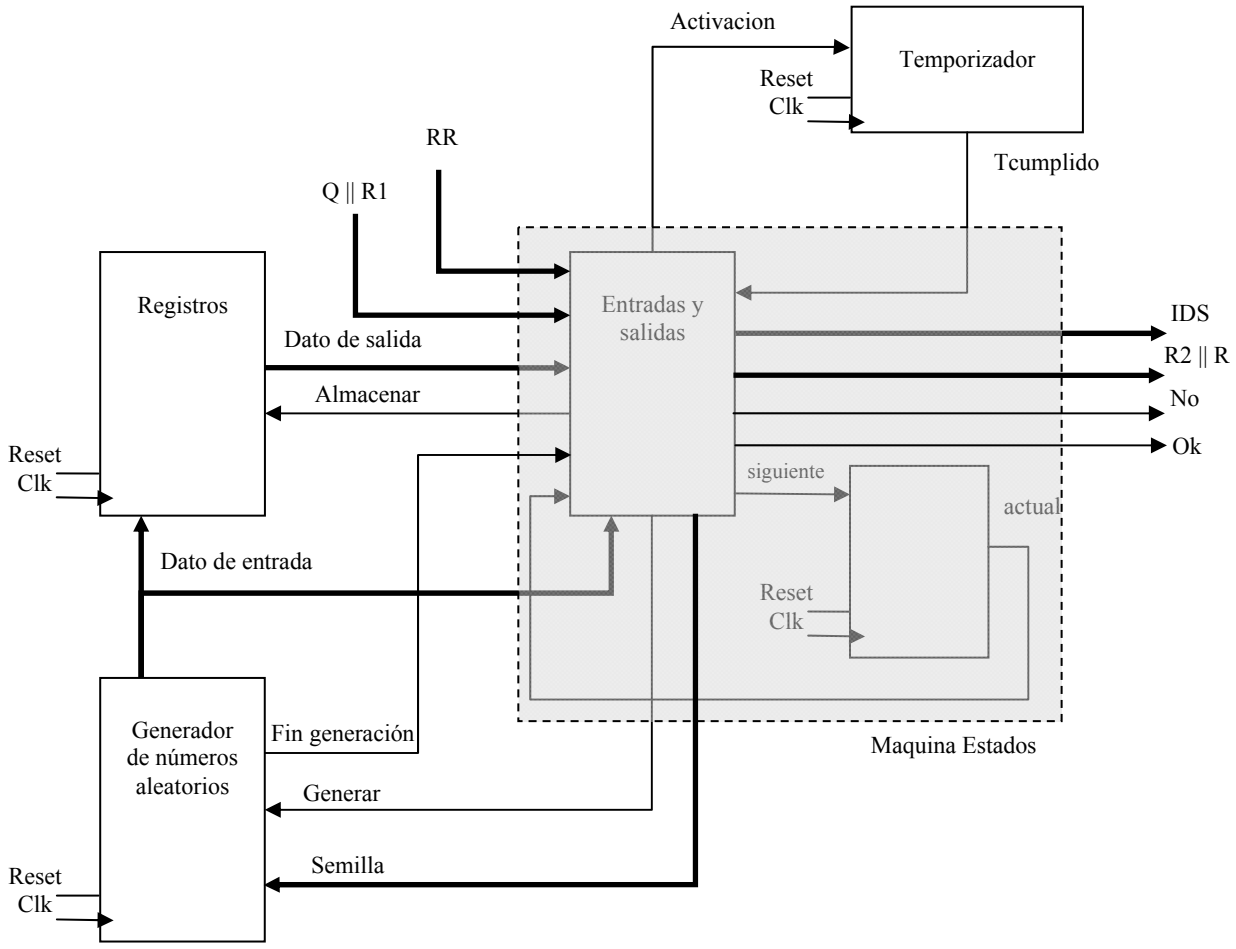

Figura 20: Arquitectura del segundo protocolo.

La máquina de estados que ha sido necesario implementar para controlar el funcionamiento del protocolo consta de los siguientes estados. Se parte de un estado de reposo en el que se permanece hasta que el *tag* recibe la señal de solicitud Q junto con el primer dato (R1) enviado por el lector, pasando al siguiente estado.

Una vez en el estado "generación_r2" se calcula R2 mediante el generador de números aleatorios y se almacena para operar con él más adelante. Cuando se ha terminado de calcular este número, se pasa al estado siguiente donde se obtendrá g'. Esta operación se llevará a cabo en varias fases. Esto dependerá del tamaño de las semillas con el que trabaje el generador de números pseudoaleatorios. Por ello los valores R1, R2 y *x* se enviarán concatenados o de forma independiente, según sea necesario. De los números que se obtendrán del generador, se cogerán los bits más significativos para conformar el número g'.

Habiendo finalizado este proceso, se pasa a la fase "calculo_sid_r", en la cual se procede a calcular SID' y R', tal y como se explicó anteriormente, enviando al lector los valores de R2 y R' concatenados, así como el de IDS. El *tag* queda entonces a la espera de la respuesta del lector. Si ésta no se produce dentro de un tiempo determinado la comunicación se aborta. Por otro lado, si el lector envía su respuesta, R'', el *tag* pasa a compararla con la R'' calculada por él mismo. En el caso de que sean diferentes, el *tag* detiene la comunicación, envía la señal "no" al lector y vuelve al estado de reposo. En el caso de que sean iguales, el *tag* envía la señal "ok" al lector, confirmando la autenticación y pasa al siguiente estado. En los dos siguientes estados, se realiza la actualización tanto de IDS como de 'x', enviando como semillas al generador la concatenación de IDS y SID' y de 'x' y g', respectivamente.

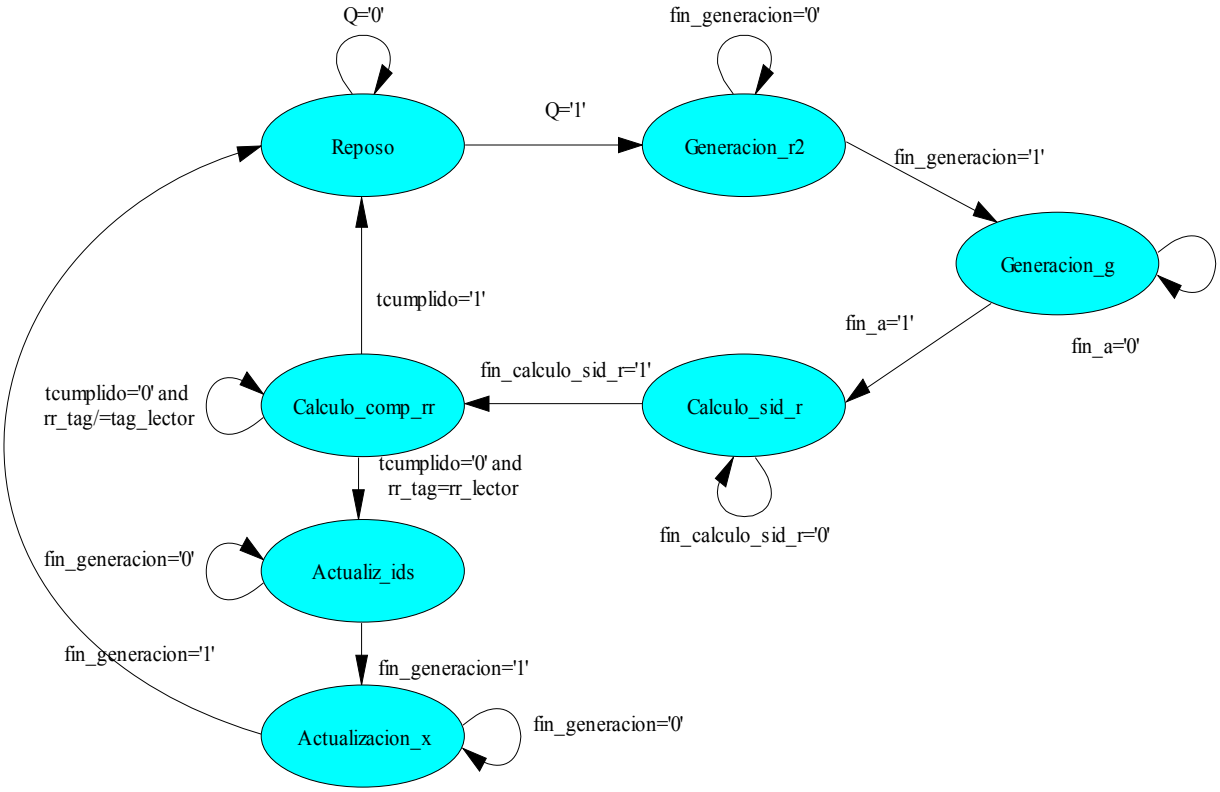

Figura 21: Máquina de estados implementada para el segundo protocolo.

4.4. GENERADORES DE NÚMEROS ALEATORIOS

Un generador de números aleatorios es un algoritmo matemático que produce una sucesión indefinida de números aparentemente aleatorios, es decir que satisfacen en las pruebas de aleatoriedad. La mayor parte de los generadores de números aleatorios son, en realidad, pseudoaleatorios, ya que precisan una semilla (X_0), a partir de la cual ir generando la sucesión de números deseada X_1, X_2, X_3, ... Si se parte de la misma semilla se obtendrá siempre la misma sucesión.

En este proyecto se han empleado en la implementación del protocolo, diferentes generadores de los que se disponía [41], con el fin de analizar la mejor solución para el caso que nos ocupa. Dichos generadores se describen a continuación.

4.4.1. Generador A

- **Opción A**

Este algoritmo es un PRNG ultraligero, basado en el uso de una función triangular. Parte de dos valores iniciales denominados semillas, que serán las variables de entrada (*x0* y *x1*). La constante *x0* será el valor inicial de *z* con el que se comenzará una serie de 64 iteraciones consistentes en la suma total de *z*, la constante *x1*, un desplazamiento de *z* a la derecha y un desplazamiento de *z* a la izquierda. Por último se realiza un filtrado de la salida con el fin de emplear el número de bits significativos necesarios.

```
x[0]= "semilla_1";
x[1]= "semilla_2";

//Non-linear function

z[0]= x[0];
for (i=0; i<64; i++){ z[0]= (z[0]>>1)+(z[0]<<1)+z[0]+x[1];}

//Filter output
// z[0] es la salida del generador
// Hay que quedarse con los bits menos significativos que son los
que varían más rápidamente
// Para el ejemplo de variables de 64 bits se cogen los 32 menos
significativos

z[0]= z[0]&0x00000000ffffffff;
```

Figura 22: Descripción del algoritmo del prng A.

- **Opción A1**

Este algoritmo es una variación del anterior. Partiendo de las mismas ideas, su implementación se basa en la utilización de un sumador de la mitad de bits que realiza los mismos cálculos que el generador de la opción A. El objetivo de esta modificación es reducir el área empleada por el diseño. Lo que se comprobará más adelante, una vez haya sido sintetizado.

4.4.2. Generador B

- **Opción B**

Del mismo modo que en el caso del algoritmo A, éste también está basado en una función triangular. Parte de dos valores iniciales, denominados semillas (*x0* y *x1*). Consta de dos funciones (A y B), cada una de las cuales se realiza durante 24 iteraciones con el fin de obtener números pseudoaleatorios criptográficamente seguros.

```
x[0]= "semilla_1";
x[1]= "semilla_2";

//Non-linear function

r1=24
/* A - Function */
z[0]=x[0]^x[1];
for(i=0;i<r1+){z[0]=(z[0]<<1)+((z[0]+(0x56AB0A72A4FB56AB))>>1);}

/* B - Function */
y[0]=x[1]^z[0];
for(i=0;i<r1++){y[0]=(y[0]>>1)+(y[0]<<1)+y[0]+(0x72A4FB56AB0A72A4);}

z[0]=z[0]^y[0];

//Filter output
// z[0] es la salida del generador
// Hay que quedarse con los bits menos significativos que son los que
varían más rápidamente
// Para el ejemplo de variables de 64 bits se cogen los 32 menos
significativos

z[0]= z[0]&0x00000000ffffffff;
```

Figura 23: Descripción del algoritmo del prng B.

La función A consiste en asignar a *z0* el resultado de realizar un XOR a nivel de bits entre *x0* y *x1*. A continuación se llevaría a cabo la suma de un desplazamiento a la izquierda de *z* más un desplazamiento a la derecha de la suma de *z* con un número en hexadecimal, repitiendo esto el número de veces indicado.

Una vez terminadas las 24 iteraciones de la función A, se pasa a la función B. Lo primero será asignar a *y0* el resultado de realizar una función XOR a nivel de bits entre *x1* y el valor obtenido de la función A. A continuación se llevaría a cabo la suma de: un desplazamiento a la izquierda de *y*, más un desplazamiento a la derecha de *y*, más *y*, más un número en hexadecimal, realizando las iteraciones pertinentes.
Por último se realizará una función XOR a nivel de bits entre el valor obtenido de la función A y el valor obtenido de la función B. De dicho resultado se selecciona el número de bits significativos necesarios.

- **Opción B1**

Este algoritmo es una variación del anterior. Partiendo de las mismas ideas, su implementación se basa en la utilización de un sumador de la mitad de bits que realiza los mismos cálculos que el generador de la opción B. El objetivo de esta modificación es reducir el área empleada por el diseño. Lo que se comprobará más adelante, una vez haya sido sintetizado.

- **Opción B2**

Este algoritmo es una variación del anterior. Partiendo de las mismas ideas, su implementación se basa en la utilización de un sumador de un cuarto de los bits que realiza los mismos cálculos que el generador de la opción B. El objetivo de esta modificación, al igual que la anterior, es reducir el área empleada por el diseño. Lo que se comprobará más adelante, una vez haya sido sintetizado.

4.4.3. Generador C

Del mismo modo que en el caso de los dos algoritmos anteriores, éste también está basado en una función triangular. Parte de dos valores iniciales, denominados semillas (*x0* y *x1*). En primer lugar se asigna a *z0* el valor de *x0*. Entonces se realizarán

una serie de operaciones dentro de un bucle, que se repetirán 32 veces, con el objetivo de obtener un grado suficiente de aleatoriedad en los datos generados. Estas operaciones comienzan por realizar la suma de un desplazamiento a la izquierda de *z0* más un desplazamiento a la derecha de la suma de *z0* con un número, asignando el resultado total a *y0*. A continuación se lleva a cabo la suma de un desplazamiento a la izquierda de *y0* más otro a la derecha más *y0* más *z0*.

Por último se realizará un XOR a nivel de bits entre el valor de *z0* y el valor de *y0*. A dicho resultado se realiza un filtrado con el fin de emplear el número de bits significativos necesarios.

```
x[0]= "semilla_1";
x[1]= "semilla_2";

//Non-linear function

r1=32
z[0]=x[0];

for(i=0;i<32;i++){z[0]=(z[0]<<1)+((z[0]+(0x56AB0A))>>1);
y[0]=z[0];
y[0]=(y[0]>>1)+(y[0]<<1)+y[0]+(x[1]); }

z[0]=z[0]^y[0];

//Filter output
// z[0] es la salida del generador
// Hay que quedarse con los bits menos significativos que son los
que varían más rápidamente
// Para el ejemplo de variables de 64 bits se cogen los 32 menos
significativos

z[0]= z[0]&0x00000000ffffffff;
```

Figura 24: Descripción del algoritmo del prng C.

CAPÍTULO 5

PRUEBAS, SIMULACIONES Y RESULTADOS

En este apartado se van a mostrar los resultados del análisis de los sistemas implementados. Primero se relizará la simulación, mediante el programa ModelSim, del código de los procolos generado en VHDL. Por otro lado se mostrarán los resultados obtenidos de la síntesis realizada en Synopsys.

5.1. CONSIDERACIONES PREVIAS:

La síntesis se ha realizado empleando la herramienta Design Vision del programa Synopsys, marcando la opción de un alto esfuerzo en área, necesario para las especificaciones de este proyecto. Esto se debe a que los *tags* RFID poseen un área limitada. Por lo general los *tags* de bajo coste tienen entre 5000 y 10000 puertas equivalentes, de las cuales tan sólo un rango de entre 250 y 4000 se emplean para funciones de seguridad [42]. Por ello, uno de los objetivos que se persigue es que los diseños de estos protocolos de autenticación se muevan en un rango de 3000-4000 puertas equivalentes (*Equivalent Gates*, EG).

Una puerta equivalente representa una unidad de medida que permite especificar la complejidad de fabricación independientemente de la tecnología que se emplee. Está basado en el número de puertas lógicas individuales que tendrían que estar interconectadas para realizar la misma función que el circuito implementado. En el diseño de circuitos digitales, para cada tecnología de fabricación, se utiliza una biblioteca de celdas estandar, la cual contiene una gran variedad de ellas.

Para las tecnologías actuales, el área de una puerta NAND se toma como área equivalente. De este modo, al dividir el área que ocupa el circuito que se implementa entre al área de la puerta NAND en la tecnología empleada, se obtiene el número de EGs. En nuestro caso la tecnología empleada es la de 90nm y el área de una puerta NAND es 5,5296 μm^2 [43].

Por otra parte, teniendo en cuenta que estamos trabajando con *tags* pasivos, ha de procurarse reducir el consumo de potencia lo máximo posible. Uno de los parámetros que más influye en este aspecto es la frecuencia del reloj [42][44].

Por ello, tal y como se indicó anteriormente, se va a tomar una frecuencia interna del reloj del *tag* RFID de 100 kHz.

Los datos que se van a observar una vez realizada la síntesis del diseño están relacionados con el área y la potencia. Los datos de área que se observarán son el número de puertos (entradas y salidas que presenta el diseño), el número de nodos (número de uniones o interconexionado), el número de celdas (número total de celdas o puertas lógicas usadas) y el número de referencias (los diferentes tipos de puertas lógicas empleadas en la síntesis del diseño). Además se obtendrá el área perteneciente tanto a la lógica combinacional como a la no combinacional (*Combinational Area, Noncombinational Area*), así como el área de interconexionado y las áreas totales (*Net Interconnect Area, Total Cell Area, Total Area*).

Por otro lado, de los datos relacionados con el consumo de potencia, observaremos la *Cell Internal Power* o potencia consumida por las celdas internamente, la *Net Switching Power* o potencia asociada al conexionado, la *Total Dynamic Power* que sería la suma de las dos anteriores y la *Cell Leakage Power* o pérdidas asociadas a las corrientes de fuga.

5.2. PROTOCOLO DE AUTENTICACIÓN 1

En primer lugar se van a mostrar los resultados obtenidos del análisis del protocolo de autenticación estándar, es decir, del protocolo implementado empleando un bloque genérico como PRNG, de este modo se podrá observar más adelante la influencia de la utilización de cada uno de los diferentes diseños de generadores de números pseudoaleatorios que se van a implementar.

Una vez realizado el diseño mediante lenguaje de descripción de hardware, se ha llevado a cabo la simulación del protocolo de autenticación. Para ello se ha creado una señal de reloj de 10 μs de periodo (f = 100 KHz). Puesto que no se requieren altas velocidades para este caso, se ha decidido emplear una frecuencia no muy elevada, procurando así reducir el consumo.

Teniendo en cuenta estos datos, se pasa a realizar la simulación del sistema mediante el programa ModelSim de *Menthors Graphics*.

5.2.1. Simulación ModelSim

A continuación se muestra la simulación del diseño en las diferentes etapas del sistema. Con el fin de comprobar todas las opciones de funcionamiento del algoritmo se han realizado varias pruebas. La primera se centra en el supuesto de que la comunicación entre *tag* y lector se lleva a cabo sin problemas ni errores, por lo que únicamente serán necesarios tres números para realizar el proceso de autenticación de forma satisfactoria, tal y como se explicó en el capítulo 4. En la segunda simulación se mostrará también el supuesto en el que sean necesarios los cinco números.

- Caso 1:

En esta simulación se puede observar el caso de funcionamiento óptimo del algoritmo. Una vez inicializados los valores de RN1 y RN2 (en rojo en (1)), se pasa al estado de reposo hasta la recepción de la señal de requerimiento *query*. En el momento en que eso ocurre, el *tag* envía el primero de los valores RN1, quedando a la espera de recibir el valor RN2, que una vez obtenido pasará a compararse (2) con el generado y almacenado anteriormente en la fase de inicialización. Si ambos coinciden, el *tag* genera y envía un tercer número RN3, pasando a la fase *espera_acep*, hasta que el lector confirme si la autenticación se ha llevado a cabo de forma correcta (3).

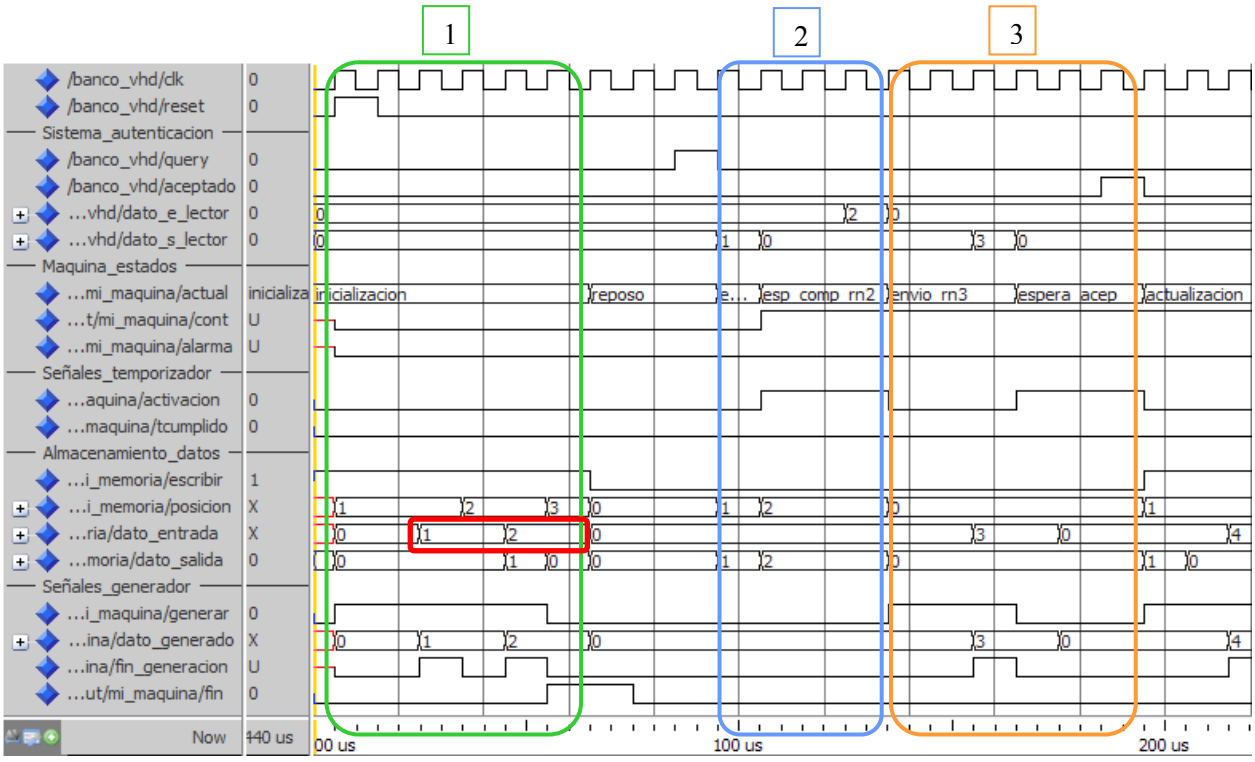

Figura 25: Simulación del caso óptimo de funcionamiento del primer protocolo implementado.

- Caso 2:

En esta simulación se puede observar el caso de funcionamiento del algotirmo con incidencias en la comunicación entre *tag* y lector. El proceso se desarrolla del mismo modo que en el caso anterior hasta la fase de comparación de RN2. En esta ocasión la comparación entre los valores no es correcta por lo que, pasado el tiempo de espera sin una comparación correcta (4'), la comunicación se interrumpe y se vuelve al estado de reposo de nuevo (4) como se ve en la Figura 26.

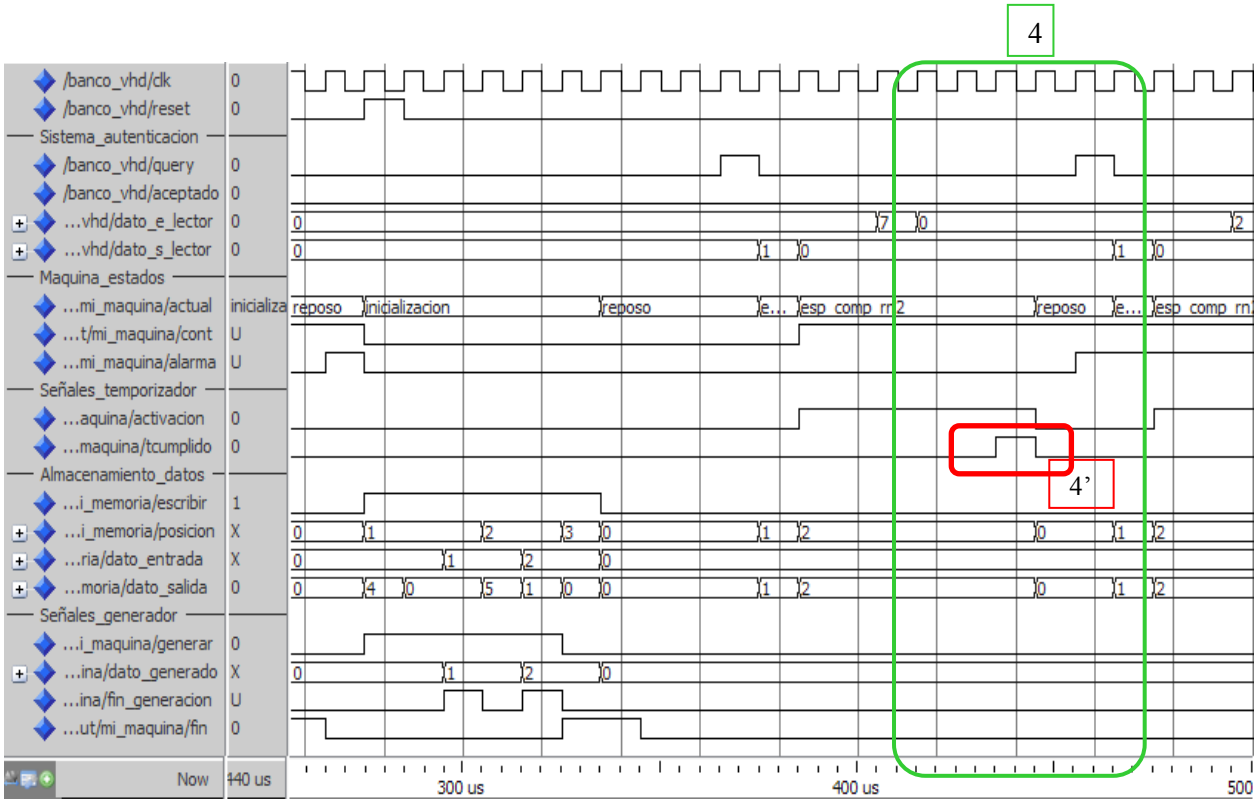

Figura 26: Simulación 1 del caso de funcionamiento con incidencias del primer protocolo implementado.

Debido a la longitud de la simulación, ésta se ha dividido en dos partes, con el fin de poder observar el funcionamiento del protocolo adecuadamente. En la Figura 26 se muestra el fallo en la comunicación entre el lector y el *tag* durante la comparación de RN2. Mientras que en la Figura 27 se observa el reinicio de la comunicación y el envío de los cinco números necesarios para llevar a cabo la autenticación una vez que ha ocurrido una incidencia en el proceso.

Habiendo vuelto al estado de reposo después de la incidencia en la comunicación, se permanece a la espera. Al recibir una nueva señal *query* por parte del lector, se reinicia el proceso, realizando las misma fases descritas anteriormente, pero después de realizar de forma correcta la comparación de los valores RN2, el algoritmo varía. En esta ocasión el número generado y enviado será RN4, quedando a la espera de recibir RN3 con el fin de realizar la comparación pertinente con el valor almacenado por el *tag* ((5) de la Figura 27). Una vez que la comparación es positiva se enviará RN5 y se esperará la confirmación por parte del lector de que el proceso de autenticación se ha realizado de forma correcta ((6) de la Figura 27).

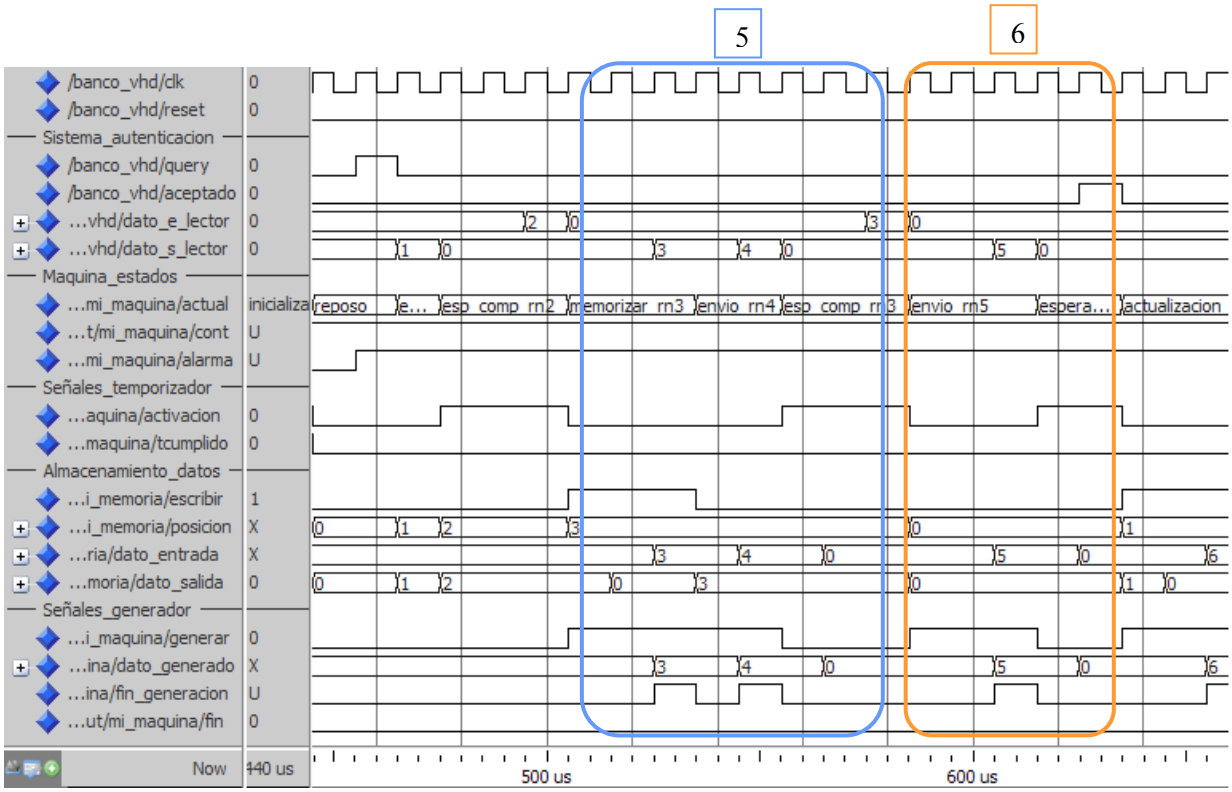

Figura 27: Simulación 2 del caso de funcionamiento con incidencias del primer protocolo implementado.

5.2.2. Síntesis Synopsys

Una vez comprobado el correcto funcionamiento del algoritmo implementado, se pasa a la fase de síntésis y obtención de resultados mediante Synopsys. Una vez compilado y sintetizado el diseño, el programa nos permite ver el esquema jerárquico del diseño.

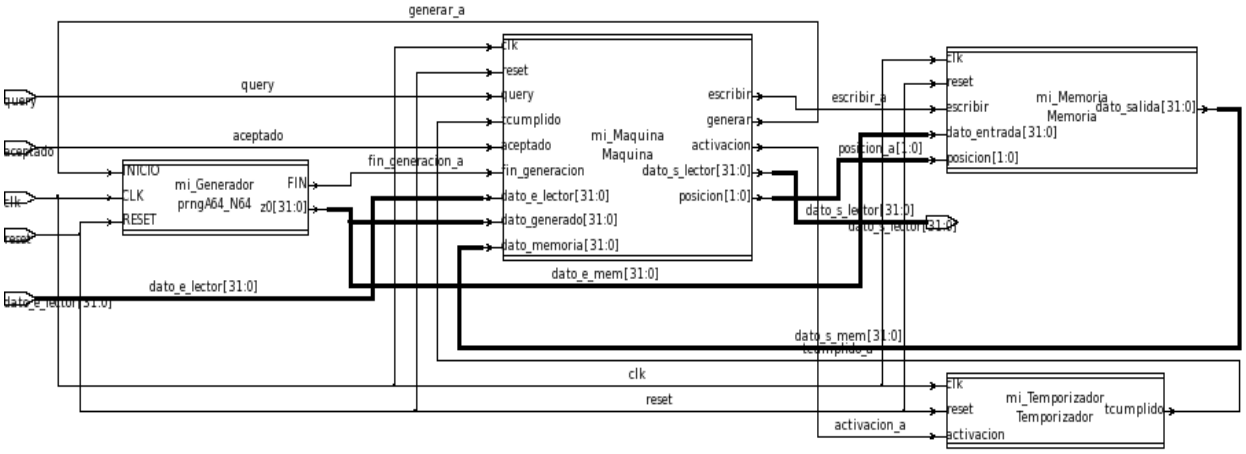

Figura 28: Esquema del diseño implementado para el primer protocolo.

Figura 29: Interfaz del diseño implementado para el primer protocolo.

Los datos que se muestran a continuación corresponden a la síntesis del diseño del protocolo sin generador de números pseudoaleatorios.

Datos de área de la síntesis:

Número de puertos	68
Número de nodos	595
Número de celdas	521
Número de referencias	31
Puertas equivalentes	1271
Área combinacional	3894,574 μm^2
Área no combinacional	3131,582 μm^2
Área de interconexión	340,878 μm^2
Área total de celdas	7026,156 μm^2
Área total	7367,034 μm^2

Datos consumo de la síntesis:

Potencia interna de las celdas	60.4950 nW	(78%)
Potencia de conmutación	17.0561 nW	(22%)
Potencia dinámica total	77.5511 nW	(100%)
Pérdidas de energías de las celdas	24.3459 uW	

5.3. PROTOCOLO DE AUTENTICACIÓN 2:

Del mismo modo que con el primer protocolo, inicialmente se van a mostrar los resultados obtenidos del análisis del protocolo de autenticación estándar, es decir, del protocolo implementado empleando un bloque genérico como PRNG, de este modo se podrá observar más adelante la influencia de la utilización de cada uno de los diferentes diseños que se van a implementar.

Una vez realizado el diseño mediante lenguaje hardware, se ha llevado a cabo la simulación del protocolo de autenticación. Para ello se ha creado una señal de reloj de 10 μs de periodo (f = 100 KHz) puesto que no se requieren altas velocidades para este caso, se ha decidido emplear una frecuencia no muy elevada, procurando así reducir el consumo.

Teniendo en cuenta estos datos, se pasa a realizar la simulación del sistema mediante el programa ModelSim de *Menthors Graphics*.

5.3.1. Simulación ModelSim

A continuación se muestra la simulación del diseño en las diferentes etapas del sistema. Con el fin de comprobar todas las opciones de funcionamiento del algoritmo se han realizado varias pruebas. La primera se centra en el supuesto de que la comunicación entre *tag* y lector se lleva a cabo sin problemas ni errores, por lo que el proceso de autenticación se realizará correctamente, tal y como se explicó en el capítulo 4. En la segunda simulación se mostrará también el supuesto en el que la comunicación se detenga por alguno de los motivos anteriormente explicados.

- Caso 1:

En la simulación de la figura 30 se puede observar el caso de funcionamiento correcto del algoritmo. Se parte del estado de reposo, en el que se permanece hasta recibir por parte del lector la señal Q|| R1 en la que Q='1', momento en el que se pasa al estado en el que se genera R2 por parte del *tag* (1). Una vez finalizado este calculo, se pasa a generar el valor g', proceso dividido en dos fases al estar trabajando con un prng de 64 bits en este caso, tal y como se indicó anteriormente (2). Se puede observar como se van envíando al generador las semillas adecuadas en cada caso (3).

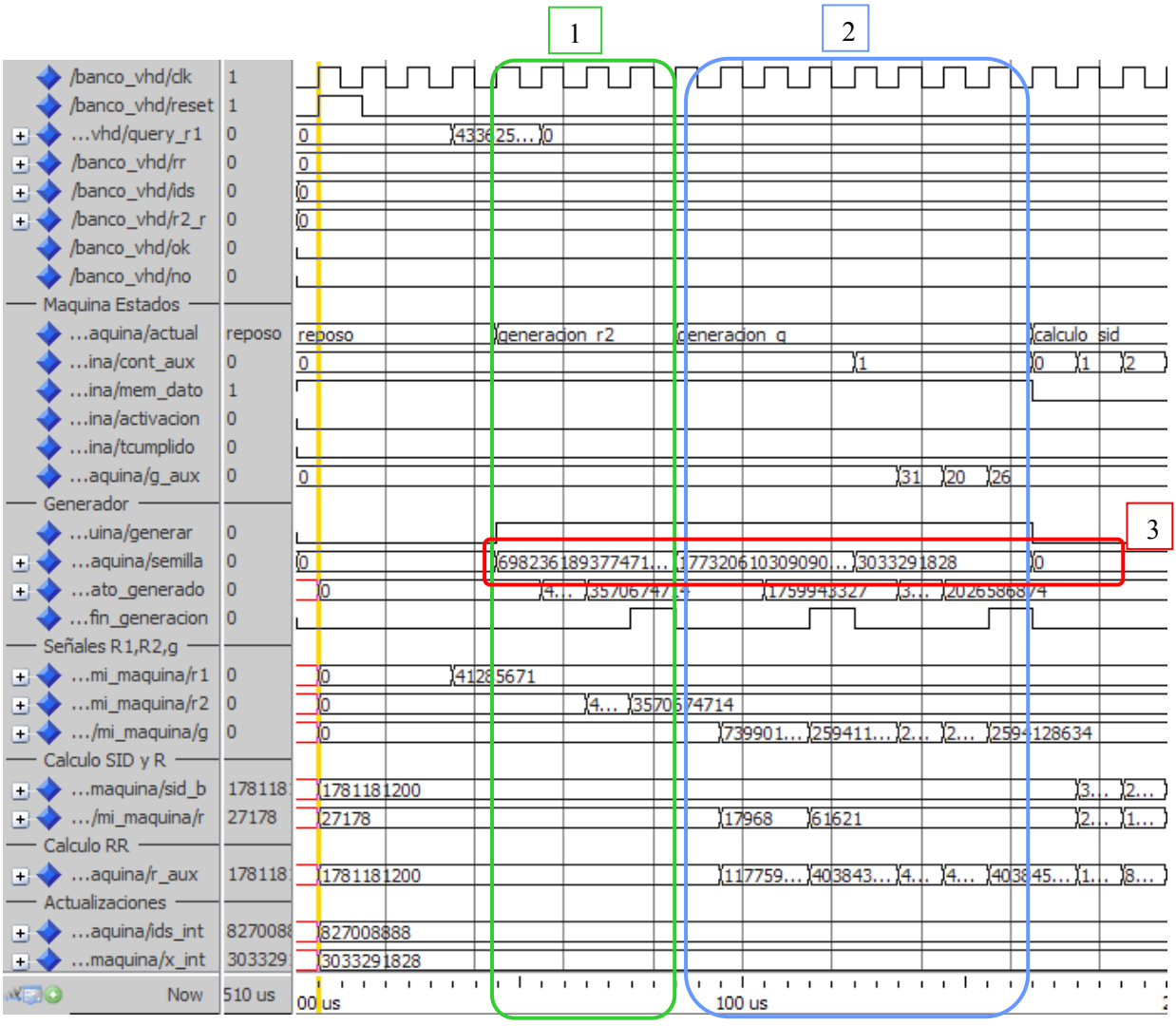

Figura 30: Simulación 1 del caso de funcionamiento correcto del segundo protocolo implementado.

En la figura 31 se muestran como continúa el protocolo. Una vez calculados tanto R2 como g', se pasa a la fase de cálculo de SID' y R' (4), etapa en la cual se envían al lector tanto IDS como los valores concatenados de R2 y R'. Una vez realizado

esto se pasa al estado siguiente (5), en el que se permanece a la espera de una respuesta por parte del lector. Una vez que ésta se recibe, se compara con el valor de RR calculado por el *tag*. En el caso de que la comparción sea positiva, se envía la señal "ok" activada al lector inidicándole que la autenticación se ha realizado de forma correcta. Una vez finalizado el proceso de autenticación se procede a la actualización de los valores de IDS y x almacenados por el *tag* (6). Al igual que antes se pueden observar como se van envíando las semilla adecuadas en cada caso (7).

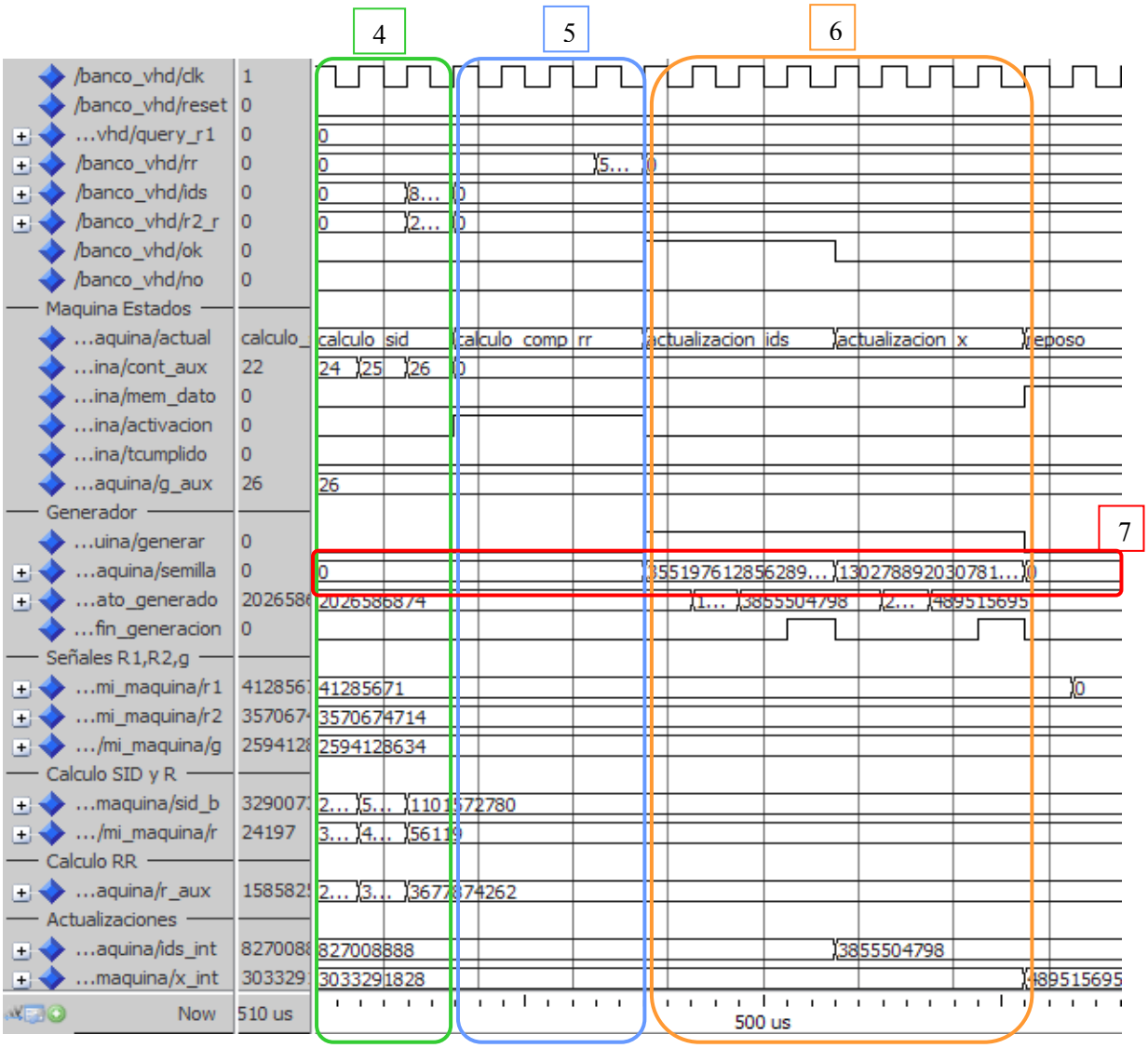

Figura 31: Simulación 2 del caso de funcionamiento correcto del segundo protocolo implementado.

- Caso 2:

En la simulación de la figura 32 se puede observar el caso de funcionamiento del algoritmo con incidencias en la comunicación entre *tag* y lector. El proceso se desarrolla del mismo modo que en el caso anterior (8) hasta la fase de comparación de RR.

En esta ocasión la comparación de valores no es correcta, por lo que el *tag* queda a la espera de recibir un valor adecuado que autentique correctamente al lector (9). Como se comentó anteriormente, este tiempo de espera es limitado, por lo que una vez superado dicho margen, el *tag* aborta la comunicación y vuelve al estado de reposo. Aunque el lector envíe finalmente su respuesta, como ésta se produce fuera del tiempo estipulado, el *tag* no la atiende, por lo que si se desea comunicar con él, el lector deberá comenzar de nuevo todo el proceso.

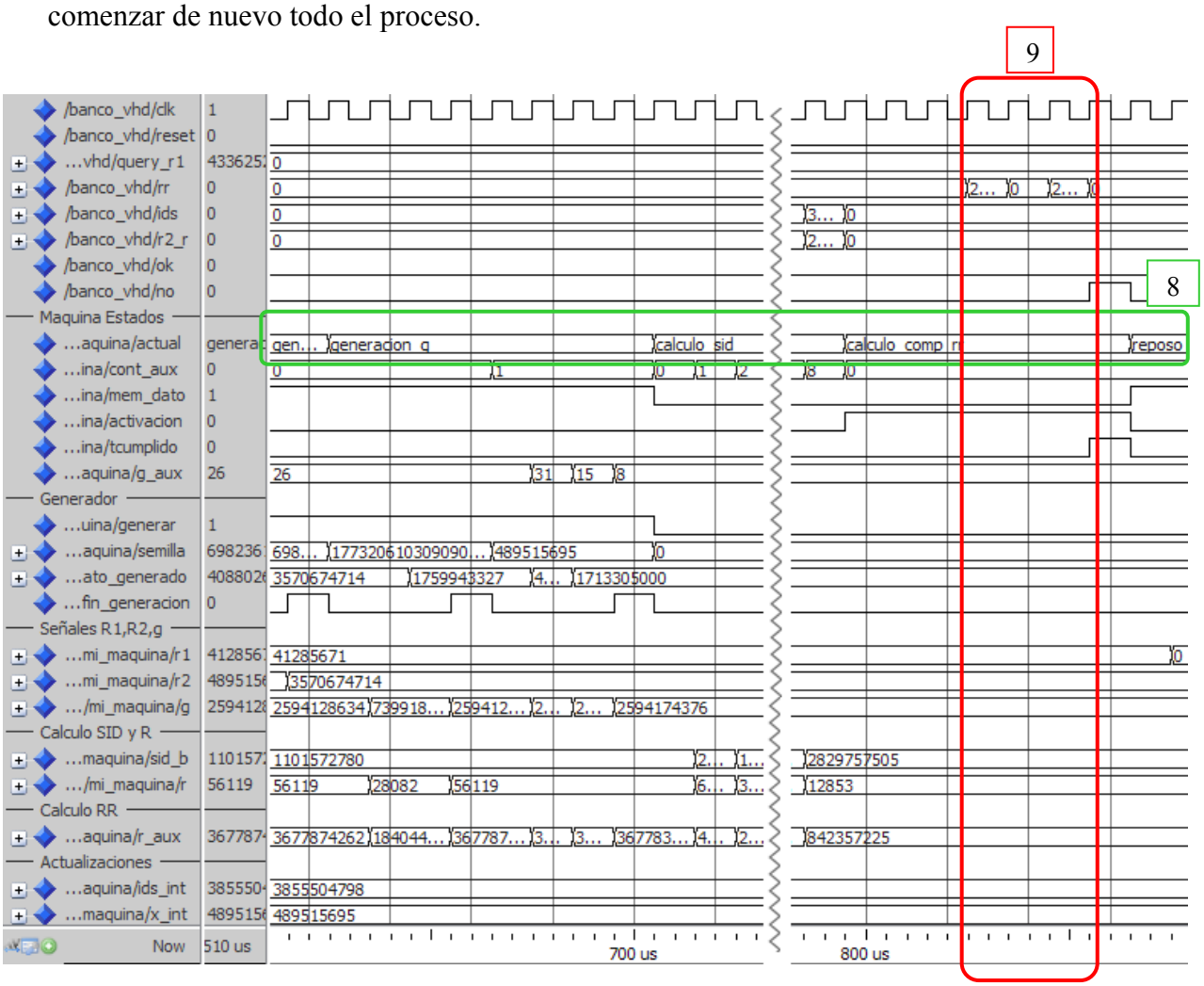

Figura 32: Simulación del caso de funcionamiento con incidencias del segundo protocolo implementado.

5.3.2. Síntesis Synopsys:

Una vez comprobado el correcto funcionamiento del algoritmo implementado, se pasa a la fase de síntésis y obtención de resultados mediante Synopsys. Una vez compilado y sintetizado el diseño, el programa nos permite ver el esquema del diseño.

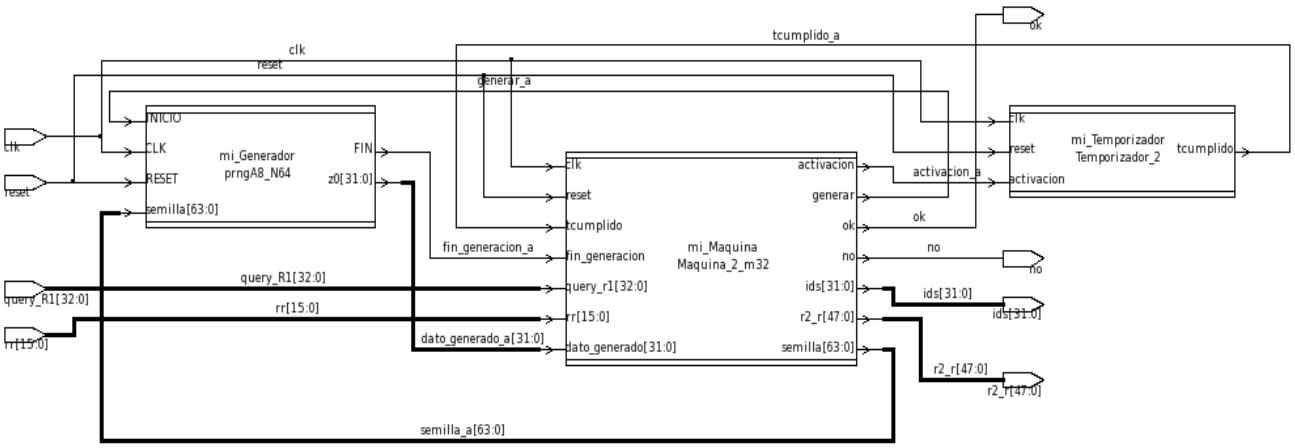

Figura 34: Esquema del diseño implementado para el segundo protocolo.

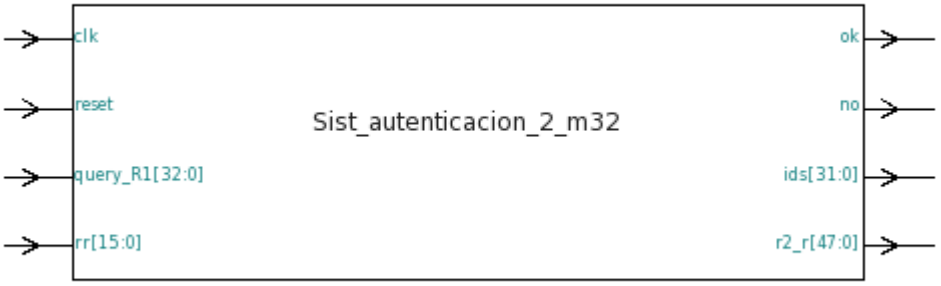

Figura 33: Interfaz del diseño implementado para el segundo protocolo.

Los datos que se muestran a continuación corresponden a la síntesis del diseño del protocolo sin generador de números pseudoaleatorios. Esta síntesis se ha realizado teniendo en cuenta las mismas pautas, tecnología y restricciones que en el caso del protocolo anterior.

Datos de área de la síntesis:

Número de puertos	133
Número de nodos	868
Número de celdas	762
Número de referencias	27
Puertas equivalentes	1968
Área combinacional	5425,462 μm^2
Área no combinacional	5455,860 μm^2
Área de interconexión	625,244 μm^2
Área total de celdas	10881,322 μm^2
Área total	11506,567 μm^2

Datos consumo de la síntesis:

Potencia interna de las celdas	114,228 nW (75%)
Potencia de conmutación	37,282 nW (25%)
Potencia dinámica total	151,510 nW (100%)
Pérdidas de energías de las celdas	40,400 uW

Se puede apreciar como este protocolo va a tener un mayor consumo, tanto de área ocupada como de potencia consumida.

5.4. PROTOCOLOS CON LOS DIFERENTES GENERADORES

Una vez simulados los sistemas, se pasa a la fase de síntesis de la que se van a extraer los datos de consumo de área y potencia que ya se indicaron anteriormente. Las síntesis de los diseños se han realizado para los casos de generadores de 32, 64 y 96 bits, que generan todos ellos una salida de 32 bits, el grado de seguridad que se considera adecuado para los protocolos que se han implementado. Con estas pruebas se pretende determinar qué nivel de seguridad puede alcanzarse, cumpliendo siempre las restricciones impuestas en este proyecto.

5.4.1. Opciones de optimización durante la síntesis

Tal y como se ha indicado anteriormente, la síntesis de los diseños mediante Synopsys se han realizado marcando la opción de alto esfuerzo en área. Al llevar a cabo la síntesis de los protocolos completos, es decir, incluyendo los diferentes generadores de números pseudoaleatorios de los que se dispone, se ha observado como en númerosas ocasiones estos diseños no cumplían las restricciones de área impuestas. Es por ello que se decide aplicar condiciones de síntesis diferentes, como es la opción "*Compile Ultra*" empleando "*Area Script*", que ofrece el programa Synopsys [45][46].

Esta opción de síntesis emplea estrategias de compilación más complejas que requieren de un mayor esfuerzo computacional, destinadas a mejorar la calidad de los resultados en términos de área, tiempo y potencia.

Con el fin de mostrar la mejora que supone este metodo de síntesis frente al empleado inicialmente, se muestran las siguientes tablas con los resultados obtenidos de la síntesis de un mismo protocolo (protocolo 1 empleando el generador A), mediante el método estándar de compilación habiendo seleccionado alto esfuerzo en área (Tabla 3) y mediante el método "*compile ultra*" con la opción "*area script*" (Tabla 4).

Resultados de la síntesis del protocolo 1 con prng A			
	32 bits	**64 bits**	**96 bits**
Número de puertos	68	68	68
Número de nodos	903	1199	1506
Número de celdas	728	895	1074
Número de referencias	39	34	34
Área combinacional (μm^2)	7398,176	10636,575	13973,226
Área no combinacional (μm^2)	4507,525	5486,270	6518,462
Área de interconexión (μm^2)	545,928	811,400	967,350
Área total de celdas (μm^2)	11905,701	16122,845	20491,688
Área total (μm^2)	12451,629	16934,245	21459,038
Puertas equivalentes	2153	2916	3706
Potencia interna	179,122 nW (69%)	204,566 nW (71%)	271,405 nW (71%)
Potencia de conmutación	80,924 nW (31%)	82,364 nW (29%)	112,431 nW (29%)
Potencia dinámica total	260,046 nW	286,930 nW	383,835 nW
Pérdidas de energía	42,519 uW	56,586 uW	72,053 uW

Tabla 3: Resultados del protocolo 1, empleando el primer método de síntesis.

Resultados de la síntesis del protocolo 1 con prng A			
	32 bits	**64 bits**	**96 bits**
Número de puertos	68	68	68
Número de nodos	902	1017	495
Número de celdas	620	808	423
Número de referencias	39	49	30
Área combinacional (μm^2)	6691,545	9193,325	11986,763
Área no combinacional (μm^2)	4454,078	5526,814	6514,774
Área de interconexión (μm^2)	576,272	831,547	1004,946
Área total de celdas (μm^2)	11145,623	14720,139	18501,537
Área total (μm^2)	11721,895	15551,686	19506,483
Puertas equivalentes	2016	2662	3346

Potencia interna	139,410 nW (63%)	244,052 nW (63%)	277,394 nW (63%)
Potencia de conmutación	81,614 nW (37%)	144,926 nW (37%)	161,006 nW (37%)
Potencia dinámica total	221,024 nW	388,978 nW	438,400 nW
Pérdidas de energía	41,418 uW	56,872 uW	68,808 uW

Tabla 4: Resultados del protocolo 1, empleando el segundo método de síntesis.

Estudiando el número de celdas, se aprecia como en el segundo caso, éste es menor debido a la utilización de celdas mayores, lo que reduce el número necesario de éstas para la implementación del circuito, esto se traduce en una reducción del área ocupado por las celdas y en un ligero aumento del área de interconexionado. De esta forma se consiguen circuitos de menor número de puertas equivalentes como se desprende de los datos presentados, uno de los objetivos principales de esta mejora. Esta reducción de E.G., después de analizar todos los casos realizados, es del 7% de media. En vista de estos resultados, se decide optar por la opción de síntesis "*compile ultra*", cuyos resultados para cada diseño se muestran en los siguientes apartados.

5.4.2. Síntesis empleando el generador A

De los datos obtenidos de estas síntesis se obtienen las siguientes tablas resumen con los resultados de los diferentes diseños, sintetizados en función del nivel de seguridad empleado en el generador de números pseudoaleatorios utilizado.

Resultados de la síntesis del protocolo 1 con prng A			
	32 bits	**64 bits**	**96 bits**
Número de puertos	68	68	68
Número de nodos	902	1017	495
Número de celdas	620	808	423
Número de referencias	39	49	30
Área combinacional (μm^2)	6691,545	9193,325	11986,763
Área no combinacional (μm^2)	4454,078	5526,814	6514,774
Área de interconexión (μm^2)	576,272	831,547	1004,946
Área total de celdas (μm^2)	11145,623	14720,139	18501,537
Área total (μm^2)	11721,895	15551,686	19506,483
Puertas equivalentes	2016	2662	3346
Potencia interna	139,410 nW	244,052 nW	277,394 nW

	(63%)	(63%)	(63%)
Potencia de conmutación	81,614 nW (37%)	144,926 nW (37%)	161,006 nW (37%)
Potencia dinámica total	221,024 nW	388,978 nW	438,400 nW
Pérdidas de energía	41,418 uW	56,872 uW	68,808 uW

Tabla 5: Resultados de la síntesis del protocolo 1 según el número de bits del prng A.

Resultados de la síntesis del protocolo 2 con prng A			
	32 bits	**64 bits**	**96 bits**
Número de puertos	133	133	133
Número de nodos	1153	1458	914
Número de celdas	964	1180	788
Número de referencias	46	35	35
Área combinacional (μm^2)	8632,412	11501,396	14402,897
Área no combinacional (μm^2)	7911,920	8733,986	9965,246
Área de interconexión (μm^2)	1170,533	1326,543	1669,418
Área total de celdas (μm^2)	16544,332	20235,382	24368,143
Área total (μm^2)	17714,865	21561,925	26037,561
Puertas equivalentes	2992	3660	4407
Potencia interna	188,857 nW (78%)	190,471 nW (79%)	327,232 nW (67%)
Potencia de conmutación	54,264 nW (22%)	51,592 nW (21%)	159,666 nW (33%)
Potencia dinámica total	243,121 nW	242,063 nW	486,898 nW
Pérdidas de energía	63,330 uW	74,784 uW	91,610 uW

Tabla 6: Resultados de la síntesis del protocolo 2 según el número de bits del prng A.

Como se puede apreciar en las tablas, el número de puertos para cada uno de los protocolos, independientemente del generador con el que se implemente y del número de bits del mismo, se va a mantener siempre igual (68 para el protocolo 1 y 133 para el protocolo 2), ya que la estructura en sí del protocolo no varía.

Por otro lado, observamos los valores del número de nodos, de celdas y de referencias, relacionados entre sí tal y como ya se ha explicado en apartados anteriores. El número de celdas y de nodos va a ser siempre superior en el caso del segundo protocolo, debido a la arquitectura más compleja de éste, esto se verá reflejado en un mayor área y por tanto un mayor número de E.G. como se podrá ver más adelante.

De ahora en adelante, no se harán referencias a estos cuatro parámetros, bien por no ser significativos a la hora del análisis de los diseños o bien porque su influencia se puede observar en el estudio de otros parámetros.

A continuación se muestran diferentes gráficas de los datos obtenidos de la síntesis que permiten realizar una comparación entre las diferentes opciones que se han implementado con el generador de números pseudoaleatorios A.

En la gráfica que aparece a continuación (Figura 35) se muestra la comparativa del número de puertas equivalentes necesarias para cada diseño, obtenidas de las síntesis de los mismos, dependiendo del número de bits del generador de números pseudoaleatorios empleado y del protocolo implementado.

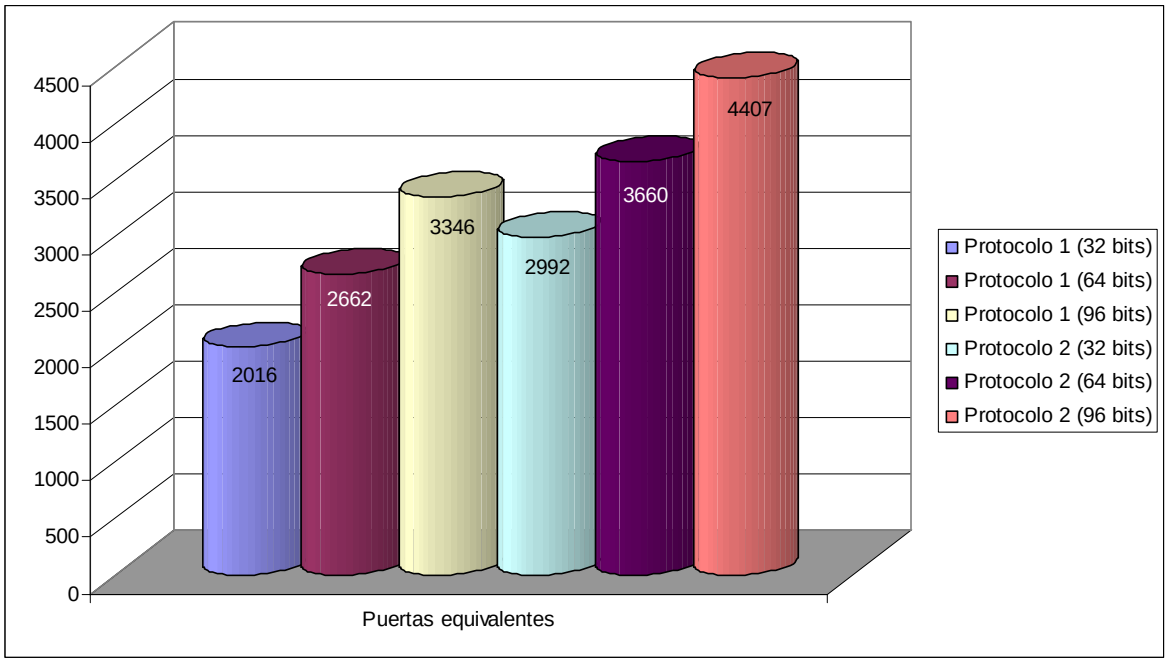

Figura 35: Número de puertas equivalentes de cada diseño según el tamaño del prng A.

Se puede apreciar como los protocolos con PRNG de 32 bits son claramente implementables, pero al reducirse el nivel de seguridad del sistema, sólo se implementarían, si las opciones de mayor número de bits no fueran factibles. Por otro lado, se ve como para ambos protocolos, se cumplen las restricciones de área impuestas para este proyecto (3000-4000 E.G.) en el caso en el que se emplea un generador de 64 bits, cumpliendo asimismo las necesidades de aleatoriedad y seguridad del protocolo. Por último, la implementación de los protocolos en el caso de 96 bits, sería factible para el primer protocolo, mientras que para el segundo sólo sería implementable en *tags* de

mayor tamaño y por tanto más caros, que no se están teniendo en cuenta en este proyecto.

En la Figura 36 se puede observar la distribución de las diferentes áreas (combinacional, no combinacional y conexionado) en cada uno de los diferentes diseños sintetizados. El área empleada para determinar el número de puertas equivalentes necesarias para cada diseño mostradas anteriormente, viene determinada por la suma del área combinacional más el área no combinacional, es decir, no se tiene en cuenta el área de interconexionado para estos cálculos. Se ve como el área consumida aumenta a medida que el número de bits del PRNG es mayor.

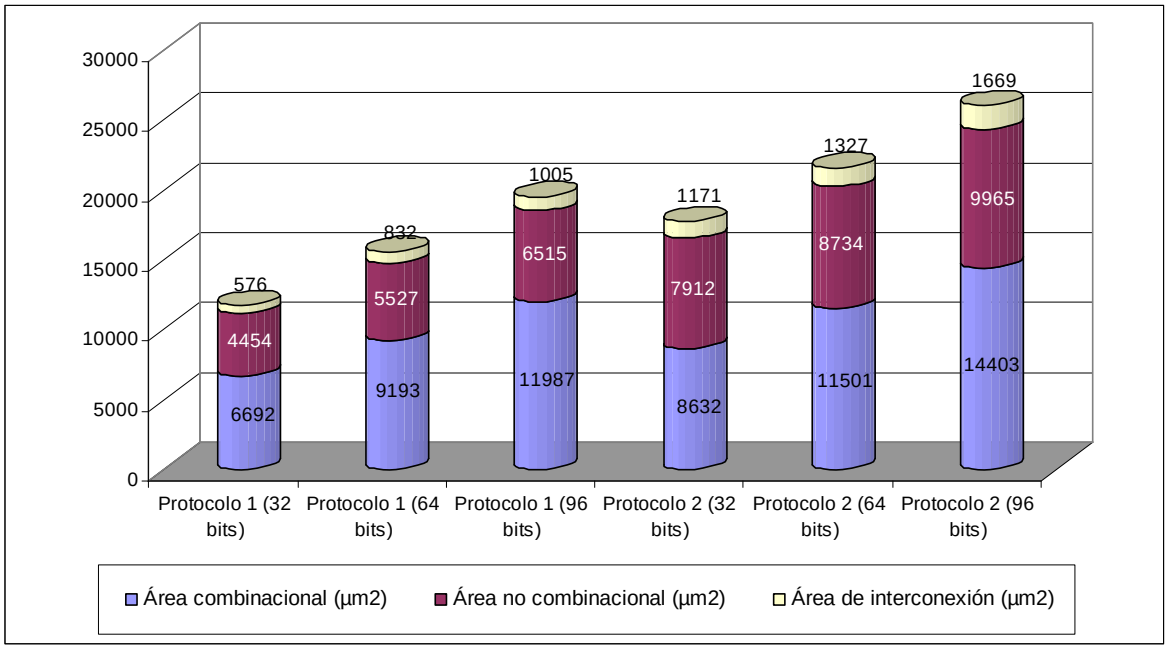

Figura 36: Distribución del área de cada diseño según el tamaño del prng A.

Por último se muestra la gráfica con la potencia consumida, así como con las pérdidas de potencia en cada diseño en función del número de bits empleados en su implementación.

En la Figura 37 se puede observar tanto el consumo de potencia dinámica como las pérdidas de energía que tiene el circuito. La potencia dinámica está compuesta por la potencia interna y por la potencia de conmutación, y depende en gran medida de la frecuencia de funcionamiento del sistema [47]. La potencia interna es la potencia disipada en la celda debido a su carga interna, mientras que la potencia de conmutación es la potencia consumida por la carga o descarga de la carga externa de la celda.

Por otro lado las perdidas de potencia o potencia estática están causadas por pequeños umbrales de tensión que impiden el apagado completo de la celda, generando una corriente de pérdidas. Esta potencia depende, por tanto, de la tensión, de la temperatura y del tipo de puerta lógica.

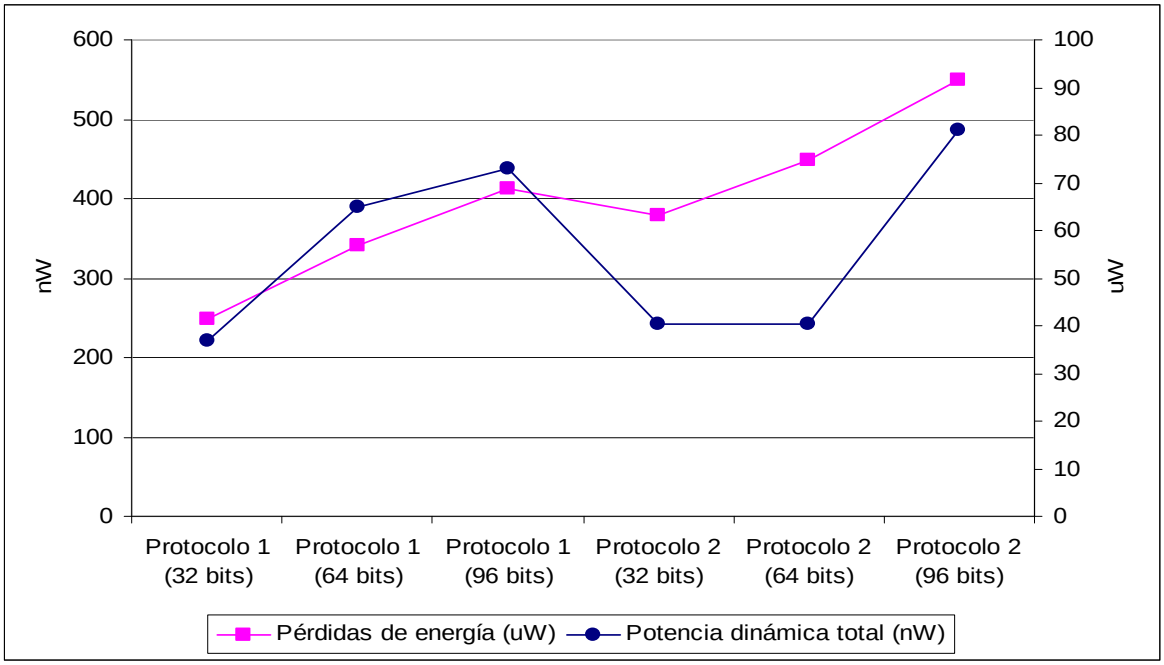

Figura 37: Potencia consumida y pérdidas de cada diseño según el tamaño del prng A.

Como se puede observar en la gráfica, mientras que las pérdidas de potencia son del orden de uW, la potencia dinámica es de nW. Esto es poco corriente en este tipo de tecnología, debido a que la tecnología CMOS se caracteriza por tener bajas potencias estáticas, que en muchos casos pueden llegar a ser despreciables frente a las potencias dinámicas.

La peculiaridad de este caso es que en este proyecto se está trabajando con frecuencias muy pequeñas (100 kHz), lo que hace obtener unas potencias dinámicas, que como ya se ha comentado dependen en gran medida de la frecuencia, inferiores a las estáticas. Se ha comprobado que en el caso de trabajar con frecuencias del orden de los MHz, la potencia dinámica aumentaria de forma considerable hasta magnitudes de mW, lo que no sería adecuado para las restricciones de potencia que presentan los *tags* de bajo coste para los que se están realizando estos diseños.

5.4.3. Síntesis empleando el generador A1

De los datos obtenidos de estas síntesis se realizan las siguientes tablas resumen con los resultados de los diferentes diseños sintetizados en función del nivel de seguridad empleado en el generador de números pseudoaleatorios utilizado.

Resultados de la síntesis del protocolo 1 con prng A1			
	32 bits	**64 bits**	**96 bits**
Número de puertos	68	68	68
Número de nodos	864	1178	493
Número de celdas	798	1098	421
Número de referencias	46	44	36
Área combinacional (μm^2)	6109,952	8366,485	10758,483
Área no combinacional (μm^2)	5547,094	7611,478	9675,862
Área de interconexión (μm^2)	738,493	1114,741	1739,644
Área total de celdas (μm^2)	11657,046	15977,963	20434,345
Área total (μm^2)	12395,539	17092,704	22173,989
Puertas equivalentes	2108	2890	3695
Potencia interna	148,738 nW (66%)	224,552 nW (60%)	340,200 nW (57%)
Potencia de conmutación	77,943 nW (34%)	152,697 nW (40%)	259,010 nW (43%)
Potencia dinámica total	226,681 nW	377,249 nW	599,209 nW
Pérdidas de energía	43,502 uW	57,396 uW	80,852 uW

Tabla 7: Resultados de la síntesis del protocolo 1 según el número de bits del prng A1.

Resultados de la síntesis del protocolo 2 con prng A1			
	32 bits	**64 bits**	**96 bits**
Número de puertos	133	133	133
Número de nodos	1239	946	904
Número de celdas	1112	822	778
Número de referencias	42	37	36
Área combinacional (μm^2)	8517,665	11341,802	13251,194
Área no combinacional (μm^2)	8980,054	11027,848	13094,078
Área de interconexión (μm^2)	1354,319	1745,016	2350,480
Área total de celdas (μm^2)	17497,719	22369,650	26345,272
Área total (μm^2)	18852,038	24114,666	28695,752
Puertas equivalentes	3164	4045	4764

Potencia interna	194,181 nW (80%)	275,746 nW (69%)	349,174 nW (63%)
Potencia de conmutación	48,678 nW (20%)	122,544 nW (31%)	201,914 nW (37%)
Potencia dinámica total	242,859 nW	398,290 nW	551,088 nW
Pérdidas de energía	65,355 uW	85,667 uW	102,334 uW

Tabla 8: Resultados de la síntesis del protocolo 2 según el número de bits del prng A1.

En la gráfica que aparece a continuación (Figura 38) se muestra la comparativa del número de puertas equivalentes necesarias para cada diseño, obtenidas de las síntesis de los mismos, dependiendo del número de bits del generador empleado y del protocolo implementado, realizando una comparación entre las diferentes opciones que se han implementado con el generador de números pseudoaleatorios A1.

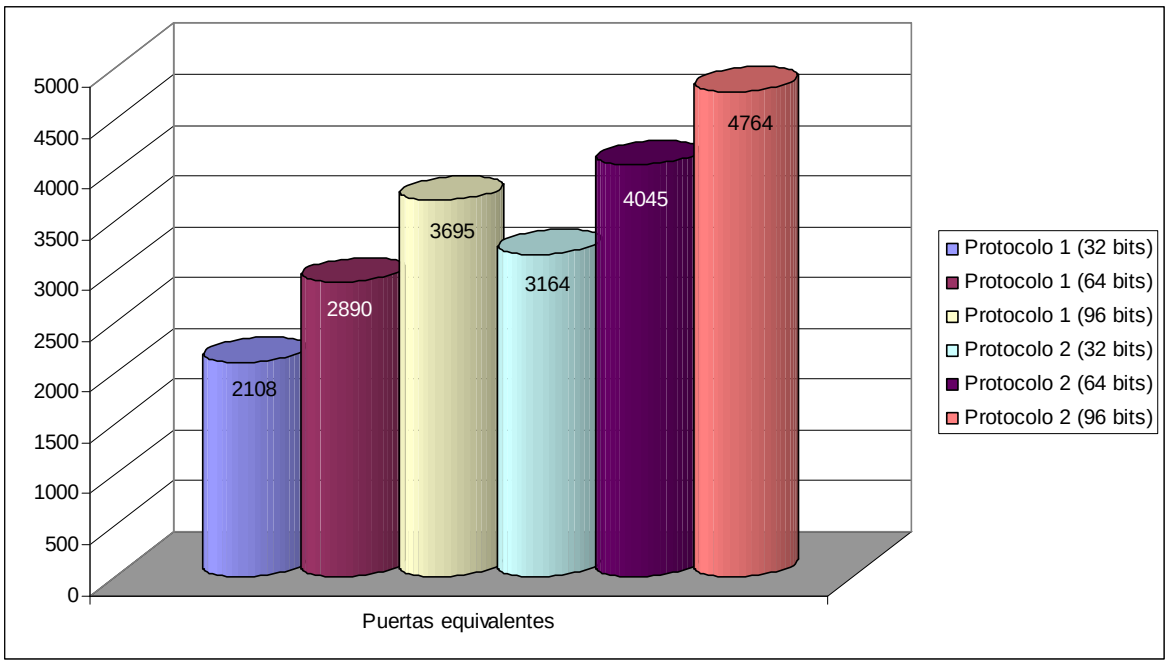

Figura 38: Número de puertas equivalentes de cada diseño según el tamaño del prng A1.

Tal y como se ha explicado anteriormente, el generador A1 pretende ser una mejora del generador A. Observando los resultados obtenidos de la síntesis de ambos casos, se puede concluir que dicha mejora no es efectiva en ninguno de los dos protocolos, puesto que para todos los diseños el número de puertas equivalentes consumidas en los protocolos con generador A1 (Figura 38) son mayores que con el generador A (Figura 35). Por este motivo, dichas implementaciones quedan descartadas para posteriores análisis de los protocolos con el fin de determinar el más adecuado.

5.4.4. Síntesis empleando el generador B

De los datos obtenidos de estas síntesis se realizan las siguientes tablas resumen con los resultados de los diferentes diseños sintetizados en función del nivel de seguridad empleado en el generador de números pseudoaleatorios utilizando.

Resultados de la síntesis del protocolo 1 con prng B			
	32 bits	**64 bits**	**96 bits**
Número de puertos	68	68	68
Número de nodos	1080	495	494
Número de celdas	882	423	422
Número de referencias	51	30	35
Área combinacional (μm^2)	8988,255	14627,074	20309,807
Área no combinacional (μm^2)	5436,500	7482,454	9546,838
Área de interconexión (μm^2)	814,984	1370,495	1995,201
Área total de celdas (μm^2)	14424,755	22109,528	29856,645
Área total (μm^2)	15239,739	23480,023	31851,846
Puertas equivalentes	2609	3998	5399
Potencia interna	168,284 nW (65%)	222,268 nW (66%)	270,808 nW (72%)
Potencia de conmutación	89,191 nW (35%)	114,763 nW (34%)	104,629 nW (28%)
Potencia dinámica total	257,476 nW	337,032 nW	375,437 nW
Pérdidas de energía	52,565 uW	78,966 uW	104,531 uW

Tabla 9: Resultados de la síntesis del protocolo 1 según el número de bits del prng B.

Resultados de la síntesis del protocolo 2 con prng B			
	32 bits	**64 bits**	**96 bits**
Número de puertos	133	133	133
Número de nodos	1468	934	906
Número de celdas	1206	810	780
Número de referencias	45	37	31
Área combinacional (μm^2)	11293,965	17675,707	23008,621
Área no combinacional (μm^2)	8882,364	10927,394	12997,310
Área de interconexión (μm^2)	1495,255	2033,643	2851,670
Área total de celdas (μm^2)	20176,329	28603,101	36005,931
Área total (μm^2)	21671,584	30636,744	38857,601
Puertas equivalentes	3649	5173	6512

Potencia interna	194,279 nW (79%)	265,979 nW (70%)	313,603 nW (78%)
Potencia de conmutación	50,099 nW (21%)	116,213 nW (30%)	90,404 nW (22%)
Potencia dinámica total	244,377 nW	382,192 nW	404,007 nW
Pérdidas de energía	74,115 uW	104,355 uW	127,311 uW

Tabla 10: Resultados de la síntesis del protocolo 2 según el número de bits del prng B.

A continuación se muestran diferentes gráficas de los datos obtenidos de la síntesis que permiten realizar una comparación entre las diferentes opciones que se han implementado con el generador de números pseudoaleatorios B. En la gráfica que aparece a continuación (Figura 39) se muestra la comparativa del número de puertas equivalentes necesarias para cada diseño, obtenidas de las síntesis de los mismos, dependiendo del número de bits del generador de números pseudoaleatorios empleado y del protocolo implementado.

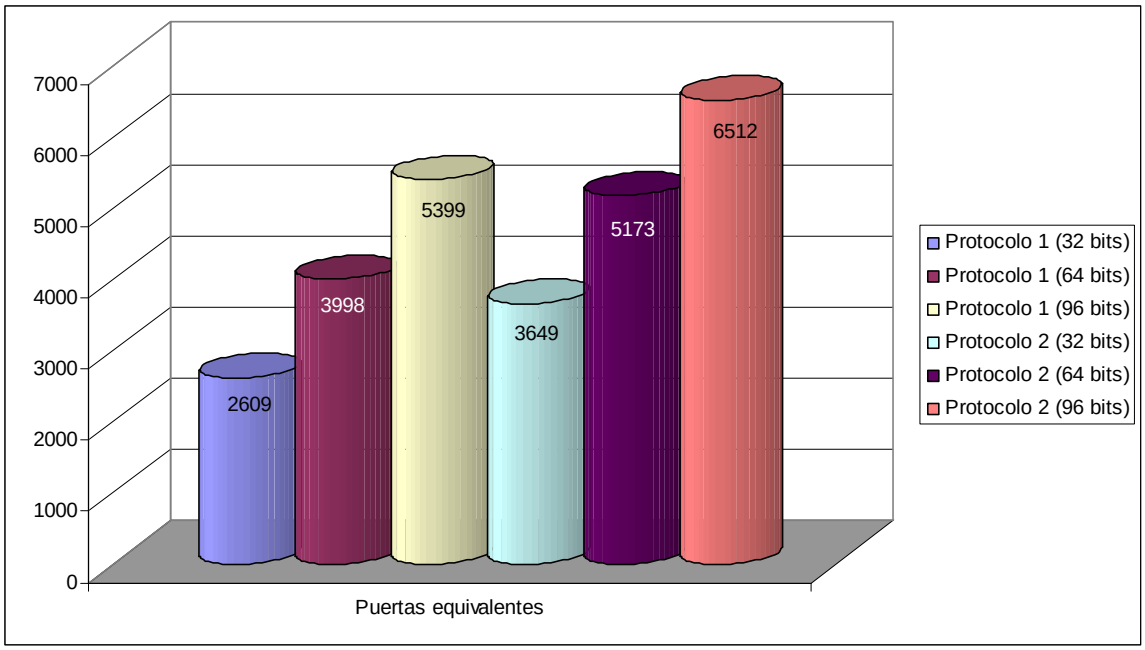

Figura 39: Número de puertas equivalentes de cada diseño según el tamaño del prng B.

Se puede apreciar cómo los protocolos con PRNG de 32 bits son claramente implementables en ambos casos, pero al reducirse el nivel de seguridad del sistema, sólo se implementarían, si las opciones de mayor número de bits no fueran factibles. Por otro lado, se ve como para el protocolo 1, se cumplen las restrisciones de área impuestas para este proyecto (3000-4000 E.G.) en el caso en el que se emplea un generador de 64

bits, mientras que en el caso del protocolo 2 se consumen más recursos de los disponibles, por lo que la única opción factible para este protocolo será trabajar con el PRNG B de 32 bits. Por último, se aprecia que la implementación de los protocolos en el caso de 96 bits, sólo sería implementable en *tags* de mayor tamaño y por tanto más caros, que no se están teniendo en cuenta en este proyecto.

En esta figura se puede apreciar la distribución de las diferentes áreas (combinacional, no combinacional y conexionado) en cada uno de los diferentes diseños sinterizados.

Como se puede apreciar en la Figura 40, tanto el área total como cada uno de los diferentes tipos de áreas van aumentando dentro de cada protocolo proporcinalmente al incremento de bits con los que trabaja el generador.

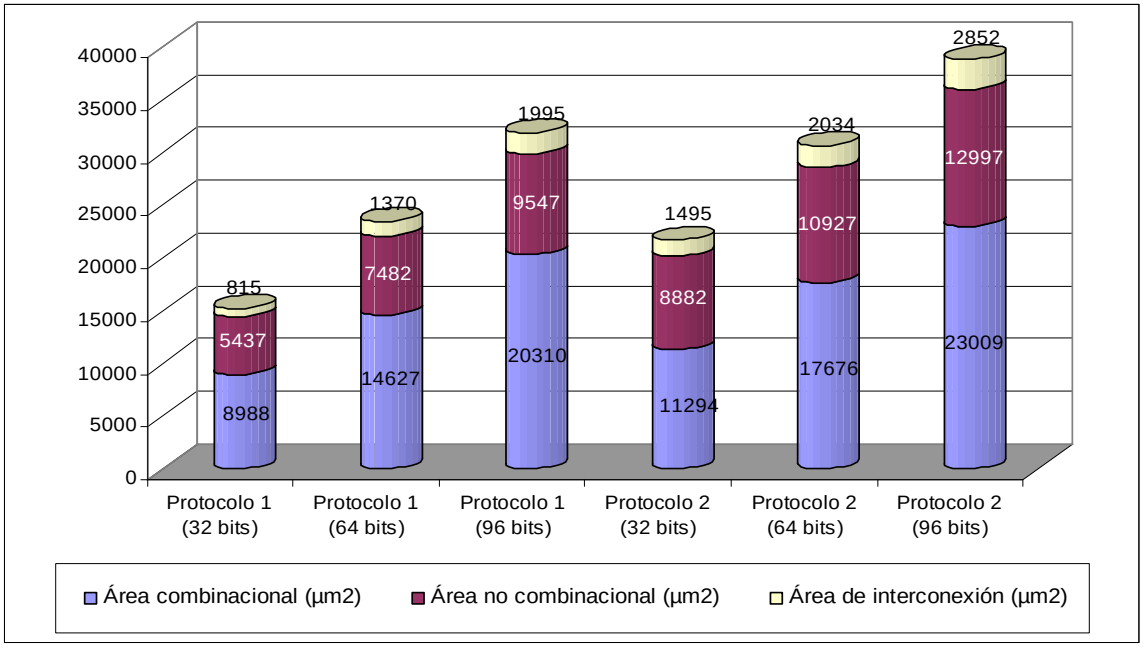

Figura 40: Distribución del área de cada diseño según el tamaño del prng B.

Por último se muestra la gráfica con la potencia consumida, así como con las pérdidas de potencia en cada diseño en función del número de bits empleados en su implementación.

Se observa en la Figura 41, que al igual que en protocolos anteriores, tanto la potencia consumida como las pérdidas, siguen la misma tendencia, son mayores siempre en el segundo protocolo y en ambos casos se van incrementando en función del tamaño del generador.

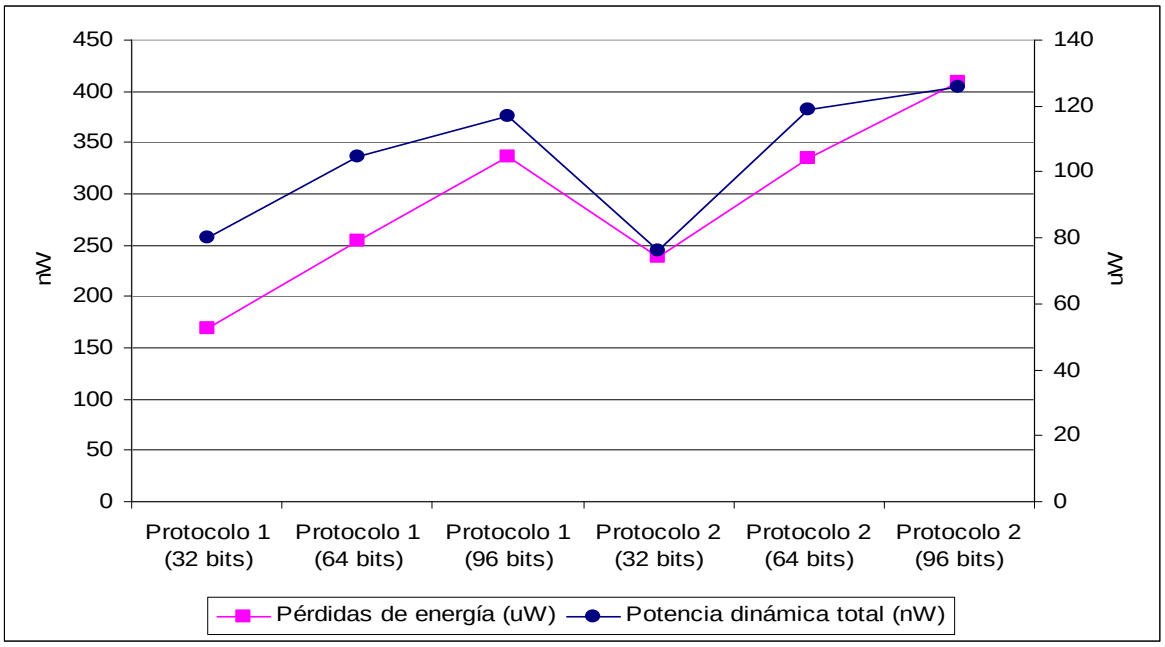

Figura 41: Potencia consumida y pérdidas de cada diseño según el tamaño del prng B.

5.4.5. Síntesis empleando el generador B1

De los datos obtenidos de estas síntesis se realizan las siguientes tablas resumen con los resultados de los diferentes diseños sintetizados en función del nivel de seguridad empleado en el generador de números pseudoaleatorios utilizado.

Resultados de la síntesis del protocolo 1 con prng B1			
	32 bits	**64 bits**	**96 bits**
Número de puertos	68	68	68
Número de nodos	496	493	494
Número de celdas	424	421	422
Número de referencias	38	35	35
Área combinacional (μm^2)	8351,704	12636,448	16589,224
Área no combinacional (μm^2)	6325,836	9141,321	12000,118
Área de interconexión (μm^2)	969,985	1665,829	2492,179
Área total de celdas (μm^2)	14677,540	21777,769	28589,342
Área total (μm^2)	15647,525	23443,598	31081,520
Puertas equivalentes	2654	3938	5170
Potencia interna	184,715 nW (67%)	252,565 nW (64%)	319,659 nW (70%)
Potencia de conmutación	89,760 nW	141,773 nW	135,315 nW

	(33%)	(36%)	(30%)
Potencia dinámica total	274,475 nW	394,338 nW	454,974 nW
Pérdidas de energía	53,290 uW	77,234 uW	99,203 uW

Tabla 11: Resultados de la síntesis del protocolo 1 según el número de bits del prng B1.

Resultados de la síntesis del protocolo 2 con prng B1			
	32 bits	**64 bits**	**96 bits**
Número de puertos	133	133	133
Número de nodos	910	938	904
Número de celdas	786	814	778
Número de referencias	40	39	36
Área combinacional (μm^2)	10680,240	15320,959	19258,601
Área no combinacional (μm^2)	9734,839	12589,947	15450,590
Área de interconexión (μm^2)	1568,102	2299,108	3234,070
Área total de celdas (μm^2)	20415,079	27910,906	34709,191
Área total (μm^2)	21983,182	30210,014	37943,261
Puertas equivalentes	3692	5048	6277
Potencia interna	210,792 nW (74%)	293,661 nW (69%)	349,638 nW (74%)
Potencia de conmutación	74,737 nW (26%)	129,510 nW (31%)	120,976 nW (26%)
Potencia dinámica total	285,529 nW	423,172 nW	470,614 nW
Pérdidas de energía	74,936 uW	101,739 uW	124,611 uW

Tabla 12: Resultados de la síntesis del protocolo 2 según el número de bits del prng B1.

A continuación se muestran diferentes gráficas de los datos obtenidos de la síntesis que permiten realizar una comparación entre las diferentes opciones que se han implementado con el generador de números pseudoaleatorios B1.

En la Figura 42 se muestra la comparativa del número de puertas equivalentes necesarias para cada diseño, dependiendo del número de bits del generador empleado y del protocolo implementado. Tal y como se ha explicado anteriormente, el generador B1 pretende ser una mejora del generador B. Observando los resultados obtenidos de la síntesis de ambos casos, se puede concluir que dicha mejora es efectiva para ambos protocolos en los casos de 64 y 96 bits, puesto que el número de puertas equivalentes consumidas en los protocolos con generador B1 (Figura 42) son menores que con el generador B (Figura 39), en dichos casos. A pesar de dicha mejora, en el caso del protocolo 2, las opciones con el generador de 64 bits y de 96 bits continúa sin ser

factible, debido al incumplimiento de las restricciones de área impuestas. Por este motivo, al igual que ocurría en el caso anterior, la única opción factible para el protocolo 2 con generador B1 sería la de 32 bits, pero debido a que éste es el único caso en que el prng B1 no supone una mejora con respecto el prng B, dicha combinación no se tendrá en cuenta para posteriores análisis.

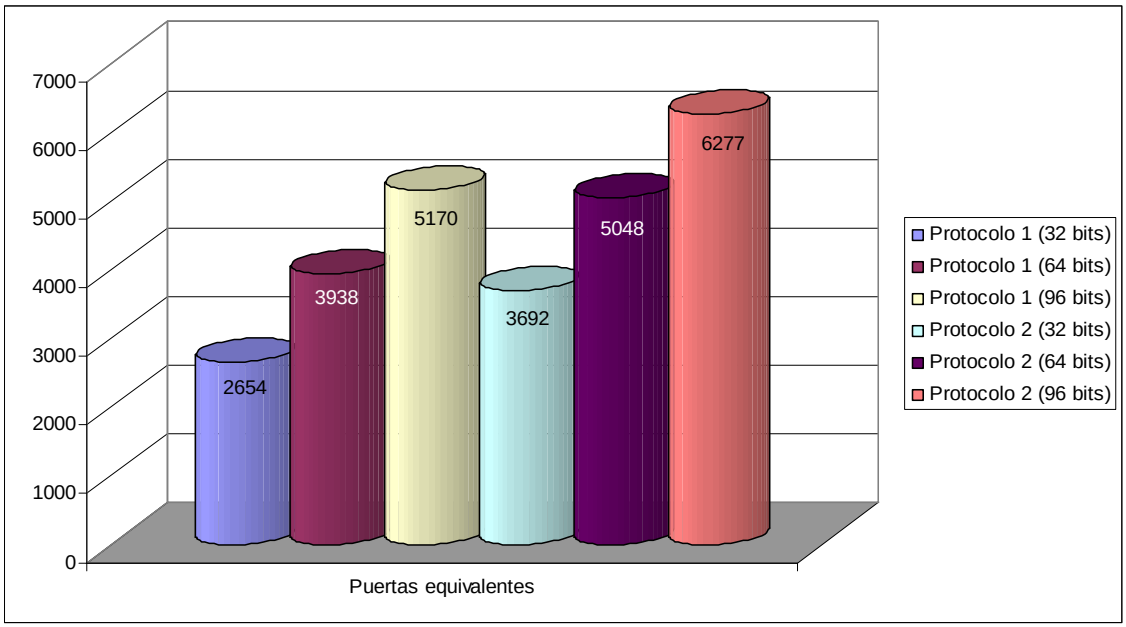

Figura 42: Número de puertas equivalentes de cada diseño según el tamaño del prng B1.

En la Figura 43 se puede apreciar la distribución de las diferentes áreas (combinacional, no combinacional y conexionado) en cada uno de los diferentes diseños sinterizados.

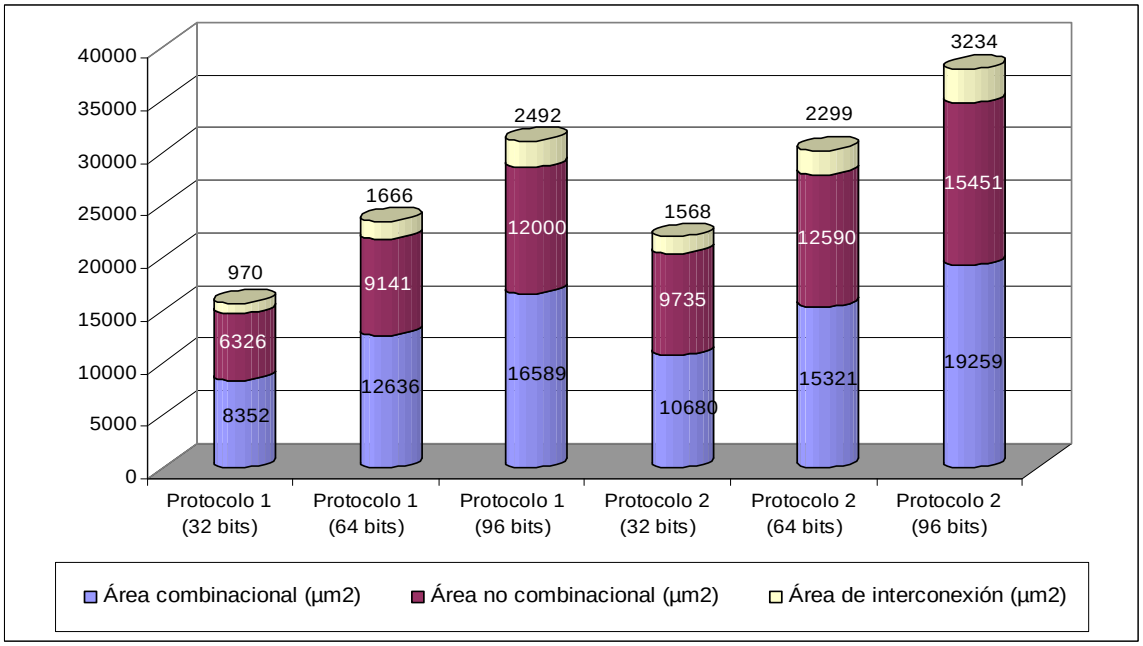

Figura 43: Distribución del área de cada diseño según el tamaño del prng B1.

Como se puede apreciar en la Figura 43, tanto el área total como cada uno de los diferentes tipos de áreas van aumentando dentro de cada protocolo proporcinalmente al incremento de bits con los que trabaja el generador.

Por otro lado, se observa en la Figura 44, que al igual que en protocolos anteriores, tanto la potencia consumida como las pérdidas, siguen la misma tendencia, son mayores siempre en el segundo protocolo y en ambos casos se van incrementando en función del tamaño del generador.

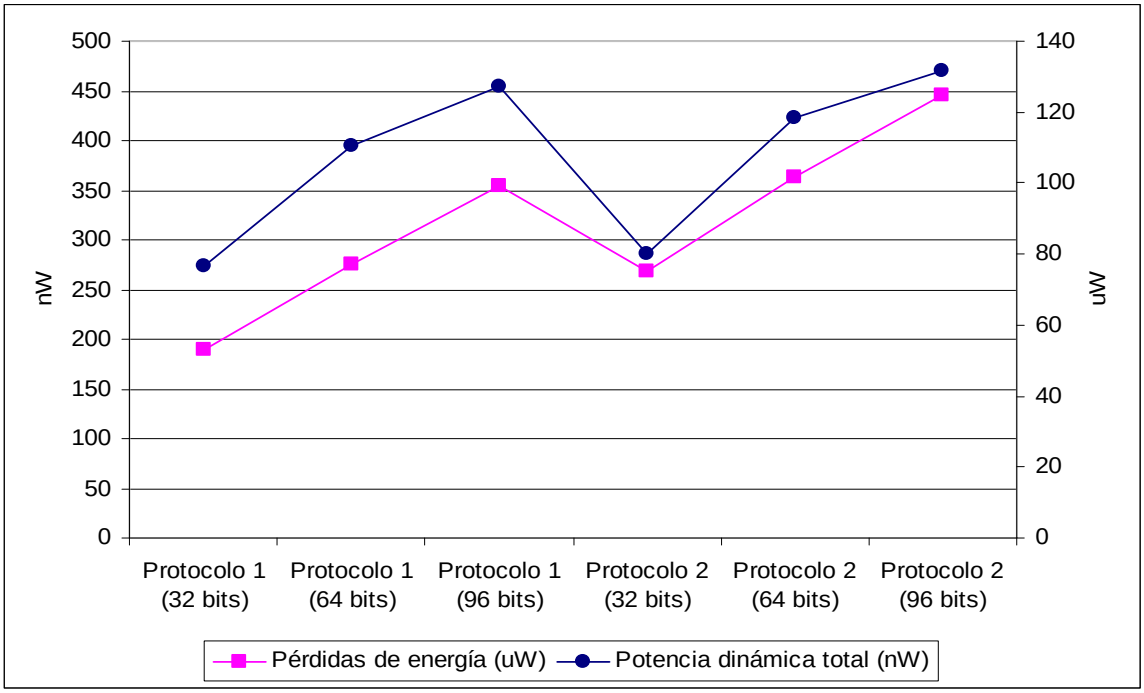

Figura 44: Potencia consumida y pérdidas de cada diseño según el tamaño del prng B1.

5.4.6. Síntesis empleando el generador B2

De los datos obtenidos de estas síntesis se realizan las siguientes tablas resumen con los resultados de los diferentes diseños sintetizados en función del nivel de seguridad empleado en el generador de números pseudoaleatorios utilizado.

Resultados de la síntesis del protocolo 1 con prng B2			
	32 bits	**64 bits**	**96 bits**
Número de puertos	68	68	68
Número de nodos	494	493	494
Número de celdas	422	421	422
Número de referencias	40	36	35

Área combinacional (μm^2)	8521,879	12737,300	16156,881
Área no combinacional (μm^2)	6358,092	9177,263	12039,746
Área de interconexión (μm^2)	986,820	1738,143	2546,673
Área total de celdas (μm^2)	14879,970	21914,563	28196,627
Área total (μm^2)	15866,791	23652,706	30473,300
Puertas equivalentes	2691	3963	5099
Potencia interna	190,941 nW (68%)	262,034 nW (66%)	308,920 nW (74%)
Potencia de conmutación	88,940 nW (32%)	136,821 nW (34%)	109,416 nW (26%)
Potencia dinámica total	279,881 nW	398,854 nW	418,336 nW
Pérdidas de energía	56,001 uW	81,207 uW	100,095 uW

Tabla 13: Resultados de la síntesis del protocolo 1 según el número de bits del prng B2.

Resultados de la síntesis del protocolo 2 con prng B2			
	32 bits	**64 bits**	**96 bits**
Número de puertos	133	133	133
Número de nodos	908	939	902
Número de celdas	784	815	776
Número de referencias	34	36	37
Área combinacional (μm^2)	10930,167	15464,711	19148,128
Área no combinacional (μm^2)	9763,409	12620,360	15488,375
Área de interconexión (μm^2)	1590,634	2247,718	3130,311
Área total de celdas (μm^2)	20693,576	28085,071	34636,503
Área total (μm^2)	22284,210	30332,789	37766,815
Puertas equivalentes	3743	5079	6264
Potencia interna	205,520 nW (76%)	284,550 nW (72%)	337,239 nW (77%)
Potencia de conmutación	65,468 nW (24%)	110,208 nW (28%)	98,822 nW (23%)
Potencia dinámica total	270,988 nW	394,758 nW	436,061 nW
Pérdidas de energía	77,385 uW	104,959 uW	127,046 uW

Tabla 14: Resultados de la síntesis del protocolo 2 según el número de bits del prng B2.

A continuación se muestran diferentes gráficas de los datos obtenidos de la síntesis que permitan realizar una comparación entre las diferentes opciones que se han implementado con el generador de números pseudoaleatorios B2. En la Figura 46 se muestran el número de puertas equivalentes empleadas en cada diseño, dependiendo del número de bits del generador empleado y del protocolo implementado

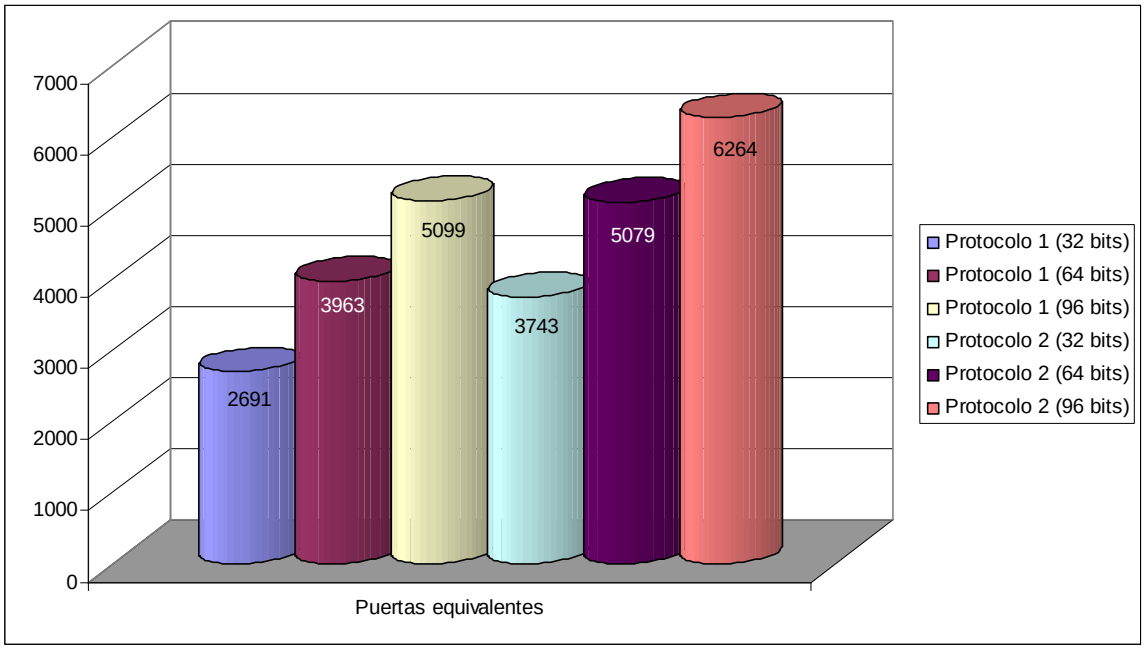

Figura 45: Número de puertas equivalentes de cada diseño según el tamaño del prng B2.

Puesto que se observaron ciertas mejorias con el generador B1 respecto al generador B, el generador B2 pretende ser una optimización de ambos, tal y como se explicó en el capítulo 4. Observando los resultados obtenidos de la síntesis de ambos protocolos, se puede concluir que dicha mejora es únicamente efectiva en el caso de los diseños con generadores de 96 bits, mientras que en el caso de 64 bits, aunque se observa mejoria con respecto a B no ocurre lo mismo con respecto a B1, por lo que estas opciones quedan descartadas, al igual que las de 32 bits, en las que no existe reducción de consumo ninguna.

En la Figura 46 se puede apreciar la distribución de las diferentes áreas (combinacional, no combinacional y conexionado) en cada uno de los diferentes diseños sinterizados. Tanto el área total como las individuales van aumentando según se incrementa el número de bits del generador.

Por último, se observa en la Figura 47, que al igual que en diseños anteriores, tanto la potencia consumida como las pérididdas son mayores siempre en el segundo protocolo y en ambos casos se van incrementando en función del tamaño del generador.

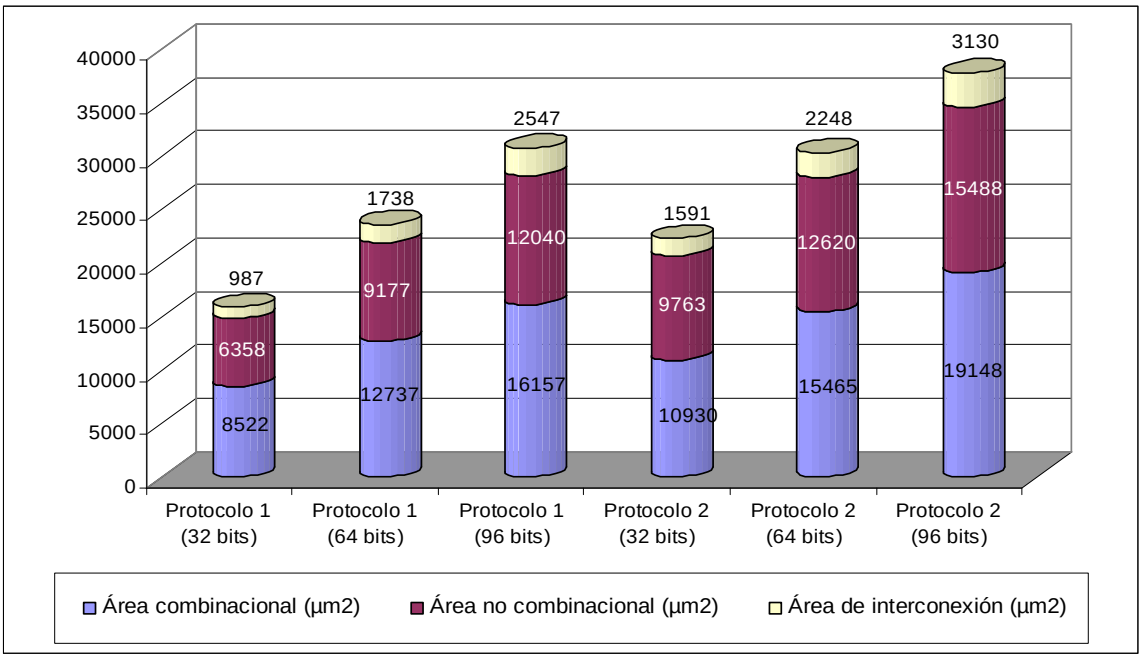

Figura 47: Distribución del área de cada diseño según el tamaño del prng B2.

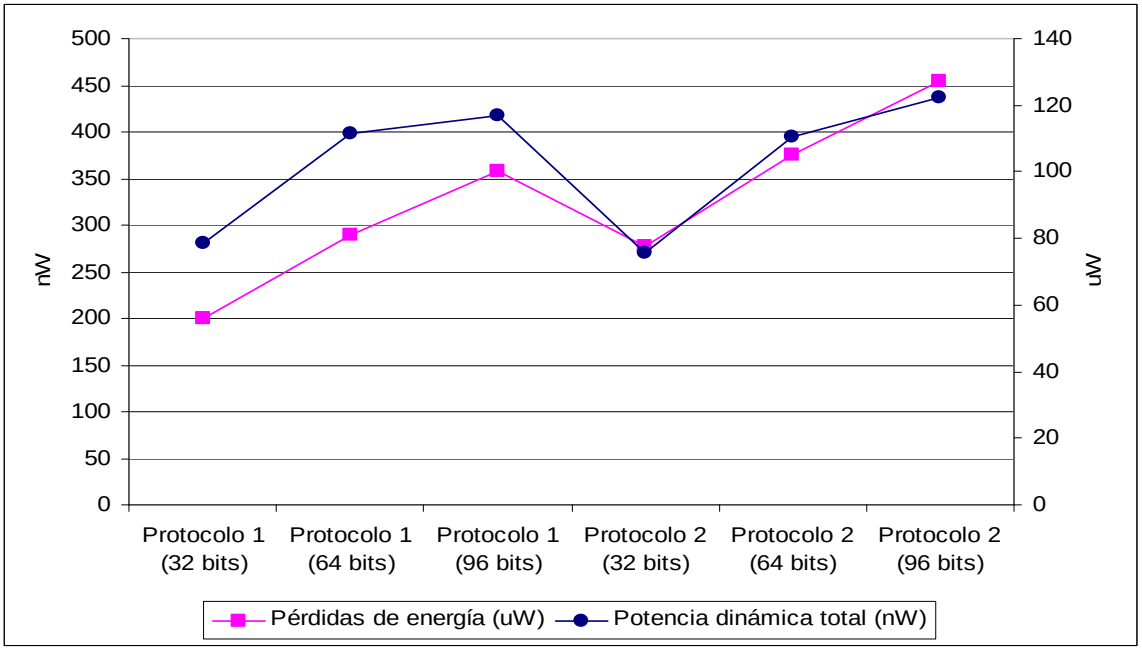

Figura 46: Potencia consumida y pérdidas de cada diseño según el tamaño del prng B2.

5.4.7. Síntesis empleando el generador C

De los datos obtenidos de estas síntesis se realizan las siguientes tablas resumen con los resultados de los diferentes diseños sintetizados en función del nivel de seguridad empleado en el generador de números pseudoaleatorios utilizado.

Resultados de la síntesis del protocolo 1 con prng B2			
	32 bits	**64 bits**	**96 bits**
Número de puertos	68	68	68
Número de nodos	1087	495	494
Número de celdas	931	423	422
Número de referencias	43	30	35
Área combinacional (μm^2)	9048,565	14659,972	20260,951
Área no combinacional (μm^2)	5539,715	7546,966	9611,350
Área de interconexión (μm^2)	785,610	1325,979	1933,137
Área total de celdas (μm^2)	14588,280	22206,938	29872,301
Área total (μm^2)	15373,890	23532,917	31805,438
Puertas equivalentes	2638	4016	5402
Potencia interna	195,248 nW (68%)	217,089 nW (63%)	276,599 nW (68%)
Potencia de conmutación	93,074 nW (32%)	127,663 nW (37%)	128,971 nW (32%)
Potencia dinámica total	288,321 nW	344,752 nW	405,570 nW
Pérdidas de energía	54,063 uW	81,193 uW	107,640 uW

Tabla 15: Resultados de la síntesis del protocolo 1 según el número de bits del prng C.

Resultados de la síntesis del protocolo 2 con prng B2			
	32 bits	**64 bits**	**96 bits**
Número de puertos	133	133	133
Número de nodos	1468	935	902
Número de celdas	1233	811	776
Número de referencias	48	39	32
Área combinacional (μm^2)	11477,824	17311,070	22853,812
Área no combinacional (μm^2)	8922,914	10959,650	13029,566
Área de interconexión (μm^2)	1623,078	2059,400	2805,672
Área total de celdas (μm^2)	20400,738	28270,720	35883,378
Área total (μm^2)	22023,816	30330,120	38689,050
Puertas equivalentes	3690	5113	6490
Potencia interna	207,779 nW (75%)	250,576 nW (76%)	291,330 nW (80%)
Potencia de conmutación	68,373 nW (25%)	80,124 nW (24%)	74,158 nW (20%)
Potencia dinámica total	276,151 nW	330,700 nW	365,488 nW
Pérdidas de energía	77,869 uW	105,230 uW	132,182 uW

Tabla 16: Resultados de la síntesis del protocolo 2 según en número de bits del prng C.

A continuación se muestran diferentes gráficas de los datos obtenidos de la síntesis que permiten realizar una comparación entre las diferentes opciones de que se han implementado con el generador de números pseudoaleatorios C. En la Figura 48 se muestra la comparativa del número de puertas equivalentes necesarias para cada diseño.

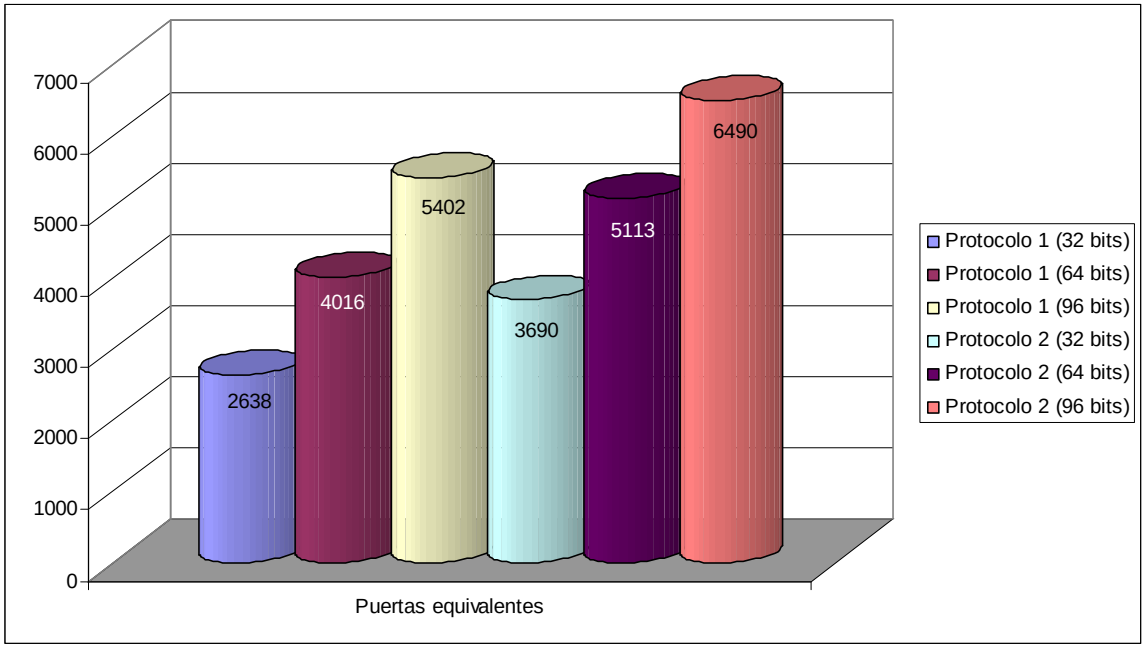

Figura 48: Número de puertas equivalentes de cada diseño según el tamaño del prng C.

Se puede apreciar como los diseños con prng de 32 bits son implementables en ambos casos, pero debido a la disminución del nivel de seguridad que esto supone, sólo se implementarían en el caso en que las opciones de mayor número de bits no fueran factibles. Por otro lado, se ve como para el primer protocolo, aunque ligeramente superior a las rectricciones, podría tomarse como válido el diseño con 64 bits. El resto de opciones, es decir, el protocolo 1 con 96 bits y los casos de 64 y 96 bits para el protocolo 2, no son implementables, ya que superan ampliamente los recursos disponibles.

En la Figura 49 se puede apreciar la distribución de las diferentes áreas en cada uno de los diferentes diseños sintetizados, y como éstas van aumentando a medida que se incrementa el número de bits con los que trabaja el generador empleado.

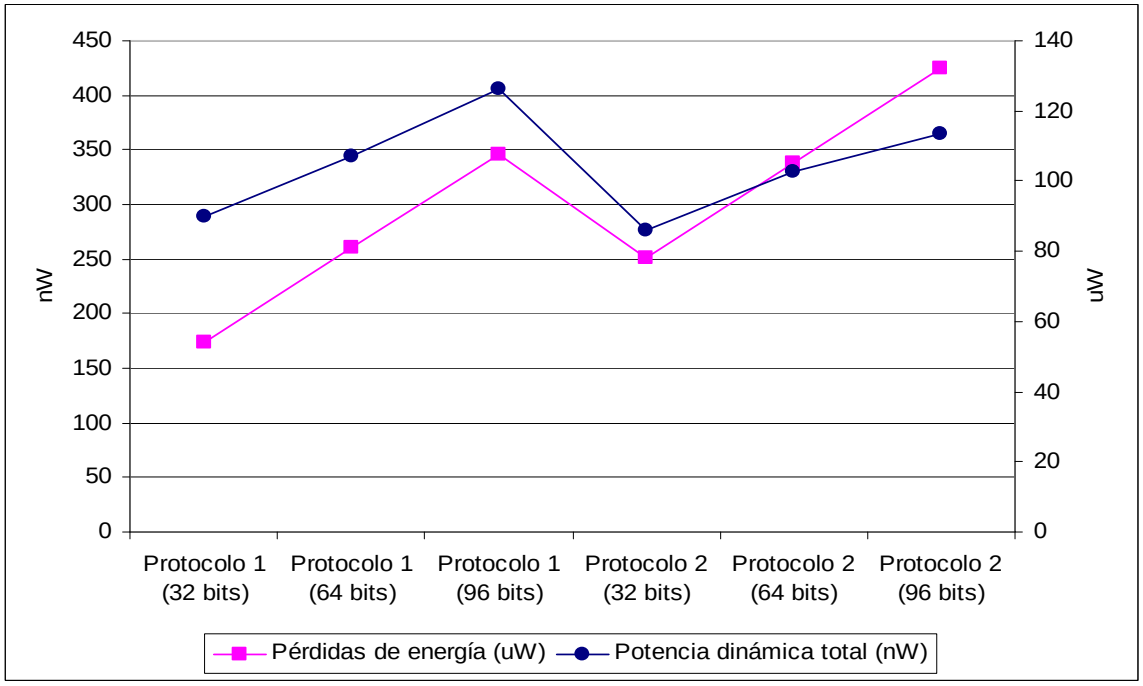

Figura 49: Potencia consumida y pérdidas de cada diseño según el tamaño del prng C.

Por último se muestra la gráfica con la potencia consumida, así como con las pérdidas de potencia en cada diseño en función del número de bits empleados en su implementación.

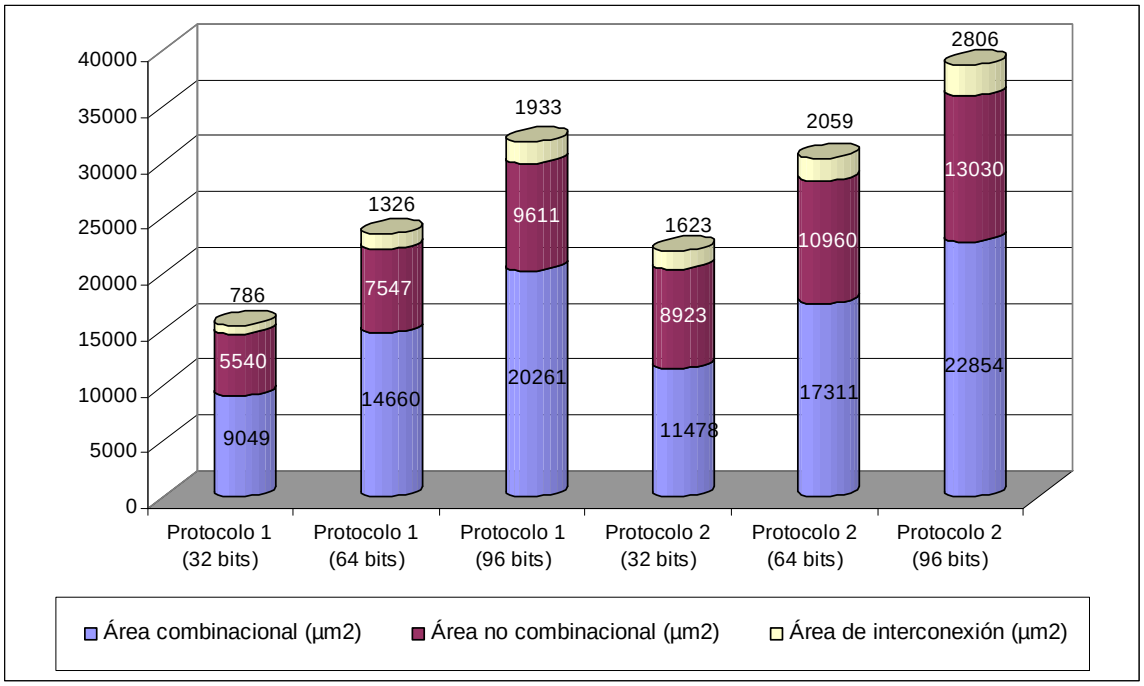

Figura 50: Distribución del área de cada diseño según el tamaño del prng C.

5.5. ANÁLISIS DE LOS RESULTADOS:

A continuación se va a proceder a realizar un análisis de los datos prensentados en los apartados anteriores, así como a la selección de la o las opciones más adecuadas para los objetivos que se persiguen en este proyecto. El objetivo principal es la implementación de un protocolo de autenticación RFID seguro y ligero. Para realizar dicha selección se tendrán en cuenta diferentes factores como son las rectricciones de área, la seguridad del sistema, tanto por parte del generador empleado como del protocolo implementado, la potencia consumida y en última instancia, el tiempo que se tarda en realizar el proceso de autenticación, factor este último, poco determinante en el caso que nos ocupa.

5.5.1. Análisis del consumo de recursos

En los apartados anteriores se han mostrado los resultados de la síntesis de los diferentes diseños, de los que se pueden extraer las siguiente conclusiones. Por un lado se ha visto, como al ir aumentando el número de bits con los que se trabaja, el consumo del sistema tanto en área como en potencia se incrementa. Por otro lado, observando por ejemplo, el número de puertas equivalentes necesarias en cada caso, se aprecia como el segundo protocolo implementado en este proyecto tiene un mayor consumo de recursos, lo que hace más complicado que cumpla las restricciones que presentan los *tags* para el mismo número de bits. Esto queda reflejado también en el tipo de operaciones que realiza cada protocolo (Tabla 17)

(n=n° de bits)		**Protocolo 1**	**Protocolo 2**
Almacenamiento	*Tag*	3*n*	3*n*
Tipos de operaciones	*Tag*	__	2(XOR)
Coste de comunicación	*Tag-Lector*	*n* o 2*n*	2½ *n*
	Lector-Tag	2*n* o 3*n*	1½ *n*

Tabla 17: Resumen de análisis de rendimiento.

A la vista de los resultados obtenidos de la síntesis de los diferentes diseños implementados en los apartados anteriores, se puede concluir que las opciones que cumplen las restricciones de área que se han impuesto en este proyecto son los siguientes.

- *Protocolo 1:*
  - PRNG A: 32, 64 y 96 bits.
  - PRNG A1: 32, 64 y 96 bits.

  Puesto que el generador A1 no ha mejorado los resultados del generador A, los diseños implementados con él no serán tenidos en cuenta a partir de ahora.
  - PRNG B: 32 y 64 bits.
  - PRNG B1: 32 y 64 bits.
  - PRNG B2: 32 y 64 bits.

  Puesto que el generador B2 no ha mejorado los resultados del generador B1, los diseños implementados con él no serán tenidos en cuenta. En el caso del generador B1 sí ha existido mejora en la opción de 64 bits, por lo que sí se considerará.
  - PRNG C: 32 y 64 bits.

- *Protocolo 2:*
  - PRNG A: 32 y 64 bits.
  - PRNG A1: 32 y 64 bits.

  Puesto que el generador A1 no ha mejorado los resultados del generador A, los diseños implementados con él no serán tenidos en cuenta a partir de ahora.
  - PRNG B: 32 bits.
  - PRNG B1: 32 bits.
  - PRNG B2: 32 bits.

  Puesto que ni el generador B1 ni el B2 han mejorado los resultados del generador B, los diseños implementados con éstos no serán tenidos en cuenta a partir de ahora.
  - PRNG C: 32 bits.

5.5.2. Análisis de seguridad

A continuación se va a analizar tanto la seguridad que aporta cada protocolo en sí, como el tipo de generador empleado, con el fin de determinar cual aportaría una mayor robustez al sistema frente a amenazas externas.

En primer lugar vamos a hablar del protocolo 1. Este protocolo es capaz de hacer frente a las amenazas de *replay*, ya que si un ataque de este tipo tiene lugar, se activa el flag de alarma y se hacen necesarios dos números más para la autenticación.

Además de esto, los valores de *RN3*, *RN4* y *RN5* nunca son los mismos lo que contribuye a evitar que un adversario pueda realizar una replica de las comunicaciones, así como deducir datos de comunicaciones anteriores.

Por otro lado, este protocolo es capaz de evitar los ataques *DOS* o de desactivación. Esto se debe a que en todo momento el *tag* y el servidor estan sincronizados y éste almacena en su base de datos tanto el valor de *RN1* nuevo como el de la sesión anterior. De esta forma, aunque un adversario impida que el *tag* pueda actualizar sus claves, siempre se podrán comunicar usando el *RN1* de la última sesión. Esto tiene el inconveniente de la posible trazabilidad de un *tag* entre sesiones, debido a la no actualización de dicho valor en ese caso.

Otro tipo de ataques frente a los que es relativamente robusto este protocolo, son los ataques activos como *man-in-the-middle*. Esto se consigue empleando un temporizador, que controla el tiempo de espera entre una comunicación y otra, evitando la intervención de agentes externos en los mensajes intercambiados. Cuanto más restrictivo sea el margen de espera, mayor será la seguridad que aporte.

En segundo lugar vamos a hablar del protocolo 2. Al igual que el anterior, este protocolo es robusto frente a los ataques de *replay*, ya que en cada sesión se actualizan los valores de *IDS* y *x*, además de generarse de nuevo *R1* y *R2*, lo que hace que el adversario vea dificultada la deducción de comunicaciones anteriores y por tanto la réplica de las mismas.

Por otro lado, este protocolo también es capaz de evitar los ataques *DOS* o de desactivación. Esto es debido a que en todo momento el *tag* y el servidor estan sincronizados y éste almacena en su base de datos tanto el valor de *IDS* nuevo como el de la última sesión. De esta modo, aunque un adversario impida que el *tag* pueda actualizar sus claves, siempre se podrán comunicar usando el *IDS* de la sesión anterior. Esto tiene el inconveniente de la posible trazabilidad de un *tag* entre sesiones, debido a la no actualización de dicho valor en ese caso.

Puesto que este protocolo también emplea un temporizador que controla los tiempos de espera entre una comunicación y otra, esto le permite ser relativamente eficaz frente a ataques activos.

Además ambos protocolos permiten la autenticación mutua mediante el intercambio de datos, que tanto *tag* como lector, deben comparar con los valores almacenados, con el fin de verificar la autenticidad de la otra parte. Igualmente, los dos protocolos implementados se benefician de la seguridad que aporta el uso de generadores de números pseudoaleatorios para la generación de los datos intercambiados, lo que permite dificultar que un adversario pueda predecir o deducir dichos datos.

La seguridad hacia delante no siempre es fácil de conseguir empleando las técnicas criptográficas en la etiquetas de bajo coste, debido a las limitaciones de recursos de las mismas. Es complejo mantener la seguridad del sistema en un futuro una vez que se ha descifrado la clave, lo único que puede hacerse es detectar lo antes posible que dicha clave ha sido comprometida, detener la comunicación y sustituirla. Para prevenir este tipo de problemas de seguridad es necesario emplear generadores suficentemente robustos que impidan al adversario llegar de descifrar la clave.

En la Tabla 18 se muestra un resumen acerca de la seguridad que presentan estos dos protocolos.

	Protocolo 1	**Protocolo 2**
Autenticación mutua	O	O
Prevención frente ataques replay	O	O
Prevención frente ataques DOS	O	O
Prevención frente ataques activos	Δ	Δ
Seguridad hacia delante	Δ	Δ
Seguridad hacia atrás	O	O
O-Satisfecho, Δ-Parcialmente satisfecho, X-No satisfecho		

Tabla 18: Resumen del análisis de seguridad de ambos protocolos.

Una ventaja que presenta el segundo protocolo frente al primero es que las operaciones que realiza para la obtenición de los números que intercambian el *tag* y el lector son más complejas. El primero obtiene los valores únicamente mediante el generador de números pseudoaleatorios, mientras que el segundo realiza operaciones básicas como XOR sobre los números obtenidos del generador, lo que aporta una mayor complejidad a los datos, dificultando de este modo que un adversario pueda deducir el modo en que su obtienen y por tanto descifrarlos.

Un factor muy importante a la hora de realizar el análisis de seguridad de estos protocolos son los generadores de números pseudoaleatorios que se emplean. Tal y como se puede ver en el capítulo 4 de este proyecto, el generador A es el más sencillo de todos puesto que las operaciones que realiza son básicamente desplazamientos y sumas de los bits, aunque por otro lado es el que realiza un mayor números de iteraciones para obtenerlos.

El generador que tiene una mayor complejidad en las operaciones que realiza, es el generador B, puesto que combina desplazamientos con operaciones sencillas como XOR, pero por otro lado es el que menor número de iteraciones realiza. En un nivel intermedio, tanto de complejidad en el proceso de calculo de los valores como de número de iteraciones para obtenerlos, se encuentra el generador C. Por ello, el nivel de seguridad que aportan cada uno de los generadores pseudoaleatorios empleados en estos diseños, es muy similar y viene proporcionado bien por un mayor número de iteraciones o bien por una mayor complejidad en sus operaciones. Lo que hará que estos generadores aporten mayor o menor seguridad a los sistemas implementados serán el número de bits con los que trabaje cada uno de ellos.

Dentro de cada combinación de protocolo-prng, se van a seleccionar aquellos diseños que aportan un nivel de seguridad más elevado por el uso de un mayor número de bits en su generador de números pseudoaleatorios.

- *Protocolo 1:*
  - PRNG A: 96 bits.
  - PRNG B1: 64 bits.
  - PRNG C: 64 bits.
- *Protocolo 2:*
  - PRNG A: 64 bits.
  - PRNG B: 32 bits.
  - PRNG C: 32 bits.

5.5.3. Selección del diseño

A la vista de los análisis de los apartados previos, se va a seleccionar el diseño más adecuado para cada protocolo. Las dos opciones más adecuadas para los objetivos del proyecto son, en ambos protocolos, el uso del generador de números

pseudoaleatorios A. Dicha seleeción se ha llevado a cabo en función de los factores indicados anteriormente.

Habiendo reducido las opciones a aquellas que cumplen las rectricciones de área y que son más seguras dentro de cada combinación protocolo-prng, se ha escogido para cada protocolo el diseño que permite emplear un mayor número de bits en su generador de números. En el caso del protocolo 1, éste puede llegar a trabajar con 96 bits, mientras que en el caso del protocolo 2 tan sólo con 64 bits. Por ese motivo el generador aporta mayor seguridad en el primer caso, pero en contrapartida el algoritmo del segundo protocolo presenta una mayor complejidad, haciendo que sea más seguro frente a las posibles amenazas.

Observando el resto de factores, el diseño implementado con el primer protocolo presenta un menor consumo de área, pero un mayor consumo de potencia. Por otro lado, el protocolo de autenticación implementado con el primer protocolo requiere menos ciclos de reloj para llevar a cabo el proceso de autenticación que el segundo protocolo. Un resumen de estas características se muestra en la Tabla 19.

	Protocolo 1	**Protocolo 2**
Consumo de área (E.G.)	3346	3660
Consumo de potencia (uW)	69,25	75,03
Seguridad del protocolo	Menor	Mayor
Seguridad del prng	96 bits	64 bits
Tiempo de ejecución	210 ciclos	370 ciclos

Tabla 19: Resumen de las características de los dos diseños seleccionados.

CAPÍTULO 6
CONCLUSIONES Y LÍNEAS FUTURAS

6.1. CONCLUSIONES

Tal y como se indicó en el capítulo introductorio de este trabajo, el objetivo del presente proyecto es la implementación y análisis de protocolos de autenticación seguros y ligeros para RFID. Este objetivo se ha alcanzado con éxito, como muestran los resultados obtenidos en el capítulo 5.

En primer lugar, se ha realizado la implementación de dos protocolos de autenticación mediante lenguaje de descripción de hardware, ambos basados en generadores de números pseudoaleatorios. Dichos protocolos han sido seleccionados por sus características de seguridad y ligereza. De este modo se pretenden cubrir dos aspectos fundamentales. Por un lado han de ser algoritmos robustos frente a las posibles amenazas a las que se enfrentan los sistemas RFID. Y por otro lado, han de ser protocolos ligeros que puedan ser implementados dentro de los limitados recursos de área que presentan los *tags* RFID.

En segundo lugar, se ha llevado a cabo la síntesis de los diferentes diseños implementados mediante, los dos protocolos seleccionados y los distintos generadores de números pseudoaleatorios de los que se disponía. Dicha síntesis se ha realizado empleando dos opciones de compilación, de las cuales se ha seleccionado finalmente la que proporcionaba resultados más optimizados. Realizando un análisis de los resultados obtenidos de dichas síntesis, en función de aspectos como el consumo y la seguridad aportada, tanto por los protocolos como por los generadores de números, se han seleccionado las opciones de diseño que mejor cumplen los objetivos de este proyecto.

Las conclusiones que se han obtenido de los resultados y análisis anteriores son las siguientes. Con respecto a los recursos, los resultados muestran como a medida que se incremeta el número de bits con los que se trabaja, aumenta el consumo de área y potencia. Los datos obtenidos permiten concluir qué implementaciones son lás más adecuadas proporcionando un compromiso entre el número de bits con los que se trabaja y los requisitos necesarios. Por otro lado, el segundo protocolo implementado siempre va a precisar un número de puertas equivalentes más elevado, debido a la mayor complejidad que presenta su algoritmo.

Con respecto a la seguridad, se ha tenido en cuenta tanto la seguridad de los protocolos en sí, como la que aportan los diferentes generadores de números pseudoaleatorios empleados. Ambos protocolos permiten la autenticación mutua del lector y el tag, satisfacen la prevención frente a ataques de replay así como de desactivación y presetan seguridad hacia atrás. Mientras que su robustez frente a ataques activos y la seguridad hacia delante quedan parcialmente satisfechas. Por otro lado, el segundo protocolo, como ya se ha mencionado, realiza operaciones más complejas para la obtención de los datos intercambiados entre el lector y el *tag* para llevar a cabo la autenticación mutua. En contrapartida, el menor consumo de recursos del primer protocolo le permite trabajar con un mayor número de bits que el protocolo 2 en los mismo casos. Tal y como se ha analizado en el capítulo anterior, el nivel de seguridad que prensentan cada uno de los generadores de números pseudoaleatorios empleados es muy similar, por lo que será el número de bits con los que trabaja cada uno de ellos lo que hará que aporten más o menos seguridad.

Teniendo en cuenta todo lo anterior, se han seleccionado como las opciones más adecuadas para los objetivos de este proyecto, los diseños implementados mediante el protocolo 1 con el generador A de 96 bits y el protocolo 2 con el generador A de 64 bits. Ambos protocolos de autenticación cumplen las restricciones de área que presentan los tags RFID y cubren unos requisitos mínimos de seguridad necesarios para este tipo de sistemas, cumpliendose de este modo los objetivos planteados en este proyecto.

6.2. LÍNEAS FUTURAS

Como posibles trabajos futuros, se puede considerar la realización de las siguientes mejoras:

a) Se puede realizar una ampliación del primer protocolo implementando un algoritmo de “refresh” o actualización de la semilla del generador cuando se considere que el sistema está comprometido. De esta forma se puede mejorar el protocolo frente a las amenazas de trazabilidad, incrementando la seguridad del mismo. Habría que estudiar si dicha ampliación es factible debido a las limitaciones de área existentes.

b) Realizar un análisis cuantitativo del nivel de seguridad que prensenta cada protocolo y cada generador.
c) Búsqueda e implementación de nuevos generadores de números pseudoaleatorios que se adapten a los sistemas que se han desarrollado, realizando diferentes arquitecturas que permitan optimizar la utilización de recursos.
d) Búsqueda e implementación de nuevos protocolos igualmente ligeros, más complejos que permitan cubrir todos los requisitos de seguridad de los sistemas RFID adecuadamente.
e) Desarrollo e implementación de la parte del lector-servidor de los protocolos implementados en este proyecto.

CAPÍTULO 7
PRESUPUESTO

En el presente capítulo se aborda la estimación de los costes derivados de la realización del proyecto, incluyendo tanto los costes de personal como los de material. Antes de llevar a cabo el presupuesto se van a establecer una serie de cuestiones que se muestran a continuación, con el fin de clarificar los datos que se aporten en el mismo.

- Los costes de personal engloban tanto el proceso de documentación, como el de implementación, pruebas y análisis de las diferentes partes del proyecto.
- La dedicación diaria a la realización del proyecto se considerará de 8 horas.
- Los meses se computarán como 20 días laborables.
- El personal necesario para llevarlo a cabo ha sido un ingeniero, ingeniero industrial en este caso.
- La dedicación del ingeniero se computa en hombres mes, considerando un hombre mes como 131,25 horas de trabajo.
- El coste hombre mes para un ingeniero se considerará de 2694,39 euros.
- Para obtener el coste de los materiales imputable al proyecto, se aplicará la siguiente fórmula de amortización de los mismo.

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

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 periodo de depreciación de los equipos informáticos 60 meses (5años), mientras que el de las licencias de software serán 12 meses.
- Los costes indirectos, es decir, aquellos que no se imputan en ninguna de las partidas mencionadas, se considerarán un 20% de los costes totales del proyecto.

Una vez realizadas estas aclaraciones, se muestra en la siguiente tabla el presupuesto del proyecto que se ha llevado a cabo.

Universidad Carlos III de Madrid

Escuela Politécnica Superior

PRESUPUESTO DE PROYECTO

1. Autor:	Elena de la Iglesia Toribios			
2. Departamento:	Tecnología Electrónica			
3. Descripción del proyecto:				
Título:	Implementación y Análisis de dos Protocolos de Autenticación Seguros y Ligeros para RFID			
Duración:	6 meses			
4. Presupuesto total del proyecto:				
Euros	24.623			
5. Desglose presupuestario: (costes directos)				

PERSONAL

Apellidos y nombre	Categoría	Dedicación (hombres mes)	Coste hombre mes	Coste (euros)
de la Iglesia Toribios, Elena	Ingeniero	7,3	2694,39	19669,05
			Total	**19669,05**

EQUIPOS

Descripción	Coste (Euro)	% Uso dedicado proyecto	Dedicación (meses)	Periodo de depreciación	Coste imputable [d)]
Licencia software Synopsys	1.800,00	100	5	12	750,00
Ordenador sobremesa	1.000,00	100	6	60	100,00
				Total	**850,00**

Tasas de costes indirectos:	20%	

6. Resumen de costes:

Concepto	Costes totales
Personal	19.669
Amortización	850
Subcontratación de tareas	0
Costes de funcionamiento	0
Costes Indirectos	4.104
Total	**24.623**

El presupuesto total de este proyecto asciende a la cantidad de *veinticuatromil seiscientos veintitrés* euros.

Leganes a 20 de Junio de 2010.

El ingeniero proyectista.

CAPÍTULO 8
BIBLIOGRAFÍA

[1] Cristina Fernández, R.: "*Estudio de la tecnología RFID y desarrollo de una aplicación para la localización de personas.*", Proyecto Fin de Carrera, Universidad Carlos III de Madrid, Julio 2009.

[2] Bueno Delgado, M. V., Martínez Sala, A. S., Egea López, E., Vales Alonso, J., García Haro, J.: "*Sistemas globales de localización y trazabilidad mediante identificación por radiofrecuencia (RFID).*" Departamento de Tecnologías de la Información y las Comunicaciones, Universidad Politécnica de Cartagena. http://repositorio.bib.upct.es/dspace/bitstream/10317/347/1/2005_AI_7.pdf

[3] Izquierdo Donoso, H.: "*Implementación hardware de algoritmos criptográficos para RFID*", Proyecto Fin de Carrera, Universidad Carlos III de Madrid, Febrero 2009.

[4] Blazquez del Toro, L. M.: "*Sistemas de identificación por radiofrecuencia*". http://www.it.uc3m.es/jmb/RFID/rfid.pdf

[5] Ahson, S. y Ilyas M.: "*RFID Handbook*", Taylor & Francis Group, 2008. http://books.google.com/books?id=q4aCyZnq0cwC&pg=PA591&lpg=PA591&dq=equivalent+gates+rfid&source=bl&ots=nK5MOtXJ27&sig=_q_5AALi1W_A77qTlpNhS2_FfBs&hl=en&ei=_WDRS42LGtuWOI_Ind8N&sa=X&oi=book_result&ct=result&resnum=2&ved=0CBAQ6AEwAQ#v=onepage&q=equivalent%20gates%20rfid&f=false

[6] Martínez Belmonte, M. J.: "*Validación de protocolos de acceso al medio probabilísticos en sistemas RFID con dispositivos pasivos*", Proyecto Fin de Carrera, Universidad Politécnica de Cartagena, septiembre 2008. http://repositorio.bib.upct.es:8080/dspace/bitstream/10317/770/1/pfc2808.pdf

[7] "*Sistemas de identificación electrónica para control y gestión de activos*" Jornada Conectividad Inalámbrica, ROBOTIKER, 26 Febrero 2004. http://www.robotiker.com/castellano/noticias/eventos_pdf/33/RBTK_Inigo.pdf

[8] Gotor Carrasco, E.: "*Estado del arte en tecnologías RFID*", Proyecto Fin de Carrera, Universidad Politécnica de Madrid, Junio 2009. http://www.criptored.upm.es/guiateoria/gt_m001s.htm

[9] Jabbar, H. y Jeong, T. T.: "*Radio Frequency Identification Fundamentals and Applications, Bringing Research to Practice*", Universidad Myongji, Corea. http://sciyo.com/download/pdf/pdfs_id/8487?...39k13d81v7ljrii**spot**11toiq

[10] http://teknik-informatika.com/perangkat-input-rfid/

[11] Ciudad Herrera, J. M. y Samá Casanovas, E.: "*Estudio, diseño y simulación de un sistema RFID basado en EPC*", Proyecto Fin de Carrera, Universidad de Cataluña, septiembre 2005. http://upcommons.upc.edu/pfc/bitstream/2099.1/3552/2/40883-2.pdf

[12] López Herrera, C.: "*Análisis de la viabilidad e implantación de sistemas de identificación basados en radiofrecuencia*", Proyecto Fin de Carrera, Universidad Carlos III, septiembre 2008.

[13] http://control-accesos.es/sistemas-de-identificacion/9

[14] Fernández Carmona, M., Urdiales, C. y Sandoval, F.: "*Lectores móviles de RFID y sus aplicaciones sanitarias*", Universidad de Málaga. http://www.istshareit.eu/shareit/Members/admin/Lectores_Integrados_de_RFID_y_sus_aplicaciones_sanitarias.pdf

[15] SCDigest, "*Logistics and RFID News: Slowly, RFID-Enabled Distribution Centers Start to Emerge*", Octubre 2008. http://www.scdigest.com/assets/newsviews/08-10-06-1.pdf

[16] "*RFID Logística*" AT4 Wireless.

http://www.at4wireless.com/download/AT4_wireless_RFID_Logistica.pdf

[17] Gardner, C.: "*La RFID en el mercado de componentes de sustitución para automóviles*", 2004.
http://www.epcglobalsp.org/papers/Automotive_Aftermarket%20_REVIS.pdf

[18] Martínez León, J. A., Martínez Sala, A. S., Egea López, E., Vales Alonso, J. y García Haro, J.: "*Megaestand: plataforma telemática para la gestión de palets reutilizables en la cadena de suministro mediante tecnologíia RFID*", Universidad Politécnica de Cartagena.
http://repositorio.bib.upct.es:8080/dspace/bitstream/10317/386/1/2006_AI_7.pdf

[19] Martínez Sala, A. S., Egea López, E., García Haro, J. y Monzó Sánchez, M.: "*Evaluación de dispositivos RFID acoplados a estanterias móviles para automatizar su gestión y control*", Universidad de Cartagena. http://w3.iec.csic.es/URSI/articulos_modernos/articulos_coruna_2003/actas_pdf/SESION%204/S4.%20Aula%202.5/1538-EVALUACION.PDF

[20] Página Oficial EPCGlobal, http://www.epcglobalsp.org/

[21] "*GS1 EPCglobal Glossary: Terms and terminology within the GS1 EPCglobal standars*", EPCGlobal, Junio 2009.
http://www.epcglobalinc.org/home/GS1_EPCglobal_Glossary_V35_KS_June_09_2009.pdf

[22] Vales Alonso, J., González Castaño, F. J., Egea López, E., Bueno Delgado, M. V., Martínez Sala, A. y García Haro, J.: "*Evaluación de CSMA no persistente como protocolo anticolisión en sistemas RFID activos*.", Área de Ingeniería Telemática. E.T:S. Ingeniería de Telecomunicación, Universidad Politécnica de Cartagena, noviembre 2007.
http://ait.upct.es/~eegea/pub/jorRFID.pdf

[23] Amezaga i Saumell, I.: "*Seguridad y privacidad en RFID*", Octubre 2008. http://www.robotiker.com/castellano/noticias/eventos_pdf/80/seguridad_y_privacidad_rfid.pdf

[24] "*Seguridad RFID*", Escuela Politécnica Nacional, Ingeniería Electrónica y de Comunicaciones, Comunicaciones Inalámbricas. http://www.docstoc.com/docs/29422149/Seguridad-RFID/

[25] Meliá-Segen, J., Herrera-Joancomartí, J. y García-Alfaro, J.: "*Análisis de seguridad y privacidad para sistemas EPC-RFID en el sector postal*" Universidad Oberta de Cataluña y Universidad Autonoma de Barcelona. http://ccd.uab.es/~joaquin/papers/recsi08-postal.pdf

[26] Lim, C. H. y Kwon, T.: "*Strong and robust RFID authentication enabling perfect ownership transfer*", Universidad de Sejong, Corea.

[27] Ohkubo, M., Suzuki, K. y Kinoshita, S.: "*Cryptographic Approach to "privacy-friendly" tags*", Compañía Nipona de Telegrafos y Telefonos, Japón.

[28] Phan, R. C.-W., Wu, J., Ouafi, K. y Stinson, D. R.: "*Privacy analysis of forward and backward untraceable RFID authentication schemes*", Universidad de Waterloo, Canada.

[29] Kwak, M., Kim, J. y Kim, K.: "*Desynchronization and cloning resistant light-weight RFID authentication protocolo using integer arithmetic for low-cost tags*", Universidad de Información y Comunicaciones de Munji-Dong, Korea. http://caislab.kaist.ac.kr/publication/paper_files/2009/SCIS2009_Minhea.pdf

[30] Rosado, A. y Bataller, M.: "*Introducción al software Xilinx ISE*", Universidad de Valencia, 2003. http://www.uv.es/rosado/dcse/prac1XilinxISEintrod.pdf

[31] Xilinx Inc., "*ISE 8.1i Quick Start Tutorial*".

[32] Xilinx Inc., "*ModelSim SE User´s Manual*", 2006.

[33] Synopsys, "*Design Vision. User Guide*", Marzo 2003, Synopsys Inc.

[34] "*Synopsys Synthesis Tutorial*" Departamento de Ingenieria Eléctrica, Universidad de San Jose. http://www.engr.sjsu.edu/tle/271_Syn_tut.pdf

[35] "*Design Compiler. User Guide*", Marzo 2007, Synopsys Inc.

[36] http://www.super-computing.org/pi-hexa_current.html

[37] Thompson, P. J. R.: "*Lightweight Cryptography for Power Constrained Microcontrollers*", Universidad de Leeds, Agosto 2009.

[38] Yang, J., Ren, K., Kim, K.: "*Security and privacy on authentication protocol for low-cost RFID*", Universidad de Información y comunicación, Korea e Instituto Plotécnico de Worcester, U.S.A., 2005. http://www.springerlink.com/content/v553732813x6262k/fulltext.pdf

[39] Burmester, M. y Munilla, J.: "*A flyweight RFID authentication protocol*" Universidad de Florida, U.S.A. y Universidad de Málaga, España. http://www.cs.fsu.edu/~burmeste/100.pdf

[40] Chien, H.-Y. y Huang, C.W.: "*A lightweight authentication protocol for low-cost RFID*", Universidad Nacional de Chin Nan, Taiwan, China, 2008. http://www.springerlink.com/content/g1775nk456v33h63/fulltext.pdf

[41] Grau Segura, M.; "*Implementación en VHDL de generadores de números pseudoaleatorios criptograficamentes seguros*", Proyecto Fin de Carrera, Universidad Carlos III de Madrid, Octubre 2009.

[42] Peris López, P.: "*Lightweight Cryptography in Radio Frequency Identification (RFID) Systems*", Tesis de Doctorado, Universidad Carlos III de Madrid, octubre 2008. http://www.lightweightcryptography.com/

[43] "*Digital Standard Cell Library. Saed_EDK90_Core. Databook*", Synopsys Inc., 2009.

[44] Feldhofer, M., Dominikus, S. y Wolkerstorfer, J.: "*Strong authentication for RFID systems using the AES algorithm*", Institute for Applied Information Processing and Communications, Austria, 2004. http://fara.cs.uni-potsdam.de/~engelman/13.%20Semester/Master%20Thesis/Materialien%20und%20Originalquellen/Arbeiten/Strong%20Authentication%20for%20RFID%20Systems%20using%20the%20AES%20Algorithm.pdf

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

[46] "*Design Compiler Ultra*" Synopsys Inc., 2010.

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

ANEXOS

ANEXO A: CÓDIGOS VHDL DE LOS PROTOCOLOS DE AUTENTICACIÓN

```
--------------------------------------------------------------------------------
--------------------------- PROTOCOLO 1 ----------------------------------------
--------------------------------------------------------------------------------

------ SISTEMA DE AUTENTICACIÓN ------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Sist_autenticacion is
    port (clk, reset: in std_logic;
        query: in std_logic;
        aceptado: in std_logic;
        dato_e_lector: in std_logic_vector (31 downto 0);
        dato_s_lector: out std_logic_vector (31 downto 0));
end Sist_autenticacion;

architecture Behavioral of Sist_autenticacion is

signal tcumplido_a, fin_generacion_a: std_logic;
signal activacion_a, generar_a: std_logic;
signal escribir_a: std_logic;
signal posicion_a: std_logic_vector (1 downto 0);
signal dato_s_mem: std_logic_vector (31 downto 0);
signal dato_e_mem: std_logic_vector (31 downto 0);

component Maquina
port(   clk : in  STD_LOGIC;
    reset : in  STD_LOGIC;
    query : in STD_LOGIC;
    dato_memoria: in std_logic_vector (31 downto 0);
    dato_generado: in std_logic_vector (31 downto 0);
    dato_e_lector: in std_logic_vector (31 downto 0);
    tcumplido : in STD_LOGIC;
    aceptado : in STD_LOGIC;
    fin_generacion: in std_logic;
    dato_s_lector: out std_logic_vector (31 downto 0);
    escribir: out std_logic;
    posicion: out std_logic_vector (1 downto 0);
    generar : out STD_LOGIC;
    activacion : out STD_LOGIC);
end component;

component Temporizador
port(   clk : in  STD_LOGIC;
    reset : in  STD_LOGIC;
    activacion : in  STD_LOGIC;
    tcumplido : out  STD_LOGIC);
end component;

component Memoria
port(   clk : in  STD_LOGIC;
    reset : in  STD_LOGIC;
    dato_entrada : in  STD_LOGIC_VECTOR (31 downto 0);
    dato_salida : out  STD_LOGIC_VECTOR (31 downto 0);
    escribir : in  STD_LOGIC;
    posicion : in  std_logic_vector(1 downto 0));
end component;

component prngA8
GENERIC(N:INTEGER:=64);
port(   z0:OUT std_logic_vector((n/2)-1 DOWNTO 0);
    INICIO: IN STD_LOGIC;
    FIN: OUT STD_LOGIC;
```

```
    CLK,RESET: IN STD_LOGIC);
end component;

begin

mi_Maquina: Maquina port map (clk, reset, query, dato_s_mem, dato_e_mem,
dato_e_lector, tcumplido_a, aceptado,
fin_generacion_a, dato_s_lector, escribir_a, posicion_a, generar_a, activacion_a);
mi_Temporizador: Temporizador port map (clk, reset, activacion_a, tcumplido_a);
mi_Memoria: Memoria port map (clk, reset, dato_e_mem, dato_s_mem, escribir_a,
posicion_a);
mi_Generador: prngA8 port map (dato_e_mem , generar_a, fin_generacion_a, clk, reset
);

end Behavioral;

------ MAQUINA DE ESTADOS ------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;


entity Maquina is
Port (clk : in  STD_LOGIC;
        reset : in  STD_LOGIC;
        query : in STD_LOGIC;
        dato_memoria: in std_logic_vector (31 downto 0);
        dato_generado: in std_logic_vector (31 downto 0);
        dato_e_lector: in std_logic_vector (31 downto 0);
        tcumplido : in STD_LOGIC;
        aceptado : in STD_LOGIC;
        fin_generacion: in std_logic;
        dato_s_lector: out std_logic_vector (31 downto 0);
        escribir: out std_logic;
        posicion: out std_logic_vector (1 downto 0);
        generar : out STD_LOGIC;
        activacion : out STD_LOGIC);
end Maquina;

architecture Maquina_a of Maquina is

type estado is (inicializacion, reposo, envio_rn1, esp_comp_rn2, envio_rn3,
memorizar_rn3,
envio_rn4, espera_acep, esp_comp_rn3, envio_rn5, actualizacion);
signal actual, siguiente: estado;
signal alarma, cont, fin, gen_aux: std_logic;
signal cont_aux: std_logic_vector (1 downto 0);

begin

process(clk, reset)
begin
    if reset='1' then
        actual <= inicializacion;
    elsif clk'event and clk='1' then
        actual <= siguiente;
    end if;
end process;

process(actual, query, tcumplido, dato_e_lector, fin, aceptado, cont_aux,
fin_generacion, dato_memoria,
dato_generado, gen_aux, alarma)
variable cont_alarma: std_logic;
begin
case actual is
    when inicializacion =>escribir<='1';
                    activacion <= '0';
                    generar<=gen_aux;
                    posicion<=cont_aux;
                    dato_s_lector<=(others=>'0');
                    if fin= '1' then
                        siguiente <= reposo;
                    else
```

```
                    siguiente <= inicializacion;
                end if;
    when reposo =>  escribir <= '0';
                activacion <= '0';
                generar <='0';
                posicion<="00";
                dato_s_lector<=(others=>'0');
                if query = '1' then
                    siguiente <= envio_rn1;
                else
                    siguiente <= reposo;
                end if;
    when envio_rn1 =>   escribir <= '0';
                    activacion <= '0';
                    generar <='0';
                    posicion<="01";
                    dato_s_lector<=dato_memoria;
                    siguiente <= esp_comp_rn2;
    when esp_comp_rn2 =>    escribir <= '0';
                    activacion <= '1';
                    generar <='0';
                    posicion<="10";
                    dato_s_lector<=(others=>'0');
                    if tcumplido = '0' then
                    if dato_memoria=dato_e_lector then
                        if alarma = '1' then
                        siguiente <= memorizar_rn3;
                        else
                        siguiente <= envio_rn3;
                        end if;
                    else
                        siguiente <= esp_comp_rn2;
                    end if;
                    else
                        siguiente <= reposo;
                    end if;
    when envio_rn3 =>   escribir <= '0';
                    activacion <= '0';
                    generar <='1';
                    posicion<="00";
                    dato_s_lector<=dato_generado;
                    if fin_generacion='1' then
                        siguiente <= espera_acep;
                    else
                        siguiente <= envio_rn3;
                    end if;
    when memorizar_rn3 =>   escribir <= '1';
                        activacion <= '0';
                        generar <='1';
                        posicion<="11";
                        dato_s_lector<=dato_generado;
                        if fin_generacion='1' then
                            siguiente <= envio_rn4;
                        else
                            siguiente<= memorizar_rn3;
                        end if;
    when envio_rn4 =>   escribir <= '0';
                    activacion <= '0';
                    generar <='1';
                    posicion<="11";
                    dato_s_lector<= dato_generado;
                    if fin_generacion='1' then
                        siguiente <= esp_comp_rn3;
                    else
                        siguiente <= envio_rn4;
                    end if;
    when espera_acep => escribir <= '0';
                    activacion <= '1';
                    generar <='0';
                    posicion<="00";
                    dato_s_lector<=(others=>'0');
                    if tcumplido = '1' then
                        siguiente <= reposo;
                    elsif tcumplido = '0' then
```

```
                        if aceptado = '1' then
                        siguiente <= actualizacion;
                        else
                        siguiente <= espera_acep;
                        end if;
                    end if;
    when esp_comp_rn3 =>    escribir <= '0';
                    activacion <= '1';
                    generar <='0';
                    posicion<="11";
                    dato_s_lector<=(others=>'0');
                    if tcumplido = '0' then
                    if dato_memoria=dato_e_lector then
                        siguiente <= envio_rn5;
                    else
                        siguiente <= esp_comp_rn3;
                    end if;
                    else
                        siguiente <= reposo;
                    end if;
    when envio_rn5 =>   escribir <= '0';
                    activacion <= '0';
                    generar <='1';
                    posicion<="00";
                    dato_s_lector<=dato_generado;
                    if fin_generacion='1' then
                        siguiente <= espera_acep;
                    else
                        siguiente <= envio_rn5;
                    end if;
    when actualizacion =>   escribir<='1';
                    activacion <= '0';
                    generar<=gen_aux;
                    posicion<=cont_aux;
                    dato_s_lector<=(others=>'0');
                    if fin = '1' then
                        siguiente <= reposo;
                    else
                        siguiente <= actualizacion;
                    end if;
end case;
end process;

process(clk, reset)
begin
    if reset='1' then
        cont_aux<="01";
    elsif clk'event and clk='1' then
        if actual=actualizacion or actual=inicializacion then
            if fin_generacion='1' then
                cont_aux<=cont_aux+'1';
            end if;
        else
            cont_aux<="01";
        end if;
    end if;
end process;
fin<='1' when cont_aux="11" else '0';
gen_aux<='1' when cont_aux>"00" and cont_aux<"11" else '0';

process(clk, reset)
begin
    if reset='1' then
        cont<='0';
        alarma<='0';
    elsif clk'event and clk='1' then
        if actual=reposo then
            alarma<=cont;
        elsif actual=envio_rn1 then
            cont<='1';
        end if;
    end if;
end process;
```

```
end Maquina_a;


------ 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)
    variable contador_aux: std_logic_vector (4 downto 0):="00000";
    begin
        if reset = '1' then
            contador_aux := "00000";
        elsif clk'event and clk='1' then
            if activacion = '1' then
                contador_aux := contador_aux + '1';
            else
                contador_aux := "00000";
            end if;
        end if;
        contador<=contador_aux;
    end process;
    tcumplido <= '1' when contador="00101" else '0';
end Temporizador_a;

------ MEMORIA ------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity Memoria is
    Port ( clk : in  STD_LOGIC;
           reset : in  STD_LOGIC;
           dato_entrada : in  STD_LOGIC_VECTOR (31 downto 0);
           dato_salida: out  STD_LOGIC_VECTOR (31 downto 0);
           escribir : in  STD_LOGIC;
           posicion : in  std_logic_vector(1 downto 0));
end Memoria;
architecture Memoria_a of Memoria is
signal rn1, rn2, rn3: std_logic_vector(31 downto 0);
begin
process(clk, reset)
variable aux: std_logic_vector(31 downto 0);
begin
    if reset='1' then
        rn3<=(others=>'0');
    elsif clk'event and clk='1' then
        if escribir='1' then
            case posicion is
                when "01" =>  rn1<=dato_entrada;
                when "10" =>  rn2<=dato_entrada;
                when "11" =>  rn3<=dato_entrada;
                when others => aux:=dato_entrada;
            end case;
        end if;
    end if;
end process;
dato_salida<=rn1 when posicion="001" else
          rn2 when posicion="010" else
          rn3 when posicion="011" else
         (others=>'0');
end Memoria_a;
```

```
--------------------------------------------------------------------------------
---------------------------- PROTOCOLO 2 ---------------------------------------
--------------------------------------------------------------------------------

------ SISTEMA DE AUTENTICACIÓN ------

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 Sist_autenticacion_2 is
    generic(m: integer := 32);
    port (clk, reset: in std_logic;
        query_R1: in std_logic_vector (m downto 0);
        rr: in std_logic_vector ((m/2)-1 downto 0);
        ids: out std_logic_vector (m-1 downto 0);
        r2_r: out std_logic_vector ((m+m/2)-1 downto 0);
        ok: out std_logic;
        no: out std_logic
        );
end Sist_autenticacion_2;

architecture Behavioral of Sist_autenticacion_2 is

signal tcumplido_a, fin_generacion_a: std_logic;
signal activacion_a, generar_a: std_logic;
signal dato_generado_a: std_logic_vector (m-1 downto 0);
signal semilla_a: std_logic_vector ((2*m)-1 downto 0);

component Maquina_2
    generic(m: integer :=32);
    port(   clk : in  STD_LOGIC;
            reset : in  STD_LOGIC;
        query_r1 : in STD_LOGIC_vector (m downto 0);
        rr: in std_logic_vector ((m/2)-1 downto 0);
        dato_generado: in std_logic_vector (m-1 downto 0);
        tcumplido : in STD_LOGIC;
        fin_generacion: in std_logic;
        ids: out std_logic_vector (m-1 downto 0);
        r2_r: out std_logic_vector ((m+m/2)-1 downto 0);
        semilla: out std_logic_vector ((2*m)-1 downto 0);
        activacion: out std_logic;
        generar : out STD_LOGIC;
        ok: out STD_LOGIC;
        no: out std_logic);
end component;

component Temporizador_2
    port(     clk : in  STD_LOGIC;
           reset : in  STD_LOGIC;
           activacion : in  STD_LOGIC;
           tcumplido : out  STD_LOGIC);
end component;

component Generador
    GENERIC(N:INTEGER:=64);
    port(   z0:OUT std_logic_vector((n/2)-1 DOWNTO 0);
         INICIO: IN STD_LOGIC;
         FIN: OUT STD_LOGIC;
            semilla: in std_logic_vector (n-1 downto 0);
         CLK,RESET: IN STD_LOGIC);
end component;

begin

mi_Maquina: Maquina_2 port map (clk, reset, query_r1, rr, dato_generado_a,
tcumplido_a, fin_generacion_a, ids, r2_r, semilla_a, activacion_a, generar_a, ok,
no);
mi_Temporizador: Temporizador_2 port map (clk, reset, activacion_a, tcumplido_a);
mi_Generador: Generador port map (dato_generado_a, generar_a, fin_generacion_a,
semilla_a, clk, reset);
```

```
end Behavioral;

------ MAQUINA DE ESTADOS ------
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 Maquina_2 is
generic (m: integer := 32);
Port ( clk : in  STD_LOGIC;
        reset : in  STD_LOGIC;
        query_r1 : in STD_LOGIC_vector (m downto 0);
        rr: in std_logic_vector ((m/2)-1 downto 0);
        dato_generado: in std_logic_vector (m-1 downto 0);
        tcumplido : in STD_LOGIC;
        fin_generacion: in std_logic;
        ids: out std_logic_vector (m-1 downto 0);
        r2_r: out std_logic_vector ((m+(m/2))-1 downto 0);
        semilla: out std_logic_vector ((2*m)-1 downto 0);
        activacion: out std_logic;
        generar : out STD_LOGIC;
        ok: out STD_LOGIC;
        no: out std_logic
);
end Maquina_2;

architecture Maquina_a of Maquina_2 is

type estado is (reposo, generacion_r2, generacion_g, calculo_sid, calculo_comp_rr,
actualizacion_ids, actualizacion_x);
signal actual, siguiente: estado;
signal r1, r2, g, sid_b: std_logic_vector (m-1 downto 0);
signal r_aux: std_logic_vector (m-1 downto 0);
signal r: std_logic_vector ((m/2)-1 downto 0);
signal mem_dato: std_logic;
signal fin_a: std_logic;
signal g_aux: integer range 0 to 31;
signal x_int: std_logic_vector ((m-1) downto 0);
signal ids_int: std_logic_vector ((m-1) downto 0);
signal cont_aux: integer range 0 to 31;
signal ena_cont, clr_cont, fin_calculo_r : std_logic;
constant sid: std_logic_vector (m-1 downto 0):= x"6A2AAB10";
constant ceros: std_logic_vector (m-1 downto 0) :=
"00000000000000000000000000000000";

begin

process(clk, reset)
begin
if reset='1' then
    actual <= reposo;
elsif clk'event and clk='1' then
    actual <= siguiente;
end if;
end process;

process(actual, query_r1, fin_generacion, rr, tcumplido, r, r1, r2, g,
ids_int, x_int, sid_b, r_aux, fin_a, cont_aux, fin_calculo_r)
begin
case actual is
when reposo =>  generar <= '0';
                semilla <= (others=>'0');
                mem_dato <= '1';
                activacion <= '0';
                ok <= '0';
                ena_cont <= '0';
                clr_cont <= '1';
                if query_r1(m) = '1' then
                    siguiente <= generacion_r2;
                else
                    siguiente <= reposo;
```

```
                end if;
when generacion_r2 =>   generar <= '1';
                        semilla <= x"60E65525F3AA55AB";
                        mem_dato <= '1';
                        activacion <= '0';
                        ok <= '0';
                        ena_cont <= '0';
                        clr_cont <= '0';
                        if fin_generacion = '1' then
                            siguiente <= generacion_g;
                        else
                            siguiente <= generacion_r2;
                        end if;
when generacion_g =>    generar <= '1';
                        ena_cont <= fin_generacion;
                        if cont_aux=0 then
                            semilla <= (r1 & r2);
                        elsif cont_aux=1 then
                            semilla <= (ceros & x_int);
                        else
                            semilla <= (others=>'0');
                        end if;
                        mem_dato <= '1';
                        activacion <= '0';
                        ok <= '0';
                        if fin_a = '1' then
                            siguiente <= calculo_sid;
                            clr_cont <= '1';
                        else
                            siguiente <= generacion_g;
                            clr_cont <= '0';
                        end if;
when calculo_sid => generar <= '0';
                    semilla <= (others=>'0');
                    mem_dato <= '0';
                    activacion <= '0';
                    ok <= '0';
                    ena_cont <= '1';
                    if fin_calculo_r = '1' then
                        clr_cont <= '1';
                        siguiente <= calculo_comp_rr;
                    else
                        clr_cont <= '0';
                        siguiente <= calculo_sid;
                    end if;
when calculo_comp_rr => generar <= '0';
                        semilla <= (others=>'0');
                        mem_dato <= '0';
                        activacion <= '1';
                        ena_cont <= '0';
                        clr_cont <= '0';
                        ok <= '0';
                        if tcumplido='0' then
                            if (r_aux ((m/2)-1 downto 0)) = rr then
                                siguiente <= actualizacion_ids;
                            else
                                siguiente <= calculo_comp_rr;
                            end if;
                        else
                            siguiente <= reposo;
                        end if;
when actualizacion_ids =>   generar <= '1';
                            ena_cont <= '0';
                            semilla <= (ids_int & sid_b);
                            mem_dato <= '0';
                            activacion <= '0';
                            ok <= '1';
                            if fin_generacion = '1' then
                                siguiente <= actualizacion_x;
                                clr_cont  <= '1';
                            else
                                siguiente <= actualizacion_ids;
                                clr_cont <= '0';
                            end if;
```

```
when actualizacion_x => generar <= '1';
                        ena_cont <= '0';
                        clr_cont <= '0';
                        semilla <= (x_int & g);
                        mem_dato <= '0';
                        activacion <= '0';
                        ok <= '0';
                        if fin_generacion = '1' then
                            siguiente <= reposo;
                        else
                            siguiente <= actualizacion_x;
                        end if;
when others => generar <= '0';
                ena_cont <= '0';
                clr_cont <= '0';
                semilla <= (others=>'0');
                mem_dato <= '0';
                activacion <= '0';
                ok <= '0';
                siguiente <= reposo;
end case;
end process;
ids <= ids_int when fin_calculo_r='1' else (others=>'0');
r2_r <= (r2 & r) when fin_calculo_r='1' else (others=>'0');
no <= '1' when tcumplido='1' else '0';

process(clk, reset)
begin
    if reset='1' then
        cont_aux <=0;
    elsif clk'event and clk='1' then
        if clr_cont = '1' then
            cont_aux<= 0;
        elsif ena_cont = '1' then
            cont_aux<=cont_aux+1;
        end if;
    end if;
end process;

fin_a<='1' when cont_aux=1 and fin_generacion = '1' else '0';

PROCESS(reset,clk)

BEGIN
if reset='1' then
    r1 <= (others => '0');
    r2 <= (others => '0');
    g <= (others => '0');
    x_int<= x"B4CC5C34";
    ids_int<= x"314B2778";
elsif clk'EVENT and clk='1' then
    if mem_dato='1' then
        if actual = reposo then
            r1 <= query_r1(m-1 downto 0);
        elsif actual = generacion_r2 then
            r2 <= dato_generado;
        elsif actual = generacion_g then
        if cont_aux=0 then
            g(m-1 downto (m/2))  <= dato_generado((m/2)-1 downto 0);
        else
            g((m/2)-1 downto 0) <= dato_generado((m/2)-1 downto 0);
        end if;
    end if;
end if;
if actual=actualizacion_ids and fin_generacion='1' then
    ids_int <= dato_generado;
end if;
if actual=actualizacion_x and fin_generacion='1' then
    x_int <= dato_generado;
end if;
end if;
end process;

process (clk, reset)
```

```
begin
if reset='1' then
    sid_b <= sid;
elsif clk'event and clk='1' then
    if actual = calculo_sid and cont_aux < g_aux then
        sid_b <= (sid_b(m-2 downto 0) & sid_b(m-1));
    end if;
end if;
end process;
fin_calculo_r <= '0' when ((cont_aux < g_aux and actual=calculo_sid) or actual/=
calculo_sid) else '1';
r_aux <= sid_b xor g;
r <= r_aux (m-1 downto m/2);
g_aux <= conv_integer(std_logic_vector(g (4 downto 0)));

end Maquina_a;


------ TEMPORIZADOR ------

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 Temporizador_2 is
    Port ( clk : in  STD_LOGIC;
           reset : in  STD_LOGIC;
           activacion : in  STD_LOGIC;
           tcumplido : out  std_logic);
end Temporizador_2;

architecture Temporizador_a of Temporizador_2 is
signal contador: std_logic_vector (4 downto 0);
begin
    process(clk, reset)
    variable contador_aux: std_logic_vector (4 downto 0):="00000";
    begin
        if reset = '1' then
            contador_aux := "00000";
        elsif clk'event and clk='1' then
            if activacion = '1' then
                contador_aux := contador_aux + '1';
            else
                contador_aux := "00000";
            end if;
        end if;
        contador<=contador_aux;
    end process;
    tcumplido <= '1' when contador="00110" else '0';
end Temporizador_a;
```

ANEXO B: CÓDIGOS VHDL DE LOS PRNGs EMPLEADOS

```
--------------------------------------------------------------------------------
-------------------------- GENERADOR A ---------------------------------
--------------------------------------------------------------------------------

-- GENERADOR A 64 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 prngA8 IS
GENERIC(N:INTEGER:=64);
PORT(   z0:OUT std_logic_vector((n/2)-1 DOWNTO 0);
    INICIO: IN STD_LOGIC;
    FIN: OUT STD_LOGIC;
    CLK,RESET: IN STD_LOGIC);
END prngA8;


--ARQUITECTURA
architecture opcionA of prngA8 is

--DECLARACION DE SEÑALES
SIGNAL z: std_logic_vector((N-1) DOWNTO 0);
SIGNAL contador_i: INTEGER RANGE 0 TO 64;
SIGNAL start:bit:='0';
SIGNAL enable:bit:='0';

TYPE ESTADO IS (Reposo,IterandoZ,final, reposo2);
SIGNAL ACTUAL,SIGUIENTE:ESTADO;

CONSTANT x0: std_logic_vector((N-1) DOWNTO 0):= x"D44AD44A53B853B8";
CONSTANT x1: std_logic_vector((N-1) DOWNTO 0):= x"4444D44A53B8A5A5";

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)
BEGIN
    CASE ACTUAL IS
        WHEN Reposo=>
            if inicio ='1' then
                fin<='0';
                    start<='0';
                    enable<='1';
                    siguiente<=iterandoz;
            else
                fin<='0';
                start<='0';
                enable<='0';
                siguiente<=reposo;
```

```
            end if;
        WHEN IterandoZ=>
            IF contador_i=64 THEN
                fin<='0';
                start<='0';
                enable<='0';
                siguiente<=final;
            ELSE
                fin<='0';
                start<='1';
                enable<='1';
                siguiente<=IterandoZ;
            END IF;
        WHEN final=>
            fin<='1';
            start<='0';
            enable<='0';
            siguiente<=reposo2;
        when reposo2=>
            if inicio ='1' then
                fin<='0';
                start<='0';
                enable<='0';
                siguiente<=iterandoz;
            else
                fin<='0';
                start<='0';
                enable<='0';
                siguiente<=reposo2;
            end if;
    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;

--REGISTROS
PROCESS(reset,clk)
BEGIN
    IF reset='1' THEN
        z <=(others=>'0');
    elsif clk'EVENT AND CLK='1' THEN
        IF enable='1' AND start='0' then
            z <= x0;
        ELSIF enable='1' AND start='1' then
            z <=('0'& z(n-1 downto 1) + (z(n-2 downto 0)&'0') + z + x1);
        ELSE
            z<=z;
        END IF;
    END IF;
END PROCESS;
z0<=z((n/2)-1 downto 0);
END opcionA;


------------------------------------------------------------------------
--------------------------- GENERADOR A1 -------------------------------
------------------------------------------------------------------------

-- ALGORITMO A 64 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 prngA64 IS

GENERIC(N:INTEGER:=64);
PORT( z0:OUT std_logic_vector((n/2)-1 DOWNTO 0);
     INICIO: IN STD_LOGIC;
     FIN: OUT STD_LOGIC;
     CLK,RESET: IN STD_LOGIC);
END prngA64;



--ARQUITECTURA

architecture opciona64 of prngA64 is
component sumador
    Port(   a : in std_logic_vector((N/2)-1 DOWNTO 0);
        b: in std_logic_vector((N/2)-1 DOWNTO 0);
        cin : in std_logic;
        sum : out std_logic_vector((N/2)-1 DOWNTO 0);
        cout: out std_logic);
END component;

--DECLARACION DE SEÑALES
SIGNAL z: std_logic_vector((N-1) DOWNTO 0);
SIGNAL contador_i: INTEGER RANGE 0 TO 64;
SIGNAL start:bit:='0';
SIGNAL enable,enablesup,enableinf:std_logic;
signal sum1,sum2,suma:std_logic_vector((N/2)-1 DOWNTO 0);
signal carryin,carryout:std_logic:='0';
signal inferior,superior:std_logic_vector((N/2)-1 DOWNTO 0);

TYPE ESTADO IS (Reposo,IterandoZ,sumax1,sumax2,sumax3,sumax4,sumax5,sumax6, reposo2
);
SIGNAL ACTUAL,SIGUIENTE:ESTADO;
constant m:integer:=N/2;

CONSTANT x0: std_logic_vector((N-1) DOWNTO 0):= x"D44AD44A53B853B8";
CONSTANT x1: std_logic_vector((N-1) DOWNTO 0):= x"4444D44A53B8A5A5";

BEGIN

sum:sumador PORT MAP(sum1,sum2,carryin,suma,carryout);

-- 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)
BEGIN
    CASE ACTUAL IS
        WHEN Reposo=>
            if inicio ='1' then
                fin<='0';
                start<='0';
                enable<='1';
                enableinf<='0';
                enablesup<='0';
                sum1<=(others=>'0');
                sum2<=(others=>'0');
```

```
            carryin<='0';
            siguiente<=iterandoz;
        else
            fin<='0';
            start<='0';
            enable<='0';
            enableinf<='0';
            enablesup<='0';
            sum1<=(others=>'0');
            sum2<=(others=>'0');
            carryin<='0';
            siguiente<=reposo;
        end if;
    WHEN IterandoZ=>
        IF contador_i=64 THEN
            fin<='1';
            start<='1';
            enable<='0';
            enableinf<='0';
            enablesup<='0';
            sum1<=(others=>'0');
            sum2<=(others=>'0');
            carryin<='0';
            siguiente<=reposo2;
        ELSE
            fin<='0';
            start<='0';
            sum1<=z(m-1 downto 0);
            sum2<=x1(m-1 downto 0);
            carryin<='0';
            enable<='0';
            enableinf<='1';
            enablesup<='0';
            siguiente<=sumax1;
        END IF;
    WHEN sumax1=>
        fin<='0';
        start<='0';
        sum1<=z(n-1 downto m);
        sum2<=x1(n-1 downto m);
        carryin<=carryout;
        enable<='0';
        enableinf<='0';
        enablesup<='1';
        siguiente<=sumax2;
    WHEN sumax2=>
        fin<='0';
        start<='0';
        sum1<=inferior;
        sum2<=(z(m-2 downto 0)&'0');
        carryin<='0';
        enable<='0';
        enableinf<='1';
        enablesup<='0';
        siguiente<=sumax3;
    WHEN sumax3=>
        fin<='0';
        start<='0';
        enable<='0';
        enableinf<='0';
        enablesup<='1';
        sum1<=superior;
        sum2<=z(n-2 downto m-1);
        carryin<=carryout;
        siguiente<=sumax4;
    WHEN sumax4=>
        fin<='0';
        start<='0';
        enable<='0';
        enableinf<='1';
        enablesup<='0';
        sum1<=inferior;
        sum2<=z(m downto 1);
        carryin<='0';
```

```
            siguiente<=sumax5;
        WHEN sumax5=>
            fin<='0';
            start<='0';
            enable<='0';
            enableinf<='0';
            enablesup<='1';
            sum1<=superior;
            sum2<=('0'& z(n-1 downto m+1));
            carryin<=carryout;
            siguiente<=sumax6;
        WHEN sumax6=>
            enable<='1';
            enableinf<='1';
            enablesup<='1';
            sum1<=(others=>'0');
            sum2<=(others=>'0');
            carryin<='0';
            fin<='0';
            start<='1';
            carryin<='0';
            siguiente<=IterandoZ;
        when reposo2=>
            if inicio ='1' then
                fin<='0';
                start<='0';
                enable<='0';
                enableinf<='0';
                enablesup<='0';
                sum1<=(others=>'0');
                sum2<=(others=>'0');
                carryin<='0';
                siguiente<=iterandoz;
            else
                fin<='0';
                start<='0';
                enable<='0';
                enableinf<='0';
                enablesup<='0';
                sum1<=(others=>'0');
                sum2<=(others=>'0');
                carryin<='0';
                siguiente<=reposo2;
            end if;
        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
            if contador_i=64 then
                contador_i<=0;
            else
                contador_i<=contador_i+1;
            end if;
        else
            contador_i<=contador_i;
        end if;
    end if;
end process;


--REGISTROS
PROCESS(reset,clk)
BEGIN
    IF reset='1' THEN
        z <=(others=>'0');
        inferior<=(others=>'0');
        superior<=(others=>'0');
    elsif clk'EVENT AND CLK='1' THEN
```

```
        IF enable='1' AND enablesup='0'  then
            z<=x0;
        ELSIF enableinf='1' and enablesup='0' then
            inferior<=suma;
        ELSIF enablesup='1' and enableinf='0' then
            superior<=suma;
        ELSIF enable='1' and enableinf='1' and enablesup='1' then
            z<=superior & inferior;
        ELSE
            z<=z;
        END IF;
    END IF;
END PROCESS;
z0<=z((n/2)-1 downto 0);
END opcionA64;

-- SUMADOR

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 sumador is
GENERIC(N:INTEGER:=64);
    port (a : in std_logic_vector((N/2)-1 DOWNTO 0);
            b : in std_logic_vector((N/2)-1 DOWNTO 0);
            cin : in std_logic;
            sum : out std_logic_vector((N/2)-1 DOWNTO 0);
            cout : out std_logic);
end sumador;

-- implementación de comportamiendo del sumador de 32 bits
architecture behavioral of sumador is
begin
p1: process(a, b, cin)
variable vsum : std_logic_vector((N/2)-1 DOWNTO 0);
variable carry : std_logic;
begin
    carry := cin;
    for i in 0 to (N/2)-1 loop
        vsum(i) := (a(i) xor b(i)) xor carry;
        carry := (a(i) and b(i)) or (carry and (a(i) or b(i)));
    end loop;
    sum <= vsum;
    cout <= carry;
end process p1;
end behavioral;


--------------------------------------------------------------------------------
-------------------------- GENERADOR B -----------------------------------------
--------------------------------------------------------------------------------

--ALGORITMO B 64 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 prngB64 IS
GENERIC(N:INTEGER:=64);
PORT(   z0: OUT std_logic_vector(((n/2)-1) DOWNTO 0);
    INICIO: IN STD_LOGIC;
    FIN: OUT STD_LOGIC;
    CLK,RESET: IN STD_LOGIC);
END prngB64;


--ARQUITECTURA
```

```
architecture opcionb64 of prngB64 is
SIGNAL z,y0: std_logic_vector((n-1) DOWNTO 0);
SIGNAL enable,enableY:bit:='0';

--DECLARACION DE SEÑALES
TYPE ESTADO IS (reposo,IterandoZ,IterandoY, reposo2);
SIGNAL ACTUAL,SIGUIENTE:ESTADO;

CONSTANT x0: std_logic_vector((N-1) DOWNTO 0):= x"2422424D2434D243";
CONSTANT x1: std_logic_vector((N-1) DOWNTO 0):= x"2422424D24342424";
SIGNAL start:bit:='0';
SIGNAL contador: INTEGER RANGE 0 TO 24;

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,contador,Inicio)
variable z,y0: std_logic_vector((n-1) DOWNTO 0);
BEGIN

CASE ACTUAL IS
    WHEN reposo=>
            if Inicio='1' then
                start<='0';
                enable<='1';
                enabley<='0';
                fin<='0';
                SIGUIENTE<=IterandoZ;
            else
                fin<='0';
                start<='0';
                enable<='0';
                enabley<='0';
                SIGUIENTE<=reposo;
            End if;
    WHEN IterandoZ=>
            IF contador=24 THEN
                start<='0';
                enable<='0';
                enabley<='1';
                fin<='0';
                SIGUIENTE<=IterandoY;
            ELSE
                start<='1';
                enable<='1';
                enabley<='0';
                fin<='0';
                SIGUIENTE<=IterandoZ;
            END IF;
    WHEN IterandoY=>
            IF contador=24 THEN
                start<='0';
                enable<='0';
                enabley<='0';
                fin<='1';
                SIGUIENTE<=reposo2;
            ELSE
                start<='1';
                enable<='0';
                enabley<='1';
                fin<='0';
                SIGUIENTE<=IterandoY;
```

```
            END IF;
    WHEN reposo2=>
            if Inicio='1' then
                start<='0';
                enable<='0';
                enabley<='0';
                fin<='0';
                SIGUIENTE<=IterandoZ;
            else
                fin<='0';
                start<='0';
                enable<='0';
                enabley<='0';
                SIGUIENTE<=reposo2;
            End if;
END CASE;
END PROCESS;


--CONTADOR----------------------------
PROCESS(reset,clk)
BEGIN
IF reset='1' THEN
    contador<= 0;
elsif clk'EVENT AND CLK='1' THEN
    if start='1' then
        contador<=contador+1;
    else
        contador<=0;
    end if;
end if;
End Process;

----------REGISTROS----------
PROCESS (reset,clk)
variable a:std_logic_vector(n-1 DOWNTO 0);
Begin
IF reset='1' THEN
    z<=(OTHERS=>'0');
    y0<=(OTHERS=>'0');
elsif clk'EVENT AND CLK='1' THEN
    IF enable='1' AND start = '0' then
        z<=x0 XOR x1;
    ELSIF enable='1' AND start = '1' THEN
        a:=z + (x"2424422A34244228");
        z<=(z(n-2 downto 0)&'0') + ('0' & a(n-2 downto 0));
    ELSIF enableY='1'AND start='0' then
        y0<= x1 XOR z;
    ELSIF enableY='1'AND start='1'  then
        y0<=('0'& y0(n-1 downto 1)) + (y0(n-2 downto 0)&'0') + y0 + x
"4424422244244223";
    ELSE
        z<= z XOR y0;
        y0<=(others=>'0');
    END IF;
END IF;
End process;
    z0 <= z(((n/2)-1) downto 0);
End opcionb64;



--------------------------------------------------------------------------------
-------------------------- GENERADOR B1 ---------------------------------
--------------------------------------------------------------------------------

--ALGORITMO B1 64 BITS

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
--ENTIDAD

ENTITY prngB64 IS
```

```
GENERIC(N:INTEGER:=64);
PORT(   z0: OUT std_logic_vector(((n/2)-1) DOWNTO 0);
    INICIO: IN STD_LOGIC;
    FIN: OUT STD_LOGIC;
    CLK,RESET: IN STD_LOGIC);
END prngB64;

--ARQUITECTURA
architecture opcionb64 of prngB64 is

component sumador
Port(   a : in std_logic_vector((N/2)-1 DOWNTO 0);
    b: in std_logic_vector((N/2)-1 DOWNTO 0);
    cin : in std_logic;
    sum : out std_logic_vector((N/2)-1 DOWNTO 0);
    cout: out std_logic);
END component;

SIGNAL z,y0: std_logic_vector((n-1) DOWNTO 0);
SIGNAL enable,enablesup,enableinf,enabley:std_logic;
signal carryin,carryout:std_logic:='0';
signal inferior,superior:std_logic_vector((N/2)-1 DOWNTO 0);

--DECLARACION DE SEÑALES
TYPE ESTADO IS (reposo,IterandoZ,IterandoY,suma1,suma11,suma12,suma22,sumaY1,sumaY2
,sumaY3,sumaY4,sumaY5,sumaY6, reposo2);
SIGNAL ACTUAL,SIGUIENTE:ESTADO;

CONSTANT x0: std_logic_vector((N-1) DOWNTO 0):= x"2422424D2434D243";
CONSTANT x1: std_logic_vector((N-1) DOWNTO 0):= x"2422424D24342424";
SIGNAL start:bit:='0';
SIGNAL contador: INTEGER RANGE 0 TO 24;

signal sum1,sum2,suma:std_logic_vector((N/2)-1 DOWNTO 0);
BEGIN
sum:sumador PORT MAP(sum1,sum2,carryin,suma,carryout);

-- 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,contador,Inicio)
BEGIN

CASE ACTUAL IS
    WHEN reposo=>
            if Inicio='1' then
                start<='0';
                sum1<=(others=>'0');
                sum2<=(others=>'0');
                enable<='1';
                enableinf<='0';
                enabley<='0';
                enablesup<='0';
                carryin<='0';
                fin<='0';
                SIGUIENTE<=IterandoZ;
            else
                sum1<=(others=>'0');
                sum2<=(others=>'0');
                fin<='0';
                carryin<='0';
                start<='0';
                enable<='0';
                enabley<='0';
                enableinf<='0';
```

```
                enablesup<='0';
                SIGUIENTE<=reposo;
            End if;
WHEN IterandoZ=>
            IF contador=24 THEN
                start<='1';
                fin<='0';
                sum1<=(others=>'0');
                sum2<=(others=>'0');
                carryin<='0';
                enable<='0';
                enabley<='1';
                enableinf<='0';
                enablesup<='0';
                SIGUIENTE<=IterandoY;
            ELSE
                sum1<=x"34244228";
                sum2<=z((n/2)-1 downto 0);
                carryin<='0';
                start<='0';
                enable<='0';
                enabley<='0';
                enableinf<='1';
                enablesup<='0';
                fin<='0';
                SIGUIENTE<=suma1;
            END IF;
WHEN suma1=>
            start<='0';
            sum1<=x"2424422A";
            sum2<=z(n-1 downto (n/2));
            carryin<=carryout;
            enable<='0';
            enabley<='0';
            enableinf<='0';
            enablesup<='1';
            fin<='0';
            SIGUIENTE<=suma11;
WHEN suma11=>
            start<='0';
            sum1<=(z((n/2)-2 downto 0)&'0');
            sum2<=superior(0)&inferior((n/2)-1 downto 1);
            carryin<='0';
            enable<='0';
            enabley<='0';
            enableinf<='1';
            enablesup<='0';
            fin<='0';
            SIGUIENTE<=suma12;
WHEN suma12=>
            start<='0';
            sum1<=z(n-2 downto (n/2)-1);
            sum2<=('0' & superior((n/2)-1 downto 1));
            enable<='0';
            enabley<='0';
            enableinf<='0';
            enablesup<='1';
            carryin<=carryout;
            fin<='0';
            SIGUIENTE<=suma22;
WHEN suma22=>
            start<='1';
            sum1<=(others=>'0');
            sum2<=(others=>'0');
            carryin<='0';
            enable<='1';
            enabley<='0';
            enableinf<='1';
            enablesup<='1';
            fin<='0';
            SIGUIENTE<=IterandoZ;
WHEN IterandoY=>
            IF contador=24 THEN
                sum1<=(others=>'0');
```

```
                sum2<=(others=>'0');
                carryin<='0';
                start<='0';
                enable<='1';
                enabley<='1';
                enableinf<='1';
                enablesup<='1';
                fin<='1';
                SIGUIENTE<=reposo2;
            ELSE
                start<='0';
                sum1<=y0((n/2)-1 downto 0);
                sum2<=x"44244223";
                carryin<='0';
                enable<='0';
                enabley<='0';
                enableinf<='1';
                enablesup<='0';
                fin<='0';
                SIGUIENTE<=sumaY1;
            END IF;
WHEN sumaY1=>
            start<='0';
            sum1<=y0(n-1 downto (n/2));
            sum2<=x"44244222";
            carryin<=carryout;
            enable<='0';
            enabley<='0';
            enableinf<='0';
            enablesup<='1';
            fin<='0';
            SIGUIENTE<=sumaY2;
WHEN sumaY2=>
            enable<='0';
            enabley<='0';
            enableinf<='1';
            enablesup<='0';
            sum1<=inferior;
            sum2<=(y0((n/2)-2 downto 0)&'0');
            carryin<='0';
            start<='0';
            fin<='0';
            SIGUIENTE<=sumaY3;
WHEN sumaY3=>
            enable<='0';
            enabley<='0';
            enableinf<='0';
            enablesup<='1';
            sum1<=superior;
            sum2<=y0(n-2 downto (n/2)-1);
            carryin<=carryout;
            start<='0';
            fin<='0';
            SIGUIENTE<=SumaY4;
WHEN sumaY4=>
            enable<='0';
            enabley<='0';
            enableinf<='1';
            enablesup<='0';
            sum1<=inferior;
            sum2<=y0((n/2) downto 1);
            carryin<='0';
            start<='0';
            fin<='0';
            SIGUIENTE<=sumaY5;
WHEN sumaY5=>
            enable<='0';
            enabley<='0';
            enableinf<='0';
            enablesup<='1';
            sum1<=superior;
            sum2<=('0'& y0(n-1 downto (n/2)+1));
            carryin<=carryout;
            start<='0';
```

```
            fin<='0';
            SIGUIENTE<=sumaY6;
WHEN sumaY6=>
            sum1<=(others=>'0');
            sum2<=(others=>'0');
            carryin<='0';
            enable<='0';
            enabley<='1';
            enableinf<='1';
            enablesup<='1';
            start<='1';
            fin<='0';
            SIGUIENTE<=IterandoY;
WHEN reposo2=>
            if Inicio='1' then
                start<='0';
                sum1<=(others=>'0');
                sum2<=(others=>'0');
                enable<='0';
                enableinf<='0';
                enabley<='0';
                enablesup<='0';
                carryin<='0';
                fin<='0';
                SIGUIENTE<=IterandoZ;
            else
                sum1<=(others=>'0');
                sum2<=(others=>'0');
                fin<='0';
                carryin<='0';
                start<='0';
                enable<='0';
                enabley<='0';
                enableinf<='0';
                enablesup<='0';
                SIGUIENTE<=reposo2;
            End if;
END CASE;
END PROCESS;


--CONTADOR----------------------------

PROCESS(reset,clk)
BEGIN
IF reset='1' THEN
    contador<= 0;
elsif clk'EVENT AND CLK='1' THEN
    if start='1' then
        if contador=24 then
            contador<=0;
        else
            contador<=contador+1;
        end if;
    else
        contador<=contador;
    end if;
        end if;
End Process;

-----------REGISTROS----------

PROCESS (reset,clk)
Begin
IF reset='1' THEN
    z<=(OTHERS=>'0');
    y0<=(OTHERS=>'0');
elsif clk'EVENT AND CLK='1' THEN
    IF enable='1' AND enablesup='0'  then
        z<=x0 XOR x1;
    ELSIF enabley='1' AND enablesup='0' then
        y0<= x1 XOR z;
    ELSIF enableinf='1' and enablesup='0' then
        inferior<=suma;
```

```
    ELSIF enablesup='1' and enableinf='0' then
        superior<=suma;
    ELSIF enable='1' and enableinf='1' and enablesup='1' and enabley='0' then
        z<=superior & inferior;
    ELSIF enable='0' and enableinf='1' and enablesup='1' and enabley='1' then
        y0<=superior & inferior;
    ELSIF enable='1' and enableinf='1' and enablesup='1' and enabley='1' then
        z<= z XOR y0;
    ELSE
        z<= z;
        y0<=y0;
        inferior<=inferior;
        superior<=superior;
    END IF;
END IF;
End process;
z0 <= z(((n/2)-1) downto 0);
End opcionb64;



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

entity sumador is
GENERIC(N:INTEGER:=64);
port (a : in std_logic_vector((N/2)-1 DOWNTO 0);
        b : in std_logic_vector((N/2)-1 DOWNTO 0);
        cin : in std_logic;
        sum : out std_logic_vector((N/2)-1 DOWNTO 0);
        cout : out std_logic);
end sumador;
-- implementación de comportamiendo del sumador de 4 bits
architecture behavioral of sumador is
begin
p1: process(a, b, cin)
variable vsum : std_logic_vector((N/2)-1 DOWNTO 0);
variable carry : std_logic;
begin
    carry := cin;
    for i in 0 to (N/2)-1 loop
        vsum(i) := (a(i) xor b(i)) xor carry;
        carry := (a(i) and b(i)) or (carry and (a(i) or b(i)));
    end loop;
sum <= vsum;
cout <= carry;
end process p1;
end behavioral;



--------------------------------------------------------------------------------
-------------------------- GENERADOR B2 --------------------------------------
--------------------------------------------------------------------------------

--ALGORITMO B 64 BITS

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

--ENTIDAD
ENTITY prngB64 IS
GENERIC(N:INTEGER:=64);
PORT(   z0: OUT std_logic_vector(((n/2)-1) DOWNTO 0);
    INICIO: IN STD_LOGIC;
    FIN: OUT STD_LOGIC;
    CLK,RESET: IN STD_LOGIC);
END prngB64;

--ARQUITECTURA
architecture opcionb64 of prngB64 is
```

```
component sumador
Port(   a : in std_logic_vector((N/4)-1 DOWNTO 0);
    b: in std_logic_vector((N/4)-1 DOWNTO 0);
    cin : in std_logic;
    sum : out std_logic_vector((N/4)-1 DOWNTO 0);
    cout: out std_logic);
END component;
SIGNAL z,y0: std_logic_vector((n-1) DOWNTO 0);
SIGNAL enable,enablesup,enableinf,enabley,enableinf1,enablesup1:std_logic;
signal carryin,carryout:std_logic:='0';
signal inferior,superior,inferior1,superior1:std_logic_vector((N/4)-1 DOWNTO 0);

--DECLARACION DE SEÑALES
TYPE ESTADO IS (reposo,IterandoZ,IterandoY,sumaX1,sumaX2,sumaX3,sumaX4,sumaX5,
sumaX6,sumaX7,sumaX8,sumaY1,sumaY2,sumaY3,sumaY4,sumaY5,sumaY6,sumaY7,sumaY8,sumaY9
,sumaY10,sumaY11,sumaY12, reposo2);
SIGNAL ACTUAL,SIGUIENTE:ESTADO;

CONSTANT x0: std_logic_vector((N-1) DOWNTO 0):= x"D224245A5A45A3F4";
CONSTANT x1: std_logic_vector((N-1) DOWNTO 0):= x"2424245A5A245A5A";
SIGNAL start:bit:='0';
SIGNAL contador: INTEGER RANGE 0 TO 24;

signal sum1,sum2,suma:std_logic_vector((N/4)-1 DOWNTO 0);
BEGIN
sum:sumador PORT MAP(sum1,sum2,carryin,suma,carryout);

-- 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,contador,Inicio)
BEGIN

CASE ACTUAL IS
WHEN reposo=>
            if Inicio='1' then
                start<='0';
                sum1<=(others=>'0');
                sum2<=(others=>'0');
                enable<='1';
                enableinf<='0';
                enabley<='0';
                enablesup<='0';
                enableinf1<='0';
                enablesup1<='0';
                carryin<='0';
                fin<='0';
                SIGUIENTE<=IterandoZ;
            else
                fin<='0';
                start<='0';
                enable<='0';
                enabley<='0';
                enableinf<='0';
                enablesup<='0';
                enableinf1<='0';
                enablesup1<='0';
                sum1<=(others=>'0');
                sum2<=(others=>'0');
                carryin<='0';
                SIGUIENTE<=reposo;
            End if;
WHEN IterandoZ=>
            IF contador=24 THEN
                start<='1';
                fin<='0';
```

```
                sum1<=(others=>'0');
                sum2<=(others=>'0');
                carryin<='0';
                enable<='0';
                enabley<='1';
                enableinf<='0';
                enablesup<='0';
                enableinf1<='0';
                enablesup1<='0';
                SIGUIENTE<=IterandoY;
            ELSE
                sum1<=x"C439";
                sum2<=z((n/4)-1 downto 0);
                carryin<='0';
                start<='0';
                enable<='0';
                enabley<='0';
                enableinf<='1';
                enablesup<='0';
                enableinf1<='0';
                enablesup1<='0';
                fin<='0';
                SIGUIENTE<=sumaX1;
            END IF;
WHEN sumaX1=>
            start<='0';
            sum1<=x"34B2";
            sum2<=z((n/2)-1 downto (n/4));
            carryin<=carryout;
            enable<='0';
            enabley<='0';
            enableinf<='0';
            enablesup<='0';
            enableinf1<='1';
            enablesup1<='0';
            fin<='0';
            SIGUIENTE<=sumaX2;
WHEN sumaX2=>
            start<='0';
            sum1<=z((n/2)+(n/4)-1 downto (n/2));
            sum2<=x"AFF0";
            carryin<=carryout;
            enable<='0';
            enabley<='0';
            enableinf<='0';
            enablesup<='1';
            enableinf1<='0';
            enablesup1<='0';
            fin<='0';
            SIGUIENTE<=sumaX3;
WHEN sumaX3=>
            start<='0';
            sum1<=z(n-1 downto (n-(n/4)));
            sum2<=x"F451";
            enable<='0';
            enabley<='0';
            enableinf<='0';
            enablesup<='0';
            enableinf1<='0';
            enablesup1<='1';
            carryin<=carryout;
            fin<='0';
            SIGUIENTE<=sumaX4;
WHEN sumaX4=>
            start<='0';
            sum1<=inferior1(0)&inferior((N/4)-1 DOWNTO 1);
            sum2<=z((n/4)-2 downto 0)&'0';
            enable<='0';
            enabley<='0';
            enableinf<='1';
            enablesup<='0';
            enableinf1<='0';
            enablesup1<='0';
            carryin<='0';
```

```
            fin<='0';
            SIGUIENTE<=sumaX5;
WHEN sumaX5=>
            start<='0';
            sum1<=superior(0)&inferior1((N/4)-1 DOWNTO 1);
            sum2<=z((n/2)-2 downto (n/4)-1);
            enable<='0';
            enabley<='0';
            enableinf<='0';
            enablesup<='0';
            enableinf1<='1';
            enablesup1<='0';
            carryin<=carryout;
            fin<='0';
            SIGUIENTE<=sumaX6;
WHEN sumaX6=>
            start<='0';
            sum1<=superior1(0)& superior((N/4)-1 DOWNTO 1);
            sum2<=z(((n/2)+(n/4)-2) downto (n/2)-1);
            enable<='0';
            enabley<='0';
            enableinf<='0';
            enablesup<='1';
            enableinf1<='0';
            enablesup1<='0';
            carryin<=carryout;
            fin<='0';
            SIGUIENTE<=sumaX7;
WHEN sumaX7=>
            start<='0';
            sum1<='0'& superior1((N/4)-1 DOWNTO 1);
            sum2<=z(n-2 downto (n/2)+(n/4)-1);
            enable<='0';
            enabley<='0';
            enableinf<='0';
            enablesup<='0';
            enableinf1<='0';
            enablesup1<='1';
            carryin<=carryout;
            fin<='0';
            SIGUIENTE<=sumaX8;
WHEN sumaX8=>
            start<='1';
            sum1<=(others=>'0');
            sum2<=(others=>'0');
            carryin<='0';
            enable<='1';
            enabley<='0';
            enableinf<='1';
            enablesup<='1';
            enableinf1<='0';
            enablesup1<='0';
            fin<='0';
            SIGUIENTE<=IterandoZ;
WHEN IterandoY=>
            IF contador=24 THEN
                sum1<=(others=>'0');
                sum2<=(others=>'0');
                carryin<='0';
                start<='0';
                enable<='1';
                enabley<='1';
                enableinf<='1';
                enablesup<='1';
                enableinf1<='0';
                enablesup1<='0';
                fin<='1';
                SIGUIENTE<=reposo2;
            ELSE
                start<='0';
                sum1<=y0((n/4)-1 downto 0);
                sum2<=x"4D2F";
                carryin<='0';
                enable<='0';
```

```
                enableY<='0';
                enableinf<='1';
                enablesup<='0';
                enableinf1<='0';
                enablesup1<='0';
                fin<='0';
                SIGUIENTE<=sumaY1;
            END IF;
WHEN sumaY1=>
            start<='0';
            sum1<=y0((n/2)-1 downto (n/4));
            sum2<=x"65FF";
            carryin<=carryout;
            enable<='0';
            enableY<='0';
            enableinf<='0';
            enablesup<='0';
            enableinf1<='1';
            enablesup1<='0';
            fin<='0';
            SIGUIENTE<=sumaY2;
WHEN sumaY2=>
            enable<='0';
            enableY<='0';
            enableinf<='0';
            enablesup<='1';
            enableinf1<='0';
            enablesup1<='0';
            sum1<=x"454A";
            sum2<=y0((n/2)+(n/4)-1 downto (n/2));
            carryin<=carryout;
            start<='0';
            fin<='0';
            SIGUIENTE<=sumaY3;
WHEN sumaY3=>
            enable<='0';
            enableY<='0';
            enableinf<='0';
            enablesup<='0';
            enableinf1<='0';
            enablesup1<='1';
            sum1<=x"A0D4";
            sum2<=y0(n-1 downto (n-(n/4)));
            carryin<=carryout;
            start<='0';
            fin<='0';
            SIGUIENTE<=SumaY4;
WHEN sumaY4=>
            enable<='0';
            enableY<='0';
            enableinf<='1';
            enablesup<='0';
            enableinf1<='0';
            enablesup1<='0';
            sum1<=inferior;
            sum2<=y0((n/4)-2 downto 0)&'0';
            carryin<='0';
            start<='0';
            fin<='0';
            SIGUIENTE<=sumaY5;
WHEN sumaY5=>
            enable<='0';
            enableY<='0';
            enableinf<='0';
            enablesup<='0';
            enableinf1<='1';
            enablesup1<='0';
            sum1<=inferior1;
            sum2<=y0((n/2)-2 downto (n/4)-1);
            carryin<=carryout;
            start<='0';
            fin<='0';
            SIGUIENTE<=sumaY6;
WHEN sumaY6=>
```

```
            enable<='0';
            enabley<='0';
            enableinf<='0';
            enablesup<='1';
            enableinf1<='0';
            enablesup1<='0';
            sum1<=superior;
            sum2<=y0(((n/2)+(n/4)-2) downto (n/2)-1);
            carryin<=carryout;
            start<='0';
            fin<='0';
            SIGUIENTE<=sumaY7;
WHEN sumaY7=>
            enable<='0';
            enabley<='0';
            enableinf<='0';
            enablesup<='0';
            enableinf1<='0';
            enablesup1<='1';
            sum1<=superior1;
            sum2<=y0(n-2 downto (n/2)+(n/4)-1);
            carryin<=carryout;
            start<='0';
            fin<='0';
            SIGUIENTE<=sumaY8;
WHEN sumaY8=>
            enable<='0';
            enabley<='0';
            enableinf<='1';
            enablesup<='0';
            enableinf1<='0';
            enablesup1<='0';
            sum1<=inferior;
            sum2<=y0((n/4) downto 1);
            carryin<='0';
            start<='0';
            fin<='0';
            SIGUIENTE<=sumaY9;
WHEN sumaY9=>
            enable<='0';
            enabley<='0';
            enableinf<='0';
            enablesup<='0';
            enableinf1<='1';
            enablesup1<='0';
            sum1<=inferior1;
            sum2<=y0((n/2) downto (n/4)+1);
            carryin<=carryout;
            start<='0';
            fin<='0';
            SIGUIENTE<=sumaY10;
WHEN sumaY10=>
            enable<='0';
            enabley<='0';
            enableinf<='0';
            enablesup<='1';
            enableinf1<='0';
            enablesup1<='0';
            sum1<=superior;
            sum2<=y0(((n/2)+(n/4)) downto (n/2)+1);
            carryin<=carryout;
            start<='0';
            fin<='0';
            SIGUIENTE<=sumaY11;
WHEN sumaY11=>
            enable<='0';
            enabley<='0';
            enableinf<='0';
            enablesup<='0';
            enableinf1<='0';
            enablesup1<='1';
            sum1<=superior1;
            sum2<='0'& y0(n-1 downto (n+1)-(n/4));
            carryin<=carryout;
```

```
            start<='0';
            fin<='0';
            SIGUIENTE<=sumaY12;
WHEN sumaY12=>
            sum1<=(others=>'0');
            sum2<=(others=>'0');
            carryin<='0';
            enable<='0';
            enabley<='1';
            enableinf<='1';
            enablesup<='1';
            enableinf1<='0';
            enablesup1<='0';
            start<='1';
            fin<='0';
            SIGUIENTE<=IterandoY;
WHEN reposo2=>
            if Inicio='1' then
                start<='0';
                sum1<=(others=>'0');
                sum2<=(others=>'0');
                enable<='0';
                enableinf<='0';
                enabley<='0';
                enablesup<='0';
                enableinf1<='0';
                enablesup1<='0';
                carryin<='0';
                fin<='0';
                SIGUIENTE<=IterandoZ;
            else
                fin<='0';
                start<='0';
                enable<='0';
                enabley<='0';
                enableinf<='0';
                enablesup<='0';
                enableinf1<='0';
                enablesup1<='0';
                sum1<=(others=>'0');
                sum2<=(others=>'0');
                carryin<='0';
                SIGUIENTE<=reposo2;
            End if;
END CASE;
END PROCESS;


--CONTADOR-------------------------
PROCESS(reset,clk)
BEGIN
IF reset='1' THEN
    contador<= 0;
elsif clk'EVENT AND CLK='1' THEN
    if start='1' then
        if contador=24 then
            contador<=0;
        else
            contador<=contador+1;
        end if;
    else
        contador<=contador;
    end if;
end if;
End Process;

-----------REGISTROS----------
PROCESS (reset,clk)
Begin
IF reset='1' THEN
z<=(OTHERS=>'0');
y0<=(OTHERS=>'0');
elsif clk'EVENT AND CLK='1' THEN
    IF enable='1' AND enablesup='0'  then
```

```
        z<=x0 XOR x1;
    ELSIF enabley='1' AND enablesup='0' then
        y0<= x1 XOR z;
    ELSIF enableinf='1' and enablesup='0' then
        inferior<=suma;
    ELSIF enablesup='1' and enableinf='0' then
        superior<=suma;
    ELSIF enableinf1='1' and enablesup='0' then
        inferior1<=suma;
    ELSIF enablesup1='1' and enableinf='0' then
        superior1<=suma;
    ELSIF enable='1' and enableinf='1' and enablesup='1' and enabley='0' then
        z<= superior1 & superior & inferior1 & inferior;
    ELSIF enable='0' and enableinf='1' and enablesup='1' and enabley='1' then
        y0<=superior1 &superior & inferior1 & inferior;
    ELSIF enable='1' and enableinf='1' and enablesup='1' and enabley='1' then
        z<= z XOR y0;
    ELSE
        z<= z;
        y0<=y0;
        inferior<=inferior;
        superior<=superior;
    END IF;
END IF;
End process;
z0 <= z(((n/2)-1) downto 0);
End opcionb64;



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

entity sumador is
GENERIC(N:INTEGER:=64);
port (a : in std_logic_vector((N/4)-1 DOWNTO 0);
        b : in std_logic_vector((N/4)-1 DOWNTO 0);
        cin : in std_logic;
        sum : out std_logic_vector((N/4)-1 DOWNTO 0);
        cout : out std_logic);
end sumador;
-- implementación de comportamiendo del sumador de 4 bits
architecture behavioral of sumador is
begin
p1: process(a, b, cin)
variable vsum : std_logic_vector((N/4)-1 DOWNTO 0);
variable carry : std_logic;
begin
    carry := cin;
    for i in 0 to (N/4)-1 loop
        vsum(i) := (a(i) xor b(i)) xor carry;
        carry := (a(i) and b(i)) or (carry and (a(i) or b(i)));
    end loop;
sum <= vsum;
cout <= carry;
end process p1;
end behavioral;


--------------------------------------------------------------------------------
--------------------------- GENERADOR C ----------------------------------------
--------------------------------------------------------------------------------


--ALGORITMO C 64 BITS
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;

--ENTIDAD
ENTITY prngC64 IS
```

```
GENERIC(N:INTEGER:=64);
PORT(   z0: OUT std_logic_vector(((n/2)-1) DOWNTO 0);
    INICIO: IN STD_LOGIC;
    FIN: OUT STD_LOGIC;
    CLK,RESET: IN STD_LOGIC);
END prngC64;

--ARQUITECTURA
architecture opcionc64 of prngC64 is
SIGNAL z,y0: std_logic_vector((n-1) DOWNTO 0);
SIGNAL enable,enableY:bit:='0';

--DECLARACION DE SEÑALES
TYPE ESTADO IS (reposo,IterandoZ,IterandoY,estableceryY, reposo2);
SIGNAL ACTUAL,SIGUIENTE:ESTADO;

CONSTANT x0: std_logic_vector((N-1) DOWNTO 0):= x"D3A8D3F2D5D3F2F2";
CONSTANT x1: std_logic_vector((N-1) DOWNTO 0):= x"2D3F2A8D3F2D5AE4";
SIGNAL start:bit:='0';
SIGNAL contador: INTEGER RANGE 0 TO 32;

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,contador,Inicio)
variable z,y0: std_logic_vector((n-1) DOWNTO 0);
BEGIN

CASE ACTUAL IS
WHEN reposo=>
        if Inicio='1' then
            start<='0';
            enable<='1';
            enabley<='0';
            fin<='0';
            SIGUIENTE<=IterandoZ;
        else
            fin<='0';
            start<='0';
            enable<='0';
            enabley<='0';
            SIGUIENTE<=reposo;
        End if;
WHEN IterandoZ=>
        IF contador=32 THEN
            start<='0';
            enable<='1';
            enabley<='1';
            fin<='0';
            SIGUIENTE<=estableceryY;
        ELSE
            start<='1';
            enable<='1';
            enabley<='0';
            fin<='0';
            SIGUIENTE<=IterandoZ;
        END IF;
WHEN estableceryY=>
            start<='1';
            enable<='0';
            enabley<='1';
            fin<='0';
```

```
            SIGUIENTE<=IterandoY;
WHEN IterandoY=>
            start<='1';
            enable<='1';
            enableY<='1';
            fin<='1';
            SIGUIENTE<=reposo2;
WHEN reposo2=>
            if Inicio='1' then
                start<='0';
                enable<='0';
                enableY<='0';
                fin<='0';
                SIGUIENTE<=IterandoZ;
            else
                fin<='0';
                start<='0';
                enable<='0';
                enableY<='0';
                SIGUIENTE<=reposo2;
            End if;
END CASE;
END PROCESS;


--CONTADOR-------------------------------------
PROCESS(reset,clk)
BEGIN
IF reset='1' THEN
    contador<= 0;
elsif clk'EVENT AND CLK='1' THEN
    if start='1' then
        contador<=contador+1;
    else
        contador<=0;
    end if;
end if;
End Process;

-----------REGISTROS----------
PROCESS (reset,clk)
variable a:std_logic_vector(n-1 DOWNTO 0);
Begin
IF reset='1' THEN
    z<=(OTHERS=>'0');
    y0<=(OTHERS=>'0');
elsif clk'EVENT AND CLK='1' THEN
    IF enable='1' AND enableY = '0' AND start = '0' then
        z<=x0;
    ELSIF enable='1' AND enableY ='0' AND start = '1' THEN
        a:=z + x"A8DA8D3F2D53F2D5";
        z<=(z(n-2 downto 0)&'0') + ('0' & a(n-1 downto 1));
    ELSIF enableY='1'AND enable='1' AND start='0' then
        y0<=z;
    ELSIF enableY='1'AND enable = '0' AND start='1'  then
        y0<=('0'& y0(n-1 downto 1)) + (y0(n-2 downto 0)&'0') + y0 + x1;
    ELSIF enable = '1' AND enableY ='1' AND start = '1' THEN
        z<= z XOR y0;
    ELSE
        z<=z;
        y0<=y0;
    END IF;
END IF;
End process;
z0 <= z(((n/2)-1) downto 0);
End opcionc64;
```

ANEXO C – RESULTADOS DE LA SÍNTESIS CON "DESIGN COMPILER"

Resultados de la síntesis del protocolo 1 con prng A			
	32 bits	**64 bits**	**96 bits**
Número de puertos	68	68	68
Número de nodos	903	1199	1506
Número de celdas	728	895	1074
Número de referencias	39	34	34
Área combinacional (μm^2)	7398,176	10636,575	13973,226
Área no combinacional (μm^2)	4507,525	5486,270	6518,462
Área de interconexión (μm^2)	545,928	811,400	967,350
Área total de celdas (μm^2)	11905,701	16122,845	20491,688
Área total (μm^2)	12451,629	16934,245	21459,038
Puertas equivalentes	2153	2916	3706
Potencia interna	179,122 nW (69%)	204,566 nW (71%)	271,405 nW (71%)
Potencia de conmutación	80,924 nW (31%)	82,364 nW (29%)	112,431 nW (29%)
Potencia dinámica total	260,046 nW	286,930 nW	383,835 nW
Pérdidas de energías	42,519 uW	56,586 uW	72,053 uW

Resultados de la síntesis del protocolo 1 con prng A1			
	32 bits	**64 bits**	**96 bits**
Número de puertos	68	68	68
Número de nodos	921	1207	1540
Número de celdas	869	1155	1488
Número de referencias	40	36	38
Área combinacional (μm^2)	6903,905	9351,671	12119,666
Área no combinacional (μm^2)	5550,782	7615,166	9679,550
Área de interconexión (μm^2)	714,290	1162,264	1490,606
Área total de celdas (μm^2)	12454,687	16966,837	21799,216
Área total (μm^2)	13168,976	18129,101	23289,822
Puertas equivalentes	2252	3068	3942
Potencia interna	160,522 nW (65%)	229.6921 nW (73%)	308.6813 nW (72%)
Potencia de conmutación	85,971 nW (35%)	85.6589 nW (27%)	120.8809 nW (28%)
Potencia dinámica total	246,493 nW	315.3510 nW	429.5622 nW
Pérdidas de energías	45,973 uW	62.5621 uW	80.3323 uW

Resultados de la síntesis del protocolo 1 con prng B			
	32 bits	**64 bits**	**96 bits**
Número de puertos	68	68	68
Número de nodos	1231	1873	2555
Número de celdas	959	1366	1805
Número de referencias	43	37	36
Área combinacional (μm^2)	10443,642	16893,697	23560,437
Área no combinacional (μm^2)	5451,246	7486,142	9550,526
Área de interconexión (μm^2)	729,286	1229,735	1614,556
Área total de celdas (μm^2)	15894,888	24379,839	33110,963
Área total (μm^2)	16624,174	25609,574	34725,519
Puertas equivalentes	2875	4409	5988
Potencia interna	167,079 nW (68%)	212,167 nW (75%)	270,003 nW (75%)
Potencia de conmutación	79,050 nW (32%)	71,167 nW (25%)	88,550 nW (25%)
Potencia dinámica total	246,130 nW	283,334 nW	358,552 nW
Pérdidas de energías	55,806 uW	84,820 uW	114,178 uW

Resultados de la síntesis del protocolo 1 con prng B1			
	32 bits	**64 bits**	**96 bits**
Número de puertos	68	68	68
Número de nodos	1290	1946	2693
Número de celdas	1147	1710	2366
Número de referencias	48	41	42
Área combinacional (μm^2)	9299,763	13972,441	19308,135
Área no combinacional (μm^2)	6284,369	9143,166	12003,806
Área de interconexión (μm^2)	930,316	1679,224	2300,370
Área total de celdas (μm^2)	15584,132	23115,607	31311,941
Área total (μm^2)	16514,449	24794,831	33612,312
Puertas equivalentes	2818	4180	5663
Potencia interna	186,9797 nW (65%)	274,289 nW (73%)	388,211 nW (71%)
Potencia de conmutación	100,385 nW (35%)	100,094 nW (27%)	158,768 nW (29%)
Potencia dinámica total	287,364 nW	374,383 nW	546,978 nW
Pérdidas de energías	55,743 uW	82,833 uW	112,776 uW

Resultados de la síntesis del protocolo 1 con prng B2			
	32 bits	**64 bits**	**96 bits**
Número de puertos	68	68	68
Número de nodos	1269	1856	2529
Número de celdas	1132	1627	2223
Número de referencias	56	47	44
Área combinacional (μm^2)	9172,712	13349,219	18226,362
Área no combinacional (μm^2)	6320,311	9175,422	12036,062
Área de interconexión (μm^2)	930,195	1691,417	2246,665
Área total de celdas (μm^2)	15493,023	22524,641	30262,424
Área total (μm^2)	16423,218	24216,058	32509,089
Puertas equivalentes	2802	4073	5473
Potencia interna	176,488 nW (67%)	243,877 nW (75%)	338,232 nW (74%)
Potencia de conmutación	85,598 nW (33%)	80,462 nW (25%)	121,351 nW (26%)
Potencia dinámica total	262,085 nW	324,340 nW	459,583 nW
Pérdidas de energías	55,706 uW	79,054 uW	110,277 uW

Resultados de la síntesis del protocolo 1 con prng C			
	32 bits	**64 bits**	**96 bits**
Número de puertos	68	68	68
Número de nodos	1265	1946	2629
Número de celdas	970	1395	1822
Número de referencias	37	36	37
Área combinacional (μm^2)	10514,930	17109,619	23736,093
Área no combinacional (μm^2)	5486,270	7550,654	9615,038
Área de interconexión (μm^2)	894,880	1260,805	1648,770
Área total de celdas (μm^2)	16001,200	24660,272	33351,130
Área total (μm^2)	16896,081	25921,078	35000,000
Puertas equivalentes	2894	4460	6031
Potencia interna	146,491 nW (73%)	224,490 nW (73%)	299,736 nW (73%)
Potencia de conmutación	54,187 nW (27%)	83,107 nW (27%)	111,493 nW (27%)
Potencia dinámica total	200,677 nW	307,597 nW	411,229 nW
Pérdidas de energías	55,245 uW	85,376 uW	113,981 uW

Resultados de la síntesis del protocolo 2 con prng A			
	32 bits	**64 bits**	**96 bits**
Número de puertos	133	133	133
Número de nodos	1413	1767	1974
Número de celdas	1188	1436	1527
Número de referencias	40	40	35
Área combinacional (μm^2)	10727,212	14992,233	17597,400
Área no combinacional (μm^2)	6786,654	7810,548	8786,526
Área de interconexión (μm^2)	1117,987	1372,885	1433,013
Área total de celdas (μm^2)	17513,866	22802,781	26383,926
Área total (μm^2)	18631,853	24175,666	27816,939
Puertas equivalentes	3167	4123	4771
Potencia interna	173,761 nW (78%)	215,447 nW (76%)	315,460 nW (73%)
Potencia de conmutación	49,564 nW (22%)	67,492 nW (24%)	115,505 nW (27%)
Potencia dinámica total	223,325 nW	282,939 nW	430,965 nW
Pérdidas de energías	61,859 uW	81,236 uW	93,265 uW

Resultados de la síntesis del protocolo 2 con prng A1			
	32 bits	**64 bits**	**96 bits**
Número de puertos	133	133	133
Número de nodos	1426	1965	2027
Número de celdas	1320	1881	1955
Número de referencias	41	42	38
Área combinacional (μm^2)	10233,037	15235,601	15867,658
Área no combinacional (μm^2)	7851,102	9907,188	11915,358
Área de interconexión (μm^2)	1325,279	1811,011	1950,358
Área total de celdas (μm^2)	18084,139	25142,789	27783,016
Área total (μm^2)	19409,418	26953,800	29733,375
Puertas equivalentes	3270	4547	5024
Potencia interna	174,920 nW (79%)	253,937 nW (76%)	306,990 nW (76%)
Potencia de conmutación	46,817 nW (21%)	81,949 nW (24%)	97,065 nW (24%)
Potencia dinámica total	221,737 nW	335,886 nW	404,055 nW
Pérdidas de energías	65,352 uW	96,527 uW	103,570 uW