UNIVERSIDAD CARLOS III MADRID

ESCUELA POLITÉCNICA SUPERIOR

DEPARTAMENTO DE TECNOLOGÍA ELECTRÓNICA

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

PROYECTO FIN DE GRADO

"IMPLEMENTACIÓN Y ANÁLISIS DE CIFRADORES DE FLUJO PARA TECNOLOGÍA RFID"

AUTOR: FRANCISCO JAVIER LEÓN GARCÍA

TUTORES: HONORIO MARTÍN GONZÁLEZ

ENRIQUE SAN MILLÁN HEREDIA

AGOSTO 2013

ÍNDICE

CAPÍTULO 1

INTRODUCCIÓN.....9

1.1.OBJETIVOS 9

1.2. ESTRUCTURA DEL CONTENIDO 11

CAPÍTULO 2

RFID.....12

2.1. ¿QUÉ ES EL RFID?..... 12

2.2. ORIGEN Y EVOLUCIÓN 12

2.3. COMPONENTES Y DESCRIPCIÓN.....15

2.3.1. Etiquetas RFID…..... 16

2.3.1.1. El transpondedor…….....17

2.3.1.2. Alimentación y etiquetas……....18

2.3.1.3. Datos almacenados.....19

2.3.1.4. Programación..... 20

2.3.2. Lector…….. ……...20

2.3.2.1. Módulo de radiofrecuencia……..22

2.3.2.2. Unidad de control……22

2.3.3 Antena..……23

2.3.4 Middleware…..……24

2.3.5. Sistemas de información……...25

2.4. FUNCIONAMIENTO DEL SISTEMA RFID……..…25

2.5. FRECUENCIAS……26

2.5.1. Baja frecuencia (125 KHz)……..…..27

2.5.2. Alta frecuencia (13,56 MHz)……..……..28

2.5.3. Ultra alta frecuencia (UHF)

(433MHz, 860MHz, 928 MHz)……....29

2.5.4. Microondas (2,45GHz , 5,8GHz)……..……...30

2.6. CODIFICACIÓN……31

2.7. ESTÁNDARES Y REGULACIÓN……..……32

2.7.1. Estádares ISO……..……33

2.7.1.1. ISO/IEC 18047……..……33

2.7.1.2. ISO/IEC 18000……33
2.7.1.3. ISO/IEC 159……34
2.7.1.4. ISO/IEC 19762……34
2.7.1.5. ISO/IEC 18046……35
2.7.1.6 ISO/IEC 27429……35
2.7.1.7. Otros estándares ISO/IEC……35
2.7.2. Trazabilidad de las personas……36
2.7.3. Trazabilidad de objetos……36
2.7.4. EPCGLOBAL NETWORK……36
2.8. VENTAJAS E INCONVENIENTES DE LA TECNOLOGÍA RFID….39
2.9. APLICACIONES DE LOS SISTEMAS RFID…..……40
2.9.1. Control de accesos……...41
2.9.2. Identificación de equipajes……41
2.9.3. Comercio a distancia……42
2.9.4. Automóviles…..……42
2.10. SEGURIDAD EN RFID……42
2.10.1. PROTOCOLOS ANTICOLISIÓN……44

CAPÍTULO 3

METODOLOGÍA.....46

3.1. MODO DE TRABAJO..... 46
3.2. FLUJO DE TRABAJO..... 47
3.3. VHDL.....49
3.4. HERRAMIENTAS UTILIZADAS……49

CAPÍTULO 4

CIFRADORES DESARROLLADOS Y ARQUITECTURAS…….....51

4.1. PROTOCOLOS DE AUTENTICACIÓN……52
4.1.1. Cifradores de flujo.....52
4.1.2. Cifrado continuo utilizando LFSRs….....54
4.3. CIFRADOR DE FLUJO UTILIZADO……55
4.4. PROTOCOLO CON DOS SUMADORES……61
4.5. PROTOCOLO CON UN SUMADOR DE 2 ENTRADAS……66

CAPÍTULO 5

TEST Y RESULTADOS……..73

5.1. TEST..... 73

5.1.1. Importancia de los test……73

5.1.2. Análisis estadísticos……73

5.1.3. Aleatoriedad en la salida……..77

5.1.4. Test de correlación clave/salida……..78

5.2. RESULTADOS Y ANÁLIS……..…80

5.2.1.Resultados de síntesis arquitectura dos sumadores……81

5.2.2. Resultados de síntesis arquitectura un sumador……85

5.2.3. Comparación de arquitecturas……88

5.2.3.1. Área……88

5.2.3.2. Consumo de potencia……89

5.2.3.3. Ciclos de reloj / throughput……90

CAPÍTULO 6

CONCLUSIONES Y LÍNEAS FUTURAS.....91

6.1. CONCLUSIONES..... 91

6.2. LÍNEAS FUTURAS..... 92

CAPÍTULO 7

PRESUPUESTO.....93

CAPÍTULO 8

BIBLIOGRAFÍA.....95

ANEXOS.....98

ANEXO A – CÓDIGO ARQUITECTURA DOS SUMADORES DE 2 ENTRADAS Y 64 BITS……..…97

ANEXO B – CÓDIGO ARQUITECTURA SUMADOR DE 2 ENTRADAS Y 64 BITS……..111

ANEXO C – CÓDIGO BANCO DE PRUEBAS PARA 64 BITS…..…126

ANEXO D – RESULTADOS DEL TEST DE NIST..……..128

ANEXO E – CÓDIGO MATLAB Y RESULTADO DEL TEST CHI-CUADRADO……..……133
ANEXO F – SIMULACIONES ARQUITECTURA DOS SUMADORES Y 64 BITS……..……134
ANEXO G – SIMULACIONES ARQUITECTURA CON UN SUMADOR DE 2 ENTRADAS Y 64 BITS……..……139
ANEXO H – CÓDIGO ARQUITECTURA CON DOS SUMADORES DE 2 ENTRADAS y 160 BITS……144
ANEXO I – CÓDIGO ARQUITECTURA SUMADOR DE 2 ENTRADAS Y 160 BITS……..……153
ANEXO J – CÓDIGO BANCO DE PRUEBAS PARA 160 BITS……163
ANEXO K – SIMULACIONES ARQUITECTURA DOS SUMADORES Y 160 BITS……..……165
ANEXO L – SIMULACIONES ARQUITECTURA CON UN SUMADOR DE 2 ENTRADAS Y 160 BITS…….....172

ÍNDICE DE FIGURAS

FIGURA 1: ELEMENTOS DE SISTEMA RFID ……..15
FIGURA 2: ETIQUETA RFID…….16
FIGURA 3: ESQUEMA TRANSPONDEDOR…….17
FIGURA 4: ESQUEMA RFID……..……26
FIGURA 5: DISMINUCIÓN DE LA INTENSIDAD DE CAMPO RESPECTO A LA DISTANCIA DE LA ANTENA……..…….28
FIGURA 6: FORMATO DE ETIQUETA EPC…….38
FIGURA 7: PROTOCOLO DE BÚSQUEDA EN ÁRBOL……...44
FIGURA 8: FLUJO DE PAQUETES EN UN CANAL ALOHA……..…..45
FIGURA 9: FLUJOGRAMA DE LA IMPLEMENTACIÓN DE LAS ARQUITECTURAS……48
FIGURA 10: ESQUEMA DE CIFRADO EN FLUJO SÍNCRONO……53
FIGURA 11: ESQUEMA DE CIFRADO EN FLUJO AUTOSINCRONIZANTE……54
FIGURA 12: S-BOX INICIAL…..……56
FIGURA 13: FLUJOGRAMA DEL ALGORITMO A IMPLEMENTAR……..…60
FIGURA 14: DIAGRAMA DE BLOQUES DEL CIRCUITO PARA LA ARQUITECTURA DE DOS SUMADORES…….....…….62
FIGURA 15: MÁQUINA DE ESTADOS PARA ARQUITECTURA DE DOS SUMADORES……..…66
FIGURA 16: DIAGRAMA DE BLOQUES DEL CIRCUITO PARA LA ARQUITECTURA DE UN SUMADOR……67
FIGURA 17: MÁQUINA DE ESTADOS PARA ARQUITECTURA DEL SUMADOR DE DOS ENTRADAS ……72
FIGURA 18: GRÁFICO PUERTAS EQUIVALENTES ARQUITECTURA DE 64 BITS CON DOS SUMADORES……83
FIGURA 19. GRÁFICO PUERTAS EQUIVALENTES ARQUITECTURA DE 160 BITS CON DOS SUMADORES…….84
FIGURA 20. GRÁFICO POTENCIA ARQUITECTURA DE 64 BITS CON DOS SUMADORES……..84
FIGURA 21. GRÁFICO POTENCIA ARQUITECTURA 160 BITS CON DOS SUMADORES……..…85

FIGURA 22. GRÁFICO PUERTAS EQUIVALENTES ARQUITECTURA DE 64 BITS Y UN SUMADOR……87
FIGURA 23. GRÁFICO PUERTAS EQUIVALENTES ARQUITECTURA DE 160 BITS Y UN SUMADOR……87
FIGURA 24. GRÁFICO POTENCIA ARQUITECTURA DE 64 BITS Y UN SUMADOR……88
FIGURA 25. GRÁFICO POTENCIA ARQUITECTURA DE 160 BITS Y UN SUMADOR ……88

ÍNDICE DE TABLAS

TABLA 1:TABLA RESUMEN ARQUITECTURA DOS SUMADORES……82
TABLA 2: TABLA RESUMEN ARQUITECTURA UN SUMADOR….....86

CAPÍTULO 1

INTRODUCCIÓN

La tecnología de identificación por radiofrecuencia RFID [1], (Radio Frequency Identification) se ha convertido en una de las tecnologías de comunicación con más crecimiento. Esta tecnología ofrece la posibilidad de realizar lectura a distancia de la información que posee una tarjeta a través de un lector, sin necesidad de contacto físico (ni siquiera visual), además de disponer de la posibilidad de realizar lecturas y escrituras simultáneamente.

El RFID es uno de los sistemas más utilizados en diversos ámbitos de la sociedad y de las empresas, como puede ser la trazabilidad y el control de inventarios, la localización y el seguimiento de animales, personas y objetos, o la seguridad e identificación en los controles de seguridad de accesos.

Esta tecnología consiste en realizar una identificación de objetos mediante radiofrecuencia, lo que supone un avanzado sistema de identificación automática y una opción diferente ante sistemas de seguimiento y control más tradicionales.

La vulnerabilidad de estos sistemas es uno de los principales puntos débiles, ya que es relativamente sencillo realizar ataques con fines malintencionados a los sistemas RFID de manera. Por ello, uno de los principales puntos a tratar a la hora de desarrollar esta tecnología consiste en la seguridad para evitar dicha intrusión, por lo que se emplean protocolos de autenticación que permiten la identificación segura de los agentes que intervienen en la comunicación.

1.1. OBJETIVOS

En este proyecto se realiza la implementación de un cifrador de flujo (streamcipher) diseñado para RFID. Se presentan varias arquitecturas con las que se intentan mejorar alguno de los pará metros característicos de la tecnología (área, consumo, throughput, etc.). Además se utilizaran varias técnicas de reducción de potencia, ya que hay que tener en cuenta que la tecnología RFID exige un área y una potencia limitadas, con un límite considerablemente pequeño.

Para la implementación del protocolo se empleará el lenguaje VHDL, el cual permite la descripción de circuitos digitales. Posteriormente, se empleará la herramienta synopsys, la cual nos permite obtener una estimación real del

área y del consumo de potencia de cada una de las arquitecturas desarrolladas.

Los objetivos a alcanzar en este proyecto son los siguientes:

1) **Estudio del cifrador de flujo:** En primer lugar se debe realizar el estudio del streamchipher (cifrado de flujo) que será utilizado en la implementación a realizar. Para ello se procederá al estudio de los cambios que se han de realizar a los datos de entrada, así como los cambios que debe sufrir la salida en relación a la clave que sea introducida y a los datos.

2) **Proposición de arquitecturas:** Una vez diseñado y estudiado el algoritmo de encriptación a utilizar, que se procede a determinar los bloques que compondrán el circuito y se estudian diferentes alternativas que pudieran ser mejores a la hora de conseguir reducir el consumo y el área del circuito.

   Para ello, se realizan diferentes arquitecturas, todas ellas basadas en el mismo cifrador de flujo, obteniendo la misma salida del circuito a través de todas ellas, pero utilizando variaciones en él.

3) **Implementación:** Determinadas las diferentes estructuras a realizar y los elementos que compondrá cada una de ellas, se procederá a su implementación en VHDL con la herramienta de xilinx, teniendo en cuenta únicamente que se cumpla con lo deseado en la salida del circuito, atendiendo en un proceso posterior a las limitaciones mencionadas a lo largo del proyecto, no haciéndolo así en este punto. Para realizar este apartado, se decidió dividir la implementación en distintos bloques, de tal forma que se pudiera identificar, claramente, la funcionalidad de todos los bloques que forman parte del circuito a llevar a cabo.

4) **Simulación y análisis:** Acabada la implementación, se procede a realizar la simulación. Para comprobar la veracidad de los resultados, se realiza paralelamente el mismo algoritmo en C, de tal forma que introduciendo diferentes claves y datos tanto al circuito en VHDL como al programa realizado en C, los resultados obtenidos han de ser los mismos, comprobando así el correcto funcionamiento del sistema.

5) **Test de aleatoriedad y correlación:** Para reforzar la seguridad de que el circuito está implementado correctamente, se someterán los resultados a ciertos análisis matemáticos, con el fin de comprobar la no correlación entre la salida y la entrada y la aleatoriedad en la salida generada por el circuito.

1.2. ESTRUCTURA DEL CONTENIDO

Este proyecto ha sido dividido en 8 capítulos.

En el primer capítulo se realiza una breve introducción a la tecnología RFID y se exponen los objetivos a alcanzar.

En el segundo capítulo se presenta el estado del arte de la tecnología, explicando en profundidad la historia de la tecnología del RFID, en qué consiste esta tecnología, ventajas e inconvenientes de utilizarla y posibles aplicaciones.

El tercer capítulo muestra las herramientas y la metodología que se han utilizado para realizar las pruebas y las sintetizaciones de los circuitos que se desarrollan en este proyecto.

En el cuarto capítulo se entra en profundidad a explicar qué son los cifradores de flujo y a detallar y explicar los que se han realizado para el presente proyecto con esquemas y definiciones paso a paso del funcionamiento de las diferentes arquitecturas.

El quinto capítulo se centra en la descripción de los test realizados para corroborar el correcto funcionamiento de las arquitecturas y se exponen los resultados obtenidos en dichos test.

El sexto capítulo muestra las conclusiones obtenidas al finalizar el proyecto y las líneas futuras que pueden presentarse.

El presupuesto del proyecto se presenta en el séptimo capítulo, dejando el octavo para la bibliografía utilizada.

Finalmente se incluye un anexo con todos los códigos VHDL realizados y las simulaciones que muestran el correcto funcionamiento del circuito.

CAPÍTULO 2

RFID

2.1. ¿QUÉ ES EL RFID?

El sistema RFID [1] es la tecnología electrónica capaz de permitir la comunicación entre un lector y una etiqueta. Los sistemas RFID pueden almacenar una cantidad considerable de datos (hasta 2kbytes) a través de comunicadores de radiofrecuencia.

El RFID es uno de los sistemas punteros en tecnología de identificación, ya que permite mejorar la gestión de la cadena de suministro de un almacén, llevar a cabo con éxito los sistemas de identificación en zonas de seguridad, y otras múltiples aplicaciones que gracias al RFID se hacen sencillas, rápidas y muy eficaces.

Uno de los principales puntos fuertes de esta tecnología es que la recuperación de la información almacenada en la etiqueta se realiza a través de radiofrecuencia y sin la necesidad de que se propicie contacto físico ni visual entre el dispositivo lector y la etiqueta, a pesar de que se requiere una distancia de proximidad entre los componentes.

Así pues, podemos observar que los sistemas RFID se utilizan de multitud de maneras, como tarjetas identificadoras sin contacto (pago en los peajes por ejemplo), inmovilizadores de vehículos (con un sistema interrogador en el vehículo y su identificador en la llave), identificar envíos de paquetes en empresas de transporte, identificadores de equipaje, inventariado, detector antirrobo en tiendas, localización de documentos, bibliotecas y más actualmente se está estudiando la posibilidad de incluir chips con el historial médico de las personas y billetes en curso legal, para evitar robos de estos últimos y facilitar su seguimiento en caso de producirse.

2.2. ORIGEN Y EVOLUCIÓN

Cabe mencionar, antes de entrar en los sistemas RFID, que dicha tecnología está ligada a los descubrimientos de James Clerk Maxwell, Heinrich Rudolf Hertz y Guglielmo Marconi , los cuales introdujeron la base fundamental de la tecnología RFID: transmisión de datos de forma inalámbrica en una frecuencia del espectro electromagnético y el uso de ondas de radio para la lectura de las Tags.

El origen del RFID [2] surge durante la segunda guerra mundial, en la cual se desarrolló la tecnología del transponedor de IFF (Identification Friend of Foe). Fue desarrollado por el ejército británico con el fin de identificar aviones aliados o enemigos. Dicho sistema utilizaba un dispositivo que llevaban a bordo los aviones aliados, el cual emitía señales codificadas y que eran detectadas y decodificadas por los radares británicos.

Terminada la segunda guerra mundial, se continuaron con las investigaciones, siendo en octubre de 1984 cuando Harry Stockman editó un artículo en la revista Proceedings of the IRE, titulado “Communications by Means of Reflected Power”, el cual se considera la investigación más próxima a la tecnología RFID.

A partir de este momento, los desarrollos e investigaciones han sido un proceso lento pero constante, hasta llegar a la época de los 50. En esta época se realizaron numeroso estudios relacionados con la tecnología, orientados en su mayoría al diseño de sistemas seguros para aplicaciones en minas de carbón, instalaciones nucleares, controles de acceso o sistemas antirrobo, llegando a publicarse dos artículos de vital importancia: “Applications of Microwave Homodyne”, de F. L. Vernon, y “Radio Transmission Systems with Modulatable Passive Responders”, de D. B. Harris.

Durante los años 60 se realizaron investigaciones en el campo de la teoría electromagnética y aparecieron las primeras pruebas de campo. En esta época se instaló la activación remota de dispositivos con batería, la comunicación por radar o los sistemas de identificación interrogación-respuesta. Es en este periodo, además, donde comienzan las actividades comerciales, fundándose Sensormatic y Checkpoint, las cuales diseñan equipos de vigilancia electrónica contra la intrusión (EAS). Electronic Article Surveillance (EAS), es el primer desarrollo de RFID y el que se ha utilizado de forma más extendida. esta fue la semilla de dicha tecnología.

En la década de los 70 investigadores, desarrolladores, centros de investigación, empresas e instituciones, apostaron por el desarrollo activo de la tecnología, lo que concluyó en notables avances. Es en este ámbito donde aparecen las primeras aplicaciones de RFID. Las grandes empresas empiezan a desarrollar tecnología de identificación electrónica, y es en 1978 cuando se desarrolla un transponedor pasivo de microondas.

A finales de la época, se habían desarrollado ya un avance muy importante en la investigación del electromagnetismo y electrónica necesarias para las redes RFID. EEUU fue el pionero, centrando sus principales intereses en sistemas para transporte, acceso de personal e identificación de animales. Más tarde aparecieron los peajes electrónicos, realizándose la primera aplicación para aduanas en 1987, en Noruega.

Entrada la época de los 90, se inicia el peaje con control electrónico en autopistas de Houston y Oklahoma (EEUU), sistema que permitía gestionar el paso de vehículos por los puestos de control. En Europa se investigaron otras aplicaciones, usándose sistemas de microondas e inductivos para billetes electrónicos y control de accesos.

El RFID introdujo un avance muy importante en la tecnología automovilística, desarrollando Texas Instruments un sistema de control del encendido de los automóviles. Philips contribuye también al avance en esta materia, desarrollando un sistema de control de combustible, la gestión del encendido y el control al acceso de los vehículos.

Las aplicaciones realizadas para autopistas y billetes electrónicos se expandieron por Asia, África, Sudamérica y Australia, lo que contribuyó a que la tecnología RFID se aplicara a otros ámbitos económicos debido a su gran popularidad y éxito.

Es en Dallas donde por vez primera un solo tag es utilizado para el acceso a una autopista, a diferentes garajes, accesos de universidad, etc. La tecnología avanza enormemente durante esta década y los desarrollos tecnológicos que se producían en multitud de ámbitos, proporcionaron la opción de fabricar circuitos más pequeños, con mayor alcance, mayor memoria y, además, con un menor coste de fabricación.

A partir del año 2000, el objetivo se instaló en el desarrollo de etiquetas a un precio muy bajo, por lo que la tecnología RFID se convirtió en una gran alternativa a los códigos de barras.

En 2002 empieza a despuntar la tecnología NFC (Near Field Communication), lo cual mejoraba las características del RFID, incorporando un único dispositivo, un emisor y un receptor RFID, pudiendo llegar a insertarse en un teléfono móvil, aportándole nuevas funcionalidades.

El año 2003 marcó un antes y un después en la tecnología RFID: Walmart y el departamento de defensa de Estados Unidos, decidieron implantar la tecnología RFID. De esta manera, es en este año cuando el centro AutoID, se convierte en EPCglobal, creando a partir de este momento estándares adoptados por Wallmart y el departamento de defensa de Estados Unidos.

Texas instruments desarrolla nuevas aplicaciones, siendo una de las pioneras en el diseño de esta tecnología. Aplicaciones como el control del encendido de los vehículos o control para pases de esquí, fueron una gran aportación al avance del RFID, lo que produjo una amplia introducción en el mercado de empresas Europeas.

En España, en 2005, se realizó un proyecto lanzado por Correos (Q-RFID), el cual incorporó las últimas tecnologías de control por radiofrecuencia para permitir la planificación de la correspondencia. Fue en 2007 cuando terminó de implantarse el Q-RFID y resultó uno de los proyectos más importantes y determinantes en Europa,

Actualmente, la tecnología RFID es utilizada en multitud de campos, ya que permite transportar datos y ser capturados electrónicamente.

En el mundo actual, basado en la información, el presente y el futuro del RFID es de suma importancia, ya que cada vez se desecha más la opción del uso de cableado y se requieren de radios de acción cada vez más elevados. El interés actual por el comercio virtual, hace del RFID uno de los sistemas más globales e importantes de los utilizados hoy en día, dejando de lado, cada vez más, a sistemas antiguos como el código de barras. En la actualidad, la tecnología RFID dispone de un gran avance, pudiendo realizarse sistemas con un área relativamente pequeña, una rapidez considerable y suficiente para las aplicaciones en las que se utiliza y un consumo cada vez menor de los circuitos. El futuro del RFID parece ir encaminado a mejorar todos estos últimos aspectos, para lo cual necesita ir de la mano de investigaciones y avances en multitud de campos de la tecnología moderna.

2.3. COMPONENTES Y DESCRIPCIÓN

El RFID [3] es una tecnología que permite el almacenamiento y recuperación de datos basándose en la utilización de etiquetas donde se almacena la información, empleando diferentes bandas para utilizar las señales de radiofrecuencia.

Los elementos básicos de todo sistema RFID son: etiqueta RFID, lector o interrogador, un ordenador, host o controlador y una antena RFID.

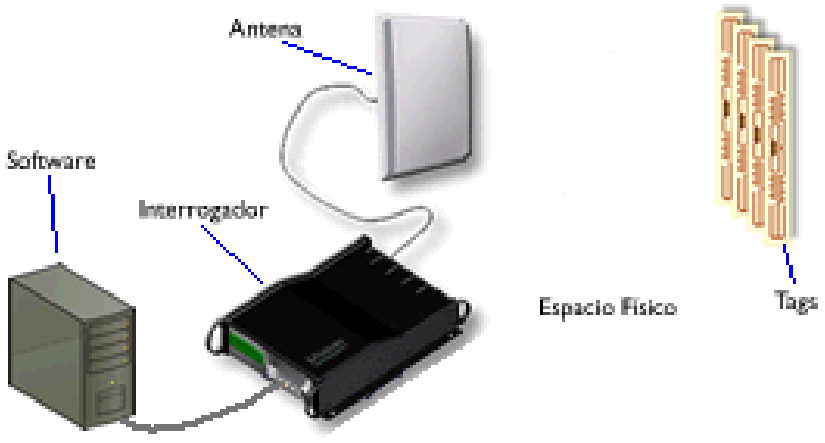


Figura 1. Elementos de sistema RFID [4]

2.3.1. Etiquetas RFID

Una etiqueta se adhiere en el objeto, animal o persona, conteniendo la información que deseamos extraer posteriormente a través del lector correspondiente.

La etiqueta se compone de un microchip en el cual se almacenan los datos deseados y una antena que permite la comunicación entre el lector y el chip a través de radiofrecuencia. Los transponedores son diseñados de tal forma que utilicen una frecuencia que sea la más adecuada al sistema en el cual se deben acoplar, estableciendo de esta forma la distancia máxima de lectura y la necesidad de que el chip soporte el ambiente al cual será sometido.

Se pueden diferenciar dos tipos principales de etiquetas: tags activos (los cuales incluyen una batería integrada) y los tags pasivos (los cuales no incluyen alimentación de forma integrada, obteniendo la energía del campo generado por el lector).

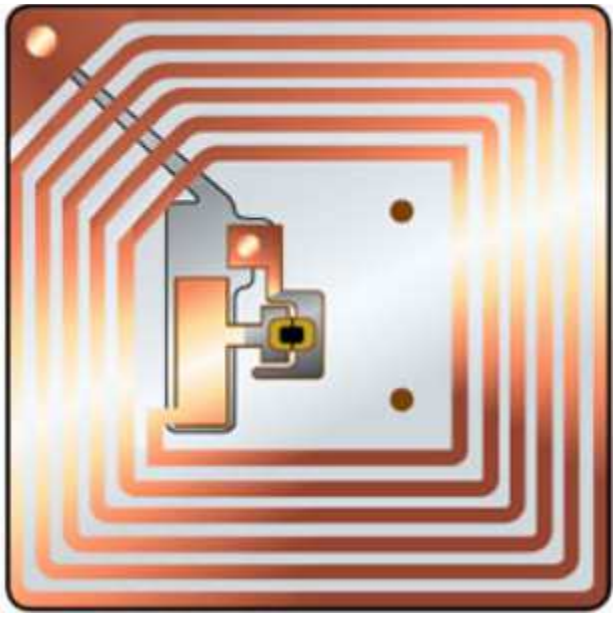


Figura 2. Etiqueta RFID [5]

Existen etiquetas de diferentes formas, tamaños y carcasas, dependiendo de la aplicación en la cual serán instaladas. Para ensamblarlas se utiliza una material como base (papel, plástico…) de una antena fabricada con materiales conductivos (cobre, plata o aluminio). Después se conecta el microchip a la antena y se protege el sistema con materiales que permitan combatir las dificultades del medio en el que se vayan a encontrar, de tal forma que sea un material adhesivo, puesto que posteriormente irán pegadas al objeto donde se desea almacenar la información.

El tamaño de la etiqueta suele ir desde milímetros hasta unos pocos centímetros. Algunas etiquetas, llevan unas medidas estandarizadas, como por ejemplo las etiquetas inteligentes RFID (85,72mm x 54,03mm x 0,76mm), mientras que en el resto, el tamaño dependerá de la aplicación y de lo que se estime necesario.

Las tags pueden ir encapsuladas en ampollas, pilas, monedas, llaves, relojes, etc. Existiendo una amplia variedad de formas y tamaños donde pueden ir instaladas.

2.3.1.1. El transpondedor

Es el dispositivo [6] que va embebido en la etiqueta y el cual contiene la información necesaria del objeto en el cual está instalado, y está compuesto por un chip y una antena.

De forma opcional, estos dispositivos pueden llevar incorporada una batería para alimentar las transmisiones o incluso, pueden disponer de circuitos con funciones adicionales de entrada/salida como registros de tiempo, entre otros.

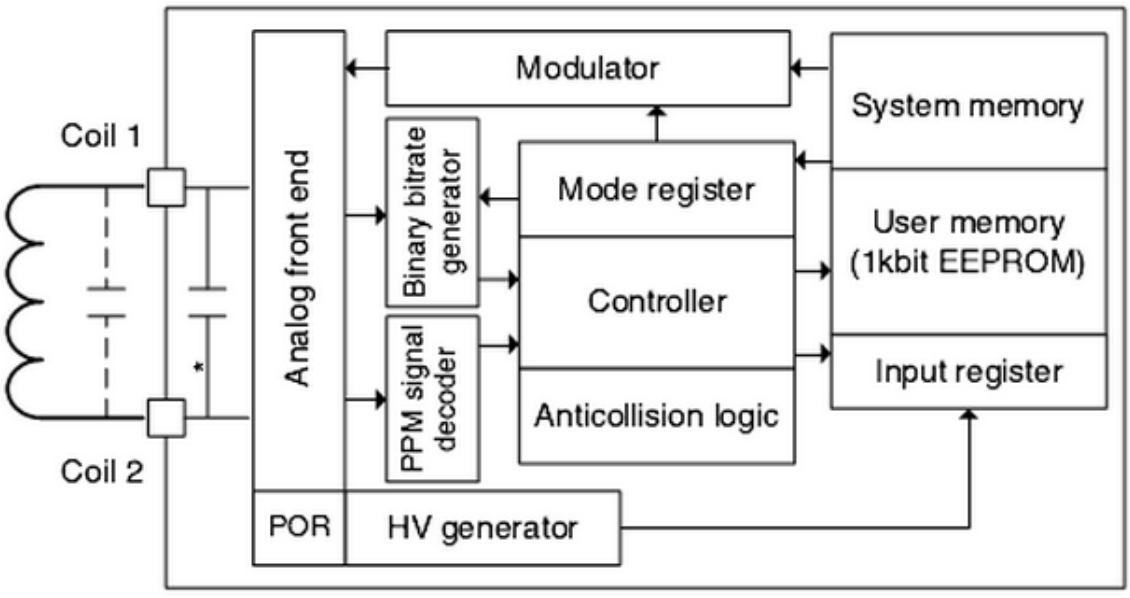


Figura 3. Esquema transponedor [7]

Los microchips incluyen:

- Circuto analógico que realiza la transferencia de datos y suministra la alimentación.

- Circuito digital que dispone la lógica de control, la lógica del microprocesador y la lógica de seguridad.

- Memoria para almacenar los datos, la cual suele estar compuesta por una memoria ROM que guarda los datos del funcionamiento del sistema y de la seguridad, una memoria RAM que permite el acceso temporal de datos y una memoria de programación no volátil (como una EEPROM), que se encarga de que los datos queden almacenados cuando el dispositivo está inactivo.

- Registros de datos que soportan los datos de entrada después de la demodulación y los de salida antes de esta.

2.3.1.2. Alimentación y etiquetas

Las etiquetas [8] deben tener alimentación para realizar la transmisión de datos, a pesar de que la que requiere estos circuitos es baja (del orden de microwatios). Según la forma en la que estos circuitos obtengan la alimentación, podemos distinguir entre etiquetas activas y pasivas.

Las etiquetas activas disponen de una batería propia que se encarga de suministrar la energía que requieren, a pesar de que se alimentan también de la energía suministrada por el lector en el momento de la transmisión. Usualmente llevan incorporadas una batería que posee una gran relación potencia-peso y que es capaz de funcionar en un rango de temperaturas amplio.

El inconveniente de las etiquetas activas frente a las pasivas, es que poseen una duración finita, ya que el agotamiento de la batería acaba con la vida de la etiqueta. No obstante, una colocación adecuada y correctamente acoplada a la circuitería de baja potencia, asegura una vida de unos 10 años aproximadamente, dependiendo del uso y las condiciones que esta deba soportar.

Las etiquetas activas son dispositivos de lectura y escritura (normalmente) y presentan una gran ventaja frente a las etiquetas pasivas, la cual es la posibilidad de gestionar otros dispositivos, como por ejemplo sensores.

Las tags activas permiten mayor radio de cobertura que las pasivas, una inmunidad superior al ruido y tasas de transmisión mucho más altas en alta frecuencia, pero presentan el inconveniente del agotamiento y de un coste más elevado que las etiquetas pasivas.

Existen dos tipos de tarjetas activas según su funcionamiento: aquellas que se encuentran desactivadas y se activan al entrar en contacto con el lector (ahorrando batería) y aquellas que envían señales de forma periódica aunque

no estén en comunicación con el lector (estas operan en frecuencias más bajas para reducir el consumo de batería).

Las etiquetas pasivas no disponen de elementos que suministren ningún tipo de alimentación, siendo obtenida la potencia necesaria del campo que genera el lector cuando se realiza la transmisión de datos.

Al no disponer de batería, las tags pasivas son más ligeras, pequeñas y flexibles que las activas, además de suponer un coste menor y una vida prácticamente ilimitada. Por el contrario, el radio de cobertura que ofrecen es menor y requieren de una cantidad de energía mayor proveniente de los lectores. Poseen limitaciones en el almacenamiento de datos y una mala inmunidad al ruido.

Existe un tipo de tarjeta pasiva que incorpora batería, pero esta no es utilizada para realizar la transmisión, sino para alimentar la circuitería del microchip.

2.3.1.3. Datos almacenados

Para la organización y almacenamiento de datos [9], se utiliza un proceso denominado codificación de fuente, los cuales permiten que los datos almacenados en las etiquetas dispongan de organización para facilitar la recuperación de datos (utilización de bits de detección de errores por ejemplo).

La aplicación en la cual será utilizada la etiqueta será la que determine la cantidad de datos que se desee almacenar, pudiéndose usarse para transportar un identificador (cadena alfanumérica que puede representar una identidad o una clave de acceso a información) o ficheros de datos denominados PDF (almacenamiento de información organizada que permite ser transmitida o iniciar ciertas acciones).

Las etiquetas más usuales son aquellas que permiten almacenar desde un único bit hasta cientos de kbytes. Las tags que disponen de un solo bit poseen dos estados: etiqueta en zona de lector y etiqueta fuera de la zona del lector. Algunos permiten activar y desactivar la etiqueta y no requieren de microchip, por lo que su coste es muy reducido.

Estas etiquetas de un bit son utilizadas principalmente en sistemas antirrobo en aplicaciones EAS (Electronic Article Surveillance), para permitir la vigilancia de manera electrónica de artículos en tiendas principalmente, gracias a que el bit de la etiqueta dispara una alarma cuando esta atraviesa el campo del lector. Otra aplicación muy usual en este tipo de tarjetas es la de recuento de individuos y objetos.

Las tags de 128 bits almacenados suelen llevar un número de serie con bits de paridad adicionales. Estas tags son de uso particular y suelen ser programadas de forma específica para la aplicación deseada.

Las etiquetas de 512 bits se programan por el usuario y suele ser común utilizarlas para guardar identificadores y datos específicos.

Las etiquetas de 64kbits almacenan ficheros de datos y su función es el transporte de estos.

La velocidad con la que los datos son leídos dependerá de la frecuencia portadora. Una frecuencia portadora alta implica una velocidad de datos mayor, a cambio de un consumo de potencia más elevado.

A la hora de utilizar una tarjeta, hay que tener en cuenta que el tiempo en el cual atraviesa el campo de lectura la tarjeta, ha de ser mayor que la velocidad de la lectura de los datos, ya que sino el lector no será capaz de realizar la lectura de forma correcta. Es por ello que la velocidad de lectura suele ser mayor cuando los datos almacenados tienen mayor envergadura.

2.3.1.4. Programación

Dependiendo de la memoria que incorpore la etiqueta, los datos pueden ser de:

- Sólo lectura: son programados por el fabricante en el momento de fabricación y son de baja capacidad. Se suelen utilizar para almacenar identificaciones o claves.

- Lectura y escritura: son programables por el usuario, permitiendo modificar los datos almacenados.

- Una sola escritura y múltiples lecturas: son programables por el usuario pero solo es posible programarlo una vez.

2.3.2. Lector

Es el encargado de leer los datos enviados por la etiqueta y de suministrar la energía necesaria en el caso de etiquetas pasivas. Dispone de un transmisor y un receptor de radiofrecuencia, una unidad de control, y una antena para establecer la comunicación con los tags.

Las funciones principales del lector consisten en servir de interfaz a la etiqueta, el procesamiento de información y el sistema de almacenamiento, interrogar a las etiquetas y recopilar la información, actuar con comandos de software, convertir ondas analógicas de radio en datos digitales, transmitir datos a otros dispositivos y soportal múltiples protocolos.

Para cumplir estas tareas mencionadas, el lector incorpora un módulo de radiofrecuencia (con emisor y receptor) una antena y una unidad de control. El lector cuenta con una interfaz a un ordenador, host o controlador, a través de

un enlace remoto o local (Ethernet, WLAN, RS232, etc), para enviar los datos al sistema de información.

Los lectores tienen tres modos de actuación:

- Chequeando su zona de cobertura de forma continua: suele utilizarse en aplicaciones donde se espera que estén pasando etiquetas por el lector continuamente.

- Chequeando su zona de cobertura periódicamente: para detectar presencia de nuevas etiquetas.

- Chequeando de forma puntual: por ejemplo, si un sensor detecta la aparición del objeto, entonces el lector entra en funcionamiento.

Un lector o interrogador se compone de un módulo de radiofrecuencia (receptor y transmisor), una unidad de control y una antena y su complejidad variará en función del transponedor que deban alimentar y las funciones que deba desempeñar. Podemos diferenciar así lectores fijos que se posicionan en zonas específicas como puertas de acceso de modo que monitoricen las etiquetas y lectores móviles los cuales suelen ser dispositivos de mano como teléfonos e incorporan una pantalla LCD para la introducción de datos y una antena integrada en la unidad portátil lo que hace que su radio de cobertura sea pequeño.

La diferencia entre unos lectores y otros pueden radicar en:

- El protocolo de funcionamiento: protocolo que ofrece el soporte del sistema (ISO, propietarios…).

- La frecuencia: banda de frecuencia en la cual el lector realiza sus operaciones (alta frecuencia, baja frecuencia y microondas).

- La regulación: regulación de frecuencia y de potencia a la que se acoge el dispositivo, siendo la máxima potencia permitida de 0,5 Watios en Europa.

- La interfaz con el ordenador: existen varios tipos de interfaz como pueden ser TCP/IP, WLAN, Ethernet o RS 485, entre otros.

- La posibilidad de multiplexación de varios lectores: se realizará a través de concentradores o a través del middleware.

- La posibilidad de actualización del software del lector on-line: se realizará vía internet o a través de la interfaz con el host.

- La capacidad del lector de gestionar varias antenas.

- La capacidad para actuar con otros productos de middleware.

- Entrada/Salida digital.

- Circuitos ASIC de control adicionales.

Finalmente, antes de detenernos en las partes que figuran en los lectores, cabe destacar los modos de interacción de estos. Se pueden diferenciar dos modos diferentes, en el primero la etiqueta transmite de forma periódica la información, esperando que algún lector pueda recibirla y en el segundo modo es el lector el que envía la orden a la etiqueta para que transmita la información que posee.

2.3.2.1. Módulo de radiofrecuencia

Consta de un transmisor y un receptor que se encargan de generar y recibir la señal de radiofrecuencia proveniente de la etiqueta. Debe, por tanto, generar la señal de radiofrecuencia para transmitir la energía a la etiqueta y activarla, modular la transmisión de la señal para enviar los datos y recibir y demodular las señales que llegan desde la tag.

2.3.2.2. Unidad de control

Está constituida por un microprocesador, aunque en algunas ocasiones, la unidad de control dispone de un circuito ASIC (circuito integrado de aplicación específica) para apoyar al microprocesador.

La unidad de control se encargará de codificar y decodificar los datos de los transponedores, comprobar la veracidad de dichos datos, almacenar los mismos, activar las etiquetas, autentificar y autorizar la transmisión de datos, detectar y corregir errores, gestionar el proceso de multilectura para evitar la colisión, cifrar y descifrar los datos y comunicarse con el sistema de información.

Una de las funciones principales de la unidad de control consiste en gestionar el acceso. En las tecnologías que permiten la transmisión de datos sin contacto físico, pueden aparecer interferencias que dañen los datos originales durante la transmisión. Para evitarlo, se utilizan procedimientos de chequeo, los cuales permitirán detectar errores en los datos una vez han sido recibidos. Los más comunes son los bits de paridad, LRC (comprobación de redundancia longitudinal) y CRC (comprobación de redundancia cíclica).

La cantidad de etiquetas que un lector puede identificar en un tiempo determinado, depende del protocolo y de la frecuencia a la cual se trabaje,

siendo a mayor frecuencia la forma en la cual se pueden leer mayor cantidad de datos.

2.3.3. Antena

Es el elemento que se encarga de habilitar la comunicación entre el lector y la etiqueta. Las antenas pueden ser de multitud de formas y tamaños, llegando en ocasiones a ser de crítica importancia para la aplicación su diseño. El diseño puede ir desde dispositivos de mano hasta grandes antenas. Podemos hacer una clasificación del tipo de antenas, de tal forma que las más comunes suelen ser:

- Antenas de puerta.
- Antenas omnidireccionales.
- Antenas polarizadas circularmente.
- Antenas polarizadas linealmente.
- Dipolos o multipolos.
- Antenas de varilla.
- Antenas adaptativas o de arrays.

El elemento principal de las antenas es la frecuencia de operación a la que trabaja el sistema. No obstante, existen otros parámetros importantes a considerar como la impedancia, la máxima potencia permitida, la ganancia o el patrón de polarización (X-Y o circular). Dichos elementos son los encargados de crear el campo de radiofrecuencia, además de la eficiencia de la antena y el tipo de acoplamiento que la antena pueda llevar con la etiqueta.

Para elegir una antena, el parámetro principal a tener en cuenta será el área de cobertura requerida, de tal forma que debe satisfacer la distancia máxima que la aplicación requiera, pero además debe evitar lecturas no válidas que puedan introducir errores en el sistema.

La orientación de la antena es otra de las consideraciones a tener en cuenta a la hora de establecer la cobertura, ya que la cantidad de potencia enviada a la etiqueta, dependerá también de la orientación del lector con respecto a la etiqueta.

Las antenas suelen presentarse como productos finales, por lo que todos los aspectos mencionados han de tenerse en consideración. A pesar de

ello, la mayoría de antenas son sintonizables para ajustarse a la frecuencia deseada para el sistema, lo que las hace susceptibles a factores como las variaciones de RF y del entorno, pérdidas debidas a proximidad de materiales, interferencias con otras fuentes, etc. Problemas que pueden ser solucionables a partir de circuitos dinámicos sintonizadores, que realimenten de forma continua la antena y la sintonicen adecuadamente.

Una vez una etiqueta es detectada, el lector realizará las operaciones requeridas sobre ella, pudiendo leer o escribir información en ella. Una vez finalizada la operación anterior, el lector ha de descartar la tag para estar disponible para detectar etiquetas posteriores. Para realizar todas estas operaciones existen un gran número de protocolos que las administran, por ejemplo el protocolo “orden-respuesta”, en el cual el lector ordena a la etiqueta que cese su transmisión cuando ha recibido la información necesaria. Otro ejemplo de protocolo es el “Sondeo Selectivo”, a partir del cual el lector busca específicamente las etiquetas que tienen una identificación determinada y extrae la información de cada una de ellas en diferentes turnos.

2.3.4. Middleware

Es el encargado de realizar la conexión entre los sistemas de información y el hardware RFID. Los sistemas RFID no servirían de nada sin el middleware, puesto que no se nos permitiría realizar ninguna operación ni gestión de la información obtenida a través de ellos. Así, el middleware se ocupa de establecer un camino a los datos entre los lectores y las etiquetas y los sistemas de información en los cuales se procesarán dichos datos y se les dará una utilización adecuada. Por tanto, la calidad y el uso de una aplicación RFID depende en su mayoría del middleware.

Sus funciones principales son las siguientes:

- Adquirir datos: se encarga de la recopilación, filtrado y organización de los datos en caso de que estos procedan de múltiples lectores, evitando el colapso de estos sistemas.

- Encaminar los datos: hace posible la integración de las redes de elementos y los sistemas RFID, dirigiendo los datos al sistema correcto dentro de la organización empresarial (comúnmente).

- Gestionar los procesos: ofrece la posibilidad de activar eventos en función de unos parámetros (envíos no autorizados, pérdidas de stock de un almacén, etc.).

- Gestionar los dispositivos: monitoriza y coordina los diferentes lectores RFID existentes en un sistema, verifica el estado de cada uno de ellos y posibilita la gestión remota.

2.3.5. Sistemas de información

El sistema de información es el encargado de comunicarse con el lector de forma maestro-esclavo, de tal forma que todas las operaciones realizadas por el lector son iniciadas por el software del sistema. Cuando un lector recibe orden del software, establece la transmisión de datos con las etiquetas, ejerciendo el primero de maestro del segundo.

Su principal objetivo es el de tratar y gestionar los datos obtenidos del lector. Por ello, el sistema debe ser lo suficientemente avanzado y sofisticado para ser capaz de manejar las múltiples lecturas a las que se vea sometido por parte del lector, así como coordinar los tiempos adecuadamente, manejar correctamente los flujos de la información, gestionar eventos, soportar realimentaciones producidas por el usuario y permitir el uso de actualizaciones, entre otros requisitos que podemos necesitar para ciertas aplicaciones específicas. No obstante, en cualquier aplicación, el software necesita realizar modificaciones para integrar los datos obtenidos de los lectores y el programador, permitiendo acceder a todas las funcionalidades que el sistema RFID proporcione, con el fin de no perder efectividad de dicho sistema.

Algunos sistemas de información más conocidos son el ERP (Enterprise Resource Planning) para la gestión de inventarios, el sistema de almacenes WMS (Warehouse Management System), el sistema de comprobantes de recogida POC (Proof Of Collection) o el sistema de albaranes POD (Proof Of Delivery).

2.4. FUNCIONAMIENTO DEL SISTEMA RFID

Los principios de funcionamiento [10] de los sistemas RFID se basan siempre en los mismos, a pesar de tener una amplia variedad de sistemas como hemos visto anteriormente. Estos principios son los siguientes:

1. Se han de equipar los objetos a identificar con una etiqueta RFID.

2. La antena instalada en el lector emite un campo de radiofrecuencia que activa la etiqueta.

3. Al encontrarse la tarjeta en el campo mencionado, utiliza la energía y la referencia para transmitir los datos almacenados en su memoria. En el caso de que la etiqueta sea activa, la energía necesaria a la hora de transmitir los datos la proporciona la batería instalada en la propia etiqueta.

4. El lector recibirá los datos y los enviará al procesador de datos correspondiente.

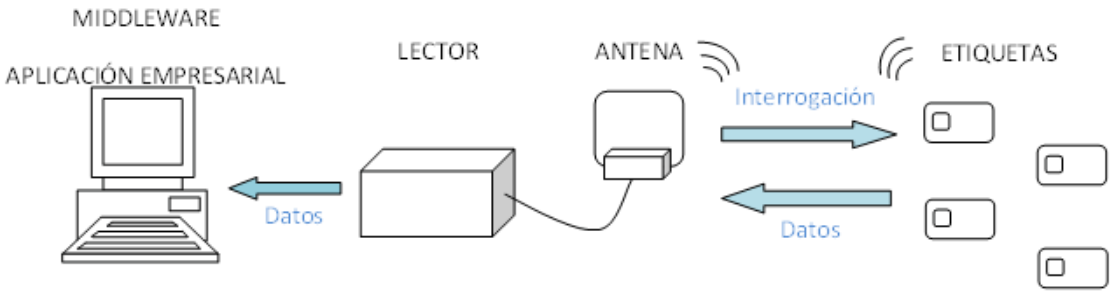


Figura 4. Esquema RFID [11]

La conexión entre el lector y el ordenador se realiza a través de un enlace de comunicaciones estándar, de forma local, remota, cableado o inalámbrico, como pueden ser Ethernet, WLAN, USB, etc.

La comunicación entre el lector y la tag, se realiza a través de un enlace de radiofrecuencia que posee sus propias características y protocolos de comunicación.

Existen tres modos de comunicación:

- **Full-Duplex (FDX):** el lector y la etiqueta se comunican de forma simultánea
- **Half-Duplex (HDX):** el lector y la etiqueta tienen turnos para establecer su comunicación, de forma que esta no puede ser simultánea.
- **Secuencial (SEQ):** La etiqueta se alimenta intermitentemente. La comunicación entre la etiqueta y el lector se realiza en los espacios de tiempo en los cuales el lector no se comunica con la etiqueta.

2.5. FRECUENCIAS

Es uno de los elementos más determinantes a tener en cuenta a la hora de diseñar un sistema RFID y dependerá de la frecuencia del hardware y el desarrollo del software elegido.

Es importante tener en cuenta el efecto de blindaje y reflexión que se produce en los sistemas de radiofrecuencia, los cuales son producidos por los metales que se encuentran dentro del área de operación.

Los sistemas inductivos RFID operan en corta distancia, por lo que la influencia de sistemas cercanos o ruido es menor que en los sistemas que trabajen en la zona UHF o microondas, como veremos detalladamente en la clasificación posterior.

Se puede trabajar en cuatro frecuencias diferentes: alta frecuencia, baja frecuencia, microondas y ultra alta frecuencia. A continuación se detalla el funcionamiento y las características más importantes en cada uno de los funcionamientos, atendiendo a la capacidad de almacenamiento que ofrece cada sistema, a la velocidad y el tiempo de lectura de datos, al área de cobertura y a las aplicaciones posibles [10].

2.5.1. Baja frecuencia (125 KHz)

Para los sistemas RFID de baja frecuencia se suelen utilizar etiquetas pasivas, las cuales utilizan el funcionamiento inductivo para realizar la transmisión de datos. Son sistemas sencillos y no requieren demasiados requisitos regulatorios.

Al utilizar etiquetas pasivas, la cantidad de datos que se pueden almacenar es baja, típicamente unos 64 bits. No obstante, se pueden utilizar etiquetas pasivas que permiten un almacenamiento de hasta 2kbits.

La velocidad y el tiempo de lectura son bajos (entre 200bps y 1kbps), ya que la energía que se puede transmitir a la etiqueta a baja frecuencia no nos permite una transferencia de la información más rápida.

El campo magnético disminuye drásticamente con la distancia y con el tamaño de la antena, por lo que en el caso de sistemas de baja frecuencia con etiquetas pasivas, la cobertura es pequeña. Esto puede ser una característica ventajosa e importante para aplicaciones en las cuales la zona de cobertura sea limitada, como pueden ser controles de producción.

Se utilizan en aplicaciones que no requieran de una gran cantidad de datos y en las cuales la velocidad de transmisión de estos no es una característica fundamental. Se suelen instalar en sistemas de control de accesos, gestión de inventario, identificación de animales o soporte a varias partes de la producción. Siendo la aplicación más extendida la del control de accesos, cabe destacar que si fuera necesario una mayor seguridad o un área de cobertura superior, deberemos emplear tarjetas activas en lugar de las tarjetas pasivas típicamente utilizadas en los sistemas RFID de baja frecuencia.

2.5.2. Alta frecuencia (13,56 MHz)

Se trata de la frecuencia más utilizada para los sistemas RFID, puesto que ofrece una capacidad de almacenamiento de datos elevada y una velocidad de transmisión buena, lo que compensa el mayor consumo de potencia que se requiere respecto a las de baja frecuencia.

La mayoría de los sistemas que operan a alta frecuencia son pasivos, no necesitando incluir una batería en las etiquetas. Esto hace, como se explica anteriormente, que el sistema tenga mayor longevidad, que se puedan utilizar en entornos más desfavorables y que el coste de las etiquetas sea menor.

Los sistemas de alta frecuencia, tienen la zona de operación situada en el campo creado junto a la antena del lector, permitiendo que se alcancen unas distancias aproximadas al diámetro de la antena (siempre que sean sistemas de una sola antena).

A continuación se muestra una gráfica de la dependencia de la intensidad del campo respecto a la distancia a una antena de 0.8m de diámetro.

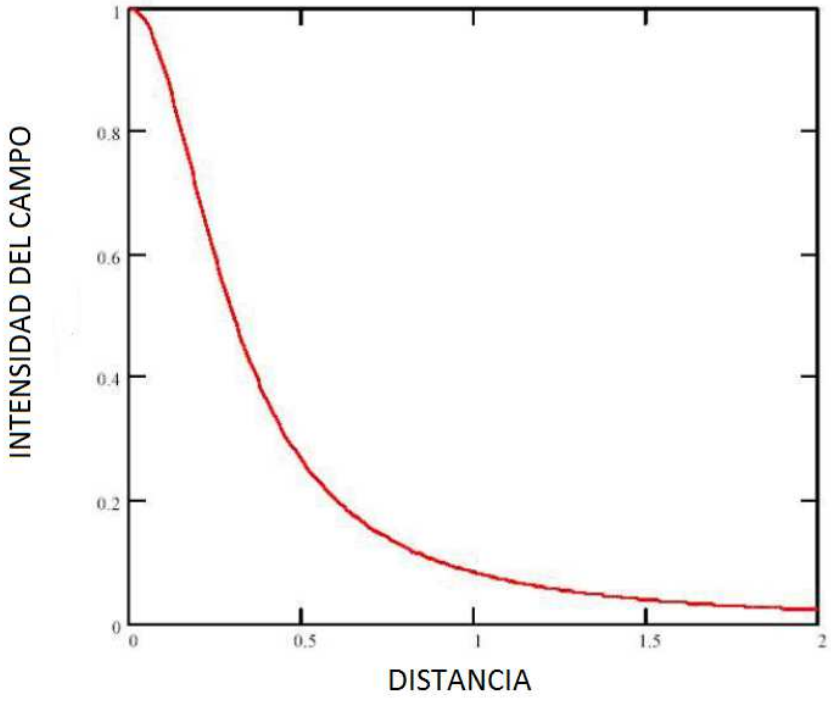


Figura 5. Disminución de la intensidad de campo respecto a la distancia de la antena

A diferencia que en otros sistemas RFID de mayor frecuencia, la radiación a 13,56 MHz no se absorbe por el agua ni la piel humana, lo que permite que las ondas emitidas se transmitan con mayor facilidad, evitando

interferencias de forma más efectiva que otros sistemas con funcionamiento a otras frecuencias.

Las etiquetas utilizadas para los sistemas de alta frecuencia son muy variadas en forma y tamaños en función de la aplicación a la que se desee ajustar. Una de las principales ventajas de esta tecnología a alta frecuencia consiste en que la antena necesita pocas vueltas en para obtener una etiqueta con un alto rendimiento, lo que conlleva un bajo coste de fabricación de estas. Las etiquetas más comunes son las ISO 14443 con un rango de 7 a 15 cm, las ISO 15693 con un rango superior a 1m, las etiquetas rígidas industriales y las etiquetas inteligentes, delgadas y flexibles para aplicaciones donde se necesiten estos requisitos.

Las etiquetas pasivas empleadas en esta frecuencia tienen típicamente desde 512 bits hasta 8kbits, divididos en bloques para direccionar los datos. Se trata de una capacidad mucho más elevada que en los sistemas de baja frecuencia, ya que permiten que la inductividad del campo genere una alimentación mucho más elevada a la etiqueta a la hora de realizar la transmisión de los datos.

25 Kbps es la velocidad típicamente utilizada en la alta frecuencia, la cual se verá reducida en el caso de introducir algoritmos de comprobación de bits que eviten errores en la transferencia de datos. No obstante, existen dispositivos que posibilitan tasas de transferencia mayores a 100Kbps.

Estos sistemas pueden leer unas 40 etiquetas por segundo, aunque podremos encontrar sistemas que puedan leer a mayor velocidad que estos funcionando en alta frecuencia.

Las etiquetas pasivas ofrecen un radio de cobertura típico de un metro aproximadamente

Estos sistemas son aptos para aplicaciones que no requieran de una cantidad elevada en la lectura de datos ni una distancia muy elevada. Se suelen emplear en las cadenas de suministros, gestión de maletas de aeropuertos o bibliotecas y servicios de alquiler.

2.5.3. Ultra alta frecuencia (UHF) (433MHz, 860MHz, 928 MHz)

Estos sistemas utilizan la propagación de las ondas electromagnéticas para la comunicación y la alimentación de las etiquetas (en caso de que estas sean pasivas), hecho que se diferencia de los sistemas de baja frecuencia, los cuales funcionan gracias a la inducción electromagnética (de forma parecida a los transformadores).

El lector envía una onda que se propaga con un frente de onda esférico y que permite que las etiquetas situadas en el interior del campo recojan la energía para su alimentación. La cantidad de energía de cada punto estará

determinada por la distancia del lector y la energía que este radia, la frecuencia de trabajo y el tamaño de la antena situada en la etiqueta.

La potencia legal que puede radiar un dispositivo RFID está limitada a 500mW en Europa, lo que implica un rango máximo de lectura de 0,7m en 860 MHz.

A esta frecuencia se disponen de etiquetas pasivas y activas que van desde los 32 bits hasta los 4kbits. Normalmente, se encuentran divididos en bloques de 128 bits para direccionar los datos.

Se encuentra en unos 28kbps típicamente, aunque existen aplicaciones que disponen de una mayor velocidad (con el mayor consumo correspondiente).

Con la velocidad citada, es posible leer unas 100 etiquetas por segundo, lo cual la convierte en una tecnología muy rápida y eficaz para aplicaciones donde la velocidad de lectura ha de ser rápida.

Las etiquetas pasivas de ultra alta frecuencia (UHF), alcanzan un radio de unos 4 metros, mientras que las etiquetas pasivas son capaces de llegar a unos 10 metros (en 433MHz).

La cobertura de este tipo de sistema viene altamente influenciada por las limitaciones que los distintos países imponen a la potencia permitida, siendo posible una cobertura mucho mayor en Estados Unidos, donde permiten hasta4W, frente a los 500mW que están permitidos en Europa.

Se utiliza en aplicaciones que requieran una alta velocidad de transmisión de los datos y una capacidad de almacenamiento superior a la baja y alta frecuencia. Se suele utilizar en trazabilidad, seguimiento de artículos y logística de cadenas de suministros.

2.5.4. Microondas (2,45GHz , 5,8GHz)

Estos sistemas se diferencian en alimentados pasivamente y alimentados activamente, según si la etiqueta lleva alimentación propia (el caso de los segundos) o no, ofreciendo una mayor cobertura y velocidad los sistemas alimentados activamente y una mayor sencillez y menor consumo los alimentados de forma pasiva.

Este tipo de sistemas se utilizan desde hace bastante tiempo, siendo las etiquetas inicialmente bastante complejas y caras de fabricar, debido a que debían procesar las señales con circuitos específicos basados en la tecnología CMOS. Actualmente, se fabrican con un único circuito integrado con alimentación pasiva, lo que se traduce en un mayor tiempo de vida y un coste muy inferior a las fabricadas en los inicios de esta tecnología.

A 2,45 GHz, se utiliza la propagación de las señales de radio para transmitir los datos y la energía necesaria para alimentar las etiquetas. A través de una antena situada en el lector se envía una señal, la cual es recibida a través de la antena situada en la etiqueta.

En el caso de etiquetas pasivas, se convierte la señal recibida en la etiqueta a corriente continua, de tal forma que se produzca la alimentación de la tag. La transmisión desde el lector hasta la etiqueta se realiza variando la amplitud, la fase o la frecuencia de la onda transmitida. La transmisión de retorno producida desde la etiqueta al lector se realizará variando la amplitud y/o la fase de la señal. Todo este proceso es el que se conoce como "modulated Backscatter".

A diferencia de los sistemas RFID que trabajan de forma inductiva, los sistemas de microondas (como los de UHF) tienen un campo de operación alejado de la antena, lo que permite una cobertura mucho mayor, alcanzando las etiquetas pasivas una distancia entre 0,5m y 12m y las etiquetas pasivas más de 30m, lo cual dependerá, de nuevo, de la regulación del país en el que nos situemos y por tanto la potencia máxima a la cual será permitida realizar la comunicación.

Las microondas se reflejan y se atenúan en el agua y en tejidos humanos, además de reflejarse en objetos metálicos, aunque los atraviesan con mayor facilidad, por lo que, al contrario que en los sistemas inductivos, es posible diseñar etiquetas que trabajen unidas a objetos metálicos.

Los sistemas de microondas disponen de etiquetas activas y pasivas, que pueden almacenar desde los 128 bits hasta los 512kbits, divididos en bloques para direccionar los datos.

La velocidad y el tiempo de lectura dependen de la etiqueta, aunque es elevada en todos los casos. La velocidad típica suele ser de unos 100Kbps (normalmente un poco inferior), llegando a alcanzar algunos dispositivos hasta 1Mbps.

Los sistemas con etiquetas pasivas alcanzan los 2 metros, mientras que los equipados con tags activas llegan hasta los 15 metros o más.

Se utilizan en aplicaciones que requieran de una amplia cobertura y una velocidad de transmisión de datos muy elevada. Son típicamente utilizados en automatización, peajes de carreteras, logística militar, etc.

2.6. CODIFICACIÓN

La transferencia de datos entre el lector y la etiqueta de la tecnología RFID requiere los siguientes bloques de funcionamiento:

- Del lector a la etiqueta:

- En el lector: codificación de señal y modulador.
- Un medio de transmisión
- En la etiqueta: demodulador y decodificador de canal.

Los sistemas de codificación de señal adquieren el mensaje en forma de señal y la adecua al canal de transmisión. Para ello, es necesario dotar a la información de protección contra interferencias y contra otras modificaciones malintencionadas de la señal.

- **Código NRZ (No return to zero):** en esta codificación se utiliza un '0' binario para representar las señales bajas y un '1' para las altas. Este tipo de codificación se utiliza para las modulaciones PSK o FSK.

- **Código unipolar RZ:** en esta codificación se utiliza un '1' binario para representar una señal alta durante la primera mitad del periodo de bit y un '0' binario para la señal baja que dura todo el periodo del bit.

- **Código DBP:** se representa con un '0' la transición en mitad del período de un bit y con un '1' lógico la ausencia de transición. El nivel de señal será invertido al inicio de cada periodo del bit.

- **Código Miller:** se representa con un '1' la transición en la mitad del período de un bit y con un '0' la continuidad del nivel de la señal hasta el período siguiente.

- **Código Manchester:** Un '1' representará una transición negativa en la mitad del período de un bit y un '0' representará la transición positiva.

- **Código pulso-pausa:** Un '1' representa una pausa de duración t antes del pulso siguiente, mientras que un '0' representará una pausa de duración 2t antes del próximo pulso.

- **Código diferencial:** Los '1' binarios transmiten que se ha de realizar un cambio en el nivel de la señal, mientras que un '0' indicará que el nivel de la misma ha de permanecer constante.

2.7. ESTÁNDARES Y REGULACIÓN

La tecnología RFID está ligada a una fuerte normativa internacional y al cumplimiento de unos requisitos básicos para su difusión y utilización como tecnología de identificación [12].

2.7.1. Estándares ISO

La entidad de normalización de este tipo de estándares es un JTC (Join Technical Committee) compuesto desde el ISO y el IEC, cuya denominación es ISO/IEC/JTC1. En este JTC, el comité 17 se encarga de abordar los temas relacionados con la trazabilidad de las personas y el comité 31 se encarga de lo relacionado con la trazabilidad de los objetos. En este último aspecto existe una subdivisión en cuatro grupos de trabajo: WG2 (Work Group on Data Structure), WG3 (Work Group on Conformance), WG4 (Work Group on RFID Item Management) y un WG5 que se encarga de la geolocalización en tiempo (RTLS).

2.7.1.1. ISO/IEC 18047

Son los estándares utilizados para establecer la interoperabilidad. El ISO ha creado las normas 18047, que se reparten como normas básicas, según frecuencia. Las normas ISO 18047 son las siguientes:

- **ISO/IEC 18047:** Information Technology – Automatic Identification and Data Capture Techniques – RFID Conformance Test Methods.

- **ISO/IEC 18047-2:** Parameters for Air Interface Communications below 135 KHz. Publicada en enero de 2006.

- **ISO/IEC 18047-3:** Parameters for Air Interface Communications at 13,56 MHz. Publicada en septiembre de 2004.

- **ISO/IEC 18047-4:** Parameters for Air Interface Communications at 2,45 GHz.

2.7.1.2. ISO/IEC 18000

ISO/IEC 18000: Information Technology - Automatic Identification and Data Capture Techniques – RFID for item Management – Air Interface.

Estos estándares proporcionan los valores para la interfaz de aire para los parámetros de una frecuencia particular y la interfaz de aire a partir de la cual se permite establecer el cumplimiento de la normativa. Las normas ISO/IEC 18000 son:

- **ISO/IEC 18000-1:** Generic Parameters for Air Interface – Communication for Globally Accepted Frequencies.

- **ISO/IEC 18000-2:** Parameters for Air Interface Communications below 135KHz.

- **ISO/IEC 18000-3:** Parameters for Air Interface Communications at 13,56 MHz.

- **ISO/IEC 18000-4:** Parameters for Air Interface Communications at 2,45GHz.

- **ISO/IEC 18000-6:** Parameters for Air Interface Communications at UHF (from 860 to 960 MHz).

- **ISO/IEC 18000-7:** Parameters for Air Interface Communications at 433 MHz.

2.7.1.3. ISO/IEC 159

Proporcionan coherencia entre las órdenes de lectura y la gestión de datos.

- **ISO/IEC 15961:** Information Technology – Automatic Identification and Data Capture Techniques – RFID for item Management – Host interrogator – Tag functional commands and other syntax features.

- **ISO/IEC 15692:** Information Technology – Automatic Identification and Data Capture Techniques – RFID for item Management – Data Syntax.

- **ISO/IEC 15963:** Information Technology – Automatic Identification and Data Capture Techniques – RFID for item Management – Unique identification of RF Tags and Registration Authority to manage the Uniqueness.

2.7.1.4. ISO/IEC 19762

Se encargar de regularizar el vocabulario utilizado en las normas ISO que se encargan de la tecnología RFID.

- **ISO/IEC 19762:** Information Technology – Automatic Identification and Data Capture Techniques – Harmonized Vocabulary.

- **ISO/IEC 19752-1:** General Terms Relating to AIDC.

- **ISO/IEC 19762-2:** Radio-Fréquency Identification (RFID).

2.7.1.5. ISO/IEC 18046

Permite a los lectores encontrar sistemas que atiendan a las necesidades de los clientes, sobre una base de prestaciones comprobadas.

Ello permitirá a los usuarios escoger entre las distintas soluciones propuestas, gracias a la interoperabilidad regulada por la norma 18000.

- **ISO/IEC 18046:** Information Technology – Automatic Identificanton and Data Capture Techniques – RFID Performance Test Methods.

2.7.1.6. ISO/IEC 27429

Se utiliza como guía para la puesta en marcha de los sistemas RFID. Tiene por objetivo el informar al usuario de la consideración del mundo en el que van a funcionar y de las características que han de cumplir para hacer frente a las posibles adversidades del medio.

- **ISO/IEC 24729:** Information Technology - Automatic Identificanton and Data Capture Techniques – Radio Frequency for item Management – Implementation Guidelines.

- **ISO/IEC 24829-1:** RFID – Disponibilidad de tarjetas.

- **ISO/IEC 24729-2:** Reciclaje de etiquetas.

- **ISO/IEC 24729-3:** RFID Instalación del interrogador de la antena.

2.7.1.7. Otros estándares ISO/IEC

- **ISO/IEC 10536 Identification cards – Contactless integrated circuit cards:** Está redactada para tarjetas de identificación inteligentes por encima de los 13,56 MHz y describe las características físicas, área de interrogación, señales electrónicas y procedimientos de reset.

- **ISO/IEC 14443 Identification Cards – Proximity integrated circuit cards:** Está redactada para tarjetas de identificación inteligentes con un rango de más de un metro y a 13,56MHz. Define las características físicas, interfaz de área, inicialización, anticolisión y protocolo de transmisión.

- **ISO/IEC 15693 Contactless integrated circuit cards – vicinity cards**: Define las características físicas, protocolos de transmisión y anticolisión y la interfaz de aérea para tarjetas sin contacto a 13,56 MHz.

- **ISO/IEC 19762 Harmonized vocabulary – Part 3:** radio-frequency identification: define términos generales y definiciones en la identificación automática y las técnicas de captura de datos.

- **ISO/IEC 18001 RFID for Item Management – Application Requirementes Profiles:** ofrece el resultado de estudios para identificar aplicaciones y usos del RFID.

2.7.2. Trazabilidad de personas

Es llevada por el subcomité internacional ISO/IEC/JTC1/SC17, existiendo dos normas relacionadas con las tarjetas sin contacto: la norma 14443 para lecturas de "vecindad" (a pocos milímetros) y la norma 15693 para lecturas de proximidad (a unos pocos centímetros). Ambas tecnologías utilizarán la frecuencia alta (13,56 MHz) y poseerán las etiquetas de formato estándar.

2.7.3. Trazabilidad de objetos

Es llevada internacionalmente por el subcomité ISO/IEC/JTC1/SC31.

Presentan soluciones a los problemas de interoperabilidad gracias a las normas 18000, además de la utilización de un protocolo común para lector y etiqueta (determinado en las normas 18000) y una organización única en la estructura de los datos que contiene la tag. Por tanto las normas 18000 se encargarán de posibilitar la interoperabilidad.

2.7.4. EPCGlobal Network

El EPC [13] (Electronic Product Code), tiene su origen en un consorcio formado por EAN (European Article International) está presente en 140 países y actualmente es denominado como GS1 US.

La creación del EPCGlobal se realizó para motivar la promoción de la EPCglobal Network, tecnología que se basaba en cambiar la cadena de suministro actual por otra con un estándar abierto y global, que hiciera posible la identificación de los productos en tiempo real en cualquier parte del mundo.

EPCglobal Network fue creada por el Auto-Id Center, que se trata de un grupo de investigación del MIT (Massachussets Institute of Technology), y fue implantado y desarrollado en más de 1000 compañías en todo el mundo.

Hoy en día, los estándar que forman EPCglobal, son estudiados por la ISO, con la única premisa de que los estándares creados por ISO sean comprobados y aceptados en los que EPCglobal crea.

El EPC [14] proporciona especificaciones para etiquetas, en relación a los datos almacenados, a la parte RF que permite la comunicación y a los protocolos de comunicación con el interrogador. Así, también proporciona especificaciones para los lectores de tarjeta, para el protocolo de la interfaz aire y para las comunicaciones que estos realizan con las etiquetas.

El estándar EPC establece una división en seis tipos de etiquetas diferentes, según la funcionalidad de estas:

- **CLASE V:** lectores. Alimentan a las etiquetas de clase I,II y III y se comunican con las etiquetas de clase IV.

- **CLASE IV:** etiquetas activas. Se pueden comunicar con otras etiquetas activas y con lectores.

- **CLASE III:** etiquetas semi-activas. Soportan comunicaciones Broadband.

- **CLASE II:** etiquetas pasivas que incluyen funciones adicionales como memoria o encriptación de datos.

- **CLASE I/ CLASE 0:** etiquetas activas de solo lectura.

La última versión del EPC data de enero de 2005 y se conoce como EPC Generación 2.

Una vez expuestos los orígenes y las clasificaciones que establece el EPC, es conveniente centrarse con mayor exactitud en lo que consiste.

El EPC es un formato que expone capacidades físicas a través de un número o código de identificación unívoco y único para los productos. Dicho código consta en todo caso de las siguientes partes:

- **Encabezado:** indica la versión numérica del código.

- **Administrador EPC:** indica la empresa que es responsable de mantener la categoría del objeto y el código. Se asigna el administrador general a una entidad, asegurándose de que los códigos sean únicos.

- **Categoría de objeto:** indica el tipo exacto del producto. Es utilizado por una entidad de gestión EPC para identificar objetos de mercado. Estos códigos han de ser únicos dentro del número del administrador general.

- **Número serial:** representa el identificador único del objeto dentro de la categoría del mismo.

Figura 6. Formato de etiqueta EPC [13]

La EPCGlobal Network tiene como objetivo la captura y compartición de la información dentro de la red. Para ello los objetos se identifican con etiquetas sencillas y de un coste bajo las cuales indican su número de identificación. Los agentes de identificación situados en las redes (lectores y antenas), recogen los códigos necesarios, los cuales serán posteriormente procesados por un Middleware y suministrados a los sistemas de información EPC. Para obtener información sobre un EPC específico, se debe acudir a los servicios de información centralizados, los cuales disponen de servicios de nombres de objetos con toda la información necesaria para los sistemas.

Para las comunicaciones del lector a la etiqueta, han de usarse modulaciones de doble banda lateral ASK o de reverso de fase ASK, con codificación de pulsointervalo . El lector, por tanto, debe esperar una respuesta de backscatter.

En la comunicación realizada entre la etiqueta y el lector, la primera deberá utilizar una señal no modulada codificada en formato FM0 o código Miller, siendo el método Half Dúplex (el lector y la etiqueta transmitirán en su turno y nunca simultáneamente) el utilizado en ambos casos.

2.8. VENTAJAS E INCONVENIENTES DE LOS SISTEMAS RFID

Debido a la infinidad de posibilidades que presentan la tecnología RFID [15] [16], en la que se pueden utilizar una cantidad considerable de datos, se ha conseguido instalar entre uno de los sistemas más utilizados en numerosas aplicaciones empresariales, y cada vez más es común su aplicación.

Los sistemas RFID presentan características que los hacen una alternativa mejor que los sistemas tradicionales de identificación, como el código de barras .

Las principales ventajas que presenta la tecnología RFID son las siguientes:

- Ofrece la posibilidad de entablar una combinación estable y eficaz entre los sistemas RFID e internet.

- Es, actualmente, la forma más precisa e inmediata de identificar y localizar de forma automática cualquier tipo de producto.

- Las etiquetas RFID permiten obtener una lectura a mayor velocidad y precisión que los sistemas tradicionales como el código de barras.

- Disminuye los niveles en el inventario y las posibles roturas en el stock.

- Ofrecen la posibilidad de clasificar cada objeto a nivel individual y no colectivo como otra tecnología más tradicional.

- Permite múltiples lecturas y la posible reprogramación de algunos tipos de etiquetas.

- No se necesita contacto visual a la hora de realizar la lectura de los datos.

- Presentan sistemas de protección más elevados que el tradicional código de barras.

- Poseen mayor durabilidad que otros sistemas.

Como se puede observar, existen numerosas ventajas para aplicar el uso de sistemas RFID, no obstante, como todo sistema, además de las citadas ventajas posee inconvenientes. Entre dichos inconvenientes se encuentran los siguientes:

- Interpretación incorrecta de los datos. Lo que conlleva a un mal uso de ellos en los procedimientos posteriores.

- Se generan colisiones e interferencias. En algunas frecuencias de forma más común que en otras.

- Alto coste y periodicidad de las etiquetas activas, las cuales podrían suponer una solución al problema citado de las interferencias, pero se descarta su utilización para productos de bajo coste debido a su elevado precio.

- Posibilidad de generarse interferencias con otros elementos que utilicen el espectro.

- Posibilidad de reflejo de la señal en materiales metálicos y absorción por parte de la piel humana, lo que puede producir atenuación de la señal o la incorrecta recepción de esta.

La tecnología RFID se presenta, por tanto, como una de las punteras en identificación, a pesar de que se debe tener en cuenta las limitaciones que esta conlleva y la forma de subsanarlas e implementarlo de la mejor manera posible.

2.9. APLICACIONES DE LOS SISTEMAS RFID

Hasta ahora se han ido comentando diferentes aplicaciones en las que se utiliza la tecnología RFID. A continuación se hace una recopilación de todas ellas [17] [18]:

- **Transporte y distribución**: seguimiento de objetos, aeronaves, vehículos, ferrocarril, contenedores, sistemas de localización en tiempo real, etc.

- **Empaquetado de artículos:** cadenas de suministro, seguimiento de palés y cajas, industria farmacéutica, inventarios, etc.

- **Industria:** Flujos de trabajo y estampación, etc.

- **Control de accesos y** seguridad: pasaportes y visados, DNI, seguimiento de animales, personas o cosas, seguimiento de equipaje, prevención de falsificaciones, accesos privados, identificación de empleados, peajes, pagos automáticos, etc.

- **Sistemas de biblioteca**

- **Monitorización de aplicaciones**

Sólo son algunas de las aplicaciones que hoy en día tiene la tecnología RFID, aunque realmente son innumerables y muy variadas. A continuación se detallan más las aplicaciones más comunes y conocidas de esta tecnología.

2.9.1. Control de accesos

Es una de las aplicaciones más numerosas en cuanto al uso del RFID. Para controlar el acceso a los recintos, se disponen de tarjetas (las cuales cada vez son más funcionales) que permiten el acceso al recinto o no, según si se cumple con las especificaciones establecidas y programadas en el sistema de control RFID. No solo se utilizan para controlar el acceso a recintos, sino que también controlan los permisos para pequeños pagos o máquinas expendedoras, por ejemplo.

2.9.2. Identificación de equipajes

Permiten la sustitución de personal a través del direccionamiento mediante sensores del equipaje a través de toda la cadena. Además, facilita la identificación ante posibles pérdidas.

Presentan claras ventajas ante los sistemas de códigos de barras (que son los más utilizados actualmente), como encajar con los sistemas de seguridad del aeropuerto, incorporar más información que permita mejor identificación en caso de pérdida y evita la continua comunicación con la base de datos, puesto que la información está alojada en la etiqueta.

Estos sistemas trabajan a 13,56 MHz y fueron los aeropuertos de Manchester y Munich en 1999 los pioneros en instalarlo.

2.9.3. Comercio a distancia

Permiten establecer la seguridad necesaria para realizar pago con ellos. Se utilizan para realizar pequeños pagos como gasolina, máquinas expendedoras o artículos de tiendas. El pago se realiza mediante teléfono móvil o con llaves especializadas para ello.

En cuanto a las empresas, permite la recogida de tendencia y gustos por parte de los clientes, por lo que supone una ventaja para vendedores y para compradores, haciendo más cómodo, rápido y seguro el pago para estos últimos.

2.9.4. Automóviles

En el inicio de la década de los 90, aparecieron los primeros sistemas RFID para automóviles, permitiendo la inmovilización de estos como medio de seguridad ante posibles robos. En la llave se instalaba el trasponedor, cada uno de los cuales disponía de un código único y fijo. Cuando el conductor giraba la llave, se generaban ondas electromagnéticas que permitían la verificación del código y el consiguiente arranque o no del vehículo.

Otra aplicación RFID aplicada en automóviles consiste en el uso de una tarjeta identificadora que permite que el vehículo se abra automáticamente sin necesidad de llave. Cuando el propietario se acerca a la distancia requerida del coche, el lector detectará la etiqueta y el coche y desbloqueará las puertas del coche.

2.10 SEGURIDAD EN RFID

Los sistemas RFID necesitan de la utilización de sistemas de seguridad para protegerlos antes posibles ataques [19], ya que estos sistemas contienen información relevante que sería una gran pérdida en algunos casos como pueden ser aplicaciones de alta seguridad o control de accesos.

Todo sistema actual necesita ser protegido, los ordenadores ante posibles ataques informáticos y pirateos, en internet de los posibles virus, etc. Así mismo los sistemas RFID necesitan ser protegidos para evitar que la información contenida sea dañada o utilizada de forma malintencionada. A continuación se nombrarán y describirán los posibles ataques y amenazas conocidos a los sistemas RFID:

Los siguientes son ataques realizados sobre las comunicaciones en radiofrecuencia:

- Spoofing: esta amenaza consiste en suministrar información errónea que aparentemente es válida, de tal forma que el sistema acepte los datos.

- Inserción: inserta comandos al sistema donde realmente deberían ir datos, de tal forma que, por ejemplo, se pueden introducir comandos SQL donde deberían estar alojados códigos EPC.

- Replay: se intercepta la señal del sistema RFID y se graban los datos. Posteriormente se suelen utilizar para transmitírselos al sistema y que los considere como válidos.

- Denegación de servicios (DOS): consiste en el colapso del sistema al introducirle una gran cantidad de datos o se anula la comunicación insertando una gran cantidad de ruido. A este último ataque se le denomina como RF jamming.

Además, existen ataques cuyo objetivo son las etiquetas. A través de técnicas malintencionadas, es posible modificar la información que estas contienen, de tal forma que si por ejemplo el contenido fuera un precio, este podría rebajarse. Uno de los sistemas más conocidos para realizar este tipo de ataques es el RF Dump, cuyo funcionamiento se realiza a través de Java, Linux o Windows y que permite cambiar los datos y volver a escribirlos en una etiqueta gracias a la instalación de un lector en el PC. El lector leerá los datos de la etiqueta y los presentará en una hoja de cálculo, donde pueden ser modificados y volver a ser escritos en la etiqueta con los valores deseados.

Existen otro tipo de ataques como los ataques man in the middle (MIM), a través del cual se suplanta una de las identidades de los sistemas que componen la red, fraudes por modificación de chips, inutilización de las etiquetas sometiéndolas a fuertes campos electromagnéticos y, cada vez más, métodos más sofisticados que consiguen burlar la defensa de estos sistemas son desarrollados.

Por ello, los sistemas RFID han de proponer e implantar soluciones para protegerse ante estos posibles ataques. Uno de los métodos más antiguos y actualmente utilizados es la encriptación, en la cual, a través de claves y códigos implantados se pueden “camuflar” los datos, de tal forma que no puedan ser descifrados en caso de ataque.

Además, se han de implementar algoritmos para prevenir que esta clave sea descubierta, por lo que deberemos instalar modos de defensa ante las lecturas no autorizadas del contenedor de la información para evitar copias o réplicas, ante la colocación de portadoras de información extraña en la zona de lectura del lector con el fin de obtener accesos, ante escuchas de las

conversaciones de radio y la recolocación de los datos imitando los originales, ante lecturas y escrituras indeseadas, ante la existencia de etiquetas falsas que puedan traspasar la seguridad del sistema o ante escuchas ilegales que tienen como objetivo la falsificación de etiquetas.

Por tanto, a la hora de elegir la implantación de un sistema RFID u otro, es de suma importancia estudiar los posibles ataques a los que se va a someter y decidir qué tipo de seguridad instalaremos en la aplicación, ya que la pérdida, copia o modificación de los datos puede ser de extrema gravedad según qué aplicaciones.

2.10.1. Protocolos anticolisión

Los lectores [20] tienen la opción de leer un conjunto de tags o de seleccionar uno de entre todos ellos. En la primera opción, es donde se produce lo que se conoce como problema de las colisiones de acceso al medio, el cual aparece en el momento en que diferentes tags intentan diferenciarse en el mismo tiempo, por lo que se requerirán protocolos anticolisión. A continuación se realizará una clasificación de estos protocolos:

- **Protocolos deterministas:** se emplea lo que se conoce como algoritmos de búsqueda en árbol. Estos algoritmos trabajan descomponiendo las tags bajo la cobertura del lector en grupos reducidos, a través de técnicas de segmentación. El lector enviará un bit, de tal forma que todas las etiquetas cuya identificación comiencen por dicho número son las que responderán al bit enviado. Si en este momento se detectan colisiones, el lector realizará la segmentación de la población de las etiquetas que anteriormente respondieron a dicho bit. De esta forma se procederá repetidas veces hasta que el lector no detecte ninguna colisión, finalizando el proceso de lectura de un subgrupo y continuando por el resto si fuera necesario.

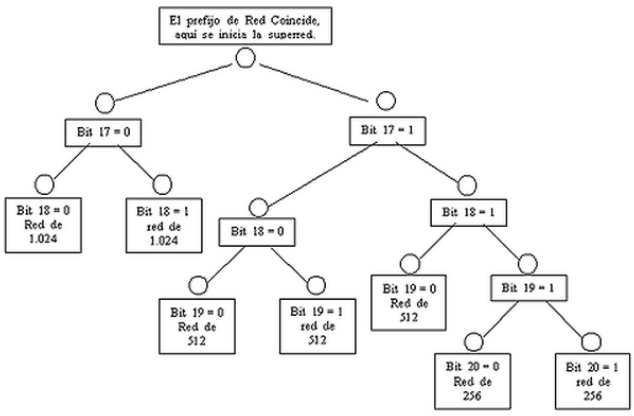


Figura 7. Protocolo de búsqueda en árbol [21]

- **Protocolos probabilísticos:** se basan en el azar, de forma que utilizando la estadística obtienen las soluciones deseadas. Este tipo de protocolos es muy utilizado en RFID, puesto que en muchas aplicaciones se desconoce el número de etiquetas que operarán con el lector. El protocolo aloha es el principal y existen otro tipo de protocolos los cuales están basados en él. El protocolo aloha funciona de tal manera que en el momento que la etiqueta entra en comunicación con el lector, esta debe enviar su código de identificación. Si la respuesta de dos o más etiquetas se solapa en el mismo tiempo, la lectura de la información no se realizará adecuadamente (se estaría produciendo una colisión) y esta sería perdida.

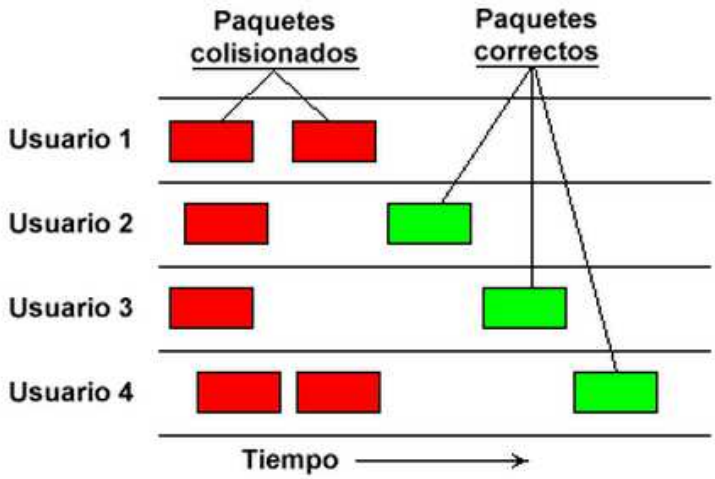


Figura 8. Flujo de paquetes en un canal ALOHA [22]

CAPÍTULO 3

METODOLOGÍA

Este capítulo tiene como objetivo el describir el modo en el que se ha trabajado en el presente proyecto, así como las herramientas utilizadas.

3.1. MODO DE TRABAJO

Para realizar este proyecto fin de grado se realizaron varios pasos y bloques diferenciados. Inicialmente se realizó el estudio del cifrador de flujo a implementar, de tal forma que considerando el tamaño que pudiera ocupar y considerando el nivel de seguridad que ofrecía, se determinó como válido para implementarlo en un sistema RFID.

Posteriormente, se realizó la implementación del protocolo seleccionado en un lenguaje de descripción hardware, en este caso VHDL. En este paso se determinaron tres tipos de arquitecturas sobre el mismo protocolo, con el fin de comparar el área ocupada, así como el throughput y el consumo que requeriría el sistema.

Una vez realizadas las tres arquitecturas, se procede a realizar el mismo protocolo para un número de bits mayor, con el propósito de observar las diferencias en los mismos términos que lo anteriormente dispuesto, es decir: área ocupada, velocidad del proceso y consumo.

Obtenidas todas las arquitecturas, se procede a la simulación de estas a través de Modelsim, de tal forma que se comprueba que el algoritmo funciona de la forma deseada. Gracias a la implementación en lenguaje C del mismo algoritmo, se pudo corroborar el correcto funcionamiento del protocolo en VHDL.

Terminada la fase en la cual los circuitos han sido diseñados y probados, se comprueba mediante métodos matemáticos que las salidas de los sistemas son las correctas. Para ello se realizó el test “chi-cuadrado”, a través de Matlab, el cual permite establecer si la salida se comporta como una distribución binomial, lo cual establece que no existe ninguna correlación entre la clave y la salida, indicando así que desencriptar la clave es una tarea complicada en caso de ataque al sistema. Además del test anteriormente mencionado, se utilizó el test de NIST para comprobar que la salida es aleatoria, lo que indica que la salida del sistema es difícil de averiguar en base a la clave y por tanto tiene una encriptación con un nivel de seguridad elevado.

Comprobado completamente el correcto funcionamiento del circuito y la validez de las salidas, se sintetizó uno a uno los diseños en la herramienta de síntesis synopsys. Esta herramienta permite determinar los parámetros de área y el consumo de cada arquitectura, de tal forma que podremos realizar la comparativa para establecer la mejor de ellas.

3.2. FLUJO DE TRABAJO

En la figura 7 se muestra un esquema que aclarará la forma y los pasos seguidos en la realización del proyecto:

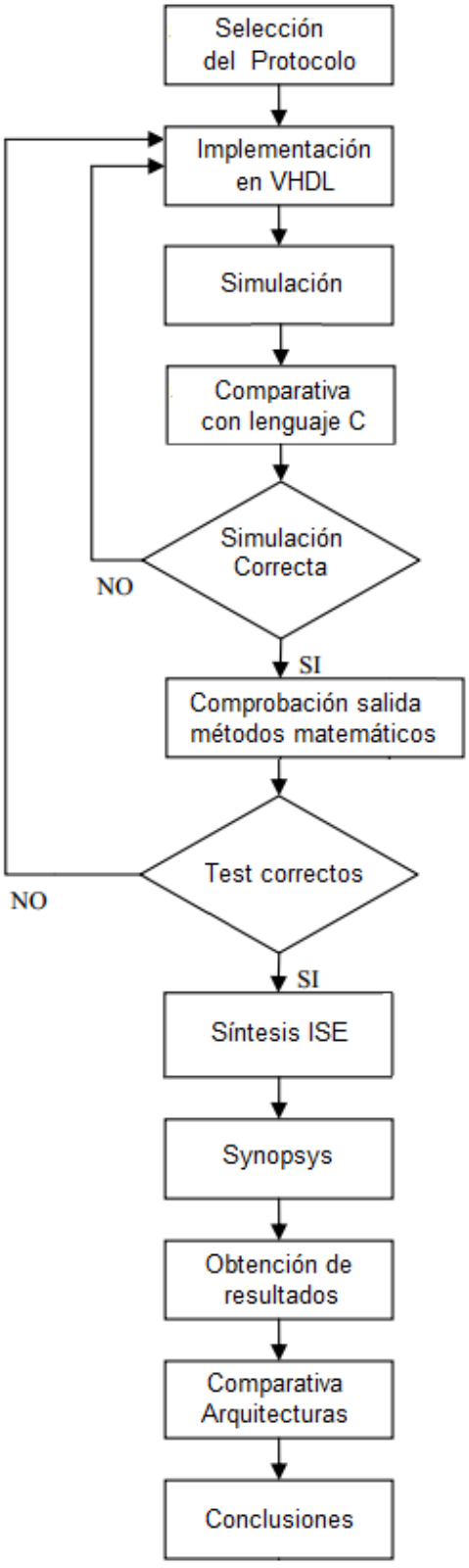


Figura 9. Flujograma de la implementación de las arquitecturas

3.3. VHDL

El lenguaje VHDL (cuyas siglas vienen de Very High Speed Integrated Circuit y Hardware Description Language), es un lenguaje estandarizado que permite describir circuitos con un nivel de abstracción deseado y posteriormente usado para implementar circuitos digitales. Dicho lenguaje puede ser posteriormente implementado en una FPGA, ASIC o PDL, entre otros.

[23][21] Este lenguaje es usado, por tanto, para modelar la arquitectura y el comportamiento de sistemas electrónicos discretos y requieren de un simulador lógico para reproducir el comportamiento del sistema modelado y verificar el diseño del sistema. Estas herramientas software que permiten el paso de las descripciones en VHDL a nivel de puertas se denominan herramientas de síntesis.

[25] El VHDL nace por iniciativa del departamento de defensa de EEUU en el comienzo de los 80. Anteriormente los circuitos se realizaban “a mano”, existiendo únicamente algunas herramientas de simulación muy básicas. Esto hacía que los diseños que era posible implementar no fueran de gran complejidad, ya que no podía llegarse al nivel del número de puertas lógicas que es posible alcanzar con el lenguaje VHDL.

En julio de 1983, las compañías Texas Instruments, IBM e Internetics, recibieron el encargo de desarrollar dicho lenguaje de descripción hardware, del cual nace su primera versión en agosto de 1985, llegándose a convertir en un estándar del IEEE en 1986.

A partir de entonces, con el estándar IEEE 1164, el lenguaje VHDL obtiene nuevas características y mejoras.

El estándar 1076 (el estándar que rige el lenguaje VHDL) ha de ser revisado cada 5 años.

3.4. HERRAMIENTAS UTILIZADAS

En este proyecto fin de grado se ha utilizado la herramienta de ISE de Xilinx para la realización del lenguaje de descripción hardware, lenguaje de programación C para corroborar el correcto funcionamiento del lenguaje VHDL descrito, ModelSim para la simulación del código de descripción hardware, VMwarePlayer de VMware Inc. para el uso de la máquina virtual del sistema operativo Red Hat Linux Fedora con el fin de utilizar el programa synopsys para la sintetización de las arquitecturas, y Matlab para la realización del test de Nist.

El programa ISE de Xilinx [26] [27] [28] [29] se utilizó para realizar el código de descripción hardware en VHDL y para su posterior simulación mediante el diseño de un banco de pruebas que permite introducir los valores deseados de cada señal perteneciente al circuito y de las entradas. El ISE de Xilinx permite el diseño en VHDL y su implementación en lógica programable.

Después de implementar cada circuito, se pudo realizar la simulación gracias a la herramienta ModelSim[30], el cual permite las simulaciones de circuitos digitales.

Synopsys [31] [32] fue utilizado una vez se tenían los circuitos comprobados y estudiados, de tal forma que permite sacar los términos de área ocupada y potencia consumida con el fin de establecer una comparativa entre las diferentes arquitecturas.

VMware Player fue utilizado para el uso de la máquina virtual con sistema operativo Red Hat Linux Fedora, con el fin de poder utilizar el programa Synopsys anteriormente descrito.

CAPÍTULO 4

CIFRADORES DESARROLLADOS Y ARQUITECTURAS

En rfid es necesario la inclusión de un sistema de seguridad que evite los peligros que se explicaron en apartados anteriores. Los sistemas de autenticación son los más utilizados y permiten establecer una seguridad elevada evitando la intrusión de terceros componentes en una red rfid entre lector y etiqueta.

Uno de los algoritmos más empleados para los sistemas de autenticación son los cifrador de flujo (los cuales se explicaron en capítulos anteriores), lo que fue una característica determinante para su elección a la hora de diseñar un protocolo de autenticación para este proyecto.

Existen numerosos protocolos de autenticación que emplean cifradores de flujo para garantizar la seguridad de los sistemas. A continuación se proponen algunos ejemplos [33]:

- **Protocolo MPPE [34] (Microsoft Point-to-Point Encryption**): en este protocolo se realiza el cifrado de flujo a través del algoritmo RC4 (uno de los algoritmos más importantes en cifradores de flujo). 128 bits de semilla son tomados de la aplicación SHA1 sobre el hash (Windows NT) de la contraseña de usuario y se generan 64 bits durante la negociación MS-CHAP. Finalmente, cada 256 paquetes PPTP, se recalcula una nueva semilla a través de la clave antigua y la clave original.

- **WEP:** basa su algoritmo en el cifrado de flujo RC4 y una clave pre-compartida. El esquema original (WEP-40) para generar la clave de flujo RC4 (de 64 bits) utiliza una clave PSK de 40 bits que se concatenará con una cadena de 24 bits que suponen el vector de identificación de la red. En el año 2000, se empezó a utilizar una clave de 128 bits (WEP-104). Este sistema es inseguro, ya que presenta problemas en el vector de inicialización y la obtención de la clave de flujo, a pesar de tratar de aumentar su fiabilidad con 128 bits, el resultado no fue satisfactorio.

- **WPA (Wi-Fi Protected Acces):** es un protocolo más avanzado que el WEP. Comenzó a utilizarse en 2003 y se basaba en el algoritmo RC4 con PSK aunque utiliza TKIP (Temporal Key Integrity protocol) para mejorar la seguridad. La TKIP controla la integridad de los paquetes, la contabilidad de estos y utiliza una función para obtener la clave de RC4 mezclando la clave de usuario con el vector de utilización de la red (en lugar de realizar una concatenación como el WEP).

En este capítulo se presentan las diferentes arquitecturas desarrolladas para el cifrador de flujo elegido, con el fin de mostrar las particularidades de cada una de ellas y las diferencias que presentan, con el fin de, posteriormente, determinar qué arquitectura resulta la más adecuada para el protocolo de autenticación de un sistema RFID.

4.1. PROTOCOLOS DE AUTENTICACIÓN

Son [35] protocolos destinados a la corroboración de la identidad de una parte del sistema a la otra, evitando de esta forma la suplantación de la identidad con fines malintencionados. A continuación se exponen una clasificación de ellos:

- **Full-fledged:** están basados en la criptografía clásica, algunos ejemplos son el cifrado simétrico o la clave pública.
- **Simples:** basados en la generación de números aleatorios y funciones hash
- **Ligeros:** basados en la generación de números aleatorios, utilizan funciones sencillas para la encriptación.
- **Ultraligeros:** realizan operaciones sencillas bit a bit como XOR, AND, OR, etc.

4.1.1. Cifradores de flujo

Los cifradores de flujo son algoritmos de cifrado que pueden realizar el cifrado incrementalmente, convirtiendo un texto plano en un texto cifrado bit a bit. Esto se logra gracias a la construcción de un generador de flujo clave.

Un flujo clave es una secuencia de bits de tamaño aleatoria que se emplean para ocultar los contenidos de un flujo de datos combinando el flujo de clave con el flujo de datos a través de una operación XOR.

A diferencia del cifrado por bloques, el cual utiliza bloques de bits de información de tamaño constante, el cifrado de flujo o cifrador de flujo utiliza un

generador de clave de flujo para realizar operaciones con cada bit o caracter de texto plano.

Los cifradores de flujo están formados [36] por un generador de claves el cual, a partir de una clave de inicialización K, produce una secuencia de bits con la misma longitud que el mensaje, la cual es empleada como clave en el proceso cifrado/descifrado. El emisor y el receptor han de contar con un generador de claves que produzcan estas de idéntica en ambos extremos de la comunicación.

Además del generador de claves, los cifradores de flujo han de contar con un algoritmo de cifrado, el cual realiza operaciones elemento a elemento en base a la clave generada por el generador de claves.

Los cifradores de flujo pueden ser clasificados en dos tipos:

- **Cifrado en flujo síncrono:** la secuencia pseudoaleatoria es completamente independiente del mensaje. Para realizar el descifrado de forma adecuada, el lector y el receptor han de tener señales de sincronización, lo cual permite que en caso de producirse un ataque mediante inserción de mensajes erróneos, este sea detectado al ser la sincronía interrumpida.

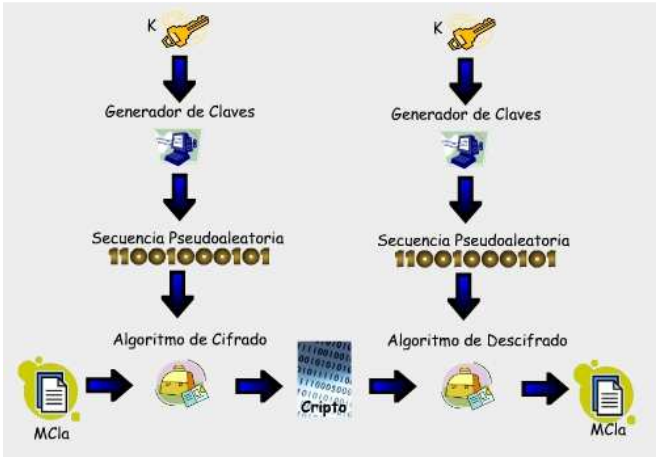


Figura 10. Esquema de cifrado en flujo síncrono [36]

- **Cifrado en flujo autosincronizante**: la secuencia pseudoaleatoria se encuentra en función del mensaje, de tal forma que no se necesite sincronía entre el emisor y el receptor, ya que en caso de pérdida de sincronía, se recuperará a través de retroalimentación. Este tipo de cifradores presentan la desventaja de que son susceptibles ante posibles ataques de inserción de mensajes erróneos en la red, lo cual ha de ser evitado mediante la emisión de mensajes de identificación de mensaje.

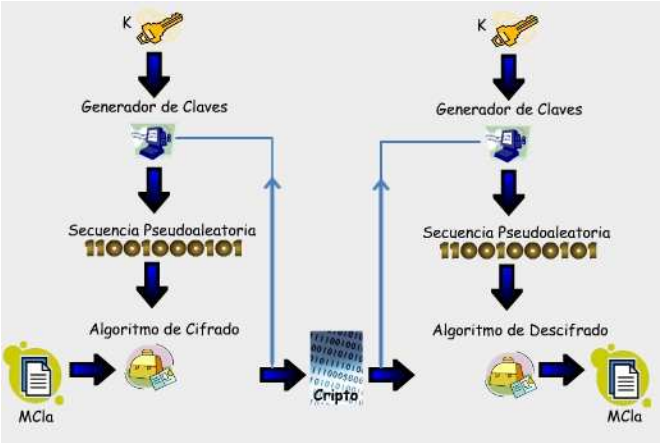


Figura 11. Esquema de cifrado en flujo autosincronizante [36]

Los cifradores de flujo tienen los siguientes componentes:

- Una cifra basada en una XOR.

- Una secuencia binaria y aleatoria S que se obtiene a través de una clave secreta compartida por emisor y receptor.

- Algoritmo de descifrado a través de una XOR.

4.1.2. Cifrado continuo utilizando LFSRs

Para realizar [37] los cifrador de flujo se suelen emplear varios LFSRs. Se emplea una secuencia aleatoria que consiste en una combinación de los bits de salida de cada LFSRs. Existen numerosas posibilidades, cada una con su complejidad y su fiabilidad. La implementación de este tipo de cifrado se realiza, generalmente, con hardware.

Los algoritmos más populares para realizar cifrador de flujo son los siguientes:

- **A5:** este algoritmo es utilizado en Global System Mobile (GSM). Se utiliza para cifrar la conexión entre la estación base y el teléfono.

- **RC4:** consiste en un algoritmo de cifrado continuo con clave de longitud variable (fue ideado por Ron Rivest).

- **SEAL:** algoritmo de cifrado continuo. Se realiza a través de software y resulta muy eficiente (diseñado por Phil Rogaway y Don Coppersmith).

- **WAKE (Word Auto Key Encription):** desarrollado por David Wheeler, su principal característica radica en que es un algoritmo que ofrece una gran velocidad de procesamiento.

- **Otros algoritmos:** PKZIP, Algoritmo M., Gifford, RAMBUTAN

4.3. CIFRADOR DE FLUJO UTILIZADO

En el presente proyecto se ha desarrollado un cifrador de flujo basado en el algoritmo RC4 y acondicionado para cumplir las especificaciones RFID, con el objetivo de que sea eficaz frente a posibles ataques de un sistema RFID.

RC4 es un algoritmo que posee dos algoritmos integrados: Key Scheduling Algorithm (KSA) y Pseudo-Random Generation Algorithm (PRGA). Ambos de estos algoritmos utilizan una s-box que consiste en un array de números únicos cuyo valor van desde 0 hasta 255 (ya que 256 es el número de bits más habitual). Cada número desde 0 hasta 255 está incluido en cada posición del array y el KSA se encargará de hacer la primera conmutación de valores. Primero el s-box es llenado con valores secuenciales desde 0 hasta 255; dicho array se denomina S. Entonces, otro array de 256 bits es llenado con el valor "semilla", repitiendo el proceso el número de veces necesario hasta que sea completamente llenado (este array se denomina K). Una vez hecho esto, el array S es conmuntado utilizando el siguiente pseudocódigo:

```
j=0;
for i = 0 to 255
{
   j = (j+S[i] + K[i]) mod 256;
   intercambia S[i] and S[j];
}
```

Una vez terminada esta primera parte, la s-box es intercambiada basándose en el valor "semilla", siendo esta la Key programada para el algoritmo.

La segunda parte es en la que se utiliza el PRGA. Dicho algoritmo dispone de dos contadores, i y j, los cuales son iniciados a 0 para comenzar las operaciones oportunas. Posteriormente, cada bit del keystream data se utiliza en el pseudocódigo siguiente:

```
i = (i + 1) mod 256;
j = (j + S[i]) mod 256;
intercambia S[i] and S[j];
t = (S[i] + S[j]) mod 256;
Exponer valor de S[t];
```

El cifrador de flujo desarrollado en este proyecto es una versión modificada con el fin de poderse implementar en un sistema RFID y utilizararse dentro de un protocolo de autenticación. En todo momento se han de tener en cuenta las limitaciones que presentan los sitemas RFID, puesto que esta tecnología posee unos recursos muy limitados en cuanto a potencia y el área destinada a la seguridad de las etiquetas. Para iniciar el protocolo, es necesaria la introducción de una clave y de la entrada que se desea encriptar. Para el proceso, se dispone de una sbox el cual es iniciado de la siguiente forma:

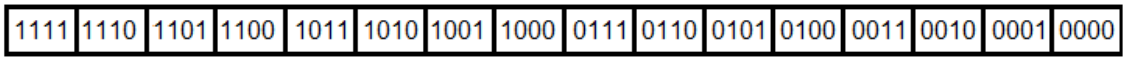


Figura 12. S-box inicial

Las posiciones de este vector (representado para 64 bits y de forma equivalente ocurriría con el de 160 bits) serán posteriormente conmutadas a través de un algoritmo basado en la clave que hemos introducido inicialmente, por tanto, en función de la clave este vector contenedor realizará unos cambios de posiciones u otros. Este proceso se repetirá durante 16 ciclos (en el caso del protocolo de 64 bits) o 32 (en el caso del protocolo de 160 bits) y es la parte que anteriormente se denominó KSA.

En una segunda fase se utilizará este vector ya conmutado y se le realizarán cambios de posición nuevamente, aunque en este caso los cambios serán independientes de la clave introducida, siendo esta utilizada únicamente para el cambio de posiciones en la primera fase. Esta parte del algoritmo es la que está basada en el PRGA.

Tras cada cambio de posición de esta segunda fase, se realizará la operación lógica XOR con la entrada de texto plano (convertida a binario si fuese necesario) obteniendo la salida bit a bit encriptada del protocolo. Este proceso se repetirá hasta completarse otros 16 ciclos (en caso de 64 bits) o 32 ciclos (en caso de 160 bits).

El pseudocódigo realizado para el presente proyecto es el siguiente:

```
//KSA
X = 0000;
Y = 0000;
Z = 0000;
for i = 0 to 16
{
   S(i%4) = S(i)+1;
}
for i = 0 to 16
{
   X = X + S(i) + K(i);
   Y = Y + S(X) + K(X+i);
   Z = Z + S(Y) + K(Y+i);
   S(i), S(X) = S(X), S(i);
   S(Y), S(Z) = S(Z), S(Y);
}
//PRGA
X = 0000;
Y = 0000;
Z = 0000;
T = 0000;
for i = 0 to 16
{
   X = X + 0001;
   Y = Y + S(X);
   Z = Z + S(Y);
   T = T + S(T);
   S(X), S(Y) = S(Y), S(X);
   S(Z), S(T) = S(T), S(Z);
}
Salida = Entrada (i) XOR S[S(X) + S(Y) + S(Z) + S(T)];
```

Los pasos, de forma resumida, que realiza el protocolo son los siguientes:

1º Inicialización de las señales del circuito

Se inicializan todas las señales que intervienen en el circuito y se almacena la clave introducida en un registro. Así mismo, se almacenarán también los bits del texto plano introducido por la entrada del mismo.

2º Suma de las variables que variarán el vector contenedor

A través de un contador, se irán realizando los diferentes ciclos del circuito, de tal forma que se pueda modificar el vector contenedor una vez haya finalizado la cuenta. Para ello utilizaremos las variables X, Y y Z, cuyo valor es el que hará cambiar en primera instancia al vector contenedor. Para realizar el cambio de posiciones en el vector contenedor se utiliza un registro auxiliar, de tal forma que se haga posible esta operación sin borrar uno de los valores al realizar el cambio.

3º Se realizan los cambios de posición en el vector contenedor

Una vez tenemos calculada la X, Y y Z en cada ciclo del contador, variaremos las posiciones del vector contenedor, de tal forma que vamos encriptando este en función de la clave introducida.

4º Segunda suma de las variables que variarán el vector contenedor

Tras finalizar la primera parte en la cual realizamos los 16 ciclos (o 64 en el caso de 160 bits) pasaremos a una segunda parte del algoritmo. Para iniciar esta segunda parte es necesario inicializar a cero los valores de X, Y y Z y posteriormente volver a calcular un nuevo valor para estos ciclo a ciclo

5º Se realizan nuevos cambios en el vector contenedor

Una vez se calcula X, Y y Z, se realizan de nuevo los cambios oportunos en el vector contenedor. Esto se realiza ciclo a ciclo, de tal forma que tendremos 16 cambios (o 64 en el caso de 160 bits) coincidiendo con los 16 valores que adoptarán X Y y Z.

6º Se realiza una XOR de la entrada con el vector contenedor

Finalizado cada cambio de posición en el vector contenedor, se realiza una operación XOR de la entrada del circuito con las posiciones de dicho vector que vendrán determinadas a través de una función basada en los valores X, Y, Z y T. Esta operación 16 veces, una por cada ciclo del contador en la segunda parte del circuito.

7º Valor de salida

Un registro irá tomando el valor de salida y lo irá almacenando, de tal forma que este será presentado en la salida del circuito únicamente cuando hayamos llegado a la cuenta final y un bit comprobador se ponga a 1, momento en el cual el registro será copiado a la salida del algoritmo.

A continuación se presenta un flujograma del cifrador de flujo a implementar

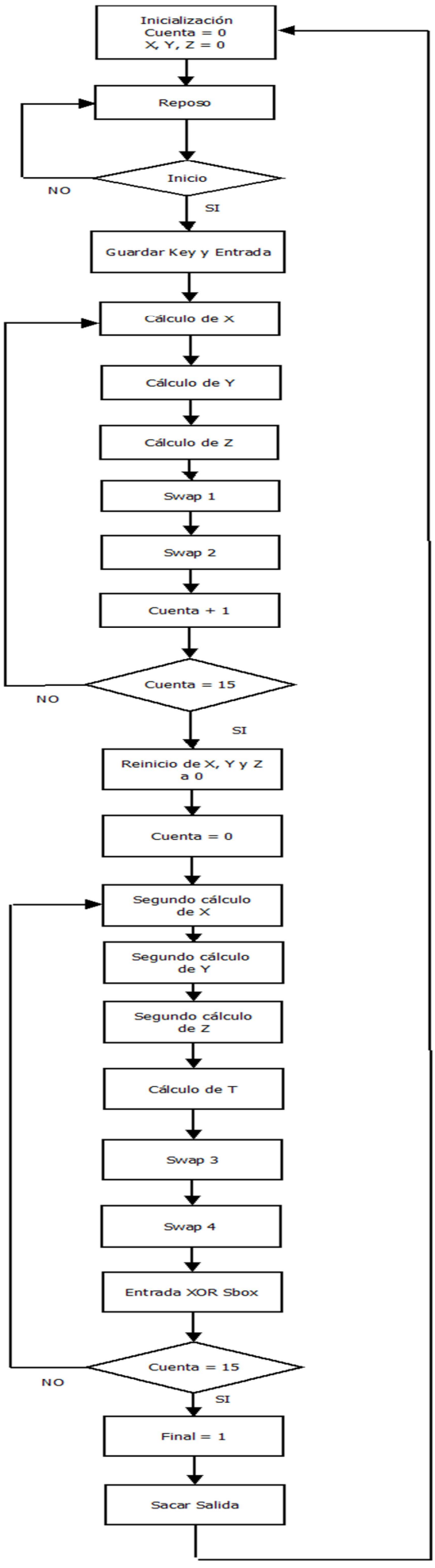


Figura 13. Flujograma del algoritmo a implementar

4.4. PROTOCOLO CON 2 SUMADORES

El primer protocolo implementado será el que dispone de dos sumadores de dos entradas en cascada para lograr obtener el mayor througput posible.

Esta implementación contará con cinco elementos principales:

- **Control:** es el encargado de comunicar la máquina de estados con el resto de bloques del circuito. será el encargado de enviar las señales a los distintos bloques.

- **Registros:** memorias donde se almacenarán los vectores necesarios para realizar las operaciones que sean requeridas por el algoritmo.

- **Máquina de estados:** es la encargada de realizar las operaciones oportunas y de gestionar los ciclos que el contador irá marcando.

- **Sumador:** se dispone de tres entradas y es el encargado de realizar la suma en binario de las operaciones a realizar. Dispone de tres entradas A, B y C y una salida de cuatro bits con el resultado. La implementación posterior serán dos sumadores de dos entradas en cascada.

- **Contador:** es el encargado de dar los ciclos al circuito. Dispone de un enable, de tal forma que solo contaremos un nuevo ciclo en el momento que se requiera y por tanto su funcionamiento no es constante. Gracias a este contador el circuito puede situarse en los distintos elementos de cada vector, ya que con cada cuenta y ciclo, estaremos indicando en qué posición del vector debemos situarnos para realizar algunas operaciones del circuito, como el cálculo de la variable X o los cambios de posición que se producen en el vector contenedor.

El diagrama de bloques del circuito es mostrado en la figura 10:

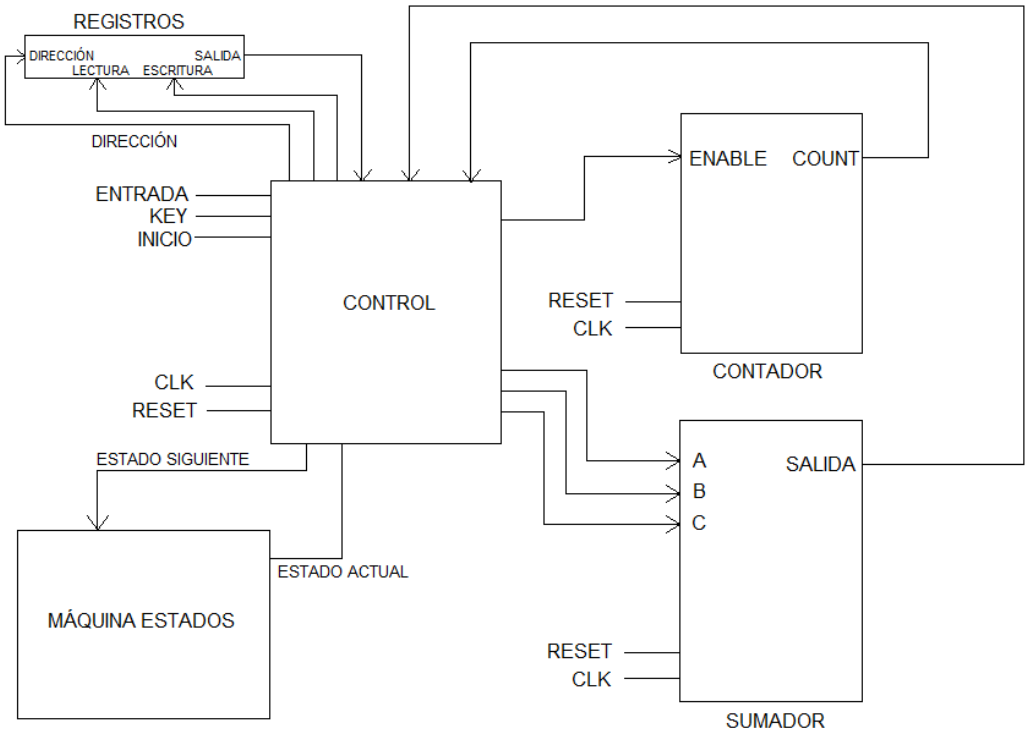


Figura 14. Diagrama de bloques del circuito para la arquitectura de dos sumadores

En esta arquitectura, la máquina de estados pasa por los siguientes estados:

- **Estado Inicio:** en este estado la máquina se encarga de reiniciar los valores de todos los registros, quedando a la espera de la señal inicio para entrar en el estado siguiente: guardar.

- **Estado guardar:** la máquina de estados da la orden para guardar las entradas del circuito en los registros que correspondan. Además, en este estado el vector contenedor inicializa sus valores y se reinician los registros de nuevo. Una vez se guarda toda la información, pasa al siguiente estado: sumax.

- **Estado sumax:** en este estado se realiza la suma de la variable X. Esta vendrá determinada por el valor de los cuatro bits de la clave en cada ciclo de reloj, el valor de X en el ciclo anterior y el valor de los cuatro bits del vector contenedor en la posición seleccionada. Es decir:
  X = X + s(contador) + K(contador).
  En el anexo F y en el anexo H se observa ciclo a ciclo los valores obtenidos de Y para una clave de ejemplo con 64 bits y 160 respectivamente.

- **Estado sumay:** se realizará la suma de la variable Y. Esta viene determinada por el valor de los cuatro bits de la clave en la posición indicada por cada valor de X sumado al valor del contador, así como el valor de Y del ciclo anterior y por los cuatro bits del vector contenedor en la posición indicada por la variable X. Esto es:
  Y = Y+ s(X) + K(X+contador).
  En el anexo F y en el anexo K se observa ciclo a ciclo los valores obtenidos de Y para una clave de ejemplo con 64 bits y 160 respectivamente.

- **Estado sumaz:** en este caso se determina el valor de Z en cada ciclo. Este dependerá del valor de Z del ciclo anterior así como de los cuatro bits del vector contenedor en la posición Y en este momento y de los cuatro bits de la clave en la posición indicada por la suma del contador más la variable Y. Esto es:
  Z = Z + s(Y) + K(Y+contador).
  En el anexo F y en el anexo K se observa ciclo a ciclo los valores obtenidos de Z para una clave de ejemplo para 64 y 160 bits.
  Además, en este ciclo se realiza el primer cambio de posiciones en el vector contenedor, en función del valor del contador y de la variable X. Se realizará como se indica:
  S(contador) ⇔S(x).

- **Estado swap1:** en este estado se tomará en el registro auxiliar de s para realizar el cambio el valor que corresponda. Se requiere un estado de la máquina únicamente para esto debido a que hay que realizar los pasos en riguroso orden y hasta que no se produzca el primer cambio en el vector contenedor (hecho en el estado sumaz), no se puede almacenar el valor de forma correcta.

- **Estado swap2:** se realizará el segundo cambio de posiciones en el vector contenedor dependiendo de la variable Y y la variable Z de la forma siguiente:
  S(Y) ⇔S(Z). En el anexo F-Resultados tras la primera parte del circuito, se observa el cambio final que se realiza en la variable s-box para 64 bits y los mismos resultados se observan para 160 bits en K – Resultados tras la primera parte del circuito.

- **Estado inicio2:** a partir de aquí se entra en la segunda parte del algoritmo, una vez que se han realizado los otros estados un total de 16 ciclos (en el caso de 64 bits) o 32 ciclos (en caso de 160 bits). En este estado se reiniciará el valor de X, Y y Z, ya que deben comenzar con valor 0 para cumplir con el cometido en la segunda parte. En el anexo F-Resultados tras la primera parte del circuito (64bits) y K-

Resultados tras la primera parte del circuito (160 bits), además del último cambio de sbox, se observa como todas las variables son inicializadas de nuevo para comenzar la segunda parte del circuito.

- **Estado sumax2:** se calculará un nuevo valor de X, de tal forma que esta dependerá únicamente del valor de la X del ciclo anterior, ya que en este estado:
  X = 0001+X.
  En el anexo F se pueden observar todos estos segundos cambios de la variable X ciclo a ciclo para una clave de ejemplo de 64bits y en el anexo K para una clave de 160.
- **Estado sumay2:** se calcula un nuevo valor de la variable Y, el cual dependerá del valor de Y del ciclo anterior y de los cuatro bits situados en el vector contenedor en la posición que indique la variable X. Esto es:
  Y = Y+ S(X).
  En el anexo F se pueden observar todos estos segundos cambios de la variable Y ciclo a ciclo para una clave de ejemplo de 64bits y en el anexo K para una clave de 160.

- **Estado sumaz2:** se calcula la variable Z de la segunda parte. Dicha variable dependerá del valor de Z en el ciclo anterior y de los cuatro bits situados en el vector contenedor en la posición que indique la variable Y. Esto es:
  Z = Z+S(Y).
  En el anexo F se pueden observar todos estos segundos cambios de la variable Z ciclo a ciclo para una clave de ejemplo de 64bits y en el anexo K para una clave de 160.

- **Estado sumat:** una nueva variable será utilizada en la segunda parte del algoritmo. Esta depende únicamente de sí misma en el estado anterior y de la posición del vector contenedor que marque ella misma. Por tanto:
  T = T+S(T).
  En el anexo F se pueden observar todos estos cambios de la variable t ciclo a ciclo para una clave de ejemplo de 64bits y en el anexo H para una clave de 160.
  Además, en este estado se realiza el tercer cambio de posiciones del vector contenedor en el algoritmo. Las posiciones se cambiarán como sigue:
  S(X) ⇔ S(Y).

- **Estado swap3:** este estado será utilizado para almacenar en el registro auxiliar el contenido de la posición Z del vector contenedor,

con el fin de realizar en el ciclo siguiente el último cambio de posiciones de dicho vector.

- **Estado swap4:** en este estado se realiza el último cambio de posiciones en el vector contenedor de la forma que se indica:
  S(Z) ⇔ S(T).

- **Estado XOR1:** denominado así porque es el primer estado que realiza funciones necesarias para realizar la XOR que obtendrá la salida del circuito. En este estado se realizará el cálculo de la posición que será sometida a la operación de la XOR con la entrada del circuito. Dicho valor de la posición será almacenado en un registro y dependerá de las posiciones del vector contenedor que indiquen las variables X, Y, Z y T. Se calcula de la forma siguiente:
  Sumas = S(X)+S(Y)+S(Z)+S(T). En este caso se introduce una suma adicional con el fin de obtener el resultado de los cuatro sumandos en un solo ciclo y no introducir un ciclo adicional para realizar esta operación.

- **Estado XOR2:** se trata del último estado del algoritmo. En él se realizará la operación XOR con la posición calculada en el estado XOR1 y la entrada del circuito. Se realizará del siguiente modo:
  Registro_Salida = Entrada XOR S(Sumas).
  Este registro almacenará la salida durante la ejecución del algoritmo, hasta que este llegue a su última iteración, de tal forma que en el momento que el contador indique que todas las operaciones XOR han sido realizadas, un bit denominado Final, indicará este evento, copiándose en ese momento el contenido de Registro_Salida en la salida del circuito. En el anexo F- Resultado final, se puede ver cómo la salida de 64 adquiere un valor en el circuito por primera vez, tomándolo del registro y en el anexo K – Resultado final, se observa esta misma situación para la arquitectura con 160 bits.

La figura 11 muestra el funcionamiento de la máquina de estados:

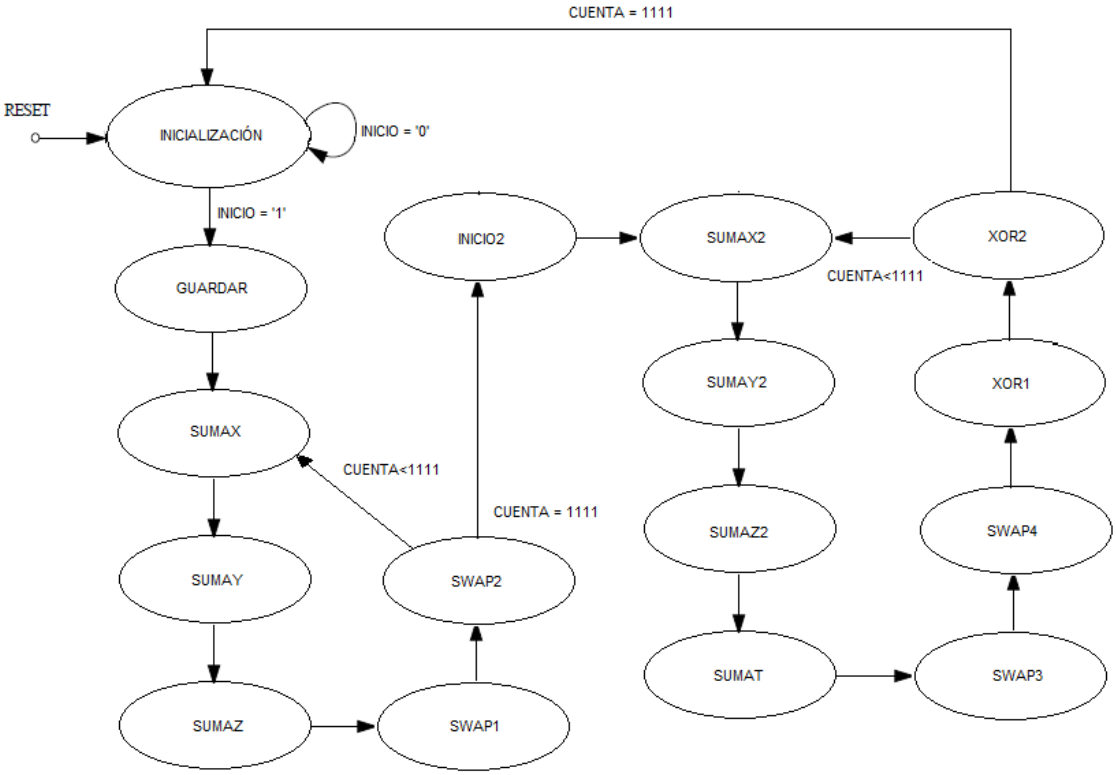


Figura 15.Máquina de estados para arquitectura de dos sumadores

4.5. PROTOCOLO CON UN SUMADOR DE 2 ENTRADAS

En este caso, se implementará el protocolo con un sumador de 2 entradas, con el objetivo de conseguir reducir el área que ocupa si lo comparamos con la arquitectura del apartado 4.4, el cual estaba diseñado con dos sumador de dos entradas en cascada. Este cambio hará que la máquina de estados necesite más estados para poder realizar las mismas operaciones.

Esta implementación contará con cinco elementos principales:

- **Control:** es el encargado de comunicar la máquina de estados con el resto de bloques del circuito. será el encargado de enviar las señales a los distintos bloques.

- **Registros:** memorias donde se almacenarán los vectores necesarios para realizar las operaciones que sean requeridas por el algoritmo.

- **Máquina de estados:** es la encargada de realizar las operaciones oportunas y de gestionar los ciclos que el contador irá marcando. Tendrá más estados que en el caso 4.4, ya que el sumador solo dispondrá de dos entradas.

- **Sumador:** dispone de dos entradas y es el encargado de realizar la suma en binario de las operaciones a realizar. Contiene dos entradas A y B y una salida de cuatro bits con el resultado.

- **Contador:** es el encargado de dar los ciclos al circuito. Dispone de un enable, de tal forma que solo contaremos un nuevo ciclo en el momento que se requiera y por tanto su funcionamiento no es constante. Gracias a este contador el circuito puede situarse en los distintos elementos de cada vector, ya que con cada cuenta y ciclo, estaremos indicando en qué posición del vector debemos situarnos para realizar algunas operaciones del circuito, como el cálculo de la variable X o los cambios de posición que se producen en el vector contenedor.

El diagrama de bloques en este caso sería similar al anterior, ya que el único cambio que se observa es la eliminación de una de las entradas del sumador:

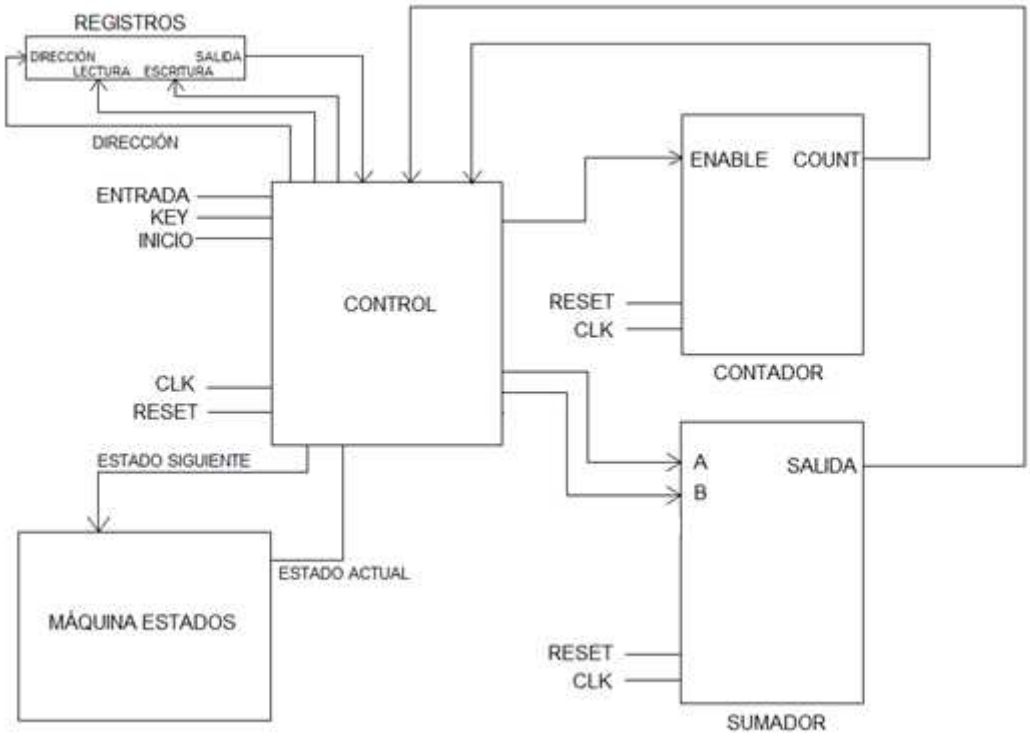


Figura 16. Diagrama de bloques del circuito para la arquitectura de un sumador de dos entradas

En esta arquitectura, la máquina de estados pasa por los siguientes estados:

- **Estado Inicio:** en este estado la máquina se encarga de reiniciar los valores de todos los registros, quedando a la espera de la señal inicio para entrar en el estado siguiente: guardar.

- **Estado guardar:** la máquina de estados da la orden para guardar las entradas del circuito en los registros que correspondan. Además, en este estado el vector contenedor inicializa sus valores y se reinician los registros de nuevo. Una vez se guarda toda la información, pasa al siguiente estado: sumax.

- **Estado sumax:** en este estado se realiza la primera iteración de la suma de la variable X. Esta vendrá determinada por el valor de X en el ciclo anterior y el valor de los cuatro bits del vector contenedor en la posición indicada por el contador. Es decir:
  X = X + s(contador).
  En el anexo G y en el L se observa ciclo a ciclo los valores obtenidos de X para una clave de ejemplo de 64 y 160 bits respectivamente

- **Estado suma2x:** se introducirán en el sumador la variable X calculada en el estado anterior y la posición de la clave determinada por el contador. Por tanto estaremos sumando al resultado de antes la posición de Key requerida:
  X = X + K(contador). Siendo la X sumada la X calculada en el estado anterior.
  En el anexo G y L se observa ciclo a ciclo los valores obtenidos de X para una clave de ejemplo de 64 y 160 bits respectivamente.

- **Estado sumay:** se realizará la primera suma para el cálculo de la variable Y. Esta viene determinada por el valor de los cuatro bits del vector contenedor en la posición indicada por X así como el valor de Y del ciclo anterior. Esto es:
  Y = Y + S(X).
  El anexo G y el anexo L permiten ver ciclo a ciclo los valores obtenidos de Y para una clave de ejemplo de 64 y 160 bits respectivamente.

- **Estado suma2y:** en este estado se le sumaran al resultado del estado anterior los cuatro bits de la posición indicada por la variable X de la clave. Esto es:

Y = Y + K(X). Donde la Y sumada es la Y resultado de la suma del estado anterior.
El anexo G y el anexo L permiten ver ciclo a ciclo los valores obtenidos de Y para una clave de ejemplo de 64 y 160 bits respectivamente.

- **Estado sumaz:** en este caso se determina la primera suma del valor de Z en cada ciclo. Este dependerá del valor de Z del ciclo anterior así como de los cuatro bits de Y en este momento, los cuales indicarán la posición del vector contenedor donde se alojan los cuatro bits a sumar. Esto es:
  Z = Z + s(Y).
  El anexo G y el anexo L permiten ver ciclo a ciclo los valores obtenidos de Z para una clave de ejemplo de 64 y 160 bits respectivamente.

- **Estado suma2z:** segunda iteración para el cálculo de la variable Z. Se sumará al resultado del estado anterior el valor de los cuatro bits en la posición indicada por Y de la clave:
  Z = Z + K(Y).
  El anexo G y el anexo L permiten ver ciclo a ciclo los valores obtenidos de Y para una clave de ejemplo de 64 y 160 bits respectivamente.
  Además, en este ciclo se realiza el primer cambio de posiciones en el vector contenedor, en función del valor del contador y de la variable X. Se realizará como se indica:
  S(contador) ⇔S(x).

- **Estado swap1:** en este estado se tomará en el registro auxiliar de s para realizar el cambio el valor que corresponda. Se requiere un estado de la máquina únicamente para esto debido a que hay que realizar los pasos en riguroso orden y hasta que no se produzca el primer cambio en el vector contenedor (hecho en el estado sumaz), no se puede almacenar el valor de forma correcta.

- **Estado swap2:** se realizará el segundo cambio de posiciones en el vector contenedor dependiendo de la variable Y y la variable Z de la forma siguiente:
  S(Y) ⇔S(Z). En el anexo G-Resultados tras la primera parte del circuito y L-Resultados tras la primera parte del circuito, se observa el cambio final que se realiza en la variable s-box para la arquitectura de 64 bits y 160 bits.

- **Estado inicio2:** a partir de aquí se entra en la segunda parte del algoritmo, una vez que se han realizado los otros estados un total de 16 ciclos (en el caso de 64 bits) o 32 ciclos (en caso de 160 bits). En este estado se reiniciará el valor de X, Y y Z, ya que deben comenzar con valor 0 para cumplir con el cometido en la segunda parte. En el anexo G-Resultados tras la primera parte del circuito, además del último cambio de sbox, se observa como todas las variables son inicializadas de nuevo para comenzar la segunda parte del circuito, al igual que podemos observarlo en el anexo L-Resultados tras la primera parte del circuito, para 160 bits.

- **Estado sumax2:** se calculará un nuevo valor de X, de tal forma que esta dependerá únicamente del valor de la X del ciclo anterior, ya que en este estado:
  X = 0001+X.
  En el anexo G y en el anexo L podemos observar ciclo a ciclo los cambios que sufre la variable X para una clave determinada como ejemplo de 64 y 160 bits respectivamente.

- **Estado sumay2:** se calcula un nuevo valor de la variable Y, el cual dependerá del valor de Y del ciclo anterior y de los cuatro bits situados en el vector contenedor en la posición que indique la variable X. Esto es:
  Y = Y+ S(X).
  En el anexo I podemos observar ciclo a ciclo los cambios que sufre la variable Y para una clave determinada como ejemplo.

- **Estado sumaz2:** se calcula la variable Z de la segunda parte. Dicha variable dependerá del valor de Z en el ciclo anterior y de los cuatro bits situados en el vector contenedor en la posición que indique la variable Y. Esto es:
  Z = Z+S(Y).
  En el anexo G podemos observar ciclo a ciclo los cambios que sufre la variable Z para una clave determinada como ejemplo.

- **Estado sumat:** una nueva variable será utilizada en la segunda parte del algoritmo. Esta depende únicamente de sí misma en el estado anterior y de la posición del vector contenedor que marque ella misma. Por tanto:
  T = T+S(T).
  En el anexo G y en el anexo L podemos observar ciclo a ciclo los cambios que sufre la variable t para una clave determinada como ejemplo.

Además, en este estado se realiza el tercer cambio de posiciones del vector contenedor en el algoritmo. Las posiciones se cambiarán como sigue:
S(X) ⇔ S(Y).

- **Estado swap3:** este estado será utilizado para almacenar en el registro auxiliar el contenido de la posición Z del vector contenedor, con el fin de realizar en el ciclo siguiente el último cambio de posiciones de dicho vector.

- **Estado swap4:** en este estado se realiza el último cambio de posiciones en el vector contenedor de la forma que se indica:
S(Z) ⇔ S(T).

- **Estado sumas1:** En este estado se realizará el cálculo de la posición que será sometida a la operación de la XOR con la entrada del circuito. Dicho valor de la posición será almacenado en un registro y dependerá de las posiciones del vector contenedor que indiquen las variables X, Y, Z y T. Se calcula de la forma siguiente:
Sumas = S(X)+S(Y)+S(Z)+S(T).
En este estado solo se realizará la primera parte de la operación, debido a que el sumador disponible solo tiene dos entradas. Por tanto la operación en este estado será:
Sumas = S(X)+S(Y).

- **Estado sumas2:** en este estado se realiza la segunda parte de la suma descrita en el estado anterior. Esto es:
Sumas = Sumas + S(Z)+S(T). Es necesario utilizar un nuevo sumador con el objetivo de no introducir un estado adicional para este cálculo, realizando una suma de 3 sumandos en lugar de los dos que permite el sumador.

- **Estado XOR2:** se trata del último estado del algoritmo. En él se realizará la operación XOR con la posición calculada en el estado XOR1 y la entrada del circuito. Se realizará del siguiente modo:
Registro_Salida = Entrada XOR S(Sumas).
Este registro almacenará la salida durante la ejecución del algoritmo, hasta que este llegue a su última iteración, de tal forma que en el momento que el contador indique que todas las operaciones XOR han sido realizadas, un bit denominado Final, indicará este evento, copiándose en ese momento el contenido de Registro_Salida en la salida del circuito. En el anexo G- Resultado final, se puede ver cómo la salida adquiere un valor en el circuito por primera vez, tomándolo

del registro y en el anexo L-Resultado final se observará lo mismo para la arquitectura de 160 bits.

A continuación se muestra un esquema del funcionamiento de la máquina de estados:

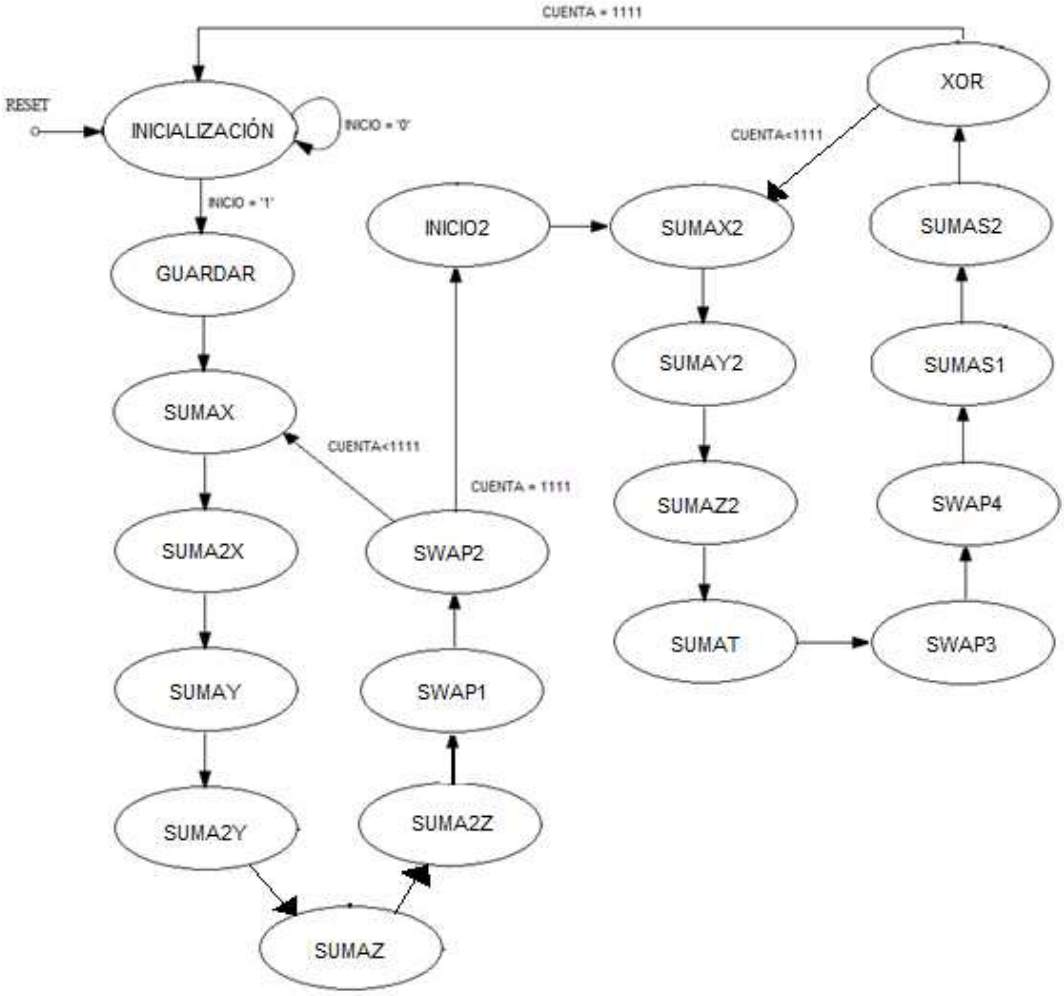


Figura 17. Máquina de estados para arquitectura del sumador de dos entradas

CAPÍTULO 5

TEST Y RESULTADOS

5.1. TEST

En este apartado se trata todo lo relativo a los test realizados con el fin de comprobar que el algoritmo es lo suficientemente seguro.

5.1.1. Importancia de los test

Los análisis estadísticos [38] de los cifrador de flujo, son más fáciles y rápidos en comparación con los análisis teóricos. Aunque no se le suele dar demasiada importancia a los test estadísticos, diversos estudios de seguridad de cifradores de flujo han demostrado la validez y veracidad de estos métodos.

El estudio de los test es de suma importancia a la hora de examinar la seguridad del cifrador de flujo, puesto que son los que nos determinarán si nuestra encriptación es lo suficientemente segura o si es vulnerable ante posibles ataques que deseen descifrarla.

En este estudio, se tratará de enfatizar la importancia de los test estadísticos para la comprobación de resultados en los cifradores de flujo, especialmente en los cifradores de flujo síncronos. Se realizarán test al algoritmo de 64 bits y al de 160, para observar su robustez estadísticamente.

Dividiremos el estudio en dos fases, en la primera se realizará el test estadístico de NIST a nuestros cifradorES de flujo y una segunda fase, en la cual se realizarán el test chi-cuadrado (a través de matlab) para saber la correlación que tiene la salida con la contraseña y comprobar que en el resultado final no se obtenga correlación entre ellas, obteniendo así una salida encriptada difícil de descifrar.

Finalmente se extraerán las conclusiones que los test realizados arrojen.

5.1.2. Análisis estadísticos

Para reforzar la elección de test estadísticos para comprobar la robustez de los sistemas, en esta sección, se resumirán y se hará un pequeño recorrido por los test estadísticos que han sido aplicados por los autores de eSREAM ciphers.

La salida de los cifradores de flujo síncronos debe ser indistinguible a partir de secuencias aleatorias en la entrada y no debe dejar escapar al exterior

información sobre la clave secreta del código ni el estado interno del sistema de cifrado.

A través del estudio de estos test, se analizará las propiedades de aleatoriedad de los cifradores de flujo. Para ello se aplicará en una primera fase el test de NIST a las secuencias de salida y en una segunda fase se realizarán test de aleatoriedad estructurales para los sistemas de cifrado.

El funcionamiento básico de los cifradores de flujo se basa en asegurar un OTP en el que el texto plano sea cifrado con un flujo de claves aleatorias a través de una función XOR. OTP es el único sistema seguro conocido que proporciona claves de forma realmente aleatorias. La seguridad de los OTP radica en que gracias a la encriptación de un texto no aleatorio con una secuencia de claves completamente aleatorias obtendremos una salida encriptada con una alta seguridad.

Se requiere una clave no más corta que el texto plano, de tal forma que se asegura un funcionamiento del encriptado fluido, evitando que sea impracticable.

Los cifradores de flujo síncronos tienen la motivación de generar claves pseudo-aleatorias cortas para ser prácticos y no muy engorrosos consiguiendo evitar las dificultades de los diseños requeridos.

La seguridad teórica de los cifradores de flujo no está altamente comprobada, pero para aplicaciones en las cuales se requiere un alto rendimiento o unas restricciones elevadas, son muy prácticos.

En pocas palabras, el objetivo de diseño de un cifrado de flujo síncrono es el de generar bits pseudoaleatorios que son prácticamente indistinguibles de los bits realmente aleatorios de forma eficiente.

Dado que los keystreams de salida son generados utilizando algoritmos determinísticos, a veces es posible distinguirlos de secuencias verdaderamente al azar con probabilidades significativas.

En estas situaciones, los cifradores de flujo están expuestos a distintos ataques. Los autores de este tipo de algoritmos defienden que los ataques débiles que utilizan largos keystreams no suponen un problema para la seguridad de los cifradores de flujo. Además, no existe una longitud de cadena determinada en términos de tamaño de clave para atacar un sistema de cifrado de forma eficaz.

Muchos enfoques teóricos se han utilizado para encontrar distinguishers a los cifradores de flujo. Los distinguishers se encuentran, generalmente, después de realizar un extenso análisis teórico, sin embargo, las pruebas de aleatoriedad estadística presentan otro modo de encontrar dichos distinguishers de forma más sencilla. Las pruebas estadísticas no tienen en

cuenta la estructura interna de los sistemas de cifrado, por lo que resultan un método más sencillo para encontrar errores al sistema.

Se obtienen las estadísticas predefinidas y de cifrado independientes de los keystream y se comparan con las distribuciones teóricas. Si se observa una desviación sistemática, se supone que el keystream de salida es inseguro y distinguible de secuencias realmente aleatorias.

El análisis estadístico de los cifradores de flujo son más sencillos y más rápidos en comparación con los análisis teóricos.

El presente estudio tiene como objetivo probar la validez de los cifradores de flujo diseñados para un sistema RFID a través de análisis estadísticos.

Para demostrar la validez de este tipo de test, se realizará un resumen con los métodos de pruebas estadísticas que han sido aplicados por los autores de cifrados eStream.

- Anashin et all demostró que la distribución de palabras de 32 bits en la salida ABC es uniforme. Los resultados obtenidos en el test de NIST no indicaron ninguna desviación con respecto a una secuencia aleatoria, por lo que quedó demostrado que la salida era válida. Además, los autores aplicaron algunos test de aleatoriedad para evaluar la propiedad de propagación del cifrado usando la distancia de Hamming y la correlación ingenua. Para una clave fija y una clave que varía con cada par IV, se obtuvieron 384 secuencias de longitud 106, las cuales se evaluaron empíricamente. Los resultados del test de NIST no mostraron ninguna desviación de comportamiento aleatorio, por lo que quedó demostrado que la salida cumplía con los prerrequisitos de forma satisfactoria.

- En Fubuki y AES se aprobaron las propiedades de difusión del sistema con un pequeño número de rondas. Se determinó que con 4 rondas AES y 2 rondas Fubuki, el sesgo de difusión queda eliminado.

- El flujo de clave de dragón es probado por los test de aleatoriedad estadísticos que figuran en el Crypt-X. Los autores aplican la frecuencia, derivado binario, punto de cambio, sub-bloque, y ejecuta las pruebas a 30 keystreams de 8 megabits de longitud. Además, las pruebas de la complejidad de secuencia lineal se aplicaron a 30 flujos de 200 kilobits. Dragón no mostró ninguna desviación de la aleatoriedad, hecho que se demostró gracias al test de NIST.

- Para el cipher Rabit, se aplicaron las pruebas de NIST, DIEHARD y ENT suites. Las pruebas fueron realizadas tanto para el estado interno como para el flujo de claves. Además, diversas pruebas fueron realizadas para la función clave de configuración y para la versión reducida de Rabit. En todos los casos no se encontraron desviaciones de la aleatoriedad, por lo que el cifrador fue aceptado como válido.

- Vuckovac informó de que la salida del Mag es la prueba de patrones en cada etapa del desarrollo mediante el uso de test de aleatoriedad estadística disponibles en la ENT, DIEHARD y Crypt-X. De acuerdo con los resultados, no se observaron desviaciones respecto a la aleatoriedad.

En todos estos casos los test demuestran una salida completamente aleatoria, a pesar de que los cifradores de flujo son realizados con métodos pseudo-aleatorios.

En el presente proyecto se realizarán las pruebas necesarias para llegar a dicho resultado. Por ello se someterán a los algoritmos de encriptación diseñados a estos test, de tal forma que podamos demostrar que la salida es completamente aleatoria.

En una primera fase se examinarán las propiedades de aleatoriedad de la secuencia de claves. Para ello se generarán una cantidad elevada de claves aleatorias con el fin de aplicar los test mencionados al cifrador de flujo.

El primer paso para realizar las pruebas es caracterizar secuencias verdaderamente al azar por un estadístico de prueba con una distribución teórica conocida. Posteriormente se realizará una comparación entre los valores teóricos y los valores obtenidos mediante el test con el objetivo de llegar a una conclusión.

Debe tenerse en cuenta que los test citados no están originalmente diseñados para probar los cifradores de flujo, sino para evaluar las propiedades de aleatoriedad de secuencias finitas. Por ello cabe mencionar que no tendrán en cuenta la estructura interna del algoritmo, clave o fase de carga de los valores.

Por tanto, en la primera fase se determinarán las propiedades de aleatoriedad de una secuencia de claves, sin tener en cuenta los efectos de la clave o el estado interno del algoritmo. En una segunda fase se trataran de medir las correlaciones entre la clave y la salida. En este test se determinará la correlación entre la clave y la salida utilizando un texto plano fijo. Por ello, la entrada del circuito permanecerá constante, mientras se introducirán una secuencia de claves aleatorias con el fin de obtener un número determinado de salidas que posteriormente serán sometidas a pruebas que determinarán la

correlación entre la key y la salida. Para realizar estas pruebas se utilizará el chi-square goodness of Fit (test de bondad de chi-cuadrado).

En resumen, se realizará un primer test de NIST con el fin de determinar la aleatoriedad de la salida de nuestro algoritmo y posteriormente se aplicará el chi-square goodness of Fit manteniendo una entrada de texto plano fija y una secuencia de claves aleatorias, obteniendo así una serie de salidas que serán sometidas a las correspondientes pruebas para determinar la correlación entre la clave y las diferentes salidas.

5.1.3. Aleatoriedad en la salida

Para corroborar que el algoritmo tiene una encriptación fiable y difícil de descifrar, se realizará un test de aleatoriedad en la salida, el cual demostrará que la misma estará lo suficientemente encriptada como para que el test la determine como aleatoria y, por tanto, difícil de descifrar.

Existen numerosos test estadísticos que indican la aleatoriedad o no de una secuencia numérica como por ejemplo ENT o Diehard. En el presente proyecto se decidió tomar el test de NIST para realizar estas pruebas debido a que es el más extendido entre todos ellos y su fiabilidad está comprobada, llegando a ser utilizado por organizaciones como la NSA (National Security Agency) de Estados Unidos.

Para realizar este test se obtendrán 100 vectores de salida con entradas aleatorias, tomando los valores decimales del número pi para ello. Posteriormente, esos 100 vectores de salida serán sometidos al test de NIST, con el fin de comprobar si el algoritmo es lo suficientemente bueno como para que el test determine que nuestra salida es aleatoria, en cuyo caso, se determinará que la encriptación realizada finalmente al texto plano es lo suficientemente segura como para que el algoritmo sea implementado en el protocolo de autenticación de un sistema RFID.

Para el test de NIST, el análisis de los resultados se realizará teniendo en cuenta la cantidad de secuencias que pasan la prueba. Esta proporción vendrá determinada por el error α, generalmente establecido en 0,01. Por tanto la prueba será pasada exitosamente siempre que el test arroje un resultado de un 96% o mayor. Esta cota se obtiene por un intervalo de confianza de un 96% para un gran número de secuencias ($n > 100$).

Los resultados de los test pueden verse en el anexo E, donde se observa que todas las pruebas se pasan con un mínimo del 97% para una muestra de 100 secuencias binarias. El mínimo ratio que se debe cumplir es de un 96% para considerarse que la prueba es pasada, por lo que el test se cumple de manera satisfactoria.

Por todo ello queda demostrado que la salida del algoritmo diseñado se puede considerar completamente aleatoria.

Tras realizar los test al protocolo implementado, en el anexo D, se observa que todos ellos son pasados satisfactoriamente y que por tanto el algoritmo implementado tiene una encriptación segura.

5.1.4. Test de correlación clave / salida

El propósito de este test es el de evaluar la correlación existente entre la clave y los primeros k bits de la salida. La alta o la baja correlación mide la posibilidad de recuperar la clave secreta utilizando flujos de claves o puede reducir el espacio de claves en el que se ha de investigar para poder realizar ataques con éxito al sistema. El objetivo será el de obtener un resultado que determine que no existe correlación entre la clave y la salida, de tal forma que la encriptación sea la máxima posible. Si este test resulta fallido, se ha de volver a la fase de diseño e implementar un nuevo algoritmo de encriptación, ya que el diseñado será vulnerable ante posibles ataques.

Para realizar este test, en primer lugar se fija un texto plano para eliminar su influencia aleatoria y se generan m valores de la clave de forma aleatoria. A continuación, se obtendrán las diferentes salidas para cada una de las claves aleatoriamente introducidas.

Para evaluar su correlación, la clave y sus correspondientes salidas serán sometidas a una operación XOR y se calcula el peso de la secuencia resultante. Los valores obtenidos del peso m se clasifican en diferentes categorías, dependiendo del tamaño de la clave, para ser comparados dichos valores de cada categoría con los valores esperados teóricamente. Es aconsejable utilizar una m mayor de 100 para realizar esta prueba. En el caso del presente proyecto se utilizará una m de 200.

Para un cifrado seguro se obtendrá una distribución binomial con parámetro k y 1/2 . Por ejemplo P(peso = k) = $\binom{n}{k}$ $(1/2)^n$ con una probabilidad de éxito 1/2. Usando estas probabilidades para k = 80, los límites de las categorías serían los siguientes: 0-35, 36-38, 39-41, 42-44, 45-80. Para k = 128 los límites serían 0-58, 59-62, 63-65, 66-69, 70-128. En el caso del proyecto presente se especificarán en la muestra de resultados.

Pequeños pesos muestran que la clave y sus correspondientes salidas son similares, es decir, están positivamente correlacionadas. Altos pesos indican una correlación negativa entre el bit i de la clave y el bit i de salida para i=1…k.

El algoritmo utilizado en matlab para la realización del test es el siguiente [39]:

$$
\begin{array}{l}
Fix\ V; \\
\textbf{for } i \leftarrow 1 \textbf{ to } m \\
\quad \textbf{do} \\
\begin{cases}
Randomly\ choose\ key\ K; \\
Generate\ first\ k\ bits\ of\ keystream,\ M = S(K,V,k); \\
w_i = weight\ of\ M \oplus K;
\end{cases} \\
Categorize\ w_i's; \\
Apply\ Chi - Square\ of\ Goodness\ of\ Fit\ test; \\
\textbf{return } (p - value)
\end{array}
$$

El objetivo del estudio está relacionado con los valores de p y ha de ser uniforme entre 0 y 1. El intervalo entre 0 y 1 se divide en 10 subintervalos iguales. Los valores esperados y observados son comparados, calculando un nuevo valor de p. Si este valor de p es mayor de 0.00001 entonces la secuencia es considerada uniformemente distribuida. Para que una segunda evaluación sea aplicable, se necesitarán al menos 50 keystreams debido a la suposición de chi-cuadrado.

En el estudio a realizar, el resultado que debemos obtener de p-valor se ha de situar entre 0,01 y 0,95, ya que el objetivo de nuestro estudio es el de demostrar que no existe correlación entre la clave y la salida, de tal forma que la encriptación sea la máxima posible.

Para nuestras pruebas, utilizaremos 200 keystreams de longitud 64 bits, utilizando claves elegidas al azar y un texto plano.

La metodología a seguir implica determinar los rangos del estudio. Para el algoritmo de 64 bits, los rangos serán los siguientes:

```
RANGOS:
00-28     29-30     31-33     34-36     36-64
```

A partir de aquí, se obtienen 200 muestras de la salida y se cuenta la cantidad de 1 que cada salida obtenía, de tal forma que se clasifica la salida del circuito en las distintas categorías según la cantidad de 1 lógico que posea. Así quedó de la forma que sigue:

```
RANGOS:
00-28     29-30     31-33     34-36     36-64
  33        44        51        37        35
```

En el anexo F se puede observar el código utilizado en matlab y el resultado del test, gracias al cual se pudo obtener que la correlación entre la

clave y la salida no es alta, por lo que la encriptación realizada es lo suficientemente segura y evitará que la clave que se utilice sea recuperable fácilmente, obteniendo un algoritmo fiable y seguro.

5.2. RESULTADOS Y ANÁLISIS

En este capítulo se realizará el estudio del área y del consumo de cada arquitectura en base a diferentes bibliotecas las cuales poseen un voltaje de alimentación diferente, una tecnología distinta y una temperatura que variará.

Las limitaciones para la tecnología rfid son estrictas, situándose estas en un límite de área de 4000 puertas destinadas a seguridad. Son puertas equivalentes y se definen como el área del circuito dividido entre el área de una puerta NAND de 2 entradas. Es una medida de normalización ampliamente extendida en RFID. Otra limitación que establece esta tecnología son 10 micro watios de consumo. Por tanto, la mejor arquitectura de entre todas las estudiadas con distintas bibliotecas será la que menos consumo requiera siempre y cuando no se supere el límite de 4000 puertas NAND.

Como podremos observar en los resultados que se presentan posteriormente, el aumentar la tensión de alimentación provocará un aumento significativo del consumo de potencia por parte del circuito. Al aumentar la tensión de alimentación, provocaremos que la corriente que circule por el circuito sea mayor, aumentando la potencia que consumirá cada puerta lógica que se encuentre en el circuito. Por ello, en tecnología rfid, donde lo ideal es un consumo bajo, requeriremos de los circuitos que consuman la menor potencia posible. Otro de los factores que influyen notablemente en el consumo de potencia del circuito es la frecuencia a la cual trabaja este. En rfid no requeriremos de una frecuencia muy elevada, ya que las etiquetas serán leídas correctamente sin necesidad de establecer una velocidad muy alta. Por ello escogemos una frecuencia de 100KHz, ya que no es muy alta ni tampoco tan baja como para que haya problemas en la sincronización de las etiquetas con los lectores.

Una mayor tensión de alimentación mejoraría la inmunidad ante el ruido del circuito, pero en este caso no es necesaria puesto que la distancia de operación no suele ser elevada.

Para los circuitos implementados en este proyecto que cumplan con la especificación de 4000 puertas lógicas establecida como límite, buscaremos el menor consumo de potencia posible, teniendo en cuenta que la velocidad del circuito no será un requisito fundamental, ya que para sistemas rfid no se requiere una gran velocidad.

Para realizar el estudio de las distintas arquitecturas, se utilizan diferentes bibliotecas que varían entre ellas en la tensión de alimentación, la tecnología utilizada y la temperatura a la que el circuito será sometido. Estas bibliotecas son las siguientes:

- **Faraday 90nm 1v25:** esta bilbioteca trabaja con una tecnología de 90nm de ancho de puerta y aporta una tensión de alimentación de 1V al circuito y lo somete a una temperatura de 25ºC.

- **Faraday 90nm 2v25:** en este caso, la biblioteca trabaja con una tecnología de 90nm de ancho de puerta y aporta una tensión de alimentación de 1,2V con una temperatura de 25ºC.

- **Faraday 90nm 1p32vm40:** esta biblioteca es utilizada para observar el comportamiento del circuito a temperaturas extremas. Para ello trabaja con una tecnología de 90nm, una tensión de alimentación de 1,32V y una temperatura de -40ºC.

- **Amis 1.32v:** aporta una tensión de alimentación de 1,32V con una tecnología de 350nm y una temperatura de 0ºC. Con esta biblioteca podremos observar la diferencia al considerar un ancho de puerta diferente respecto a las bibliotecas Faraday, además de tratarlo a una temperatura diferente y una tensión de alimentación más elevada.

- **Amis 1.98v:** esta biblioteca se utiliza para comparar un aumento de tensión respecto a la biblioteca expuesta anteriormente. Para ello se realizarán las pruebas con una tecnología de 350nm, una temperatura de 0ºC y una tensión de alimentación de 1,98V, lo que supone un aumento de tensión en 0,66V.

5.2.1. Resultados de síntesis arquitectura dos sumadores

En este apartado se mostrarán todos los resultados obtenidos en cuanto a área y potencia de la arquitectura de dos sumadores, variando la tensión de alimentación dentro de las diferentes tecnologías, por lo que obtendremos resultados que nos permitirán comparar el consumo y el área dentro de cada tecnología a diferentes tensiones y entre las propias tecnologías. Las diferentes bibliotecas poseen una temperatura distinta, por lo que además del efecto del cambio de tecnología observaremos cómo afecta la temperatura al circuito. Además se realizará un estudio a una temperatura de frío extremo para ver el comportamiento del circuito y cómo varía respecto a una temperatura más común.

Con esta arquitectura lo que se pretende es conseguir el menor número de ciclos, utilizando para ello dos sumadores en cascada en lugar de un único

sumador, con el fin de reducir reducir las operaciones de la máquina de estados.

Además se comparará el consumo de potencia respecto a la arquitectura de un sumador, ya que un menor consumo será ventajoso para nuestro circuito favoreciendo una mayor duración de las baterías en etiquetas activas y una transferencia menor del lector a la etiqueta en el caso de etiquetas pasivas, las cuales deben ser alimentadas a la hora de realizar la transferencia de datos.

De los datos recopilados se obtiene la tabla resumen con los resultados de la arquitectura, sintetizados en función del número de bits y de las distintas bibliotecas utilizadas para realizar las pruebas a diferentes tensiones de alimentación y tecnología.

	biblioteca faraday 90nm 1v25 (25ºC, 1V)		biblioteca faraday 90nm 2v25 (25ºC, 1,2V)	
	WIDHT_state = 64 bits	WIDHT_state = 160 bits	WIDHT_state = 64 bits	WIDHT_state = 160 bits
Área (μm^2)	5316,304012	12038,32001	5315,520011	11988,144015
Potencia interna (nW)	112,2668	258,005	162,7470	373,585200
Potencia conmutación (nW)	21,6287	37,8518	33,0750	57,596800
Potencia dinámica total (nW)	133,8955	295,8568	195,8220	431,182000
Pérdidas de potencia (nW)	8,1759	19,1316	13,0168	30,132300
Puertas equivalentes	1682,374687	3809,59494	1682,126586	3793,71646
Tiempo de ejecución	211,5 (2114,969 ns)	418,5 (4185,0 ns)	211,5 (2114,969 ns)	418,5 (4185,0 ns)
Throughput (Kbps)	30,274	38,232	30,274	38,232
	biblioteca amis 1.32v (0ºC, 1,32V)		biblioteca amis 1.98v (0ºC, 1,98V)	
	WIDHT_state = 64 bits	WIDHT_state = 160 bits	WIDHT_state = 64 bits	WIDHT_state = 160 bits
Área (μm^2)	2192,664494	4324,664576	1970,993695	4651,660585
Potencia interna (µW)	1,1670	2,6222	2,6745	6,2597
Potencia conmutación (nW)	58,8273	119,8085	126,073200	244,5168
Potencia dinámica total (µW)	1,2258	2,7420	2,800600	6,5042
Pérdidas de potencia (nW)	20,2703	50,4184	21,0329	49,7666
Puertas equivalentes	2192,664494	4324,664576	1970,993695	4651,660585
Tiempo de ejecución	211,5 (2114,969 ns)	418,5 (4185,0 ns)	211,5 (2114,969 ns)	38,232
Throughput (Kbps)	30,274	38,232	30,274	38,232
	biblioteca faraday 90nm 1p32vm40 (-40ºC, 1,32V)			
	WIDHT_state = 64 bits	WIDHT_state = 160 bits		
Área (μm^2)	5321,008011	12145,72801		
Potencia interna (nW)	201,3079	462,2578		
Potencia conmutación (nW)	42,2415	72,8252		
Potencia dinámica total (nW)	243,5494	535,0831		
Pérdidas de potencia (nW)	9,3922	20,9747		
Puertas equivalentes	1683,863295	3843,584812		
Tiempo de ejecución	211,5 (2114,969 ns)	418,5 (4185,0 ns)		
Throughput (Kbps)	30,274	38,232		

TABLA 1. Tabla resumen arquitectura dos sumadores

Observando la tabla 1, podemos afirmar que los resultados muestran lo esperado. Se comprueba que la arquitectura con dos sumadores cumple para 64 bits con las especificaciones de 4000 puertas y 10 µW en todas las bibliotecas utilizadas, por lo que serían factibles para instalar en el protocolo de autenticación de un sistema RFID.

Sin embargo, esto no ocurre con las arquitecturas de 160 bits, ya que para una tecnología de 350nm, todas las pruebas realizadas superan las 4000 puertas, por lo que no serían factibles a la hora de implementarlo en un circuito rfid.

La utilización de 160 bits ofrece una mayor fiabilidad y una mayor capacidad de almacenamiento de datos, sin embargo, los ciclos necesarios para las operaciones, el consumo y el área es mucho mayor que para 64 bits. En este circuito, gracias a los test, quedó demostrado que una utilización de 64 bits ofrece unos resultados iguales que los de 160 bits, por lo que la seguridad del sistema es elevada en ambos casos y priorizaremos el bajo consumo y el menor área ocupada.

También se puede observar que a mayor tensión de alimentación, el consumo cada vez es mayor tanto en 64 bits como en 160. Además, el aumento de tecnología provoca un aumento muy considerable tanto de área como de consumo, por lo que debido a las restricciones que se presentan en RFID es mejor utilizar la tecnología que ofrezca un menor tamaño de puerta.

La última biblioteca empleada en la tabla 1 se trata del estudio hecho a una temperatura extrema de -40ºC. A esta temperatura el comportamiento del circuito es difícil de predecir, aunque observando la tabla se comprueba que el área necesaria no varía y la potencia consumida se eleva de forma considerable, por lo que trabajar a temperaturas extremas supone un esfuerzo y unas limitaciones extras para nuestro circuito.

Para recopilar los datos y poder observarlos con mayor facilidad, en la figura 14 y 15 se realiza un gráfico de las puertas equivalentes y el consumo para esta arquitectura, con el fin de poder compararlos visualmente.

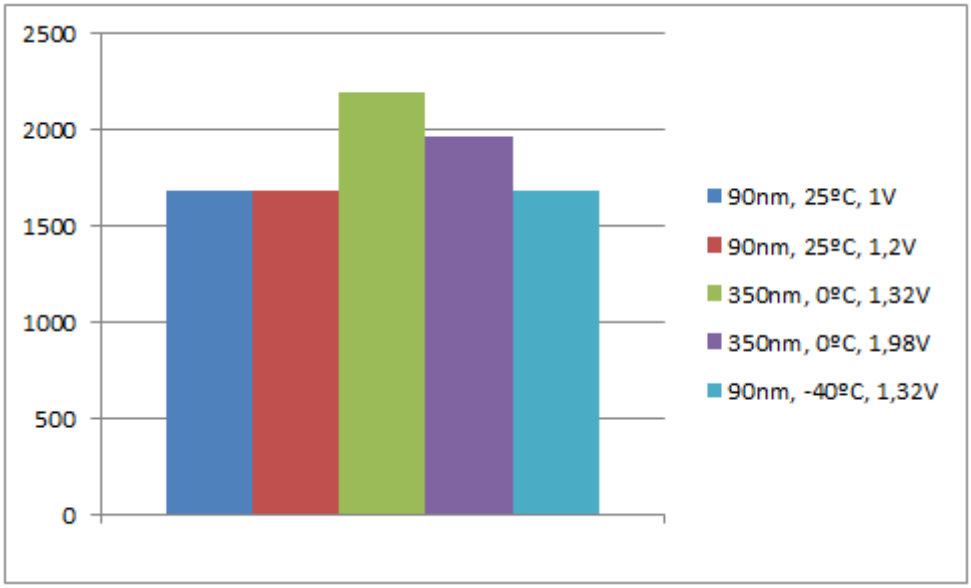


Figura 18. Gráfico puertas equivalentes arquitectura de 64 bits con dos sumadores

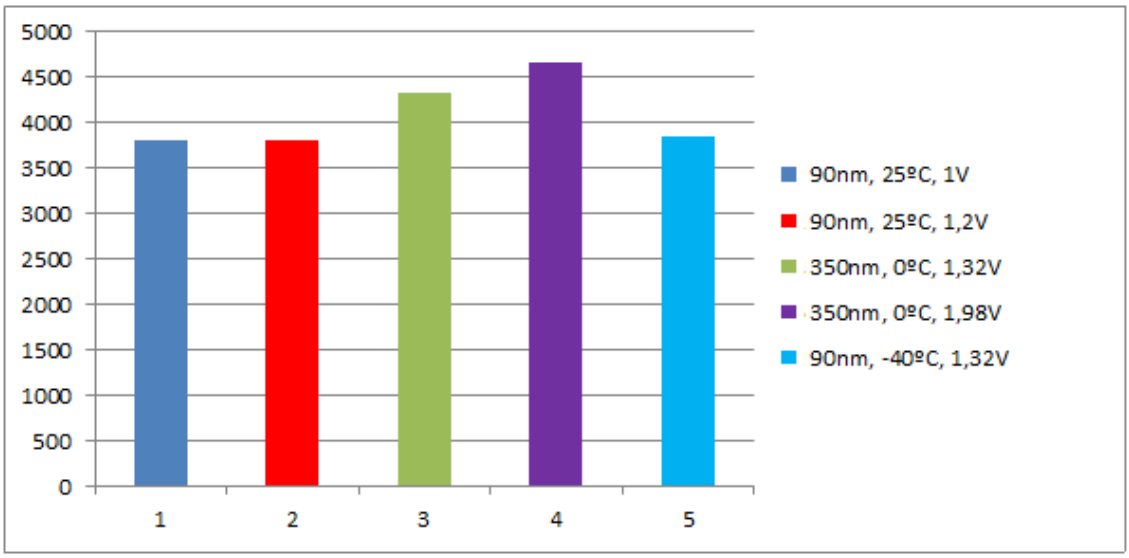


Figura 19. Gráfico puertas equivalentes arquitectura de 160 bits con dos sumadores

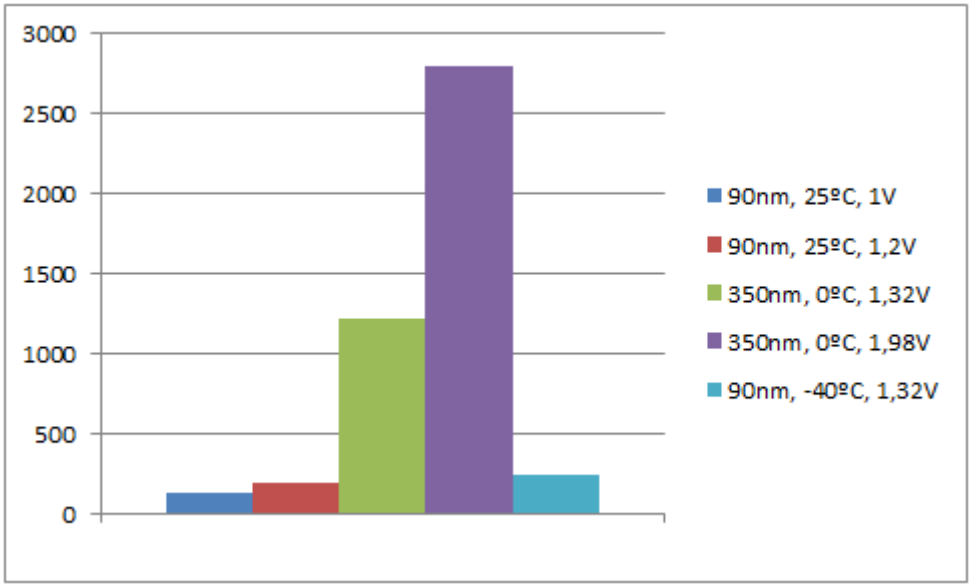


Figura 20. Gráfico potencia arquitectura de 64 bits con dos sumadores

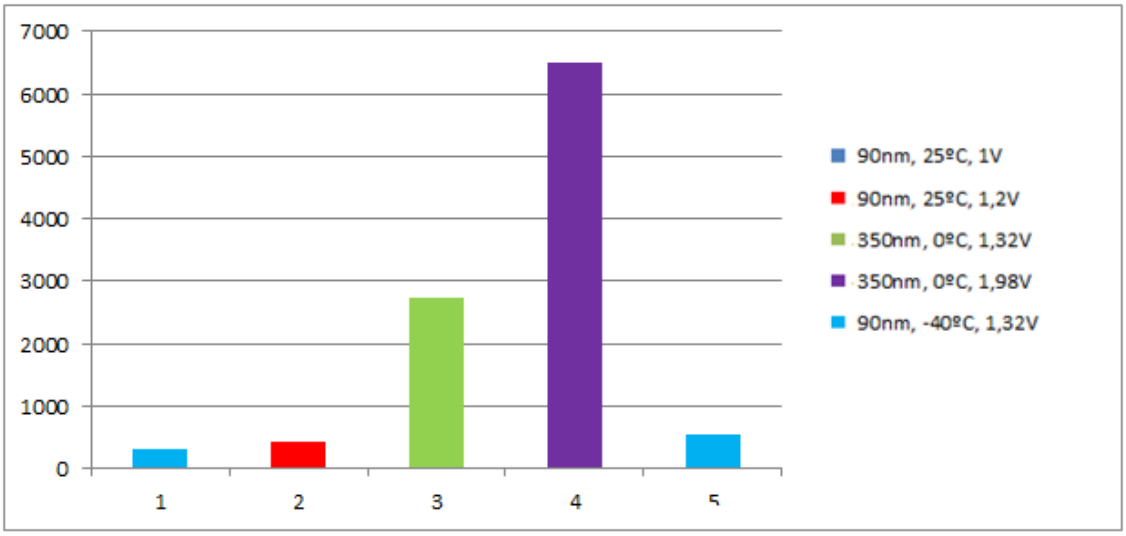


Figura 21. Gráfico potencia arquitectura de 160 bits con dos sumadores

5.2.2. Resultados síntesis arquitectura con un sumador

En este apartado se mostrarán todos los resultados obtenidos en cuanto a área y potencia de la arquitectura de un sumador, variando la tensión de alimentación dentro de las diferentes tecnologías, por lo que obtendremos resultados que nos permitirán comparar el consumo y el área dentro de cada tecnología a diferentes tensiones y entre las propias tecnologías. Las diferentes bibliotecas poseen una temperatura distinta, por lo que además del efecto del cambio de tecnología observaremos cómo afecta la temperatura al circuito. Además se realizará un estudio a una temperatura de frío extremo para ver el comportamiento del circuito y cómo varía respecto a una temperatura más común.

Con esta arquitectura lo que se pretende es conseguir el menor área posible, utilizando para ello un sumador en lugar de dos sumadores en cascada, con el fin de reducir el área que ocupa el circuito, siempre teniendo en cuenta el consumo.

De los datos recopilados se obtiene la tabla resumen con los resultados de la arquitectura, sintetizados en función del número de bits y de las distintas bibliotecas utilizadas para realizar las pruebas a diferentes tensiones de alimentación y tecnología.

	biblioteca faraday 90nm 1v25 (25ºC, 1V)		biblioteca faraday 90nm 2v25 (25ºC, 1,2V)	
	WIDHT_state = 64 bits	WIDHT_state = 160 bits	WIDHT_state = 64 bits	WIDHT_state = 160 bits
Área (μm^2)	5416,656013	14732,14399	5415,872012	10416,224071
Potencia interna (nW)	111,245	263,1907	161,4956	372,198600
Potencia conmutación (nW)	17,7874	44,468	26,7924	43,857500
Potencia dinámica total (nW)	129,0324	307,6586	188,2879	416,056100
Pérdidas de potencia (nW)	8,2514	21,8414	13,1607	25,677900
Puertas equivalentes	1714,13165	4662,070881	1713,883548	3296,273440
Tiempo de ejecución	274,5 (2745,0 ns)	546,5(5465ns)	274,5 (2745,0 ns)	546,5(5465 ns)
Throughput (Kbps)	23,315	29,28	23,315	29,28
	biblioteca amis 1.32v (0ºC, 1,32V)		biblioteca amis 1.98v (0ºC, 1,98V)	
	WIDHT_state = 64 bits	WIDHT_state = 160 bits	WIDHT_state = 64 bits	WIDHT_state = 160 bits
Área (μm^2)	2035,666885	4479,999464	2226,999383	4463,998664
Potencia interna (µW)	1,1532	2,772000	2,684900	6,5345
Potencia conmutación (nW)	53,2042	104,258300	99,187900	246,9003
Potencia dinámica total (µW)	1,2064	2,876200	2,784100	6,7814
Pérdidas de potencia (nW)	21,8298	24,370200	17,0792	28,6126
Puertas equivalentes	2035,666885	4479,999464	2226,999383	4463,998664
Tiempo de ejecución	274,5 (2745,0 ns)	546,5(5465 ns)	274,5 (2745,0 ns)	546,5(5465 ns)
Throughput (Kbps)	23,315	29,28	23,315	29,28
	biblioteca faraday 90nm 1p32vm40 (-40ºC, 1,32V)			
	WIDHT_state = 64 bits	WIDHT_state = 160 bits		
Área (μm^2)	5418,224012	10536,96007		
Potencia interna (nW)	147,7314	461,4749		
Potencia conmutación (nW)	112,6632	55,9471		
Potencia dinámica total (nW)	260,3946	517,422		
Pérdidas de potencia (nW)	9,4779	17,7177		
Puertas equivalentes	1714,627852	3334,48103		
Tiempo de ejecución	274,5 (2745,0 ns)	546,5(546 5ns)		
Throughput (Kbps)	23,315	29,28		

TABLA 2. Tabla resumen arquitectura un sumador

Observando la tabla 2 se comprueba que todos los resultados de 64 bits para las diferentes bibliotecas cumplen con las especificaciones de 4000 GE y el límite de 10 µW de potencia, por lo que podríamos instalar el protocolo en un sistema de autenticación RFID.

Al igual que en la arquitectura con los dos sumadores en cascada, a medida que aumentamos la tensión de alimentación y la tecnología, el área necesaria aumenta y el consumo hace lo propio, por lo que el circuito tiene peores características para el objetivo perseguido.

Para 64 bits volvemos a observar que cumple perfectamente las especificaciones de las 4000 puertas y los 10 µW de consumo, hecho que no ocurre para 160 bits, ya que solo se sitúa por debajo de 4000 puertas en dos casos: para una tensión de 1,2V y para una temperatura extrema de -40ºC, y, por tanto, solo sería posible implementarlo en esos casos.

La utilización de 160 bits ofrece una mayor seguridad y la posibilidad de guardar claves y entradas con más dígitos y, por tanto, más diferenciadas. Tras realizar los test para 64 y 160 bits se observó que los resultados eran idénticos, por lo que la seguridad para 64 bits cumple con las especificaciones de forma satisfactoria y no será necesario emplear 160 bits.

El aumento de la potencia y de la tecnología hace que el tiempo de ejecución sea menor, aunque no es una característica clave de nuestro circuito, ya que con la velocidad de ejecución alcanzada con una menor tensión de alimentación es suficiente.

Para recopilar los datos y poder observarlos con mayor facilidad, en la figura 16 y 17 se realiza un gráfico de las puertas equivalentes y el consumo para esta arquitectura, con el fin de poder compararlos visualmente.

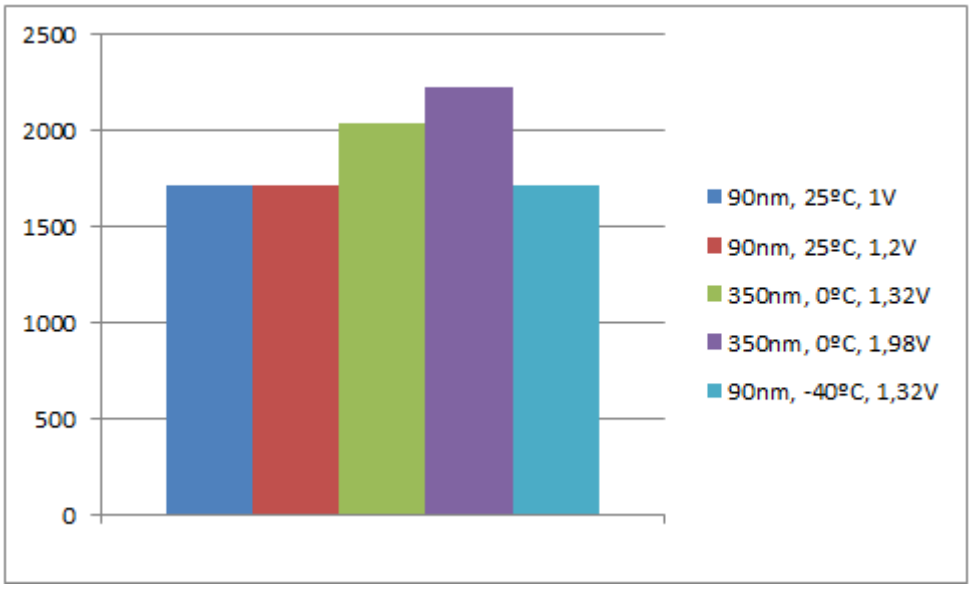


Figura 22. Gráfico puertas equivalentes arquitectura de 64 bits y un sumador

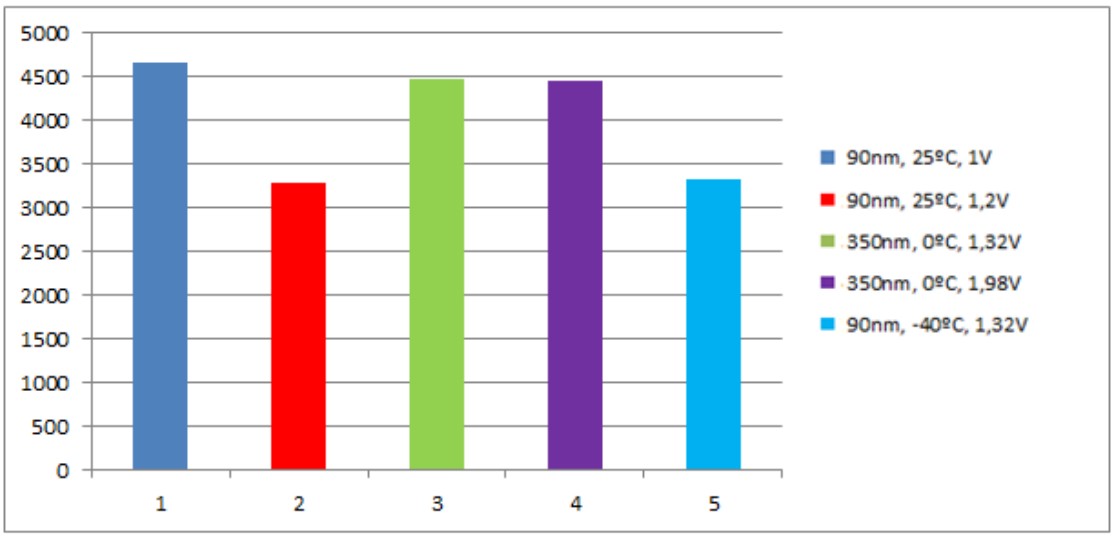


Figura 23. Gráfico puertas equivalentes arquitectura de 160 bits y un sumador

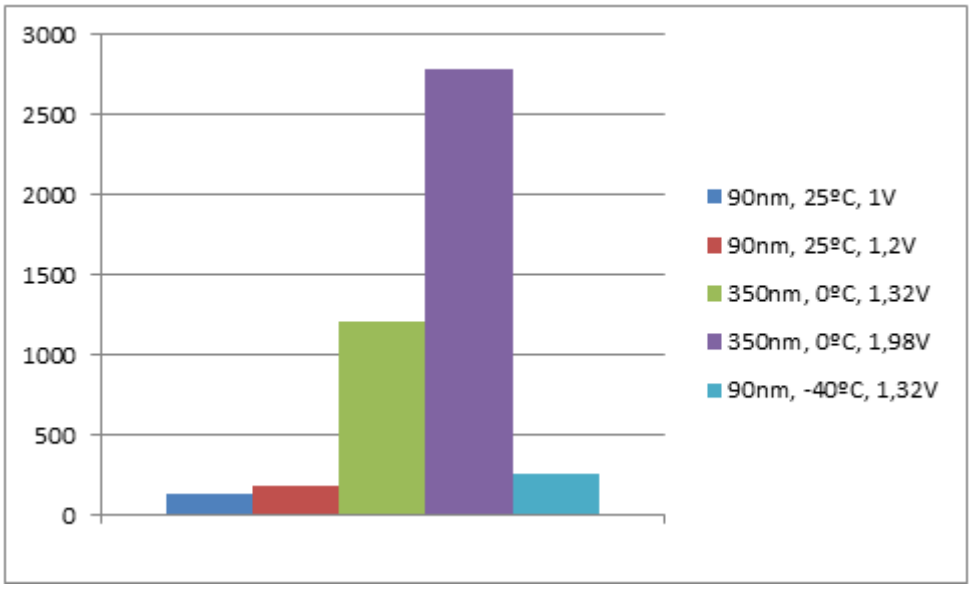


Figura 24. Gráfico potencia arquitectura de 64 bits y un sumador

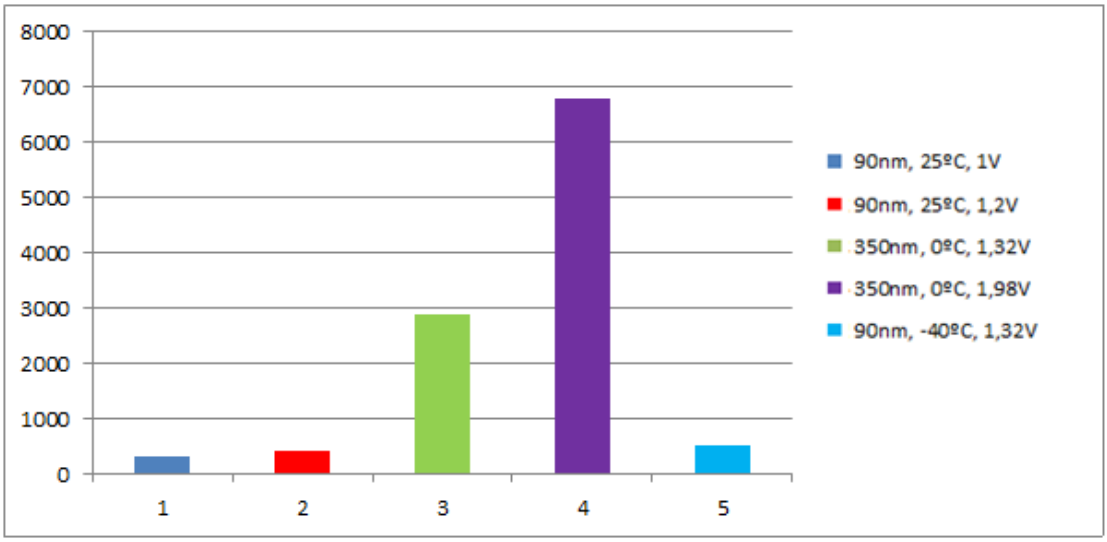


Figura 25. Gráfico potencia arquitectura de 160 bits y un sumador

5.2.3. Comparación de arquitecturas

Una vez expuestos los resultados para cada arquitectura utilizando las diferentes bibliotecas, se realiza una comparativa entre ambas, con el fin de determinar cuál de ellas cumple mejor las especificaciones perseguidas.

5.2.3.1. Área

El área de las arquitecturas dependerá de los registros necesarios para implementarla, así como del tamaño de los elementos que componen dicho circuito y la tecnología que utilicemos.

Con las tablas 1 y 2, se observa que la arquitectura del sumador de dos entradas ocupa un área de mayor tamaño. Esto, aunque inicialmente parezca contradictorio, se debe a que la lógica combinacional y los registros adicionales empleados para realizar las sumas y operaciones en más ciclos hacen que el área ocupada sea mayor que utilizando menos ciclos con un sumador con más entradas, por lo que la ganancia de área obtenida al suprimir un sumador, no se ve compensada por el aumento de área que supone la lógica adicional necesaria.

Cabe destacar que el área también dependerá de la temperatura de trabajo de los circuitos, ya que a mayor temperatura el calor que se ha de disipar también será más elevado y por tanto los encapsulados de los circuitos aumentarán de tamaño. Esto se puede observar en los resultados obtenidos en cada una de las arquitecturas, ya que a pesar de aumentar la tensión de alimentación hasta 1,32V, gracias a la temperatura de -40ºC el circuito no presenta cambios de área frente a la alimentación de 1V.

5.2.3.2. Consumo de potencia

En cuanto a la potencia consumida, son varios los factores que influyen en ella. La potencia variará en función de la lógica del circuito, puesto que para realizar cambios en los registros o comunicar una parte del circuito con otra se requerirá de ella. Otro factor a tener en cuenta es la tensión de alimentación. A mayor tensión de alimentación el circuito funcionará de forma más veloz y con un menor ruido, pero el consumo por parte de los componentes del circuito también será más elevado. El número de puertas es otro factor a tener en cuenta en un circuito, ya que cada puerta lógica requiere de tensión de alimentación y, por tanto, a mayor número de puertas lógicas el consumo será más elevado.

La potencia también dependerá de la temperatura [40] ambiente y de la temperatura del circuito de la forma siguiente:

$$P = \frac{Tj - Ta}{\theta ja}$$

Donde:

P potencia

Tj mx temperatura en el CI

Ta temperatura ambiente

θja resistencia térmica

Además de estas dos arquitecturas, se estudió la posibilidad de realizar el circuito con la tecnología denominada clock gating, la cual puede permitir la reducción del consumo de potencia.

El clock gating [41] consiste en una técnica a través de la cual es posible reducir la capacidad a conmutar mediante la deshabilitación total o parcial de los módulos que en ese momento no deban realizar ninguna computación. Esto no debe limitarse al uso de enables, ya que una parte del consumo dinámico seguirá produciéndose si la señal de reloj que controla al bloque sigue conmuntando todas las entradas de reloj del mismo. La inhabilitación de la señal de reloj de un bloque cuando este no necesita realizar operaciones es lo que se conoce como clock gating.

En el presente proyecto se estudió la posibilidad de realizar una tercera arquitectura mediante el uso de clock gating, pero se rechazó la posibilidad debido a que a una frecuencia de 100KHz el consumo dinámico no es muy elevado y la técnica de clock gating no resultaría muy ventajosa.

5.2.6.2. Ciclos de reloj / throughput

Se requiere de más ciclos de reloj en la arquitectura de un sumador de dos entradas, ya que al suprimir un sumador, se requiere de una máquina de estados que posea más estados y por lo tanto necesite más ciclos de reloj para realizar las operaciones.

El throughput es menor también en la arquitectura de un sumador de dos entradas, puesto que para sacar la el mismo número de bits que con las arquitecturas de dos sumadores en cascada, el circuito necesita de más tiempo para finalizar las operaciones, por lo que el número de kilo bytes por segundo será menor.

CAPÍTULO 6

CONCLUSIONES Y LÍNEAS FUTURAS

6.1. CONCLUSIONES

El objetivo de este trabajo es el de la implementación y el análisis de protocolos de autenticación cifradores de flujo. Dichos objetivos se han alcanzado con éxito a lo largo del presente proyecto.

Inicialmente se ha estudiado el cifrador de flujo para que fuera óptimo para la utilización en tecnología rfid, teniendo en cuenta las limitaciones de área y potencia que esta impone. Posteriormente se han implementado en VHDL dos arquitecturas para diferentes números de bits, con el fin de realizar una comparación entre el consumo y el área de cada una de ellas, determinando que arquitectura resultaría mejor para implementar en un sistema rfid.

Finalmente se realizaron test para comprobar que la encriptación del código es segura y por lo tanto es difícil realizar ataques con éxito a los algoritmos implementados a lo largo del proyecto.

Por tanto los objetivos siguientes han quedado cumplidos:

- Estudio del cifrador de flujo: se explica y se presenta el estudio que se ha realizado sobre el cifrador de flujo y las características básicas que este debe poseer.

- Implementación: se realizaron las distintas implementaciones para cada una de las arquitecturas con la herramienta synopsys y el lenguaje de programación VHDL.

- Simulación y análisis: se simularon los circuitos y se comprobó que los resultados que ofrecían eran los correctos. Se obtuvieron los términos de potencia y área y se determinaron las arquitecturas más convenientes para el caso de estudio.

- Test de aleatoriedad y correlación: se realizaron los test de aleatoriedad y correlación, quedando demostrado que la salida del algoritmo mostraba una completa aleatoriedad y no tenía correlación con la clave de entrada, lo que hace del sistema implementado un sistema seguro ante posibles ataques a la red.

6.2. LÍNEAS FUTURAS

Se proponen como trabajos futuros los siguientes puntos:

- Búsqueda y diseño de nuevos cifradores de flujo que puedan ser implementados en sistemas rfid, con mejor relación consumo/área y mayor seguridad ante posibles ataques.

- Implementación de un generador de números aleatorios robusto que alimente al cifradores de flujo con claves aleatorias.

- Realizar la síntesis con distintas librerías que las realizadas en este proyecto con el fin de comparar las diferentes arquitecturas a distintas temperaturas y tensiones de alimentación.

- Diseño del circuito con clock gating, que aunque no suponga un ahorro significativo de la energía, puede suponer cierta mejora en este aspecto.

- Realización de nuevos test que puedan corroborar la seguridad de los algoritmos implementados frente a ataques al sistema.

CAPÍTULO 7

PRESUPUESTO

En el presente capitulo se calculan los costes que se han generado para realizar el proyecto, incluyendo costes materiales y personales [42]. En cuanto a los segundos, se tomará un coste de de ingeniero de 1500 euros al mes, teniendo en cuenta un 50%, ya que no se trabajó las 8 horas al día que implicarían los 1500 euros. Los costes materiales incluirán el ordenador y los programas utilizados.

Para los costes imputables al proyecto, se aplicará la fórmula de amortización mostrada a continuación:

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

Siendo:

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

B = periodo de depreciación (meses).

C = coste del equipo (sin IVA).

D = % del uso que se dedica al proyecto.

El periodo de depreciación de los equipos informáticos es de unos 60 meses y la licencia de los programas de un año (12 meses). Los costes indirectos suponen un 20% de los costes totales. Por tanto los gastos serán los siguientes:

AUTOR:	Fco. Javier León García				
PROYECTO:	Implementación y análisis de cifradores de flujo para tecnología RFID				
GASTOS PERSONALES					
Personal	Categoría	Meses trabajados	Coste/mes	dedicación	Total (euros)
Fco. Javier León García	Ingeniero	6	1500	50%	4500
GASTOS MATERIALES					
Equipo	Coste (euros)	% Uso dedicado al proyecto	meses trabajados	Periodo depreciación	Costes imputables
PC	700	50	7	60	15
Licencia Synopsys	1800	100	7	12	500
Licencia Modelsim	100	100	7	12	900
Costes materiales	1415				

RESUMEN	EUROS
Personal	4500
Amortización	1415
Coste indirecto	1473
Total	**7388**

CAPÍTULO 8

BIBLIOGRAFÍA

[1] Patrick J. Sweeney II, RFID for Dummies, wiley Publishing.

[2] Blazquez del Toro, L. M.: “Sistemas de identificación por radiofrecuencia”.

[3] Cristina Fernández, R.: “Estudio de la tecnología RFID y desarrollo de una aplicación para localización de personas.”, Proyecto fin de carrera. Universidad Carlos III Madrid, julio 2009.

[4] http://www.dipolerfid.com/products/RFID_readers/RFID_antennas/Default.aspx

[5] http://www.aidet.es/productos/rfid-lectores-etiquetas-accesos-impresoras-tags-tarjetas-pulseras.html

[6] Klaus Finkenzeller: RFID Handbook: Fundamentals and applications in Contactless Smart Cards and Identification

[7] http://html.rincondelvago.com/rfid.html

[8] Pete Sorrells: “Pasive RFID Basics”, Microchip technology Inc.

[9] V. Daniel Hunt, Albert Puglia, Wiley Interscience: RFID A guide to Radiofrecuency Identification, 2007.

[10] José María Ciudad Herrera, Eduard Samà Casanovas: “ Estudio, diseño y simulación de un sistema de RFID basado en EPC”

[11] http://www.tadic-solutions.com/rfid/

[12] http://www.datacollection.eu/contents/articles/es/20060109/rfid_normativas_y_estandares

[13] http://www.rfidpoint.com/fundamentos/el-estandar-epc/.

[14] EPC Tag Data Standards Version 1.1, Standard specification 01, april 2004. EPCglobal inc.

[15] http://www.informatica-hoy.com.ar/rfid/Ventajas-de-las-tecnologias-RFID.php

[16] Chavarrías López, J.L.: “Implementación Hardware de función resumen para tecnología RFID”, Proyecto fin de carrera, Carlos III Madrid, octubre 2012.

[17] Godínez, Miguel: RFID: Oportunidades y riesgos, su aplicación práctica; Primera edición, Alfaomega, México D.F. 2007.

[18] Gotor Carrasco, E.: “Estado del arte en tecnologías RFID”, Proyecto Fin de Carrera, Universidad Politécnica de Madrid, Junio 2009.

[19] Instituto nacional de tecnologías de la comunicación: “Guía sobre seguridad y privacidad de la tecnología RFID”.

[20] 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

[21] http://www.monografias.com/trabajos7/protoip/protoip.shtml

[22] http://www.upv.es/satelite/trabajos/Grupo13_99.00/tema5.html

[23] Universidad politécnica de Madrid, departamento de sistemas electrónicos y control: “introducción al lenguaje VHDL”.

http://www.eweb.unex.es/eweb/fisteor/antonio_astillero/ec/vhdl/Manual%20VHDL.pdf

[24] Lluís Terés, Yago Torroja, Serafín Olcoz, Eugenio Villar: “VHDL Lenguaje estándar de diseño electrónico”, Ed. Mc Graw-Hill, 1998

[25] Juán González, UAM: “Introducción al lenguaje de descripción Hardware VHDL”.

[26] Design Compiler® Tutorial Using Design Vision™

[27] Departamento de Tecnología Electrónica de la escuela politécnica superior de la UC3M, de Casado Ortiz, Fernando y Mengibar Pozo, Luis : “Manual básico de uso de la herramienta ModelSim”

[28] Departamento de Tecnología Electrónica de la escuela politécnica superior de la UC3M, de Casado Ortiz, Fernando y Mengibar Pozo, Luis.: “Xilinx ISE. Manual básico”

[29] Xilinx Inc., “ModelSim SE User´s Manual”, 2006

[30] http://www.cannic.uab.es/docencia/vhdli06/TutorialModelsim.htm

[31] “Design Compiler. User Guide”, Marzo 2007, Synopsys Inc

[32] Bruce Schneier : Applied Cryptography, segunda edición.

[33] http://guimi.net/monograficos/G-Redes_de_comunicaciones/G-RCnode63.html

[34] José María Sierra: “protocolos de autenticación en redes”, departamento de informática Universidad Carlos III de Madrid.

[35] Kwak, M., Kim, J. y Kim, K.: “Desynchronization and cloning resistant lightweight RFID authentication protocolo using integer arithmetic for low-cost tags”, Universidad de Información y Comunicaciones de Munji-Dong, Korea.

[36] http://seguridadinformaticaequipo7.blogspot.com.es/2012/10/cifrado-en-flujo.html

[37] http://campusvirtual.unex.es/cala/epistemowikia/index.php?title=Encriptaci%C3%B3n/Desencriptaci%C3%B3n#Cifrado_de_flujo

[38] Meltem S¨onmez Turan, Ali Do˘ganaksoy, C¸ a˘gda¸s C¸ alık, Institute of Applied Mathematics, Middle East Technical University, Ankara, Turkey: “Detailed Statistical Analysis of Synchronous Stream Ciphers”

[39] http://www.mathworks.es/es/help/stats/chi2gof.html

[40] Juan Lanchares: “diseño de circuitos integrados I”.

[41] universidad politécnica de Cataluña: diseño de circuitos y sistemas integrados.

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

ANEXOS

ANEXO A – CÓDIGO ARQUITECTURA CON DOS SUMADORES DE 2 ENTRADAS y 64 BITS

```
------------------Entidad Control-----------------
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity Control is
Port ( Clk : in STD_LOGIC;
Reset : in STD_LOGIC;
Entrada : in STD_LOGIC_VECTOR (63 downto 0);
Inicio : in STD_LOGIC;
Key : in STD_LOGIC_VECTOR (63 downto 0);
Final : out STD_LOGIC;
Salida : out STD_LOGIC_VECTOR (63 downto 0));
end Control;
architecture Behavioral of Control is

component Contador is
Port (Clk : in STD_LOGIC;
Enable : in STD_LOGIC;
Reset : in STD_LOGIC;
Cuenta : out STD_LOGIC_VECTOR (3 downto 0));
end component;
component Sumador is
Port (A : in STD_LOGIC_VECTOR (3 downto 0);
B : in STD_LOGIC_VECTOR (3 downto 0);
C : in STD_LOGIC_VECTOR (3 downto 0);
Suma : out STD_LOGIC_VECTOR (3 downto 0));
end component;

Signal Contador_Enable: STD_LOGIC;
Signal Cuenta: STD_LOGIC_VECTOR (3 downto 0);
Signal Sumador_A: STD_LOGIC_VECTOR (3 downto 0);
Signal Sumador_B: STD_LOGIC_VECTOR (3 downto 0);
```

```
Signal Sumador_C: STD_LOGIC_VECTOR (3 downto 0);
Signal Sumador_Salida: STD_LOGIC_VECTOR (3 downto 0);
Signal X: STD_LOGIC_VECTOR (3 downto 0);
Signal Y: STD_LOGIC_VECTOR (3 downto 0);
Signal Z: STD_LOGIC_VECTOR (3 downto 0);
Signal T: STD_LOGIC_VECTOR (3 downto 0);
Signal a_x: STD_LOGIC_VECTOR (3 downto 0);
Signal a_y: STD_LOGIC_VECTOR (3 downto 0);
Signal a_z: STD_LOGIC_VECTOR (3 downto 0);
Signal a_t: STD_LOGIC_VECTOR (3 downto 0);
type dato is array (0 to 15) of std_logic_vector (3 downto 0);
Signal K: STD_LOGIC_VECTOR (63 downto 0);
Signal a_k: STD_LOGIC_VECTOR (63 downto 0);
Signal s: STD_LOGIC_VECTOR (63 downto 0);
Signal a_s: STD_LOGIC_VECTOR (63 downto 0);
type estados is (inicia, guardar, sumax, sumay, sumaz, swap1, swap3, swap4, inicio2,
sumax2, sumay2, sumaz2, sumat, swap2, estado_XOR1, estado_XOR2);
signal estado_actual: estados;
signal estado_siguiente: estados;
Signal registro_aux_s: STD_LOGIC_VECTOR (3 downto 0);
Signal a_registro_aux_s: STD_LOGIC_VECTOR (3 downto 0);
Signal registro_salida: STD_LOGIC_VECTOR (63 downto 0);
Signal a_registro_salida: STD_LOGIC_VECTOR (63 downto 0);
Signal sumas: STD_LOGIC_VECTOR (3 downto 0);
Signal a_sumas: STD_LOGIC_VECTOR (3 downto 0);

begin

Contador_h: Contador port map (Clk => Clk,
Enable => Contador_Enable,
Reset => Reset,
Cuenta => Cuenta);

Sumador_h: Sumador port map (A => Sumador_A,
B => Sumador_B,
C => Sumador_C,
Suma => Sumador_Salida);

process (Clk, Reset)
begin
```

```
if (reset = '1') then
X <= "0000";
Y <= "0000";
Z <= "0000";
T <= "0000";
elsif Clk'event and Clk='1' then
X <= a_x;
Y <= a_y;
Z <= a_z;
T <= a_t;
end if;
end process;

process (Clk, Reset)
begin
if (reset = '1') then
s <= (others => '0');
elsif Clk'event and Clk='1' then
s <= a_s;
end if;
end process;

process (Clk, Reset)
begin
if (reset = '1') then
registro_aux_s <= "0000";
elsif Clk'event and Clk='1' then
registro_aux_s <= a_registro_aux_s;
end if;
end process;

process (Clk, Reset)
begin
if (reset = '1') then
sumas <= "0000";
elsif Clk'event and Clk='1' then
sumas <= a_sumas;
end if;
end process;
```

```
process (Clk, Reset)
begin
if (reset = '1') then
registro_salida <= (others => '0');
elsif Clk'event and Clk='1' then
registro_salida <= a_registro_salida;
end if;
end process;

process (Clk, Reset)
begin
if (reset = '1') then
k <= (others => '0');
elsif Clk'event and Clk='1' then
k <= a_k;
end if;
end process;

process (Clk, Reset)
begin
if (Reset='1') then
estado_actual <= inicia;
elsif(Clk'event and Clk='1') then
estado_actual <= estado_siguiente;
end if;
end process;

process (estado_actual, cuenta, Key, k, s, X, Y, Z, T, Sumador_Salida, Entrada, Sumas,
registro_aux_s, Sumas, registro_salida, Inicio)
begin
a_s <= s;
a_registro_salida <= registro_salida;

case estado_actual is
when inicia =>
a_x <= "0000";
a_y <= "0000";
a_z <= "0000";
a_t <= "0000";
```

```
a_registro_aux_s <= "0000";
Contador_Enable <='0';
a_k <= Key;
a_s <=
"1111111011011100101110101001100001110110010101000011001000010000";
a_registro_salida <= (others => '0');
Sumador_A <= "0000";
Sumador_B <= "0000";
Sumador_C <= "0000";
a_sumas <= "0000";
Salida <= registro_salida;
Final <= '0';
if Inicio = '1' then
estado_siguiente <= guardar;
else
estado_siguiente <= inicia;
end if;
when guardar =>
Salida <= (others => '0');
a_registro_aux_s <= registro_aux_s;
Contador_Enable <='0';
a_k <= Key;
a_s <=
"1111111011011100101110101001100001110110010101000011001000010000";
a_registro_salida <= registro_salida;
a_x <= X;
a_y <= Y;
a_z <= Z;
a_t <= T;
Sumador_A <= "0000";
Sumador_B <= "0000";
Sumador_C <= "0000";
a_sumas <= sumas;
Final <= '0';
estado_siguiente <= sumax;
when sumax =>
Salida <= (others => '0');
a_registro_salida <= registro_salida;
a_registro_aux_s <= registro_aux_s;
a_k <= k;
```

```
a_s <= s;
Sumador_A <= X;
Sumador_B <= s((((conv_integer (cuenta))*4)+3) downto ((conv_integer (
cuenta))*4));
Sumador_C <= K((((conv_integer (cuenta))*4)+3) downto ((conv_integer (
cuenta))*4));
a_x <= Sumador_Salida;
a_y <= Y;
a_z <= Z;
a_t <= T;
a_sumas <= sumas;
Contador_Enable <='0';
Final <= '0';
estado_siguiente <= sumay;
when sumay =>
Salida <= (others => '0');
a_registro_salida <= registro_salida;
a_registro_aux_s <= s((((conv_integer (cuenta))*4)+3) downto ((
conv_integer (cuenta))*4));
a_k <= k;
a_s <= s;
Sumador_A <= Y;
Sumador_B <= s((((conv_integer (X))*4)+3) downto ((conv_integer (X))*4
));
Sumador_C <= K((((conv_integer (X+cuenta))*4)+3) downto ((conv_integer
(X+cuenta))*4));
a_x <= X;
a_y <= Sumador_Salida;
a_z <= Z;
a_t <= T;
a_sumas <= sumas;
Contador_Enable <='0';
Final <= '0';
estado_siguiente <= sumaz;
when sumaz =>
Salida <= (others => '0');
a_registro_salida <= registro_salida;
a_registro_aux_s <= registro_aux_s;
a_k <= k;
a_s((((conv_integer (cuenta))*4)+3) downto ((conv_integer (cuenta))*4))
```

```
<= s((((conv_integer (X))*4)+3) downto ((conv_integer (X))*4));
a_s((((conv_integer (X))*4)+3) downto ((conv_integer (X))*4)) <=
registro_aux_s;
Sumador_A <= Z;
Sumador_B <= s((((conv_integer (Y))*4)+3) downto ((conv_integer (Y))*4
));
Sumador_C <= K((((conv_integer (Y+cuenta))*4)+3) downto ((conv_integer
(Y+cuenta))*4));
a_x <= X;
a_y <= Y;
a_z <= Sumador_Salida;
a_t <= T;
a_sumas <= sumas;
Contador_Enable <= '0';
Final <='0';
estado_siguiente <= swap1;
when swap1 =>
Salida <= (others => '0');
a_registro_salida <= registro_salida;
a_registro_aux_s <= s((((conv_integer (Y))*4)+3) downto ((conv_integer
(Y))*4));
Contador_Enable <='0';
a_k <= k;
a_s <= s;
Sumador_A <= "0000";
Sumador_B <= "0000";
Sumador_C <= "0000";
a_x <= X;
a_y <= Y;
a_z <= Z;
a_t <= T;
a_sumas <= sumas;
Final <= '0';
estado_siguiente <= swap2;


when swap2 =>
Salida <= (others => '0');
a_registro_salida <= registro_salida;
a_registro_aux_s <= registro_aux_s;
```

```
Contador_Enable <='1';
a_k <= k;
Sumador_A <= "0000";
Sumador_B <= "0000";
Sumador_C <= "0000";
a_x <= X;
a_y <= Y;
a_z <= Z;
a_t <= T;
a_sumas <= sumas;
a_s((((conv_integer (Y))*4)+3) downto ((conv_integer (Y))*4)) <= s((((
conv_integer (Z))*4)+3) downto ((conv_integer (Z))*4));
a_s((((conv_integer (Z))*4)+3) downto ((conv_integer (Z))*4)) <=
registro_aux_s;
Final <= '0';
if cuenta="1111" then
estado_siguiente <= inicio2;
else
estado_siguiente <= sumax;
end if;

when inicio2 =>
Salida <= (others => '0');
a_registro_salida <= registro_salida;
a_registro_aux_s <= registro_aux_s;
a_x <= "0000";
a_y <= "0000";
a_z <= "0000";
a_t <= "0000";
Contador_Enable <='0';
a_k <= k;
a_s <= s;
a_sumas <= sumas;
Sumador_A <= "0000";
Sumador_B <= "0000";
Sumador_C <= "0000";
Final <= '0';
estado_siguiente <= sumax2;
when sumax2 =>
a_registro_salida <= registro_salida;
```

```
Salida <= (others => '0');
a_registro_aux_s <= registro_aux_s;
a_k <= k;
a_s <= s;
Contador_Enable <='0';
Sumador_A <= X;
Sumador_B <= "0001";
Sumador_C <= "0000";
a_sumas <= sumas;
a_x <= Sumador_Salida;
a_y <= Y;
a_z <= Z;
a_t <= T;
Final <= '0';
estado_siguiente <= sumay2;
when sumay2 =>
a_registro_salida <= registro_salida;
Salida <= (others => '0');
a_registro_aux_s <= registro_aux_s;
a_k <= k;
a_s <= s;
Sumador_A <= Y;
Sumador_B <= s((((conv_integer (X))*4)+3) downto ((conv_integer (X))*4
));
Sumador_C <= "0000";
a_sumas <= sumas;
a_x <= X;
a_y <= Sumador_Salida;
a_z <= Z;
a_t <= T;
Contador_Enable <='0';
Final <='0';
estado_siguiente <= sumaz2;
when sumaz2 =>
a_registro_salida <= registro_salida;
Salida <= (others => '0');
a_registro_aux_s <= s((((conv_integer (X))*4)+3) downto ((conv_integer
(X))*4));
a_k <= k;
a_s <= s;
```

```
Sumador_A <= Z;
Sumador_B <= s((((conv_integer (Y))*4)+3) downto ((conv_integer (Y))*4
));
Sumador_C <= "0000";
a_sumas <= sumas;
a_x <= X;
a_y <= Y;
a_z <= Sumador_Salida;
a_t <= T;
Contador_Enable <= '0';
Final <= '0';
estado_siguiente <= sumat;
when sumat =>
a_registro_salida <= registro_salida;
Salida <= (others => '0');
a_registro_aux_s <= registro_aux_s;
a_k <= k;
a_s((((conv_integer (X))*4)+3) downto ((conv_integer (X))*4)) <= s((((
conv_integer (Y))*4)+3) downto ((conv_integer (Y))*4));
a_s((((conv_integer (Y))*4)+3) downto ((conv_integer (Y))*4)) <=
registro_aux_s;
Sumador_A <= T;
Sumador_B <= s((((conv_integer (T))*4)+3) downto ((conv_integer (T))*4
));
Sumador_C <= "0000";
a_sumas <= sumas;
a_x <= X;
a_y <= Y;
a_z <= Z;
a_t <= Sumador_Salida;
Contador_Enable <='0';
Final <='0';
estado_siguiente <= swap3;
when swap3 =>
a_registro_salida <= registro_salida;
Salida <= (others => '0');
a_registro_aux_s <= s((((conv_integer (Z))*4)+3) downto ((conv_integer
(Z))*4));
a_k <= k;
a_s <= s;
```

```
Sumador_A <= "0000";
Sumador_B <= "0000";
Sumador_C <= "0000";
a_sumas <= Sumas;
a_x <= X;
a_y <= Y;
a_z <= Z;
a_t <= T;
Contador_Enable <= '0';
Final <= '0';
estado_siguiente <= swap4;
when swap4 =>
a_registro_salida <= registro_salida;
Salida <= (others => '0');
a_registro_aux_s <= registro_aux_s;
a_k <= k;
a_s((((conv_integer (Z))*4)+3) downto ((conv_integer (Z))*4)) <= s((((
conv_integer (T))*4)+3) downto ((conv_integer (T))*4));
a_s((((conv_integer (T))*4)+3) downto ((conv_integer (T))*4)) <=
registro_aux_s;
Sumador_A <= "0000";
Sumador_B <= "0000";
Sumador_C <= Sumas;
a_sumas <= Sumador_Salida;
a_x <= X;
a_y <= Y;
a_z <= Z;
a_t <= T;
Contador_Enable <= '0';
Final <= '0';
estado_siguiente <= estado_XOR1;
when estado_XOR1 =>
a_registro_salida <= registro_salida;
Salida <= (others => '0');
a_registro_aux_s <= registro_aux_s;
a_k <= k;
a_s <= s;
Sumador_A <= s((((conv_integer (X))*4)+3) downto ((conv_integer (X))*4
));
Sumador_B <= s((((conv_integer (Y))*4)+3) downto ((conv_integer (Y))*4
```

```
));
Sumador_C <= s((((conv_integer (Z))*4)+3) downto ((conv_integer (Z))*4
));
a_sumas <= Sumador_Salida + s((((conv_integer (T))*4)+3) downto ((
conv_integer (T))*4));
a_x <= X;
a_y <= Y;
a_z <= Z;
a_t <= T;
Contador_Enable <= '0';
Final <= '0';
estado_siguiente <= estado_XOR2;

when estado_XOR2 =>
a_registro_aux_s <= registro_aux_s;
a_k <= k;
a_s <= s;
Sumador_A <= "0000";
Sumador_B <= "0000";
Sumador_C <= "0000";
a_x <= X;
a_y <= Y;
a_z <= Z;
a_t <= T;
Contador_Enable <='1';
a_sumas <= sumas;
a_registro_salida(conv_integer (cuenta)*4) <= Entrada(conv_integer (
cuenta)*4) XOR s(conv_integer(sumas)*4);
a_registro_salida((conv_integer (cuenta)*4)+1) <= Entrada((conv_integer
(cuenta)*4+1)) XOR s((conv_integer(sumas)*4+1));
a_registro_salida((conv_integer (cuenta)*4)+2) <= Entrada((conv_integer
(cuenta)*4+2)) XOR s((conv_integer(sumas)*4+2));
a_registro_salida((conv_integer (cuenta)*4)+3) <= Entrada((conv_integer
(cuenta)*4+3)) XOR s((conv_integer(sumas)*4+3));
if cuenta="1111" then
Final <= '1';
Salida <= registro_salida;
estado_siguiente <= inicia;
else
Final <= '0';
```

```
Salida <= (others => '0');
estado_siguiente <= sumax2;
end if;

end case;
end process;
end Behavioral;
```

----------Entidad Contador----------

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity Contador is
Port ( Clk : in STD_LOGIC;
Enable : in STD_LOGIC;
Reset : in STD_LOGIC;
Cuenta : out STD_LOGIC_VECTOR (3 downto 0));
end Contador;

architecture Behavioral of Contador is
signal Count: std_logic_vector (3 downto 0);
begin
process (Clk, Enable, Reset)
begin
if (Reset='1') then
Count <= "0000";
elsif (Clk'event and Clk='1') then
if (Enable='1') then
Count <= Count + '1';
end if;
end if;
end process;
Cuenta <= Count;



end Behavioral;
```

```
------------------Entidad Sumador-----------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity Sumador is
Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
B : in STD_LOGIC_VECTOR (3 downto 0);
C : in STD_LOGIC_VECTOR (3 downto 0);
Suma : out STD_LOGIC_VECTOR (3 downto 0));
end Sumador;

architecture Behavioral of Sumador is

begin

process (A,B,C)
begin
Suma <= A+B+C;
end process;

end Behavioral;
```

ANEXO B – CÓDIGO ARQUITECTURA SUMADOR DE 2 ENTRADAS Y 64 BITS

```
------------------Entidad Control-----------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

34 entity Control is
35 Port ( Clk : in STD_LOGIC;
36 Reset : in STD_LOGIC;
37 Entrada : in STD_LOGIC_VECTOR (159 downto 0);
38 Inicio : in STD_LOGIC;
```

```
39 Key : in STD_LOGIC_VECTOR (159 downto 0);
40 Final: out STD_LOGIC;
41 Salida : out STD_LOGIC_VECTOR (159 downto 0));
42 end Control;

architecture Behavioral of Control is

component Contador is
Port (Clk : in STD_LOGIC;
Enable : in STD_LOGIC;
Reset : in STD_LOGIC;
Cuenta : out STD_LOGIC_VECTOR (4 downto 0));
end component;

component Sumador is
Port (A : in STD_LOGIC_VECTOR (4 downto 0);
B : in STD_LOGIC_VECTOR (4 downto 0);
Suma : out STD_LOGIC_VECTOR (4 downto 0));
end component;

Signal Contador_Enable: STD_LOGIC;
Signal Cuenta: STD_LOGIC_VECTOR (4 downto 0);
Signal Sumador_A: STD_LOGIC_VECTOR (4 downto 0);
Signal Sumador_B: STD_LOGIC_VECTOR (4 downto 0);
Signal Sumador_Salida: STD_LOGIC_VECTOR (4 downto 0);
Signal X: STD_LOGIC_VECTOR (4 downto 0);
Signal Y: STD_LOGIC_VECTOR (4 downto 0);
Signal Z: STD_LOGIC_VECTOR (4 downto 0);
Signal T: STD_LOGIC_VECTOR (4 downto 0);
Signal a_x: STD_LOGIC_VECTOR (4 downto 0);
Signal a_y: STD_LOGIC_VECTOR (4 downto 0);
Signal a_z: STD_LOGIC_VECTOR (4 downto 0);
Signal a_t: STD_LOGIC_VECTOR (4 downto 0);
Signal K: STD_LOGIC_VECTOR (159 downto 0);
Signal a_k: STD_LOGIC_VECTOR (159 downto 0);
Signal s: STD_LOGIC_VECTOR (159 downto 0);
Signal a_s: STD_LOGIC_VECTOR (159 downto 0);
76 type estados is (inicia, guardar, sumax, sumay, sumaz, swap1, swap3, swap4, inicio2,
sumax2, sumay2, sumaz2, sumat, swap2, estado_XOR, suma2x, suma2y, suma2z, sumas1,
sumas2);
```

```
signal estado_actual: estados;
signal estado_siguiente: estados;
Signal registro_aux_s: STD_LOGIC_VECTOR (4 downto 0);
Signal a_registro_aux_s: STD_LOGIC_VECTOR (4 downto 0);
Signal registro_salida: STD_LOGIC_VECTOR (159 downto 0);
Signal a_registro_salida: STD_LOGIC_VECTOR (159 downto 0);
Signal sumas: STD_LOGIC_VECTOR (4 downto 0);
Signal a_sumas: STD_LOGIC_VECTOR (4 downto 0);


begin

Contador_h: Contador port map (Clk => Clk,
Enable => Contador_Enable,
Reset => Reset,
Cuenta => Cuenta);

Sumador_h: Sumador port map (A => Sumador_A,
B => Sumador_B,
Suma => Sumador_Salida);

process (Clk, Reset)
begin
if (reset = '1') then
X <= (others => '0');
Y <= (others => '0');
Z <= (others => '0');
T <= (others => '0');
elsif Clk'event and Clk='1' then
X <= a_x;
Y <= a_y;
Z <= a_z;
T <= a_t;
end if;
end process;

process (Clk, Reset)
begin
if (reset = '1') then
s <= (others => '0');
```

```
elsif Clk'event and Clk='1' then
s <= a_s;
end if;
end process;

process (Clk, Reset)
begin
if (reset = '1') then
registro_aux_s <= (others => '0');
elsif Clk'event and Clk='1' then
registro_aux_s <= a_registro_aux_s;
end if;
end process;

process (Clk, Reset)
begin
if (reset = '1') then
sumas <= (others => '0');
elsif Clk'event and Clk='1' then
sumas <= a_sumas;
end if;
end process;


process (Clk, Reset)
begin
if (reset = '1') then
registro_salida <= (others => '0');
elsif Clk'event and Clk='1' then
registro_salida <= a_registro_salida;
end if;
end process;

process (Clk, Reset)
begin
if (reset = '1') then
k <= (others => '0');
elsif Clk'event and Clk='1' then
k <= a_k;
end if;
```

```
end process;

process (Clk, Reset)
begin
if (Reset='1') then
estado_actual <= inicia;
elsif(Clk'event and Clk='1') then
estado_actual <= estado_siguiente;
end if;
end process;

process (estado_actual, cuenta, Key, k, s, X, Y, Z, T, Sumador_Salida, Entrada, Sumas,
registro_aux_s, Sumas, registro_salida, Inicio)

begin
a_s <= s;
a_registro_salida <= registro_salida;

case estado_actual is
when inicia =>
a_x <= (others => '0');
a_y <= (others => '0');
a_z <= (others => '0');
a_t <= (others => '0');
a_registro_aux_s <= (others => '0');
Contador_Enable <='0';
a_k <= Key;
a_s <=
"11111111101110111100110111101011001110001011110110101011010010011100101000110000011110
11100110101100010110101001001010000011100110001010010000011000100000100000";
a_registro_salida <= (OTHERS => '0');
Sumador_A <= (OTHERS => '0');
Sumador_B <= (OTHERS => '0');
a_sumas <= (OTHERS => '0');
Salida <= registro_salida;
Final <= '0';
if Inicio = '1' then
estado_siguiente <= guardar;
else
```

```
estado_siguiente <= inicia;
end if;
when guardar =>
Salida <= (OTHERS => '0');
a_registro_aux_s <= registro_aux_s;
Contador_Enable <='0';
a_k <= Key;
a_s <=
"11111111101110111100110111101011001110001011110110101011010010011100101000110000011110
11100110101100010110101001001010000011100110001010010000011000100000100000";
a_registro_salida <= registro_salida;
a_x <= X;
a_y <= Y;
a_z <= Z;
a_t <= T;
Sumador_A <= (OTHERS => '0');
Sumador_B <= (OTHERS => '0');
a_sumas <= sumas;
Final <= '0';
estado_siguiente <= sumax;
when sumax =>
Salida <= (OTHERS => '0');
a_registro_salida <= registro_salida;
a_registro_aux_s <= registro_aux_s;
a_k <= k;
a_s <= s;
Sumador_A <= X;
Sumador_B <= s((((conv_integer (cuenta))*5)+4) downto ((conv_integer (
cuenta))*5));
a_x <= Sumador_Salida;
a_y <= Y;
a_z <= Z;

a_t <= T;
a_sumas <= sumas;
Contador_Enable <='0';
Final <= '0';
estado_siguiente <= suma2x;
when suma2x =>
```

```
Salida <= (OTHERS => '0');
a_registro_salida <= registro_salida;
a_registro_aux_s <= registro_aux_s;
a_k <= k;
a_s <= s;
Sumador_A <= X;
Sumador_B <= K((((conv_integer (cuenta))*5)+4) downto ((conv_integer (
cuenta))*5));
a_x <= Sumador_Salida;
a_y <= Y;
a_z <= Z;
a_t <= T;
a_sumas <= sumas;
Contador_Enable <='0';
Final <= '0';
estado_siguiente <= sumay;

when sumay =>
Salida <= (OTHERS => '0');
a_registro_salida <= registro_salida;
a_registro_aux_s <= registro_aux_s;
a_k <= k;
a_s <= s;
Sumador_A <= Y;
Sumador_B <= s((((conv_integer (X))*5)+4) downto ((conv_integer (X))*5
));
a_x <= X;
a_y <= Sumador_Salida;
a_z <= Z;
a_t <= T;
a_sumas <= sumas;
Contador_Enable <='0';
Final <= '0';
estado_siguiente <= suma2y;
when suma2y =>
Salida <= (OTHERS => '0');
a_registro_salida <= registro_salida;
a_registro_aux_s <= registro_aux_s;
a_k <= k;
a_s <= s;
```

```
Sumador_A <= Y;
Sumador_B <= K((((conv_integer (X+cuenta))*5)+4) downto ((conv_integer
(X+cuenta))*5));
a_x <= X;
a_y <= Sumador_Salida;
a_z <= Z;
a_t <= T;
a_sumas <= sumas;
Contador_Enable <='0';
Final <= '0';
estado_siguiente <= sumaz;
when sumaz =>
Salida <= (OTHERS => '0');
a_registro_salida <= registro_salida;
a_registro_aux_s <= s((((conv_integer (cuenta))*5)+4) downto ((
conv_integer (cuenta))*5));
a_k <= k;
a_s <= s;
Sumador_A <= Z;
Sumador_B <= s((((conv_integer (Y))*5)+4) downto ((conv_integer (Y))*5
));
a_x <= X;
a_y <= Y;
a_z <= Sumador_Salida;
a_t <= T;
a_sumas <= sumas;
Contador_Enable <= '0';
Final <= '0';
estado_siguiente <= suma2z;
when suma2z =>
Salida <= (OTHERS => '0');
a_registro_salida <= registro_salida;
a_registro_aux_s <= registro_aux_s;
a_k <= k;
a_s <= s;
a_s((((conv_integer (cuenta))*5)+4) downto ((conv_integer (cuenta))*5))
<= s((((conv_integer (X))*5)+4) downto ((conv_integer (X))*5));
a_s((((conv_integer (X))*5)+4) downto ((conv_integer (X))*5)) <=
registro_aux_s;
Sumador_A <= Z;
```

```
Sumador_B <= K((((conv_integer (Y+cuenta))*5)+4) downto ((conv_integer
(Y+cuenta))*5));
a_x <= X;
a_y <= Y;
a_z <= Sumador_Salida;
a_t <= T;
a_sumas <= sumas;
Contador_Enable <= '0';
Final <= '0';
estado_siguiente <= swap1;
when swap1 =>
Salida <= (OTHERS => '0');
a_registro_salida <= registro_salida;
a_registro_aux_s <= s((((conv_integer (Y))*5)+4) downto ((conv_integer
(Y))*5));
Contador_Enable <='0';
a_k <= k;
a_s <= s;
Sumador_A <= (OTHERS => '0');
Sumador_B <= (OTHERS => '0');
a_x <= X;
a_y <= Y;
a_z <= Z;
a_t <= T;
a_sumas <= sumas;
Final <= '0';
estado_siguiente <= swap3;

when swap3 =>
Salida <= (OTHERS => '0');
a_registro_salida <= registro_salida;
a_registro_aux_s <= registro_aux_s;
Contador_Enable <='1';
a_k <= k;
Sumador_A <= (OTHERS => '0');
Sumador_B <= (OTHERS => '0');
a_x <= X;
a_y <= Y;
a_z <= Z;
a_t <= T;
```

```
a_sumas <= sumas;
a_s((((conv_integer (Y))*5)+4) downto ((conv_integer (Y))*5)) <= s((((
conv_integer (Z))*5)+4) downto ((conv_integer (Z))*5));
a_s((((conv_integer (Z))*5)+4) downto ((conv_integer (Z))*5)) <=
registro_aux_s;
Final <= '0';
if cuenta="11111" then
estado_siguiente <= inicio2;
else
estado_siguiente <= sumax;
end if;

when inicio2 =>
Salida <= (OTHERS => '0');
a_registro_salida <= registro_salida;
a_registro_aux_s <= registro_aux_s;
a_x <= (OTHERS => '0');
a_y <= (OTHERS => '0');
a_z <= (OTHERS => '0');
a_t <= (OTHERS => '0');
Contador_Enable <='0';
a_k <= k;
a_s <= s;
a_sumas <= sumas;
Sumador_A <= (OTHERS => '0');
Sumador_B <= (OTHERS => '0');
Final <= '0';
estado_siguiente <= sumax2;
when sumax2 =>
a_registro_salida <= registro_salida;
Salida <= (OTHERS => '0');
a_registro_aux_s <= registro_aux_s;
a_k <= k;
a_s <= s;
Contador_Enable <='0';
Sumador_A <= X;
Sumador_B <= "00001";
a_sumas <= sumas;
a_x <= Sumador_Salida;
a_y <= Y;
```

```
a_z <= Z;
a_t <= T;
Final <= '0';
estado_siguiente <= sumay2;
when sumay2 =>
a_registro_salida <= registro_salida;
Salida <= (OTHERS => '0');
a_registro_aux_s <= registro_aux_s;
a_k <= k;
a_s <= s;
Sumador_A <= Y;
Sumador_B <= s((((conv_integer (X))*5)+4) downto ((conv_integer (X))*5
));
a_sumas <= sumas;
a_x <= X;
a_y <= Sumador_Salida;
a_z <= Z;
a_t <= T;
Contador_Enable <='0';
Final <= '0';
estado_siguiente <= sumaz2;
when sumaz2 =>
a_registro_salida <= registro_salida;
Salida <= (OTHERS => '0');
a_registro_aux_s <= s((((conv_integer (X))*5)+4) downto ((conv_integer
(X))*5));
a_k <= k;
a_s <= s;
Sumador_A <= Z;
Sumador_B <= s((((conv_integer (Y))*5)+4) downto ((conv_integer (Y))*5
));
a_sumas <= sumas;
a_x <= X;
a_y <= Y;
a_z <= Sumador_Salida;
a_t <= T;
Contador_Enable <= '0';
Final <= '0';
estado_siguiente <= sumat;
when sumat =>
```

```
a_registro_salida <= registro_salida;
Salida <= (OTHERS => '0');
a_registro_aux_s <= registro_aux_s;
a_k <= k;
a_s((((conv_integer (X))*5)+4) downto ((conv_integer (X))*5)) <= s((((
conv_integer (Y))*5)+4) downto ((conv_integer (Y))*5));
a_s((((conv_integer (Y))*5)+4) downto ((conv_integer (Y))*5)) <=
registro_aux_s;
Sumador_A <= T;
Sumador_B <= s((((conv_integer (T))*5)+4) downto ((conv_integer (T))*5
));
a_sumas <= sumas;
a_x <= X;
a_y <= Y;
a_z <= Z;
a_t <= Sumador_Salida;
Contador_Enable <='0';
Final <= '0';
estado_siguiente <= swap2;
when swap2 =>
a_registro_salida <= registro_salida;
Salida <= (OTHERS => '0');
a_registro_aux_s <= s((((conv_integer (Z))*5)+4) downto ((conv_integer
(Z))*5));
a_k <= k;
a_s <= s;
Sumador_A <= X;
Sumador_B <= Y;
a_sumas <= sumas;
a_x <= X;
a_y <= Y;
a_z <= Z;
a_t <= T;
Contador_Enable <= '0';
Final <= '0';
estado_siguiente <= swap4;
when swap4 =>
a_registro_salida <= registro_salida;
Salida <= (OTHERS => '0');
a_registro_aux_s <= registro_aux_s;
```

```
a_k <= k;
a_s((((conv_integer (Z))*5)+4) downto ((conv_integer (Z))*5)) <= s((((
conv_integer (T))*5)+4) downto ((conv_integer (T))*5));
a_s((((conv_integer (T))*5)+4) downto ((conv_integer (T))*5)) <=
registro_aux_s;
Sumador_A <= (OTHERS => '0');
Sumador_B <= (OTHERS => '0');
a_sumas <=Sumas;
a_x <= X;
a_y <= Y;
a_z <= Z;
a_t <= T;
Contador_Enable <= '0';
Final <= '0';
estado_siguiente <= sumas1;
when sumas1 =>
a_registro_salida <= registro_salida;
Salida <= (OTHERS => '0');
a_registro_aux_s <= registro_aux_s;
a_k <= k;
a_s <= s;
Sumador_A <= s((((conv_integer (X))*5)+4) downto ((conv_integer (X))*5
));
Sumador_B <= s((((conv_integer (Y))*5)+4) downto ((conv_integer (Y))*5
));
a_sumas <= Sumador_Salida;
a_x <= X;
a_y <= Y;
a_z <= Z;
a_t <= T;
Contador_Enable <= '0';
Final <= '0';
estado_siguiente <= sumas2;
when sumas2 =>
a_registro_salida <= registro_salida;
Salida <= (OTHERS => '0');
a_registro_aux_s <= registro_aux_s;
a_k <= k;
a_s <= s;
Sumador_A <= s((((conv_integer (Z))*5)+4) downto ((conv_integer (Z))*5
```

```
));
Sumador_B <= s((((conv_integer (T))*5)+4) downto ((conv_integer (T))*5
));
a_sumas <= Sumador_Salida + Sumas;
a_x <= X;
a_y <= Y;
a_z <= Z;
a_t <= T;
Contador_Enable <= '0';
Final <= '0';
estado_siguiente <= estado_XOR;

when estado_XOR =>
a_registro_aux_s <= registro_aux_s;
a_k <= k;
a_s <= s;
Sumador_A <= (OTHERS => '0');
Sumador_B <= (OTHERS => '0');
a_x <= X;
a_y <= Y;
a_z <= Z;
a_t <= T;
Contador_Enable <='1';
a_sumas <= sumas;
a_registro_salida(conv_integer (cuenta)*5) <= Entrada(conv_integer (
cuenta)*5) XOR s(conv_integer(sumas)*5);
a_registro_salida((conv_integer (cuenta)*5)+1) <= Entrada((conv_integer
(cuenta)*5+1)) XOR s((conv_integer(sumas)*5+1));
a_registro_salida((conv_integer (cuenta)*5)+2) <= Entrada((conv_integer
(cuenta)*5+2)) XOR s((conv_integer(sumas)*5+2));
a_registro_salida((conv_integer (cuenta)*5)+3) <= Entrada((conv_integer
(cuenta)*5+3)) XOR s((conv_integer(sumas)*5+3));
a_registro_salida((conv_integer (cuenta)*5)+4) <= Entrada((conv_integer
(cuenta)*5+4)) XOR s((conv_integer(sumas)*5+4));
if cuenta="11111" then
Final <= '1';
Salida <= registro_salida;
estado_siguiente <= inicia;
else
Final <= '0';
```

```
Salida <= (OTHERS => '0');
estado_siguiente <= sumax2;
end if;

end case;
end process;
end Behavioral;
```

----------Entidad Contador----------

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity Contador is
Port ( Clk : in STD_LOGIC;
Enable : in STD_LOGIC;
Reset : in STD_LOGIC;
Cuenta : out STD_LOGIC_VECTOR (4 downto 0));
end Contador;

architecture Behavioral of Contador is
signal Count: std_logic_vector (4 downto 0);
begin
process (Clk, Enable, Reset)
begin
if (Reset='1') then
Count <= "00000";
elsif (Clk'event and Clk='1') then
if (Enable='1') then
Count <= Count + '1';
end if;
end if;
end process;
Cuenta <= Count;

end Behavioral;
```

----------Entidad Sumador----------

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity Sumador is
Port ( A : in STD_LOGIC_VECTOR (4 downto 0);
B : in STD_LOGIC_VECTOR (4 downto 0);
Suma : out STD_LOGIC_VECTOR (4 downto 0));
end Sumador;

architecture Behavioral of Sumador is

begin

process (A,B)
begin
Suma <= A+B;
end process;

end Behavioral;
```

ANEXO C – CÓDIGO BANCO DE PRUEBAS PARA 64 BITS

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY prueba IS
END prueba;

ARCHITECTURE behavior OF prueba IS

COMPONENT Control
PORT(
Clk : IN std_logic;
Reset : IN std_logic;
Entrada : IN std_logic_vector(63 downto 0);
```

```
Inicio : IN std_logic;
Key : IN std_logic_vector(63 downto 0);
Salida : OUT std_logic_vector(63 downto 0)
);
END COMPONENT;


--Inputs
signal Clk : std_logic := '0';
signal Reset : std_logic := '0';
signal Entrada : std_logic_vector(63 downto 0) := (others => '0');
signal Inicio : std_logic := '0';
signal Key : std_logic_vector(63 downto 0) := (others => '0');

--Outputs
signal Salida : std_logic_vector(63 downto 0);

-- Clock period definitions
constant Clk_period : time := 10 ns;

BEGIN

-- Instantiate the Unit Under Test (UUT)
uut: Control PORT MAP (
Clk => Clk,
Reset => Reset,
Entrada => Entrada,
Inicio => Inicio,
Key => Key,
Salida => Salida
);

-- Clock process definitions
Clk_process :process
begin
Clk <= '0';
wait for Clk_period/2;
Clk <= '1';
wait for Clk_period/2;
end process;
```

```
-- Stimulus process
stim_proc: process
begin
Reset <= '1';
Inicio <='1';
wait for 100 ps;
Inicio <= '1';
Reset <= '0';
wait for 100 ps;
Inicio <= '1';
-- Se introduce clave y entrada iguales manualmente de forma aleatoria para saber la salida que habrá de dar el circuito
Entrada <= "11111111111100111111000001010111001110001110001110001111111111111";
Key <= "11111111111100111111000001010111001110001110001110001111111111111";
wait;
end process;

END;
```

ANEXO D – RESULTADOS DEL TEST DE NIST

```
------------------------------------------------------------------------------
RESULTS FOR THE UNIFORMITY OF P-VALUES AND THE PROPORTION OF PASSING SEQUENCES
------------------------------------------------------------------------------
   generator is <salida>
------------------------------------------------------------------------------
 C1  C2  C3  C4  C5  C6  C7  C8  C9 C10  P-VALUE  PROPORTION  STATISTICAL TEST
------------------------------------------------------------------------------
  7   8   6  10  12   8  18   9  15   7  0.137282    100/100     Frequency
  1   2   2   8   5  16  10   8  17  31  0.000000 *  100/100     BlockFrequency
  8   9   7   8   8  12   9  12  14  13  0.779188    100/100     CumulativeSums
  6   7   6  12   8   5  19   9  13  15  0.025193    100/100     CumulativeSums
 11  15  12   5  14   7  11   8  10   7  0.401199     98/100     Runs
 11  11  12  12   9   4  11   7  13  10  0.678686     97/100     LongestRun
 10   9  11   8   8  13   8  10  10  13  0.955835    100/100     Rank
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  FFT
 14  13  10  11   9  11   9   5   8  10  0.759756    100/100     NonOverlappingTemplate
  9   9   8  20  14  12  10   5   5   8  0.035174     98/100     NonOverlappingTemplate
  6   8  10   7  11  11  17  10  12   8  0.455937    100/100     NonOverlappingTemplate
```

11	15	9	6	9	12	7	10	11	10	0.759756	98/100	NonOverlappingTemplate
7	7	13	14	13	11	3	6	11	15	0.108791	100/100	NonOverlappingTemplate
15	11	13	10	3	7	9	12	12	8	0.304126	98/100	NonOverlappingTemplate
6	12	10	12	9	9	8	14	12	8	0.798139	100/100	NonOverlappingTemplate
14	11	5	16	10	9	19	5	1	10	0.001628	99/100	NonOverlappingTemplate
9	12	9	7	14	10	8	12	12	7	0.816537	99/100	NonOverlappingTemplate
10	10	9	12	5	14	8	11	8	13	0.699313	100/100	NonOverlappingTemplate
12	10	9	4	4	6	15	14	9	17	0.030806	99/100	NonOverlappingTemplate
15	7	15	8	9	10	9	13	9	5	0.350485	99/100	NonOverlappingTemplate
13	15	15	11	11	6	7	6	8	8	0.275709	99/100	NonOverlappingTemplate
9	10	7	10	8	16	10	11	10	9	0.816537	98/100	NonOverlappingTemplate
15	5	7	10	13	10	10	12	9	9	0.595549	99/100	NonOverlappingTemplate
12	8	14	10	8	12	15	7	7	7	0.494392	97/100	NonOverlappingTemplate
5	13	10	11	7	13	15	10	10	6	0.401199	99/100	NonOverlappingTemplate
9	14	6	13	11	12	9	8	9	9	0.798139	98/100	NonOverlappingTemplate
8	12	9	12	9	14	13	4	9	10	0.574903	100/100	NonOverlappingTemplate
12	13	10	14	7	10	4	9	11	10	0.574903	99/100	NonOverlappingTemplate
12	12	8	11	10	7	11	9	8	12	0.955835	100/100	NonOverlappingTemplate
7	8	11	17	7	12	11	12	4	11	0.224821	98/100	NonOverlappingTemplate
16	9	9	13	8	13	11	5	8	8	0.401199	98/100	NonOverlappingTemplate
12	11	10	15	7	6	11	10	10	8	0.739918	97/100	NonOverlappingTemplate
10	17	5	10	10	7	10	10	10	11	0.494392	99/100	NonOverlappingTemplate
9	13	13	9	14	4	14	8	12	4	0.153763	97/100	NonOverlappingTemplate
6	8	14	14	10	13	4	8	11	12	0.304126	98/100	NonOverlappingTemplate
7	10	10	10	11	8	12	15	6	11	0.739918	99/100	NonOverlappingTemplate
13	12	7	12	7	5	11	16	10	7	0.304126	99/100	NonOverlappingTemplate
17	12	9	11	12	8	4	9	9	9	0.334538	99/100	NonOverlappingTemplate
11	9	11	4	13	11	8	13	11	9	0.699313	99/100	NonOverlappingTemplate
13	9	13	10	12	7	5	18	9	4	0.071177	98/100	NonOverlappingTemplate
9	10	10	5	17	11	9	7	12	10	0.437274	100/100	NonOverlappingTemplate
9	12	7	15	12	11	11	7	7	9	0.699313	99/100	NonOverlappingTemplate
8	8	12	11	7	8	7	12	17	10	0.455937	100/100	NonOverlappingTemplate
14	8	10	14	4	14	7	7	10	12	0.275709	97/100	NonOverlappingTemplate
12	10	10	12	9	11	11	6	13	6	0.816537	97/100	NonOverlappingTemplate
10	9	12	13	7	9	14	10	6	10	0.779188	100/100	NonOverlappingTemplate
10	13	15	7	7	7	14	9	4	14	0.162606	100/100	NonOverlappingTemplate
12	9	12	4	12	10	12	10	12	7	0.678686	98/100	NonOverlappingTemplate
8	6	15	9	14	5	10	9	10	14	0.319084	98/100	NonOverlappingTemplate
21	14	7	8	10	9	9	7	5	10	0.028817	98/100	NonOverlappingTemplate
11	9	12	11	10	6	14	10	10	7	0.851383	99/100	NonOverlappingTemplate
13	12	8	12	4	6	11	13	11	10	0.494392	99/100	NonOverlappingTemplate
6	12	10	9	14	7	10	13	11	8	0.739918	100/100	NonOverlappingTemplate
7	16	7	7	11	10	10	9	10	13	0.595549	99/100	NonOverlappingTemplate
11	7	10	15	12	8	10	14	9	4	0.383827	98/100	NonOverlappingTemplate
9	15	9	12	6	12	4	10	10	13	0.383827	99/100	NonOverlappingTemplate
8	12	11	11	5	17	6	10	9	11	0.334538	98/100	NonOverlappingTemplate

7	4	6	13	15	10	8	13	14	10	0.191687	100/100		NonOverlappingTemplate
8	16	14	10	8	11	12	7	9	5	0.350485	99/100		NonOverlappingTemplate
7	15	7	8	15	15	9	7	10	7	0.236810	98/100		NonOverlappingTemplate
12	10	11	6	10	8	7	15	11	10	0.739918	100/100		NonOverlappingTemplate
10	12	9	9	13	13	5	9	8	12	0.759756	99/100		NonOverlappingTemplate
17	6	10	5	11	12	6	9	15	9	0.129620	100/100		NonOverlappingTemplate
7	5	14	11	11	9	8	15	14	6	0.249284	100/100		NonOverlappingTemplate
8	9	12	8	12	9	10	8	11	13	0.955835	98/100		NonOverlappingTemplate
17	9	13	10	8	6	11	8	6	12	0.319084	98/100		NonOverlappingTemplate
13	13	13	9	10	6	10	8	11	7	0.759756	100/100		NonOverlappingTemplate
9	11	9	7	14	13	9	12	9	7	0.816537	99/100		NonOverlappingTemplate
12	6	11	6	13	9	12	9	11	11	0.798139	99/100		NonOverlappingTemplate
11	9	10	10	14	16	11	6	8	5	0.350485	100/100		NonOverlappingTemplate
8	11	9	11	3	11	10	15	9	13	0.419021	100/100		NonOverlappingTemplate
7	7	13	8	14	9	14	6	11	11	0.514124	98/100		NonOverlappingTemplate
11	7	7	9	10	11	10	10	11	14	0.924076	99/100		NonOverlappingTemplate
14	7	12	8	10	8	9	7	14	11	0.699313	99/100		NonOverlappingTemplate
6	12	11	10	12	12	8	13	11	5	0.657933	100/100		NonOverlappingTemplate
11	12	11	5	9	10	14	8	9	11	0.798139	99/100		NonOverlappingTemplate
18	12	7	6	10	9	7	10	12	9	0.289667	96/100		NonOverlappingTemplate
8	11	8	12	9	12	4	11	13	12	0.657933	97/100		NonOverlappingTemplate
13	11	6	6	9	8	16	14	8	9	0.319084	98/100		NonOverlappingTemplate
5	10	18	8	9	10	10	13	8	9	0.289667	100/100		NonOverlappingTemplate
8	9	11	12	12	10	11	10	9	8	0.991468	100/100		NonOverlappingTemplate
12	13	12	8	7	6	10	15	3	14	0.137282	99/100		NonOverlappingTemplate
14	13	10	12	8	11	9	5	8	10	0.699313	100/100		NonOverlappingTemplate
17	5	12	10	8	8	8	9	14	9	0.289667	95/100	*	NonOverlappingTemplate
10	5	14	9	11	10	8	8	13	12	0.699313	100/100		NonOverlappingTemplate
13	5	8	5	7	15	15	8	12	12	0.145326	99/100		NonOverlappingTemplate
12	10	6	9	11	11	9	7	14	11	0.834308	98/100		NonOverlappingTemplate
12	8	10	12	12	7	11	9	10	9	0.971699	99/100		NonOverlappingTemplate
10	10	7	12	12	13	12	12	8	4	0.595549	100/100		NonOverlappingTemplate
10	5	13	13	6	9	9	13	11	11	0.616305	99/100		NonOverlappingTemplate
5	9	13	7	8	17	6	7	13	15	0.075719	100/100		NonOverlappingTemplate
7	6	10	16	14	8	11	12	8	8	0.401199	99/100		NonOverlappingTemplate
13	7	7	11	14	10	10	8	9	11	0.834308	100/100		NonOverlappingTemplate
8	7	13	9	9	6	10	10	13	15	0.595549	99/100		NonOverlappingTemplate
12	8	10	9	13	11	13	5	14	5	0.401199	99/100		NonOverlappingTemplate
11	10	8	9	12	15	7	6	10	12	0.699313	99/100		NonOverlappingTemplate
5	8	11	12	10	10	14	12	13	5	0.455937	100/100		NonOverlappingTemplate
3	12	14	12	13	8	7	6	16	9	0.096578	100/100		NonOverlappingTemplate
15	8	9	12	7	13	9	6	11	10	0.637119	99/100		NonOverlappingTemplate
10	13	7	11	9	10	8	12	11	9	0.964295	99/100		NonOverlappingTemplate
16	12	8	15	6	11	6	10	10	6	0.224821	96/100		NonOverlappingTemplate
11	9	8	9	9	13	12	7	10	12	0.946308	100/100		NonOverlappingTemplate
22	15	5	10	4	8	10	5	9	12	0.001757	95/100	*	NonOverlappingTemplate

17	9	7	13	10	9	10	7	9	9	0.534146	98/100		NonOverlappingTemplate
11	7	6	16	9	5	9	19	9	9	0.045675	100/100		NonOverlappingTemplate
14	11	8	11	12	9	14	12	5	4	0.289667	97/100		NonOverlappingTemplate
16	12	9	6	13	5	9	11	8	11	0.366918	97/100		NonOverlappingTemplate
6	10	15	10	6	11	9	14	12	7	0.455937	99/100		NonOverlappingTemplate
15	8	8	14	13	9	12	6	8	7	0.419021	98/100		NonOverlappingTemplate
4	11	9	6	17	14	7	9	7	16	0.042808	100/100		NonOverlappingTemplate
10	13	8	7	10	9	15	10	11	7	0.759756	100/100		NonOverlappingTemplate
8	11	12	9	8	8	16	10	10	8	0.759756	100/100		NonOverlappingTemplate
10	8	11	10	8	11	10	13	8	11	0.983453	100/100		NonOverlappingTemplate
9	10	8	12	13	7	7	12	11	11	0.897763	100/100		NonOverlappingTemplate
10	10	8	6	7	13	13	10	12	11	0.816537	99/100		NonOverlappingTemplate
14	8	10	10	7	10	13	10	8	10	0.897763	98/100		NonOverlappingTemplate
10	8	12	7	6	9	16	10	6	16	0.202268	99/100		NonOverlappingTemplate
11	9	10	8	10	15	12	12	8	5	0.657933	98/100		NonOverlappingTemplate
8	8	10	2	10	16	12	12	14	8	0.137282	100/100		NonOverlappingTemplate
5	10	3	9	18	11	12	8	10	14	0.058984	100/100		NonOverlappingTemplate
9	11	8	8	6	7	16	14	9	12	0.419021	98/100		NonOverlappingTemplate
13	11	7	8	4	12	11	13	8	13	0.474986	99/100		NonOverlappingTemplate
17	10	12	12	8	8	11	8	9	5	0.383827	100/100		NonOverlappingTemplate
9	9	14	9	11	13	13	9	8	5	0.657933	98/100		NonOverlappingTemplate
5	12	7	15	8	9	11	15	7	11	0.319084	100/100		NonOverlappingTemplate
11	6	12	9	12	11	7	11	12	9	0.897763	98/100		NonOverlappingTemplate
13	14	10	11	7	10	11	9	7	8	0.834308	97/100		NonOverlappingTemplate
24	8	10	10	8	10	8	3	10	9	0.002203	96/100		NonOverlappingTemplate
7	7	10	14	10	12	13	7	14	6	0.455937	98/100		NonOverlappingTemplate
19	10	9	10	7	10	8	12	8	7	0.262249	94/100	*	NonOverlappingTemplate
10	13	9	10	11	8	11	9	8	11	0.987896	100/100		NonOverlappingTemplate
10	12	8	8	16	7	10	12	8	9	0.678686	99/100		NonOverlappingTemplate
9	12	10	14	5	8	9	12	10	11	0.779188	100/100		NonOverlappingTemplate
5	17	9	8	14	10	10	6	9	12	0.236810	100/100		NonOverlappingTemplate
12	9	11	9	7	10	12	8	10	12	0.971699	99/100		NonOverlappingTemplate
9	10	8	13	7	13	8	8	9	15	0.678686	99/100		NonOverlappingTemplate
8	11	6	13	12	15	9	11	10	5	0.474986	100/100		NonOverlappingTemplate
12	11	7	9	15	10	13	7	9	7	0.657933	99/100		NonOverlappingTemplate
11	11	6	6	15	6	7	17	9	12	0.129620	99/100		NonOverlappingTemplate
8	10	10	5	16	10	10	10	11	10	0.678686	100/100		NonOverlappingTemplate
24	8	8	4	14	10	11	8	7	6	0.000757	94/100	*	NonOverlappingTemplate
9	14	9	10	10	6	9	10	14	9	0.816537	99/100		NonOverlappingTemplate
11	9	18	7	12	11	7	5	11	9	0.236810	99/100		NonOverlappingTemplate
11	7	6	16	11	5	8	15	13	8	0.162606	98/100		NonOverlappingTemplate
8	9	11	8	12	12	14	6	13	7	0.657933	99/100		NonOverlappingTemplate
14	11	8	9	10	10	10	9	12	7	0.935716	100/100		NonOverlappingTemplate
7	12	6	12	13	10	9	11	8	12	0.816537	100/100		NonOverlappingTemplate
12	4	9	10	8	10	10	12	13	12	0.719747	100/100		NonOverlappingTemplate
27	15	11	9	8	8	3	4	9	6	0.000003 *	94/100	*	NonOverlappingTemplate

11	12	14	9	11	10	11	8	8	6	0.851383	99/100	NonOverlappingTemplate
6	7	11	11	10	9	11	14	10	11	0.867692	100/100	NonOverlappingTemplate
9	10	13	8	4	11	10	12	12	11	0.739918	100/100	NonOverlappingTemplate
14	18	15	3	5	9	8	7	11	10	0.021999	98/100	NonOverlappingTemplate
8	14	12	7	8	12	6	8	10	15	0.474986	100/100	NonOverlappingTemplate
11	6	12	7	9	12	7	11	15	10	0.637119	100/100	NonOverlappingTemplate
12	13	12	8	7	6	10	15	3	14	0.137282	99/100	NonOverlappingTemplate
12	15	11	12	8	7	7	11	9	8	0.719747	100/100	OverlappingTemplate
0	100	0	0	0	0	0	0	0	0	0.000000 *	100/100	Universal
37	22	14	6	5	1	10	2	2	1	0.000000 *	92/100 *	ApproximateEntropy
1	2	2	0	0	0	0	2	2	3	0.213309	12/12	RandomExcursions
1	0	0	1	4	2	0	1	2	1	0.122325	12/12	RandomExcursions
1	1	1	1	1	1	2	1	1	2	0.991468	12/12	RandomExcursions
1	2	1	1	2	2	1	0	1	1	0.911413	12/12	RandomExcursions
2	0	0	2	1	2	1	1	1	2	0.739918	12/12	RandomExcursions
2	0	3	2	1	0	1	1	2	0	0.350485	12/12	RandomExcursions
1	3	1	3	2	0	0	0	1	1	0.213309	12/12	RandomExcursions
0	3	0	1	1	1	0	3	1	2	0.213309	12/12	RandomExcursions
3	1	1	2	2	0	0	2	1	0	0.350485	12/12	RandomExcursionsVariant
4	1	0	1	2	1	1	1	0	1	0.213309	12/12	RandomExcursionsVariant
1	4	0	1	2	0	1	0	1	2	0.122325	12/12	RandomExcursionsVariant
1	2	3	0	1	0	0	1	3	1	0.213309	12/12	RandomExcursionsVariant
3	1	0	0	2	2	3	0	0	1	0.122325	12/12	RandomExcursionsVariant
3	0	3	0	2	1	1	1	0	1	0.213309	12/12	RandomExcursionsVariant
4	0	0	0	2	2	1	3	0	0	0.017912	12/12	RandomExcursionsVariant
1	3	1	1	0	1	1	0	0	4	0.066882	12/12	RandomExcursionsVariant
1	0	1	3	1	2	1	2	1	0	0.534146	12/12	RandomExcursionsVariant
2	2	0	1	1	2	0	1	0	3	0.350485	12/12	RandomExcursionsVariant
2	2	1	0	0	0	1	2	0	4	0.066882	12/12	RandomExcursionsVariant
1	3	0	1	0	1	0	2	2	2	0.350485	11/12	RandomExcursionsVariant
2	1	2	0	2	0	0	1	3	1	0.350485	11/12	RandomExcursionsVariant
3	0	1	0	1	1	1	1	3	1	0.350485	12/12	RandomExcursionsVariant
1	2	0	0	2	2	1	0	2	2	0.534146	12/12	RandomExcursionsVariant
1	2	0	0	3	0	1	0	3	2	0.122325	12/12	RandomExcursionsVariant
2	0	1	0	1	3	2	2	1	0	0.350485	12/12	RandomExcursionsVariant
1	1	0	4	1	2	0	1	1	1	0.213309	12/12	RandomExcursionsVariant
15	14	8	8	7	12	6	8	14	8	0.334538	99/100	Serial
10	9	12	5	14	3	11	12	6	18	0.035174	99/100	Serial
15	3	8	12	16	9	6	10	12	9	0.122325	99/100	LinearComplexity

- -

The minimum pass rate for each statistical test with the exception of the
random excursion (variant) test is approximately = 96 for a
sample size = 100 binary sequences.

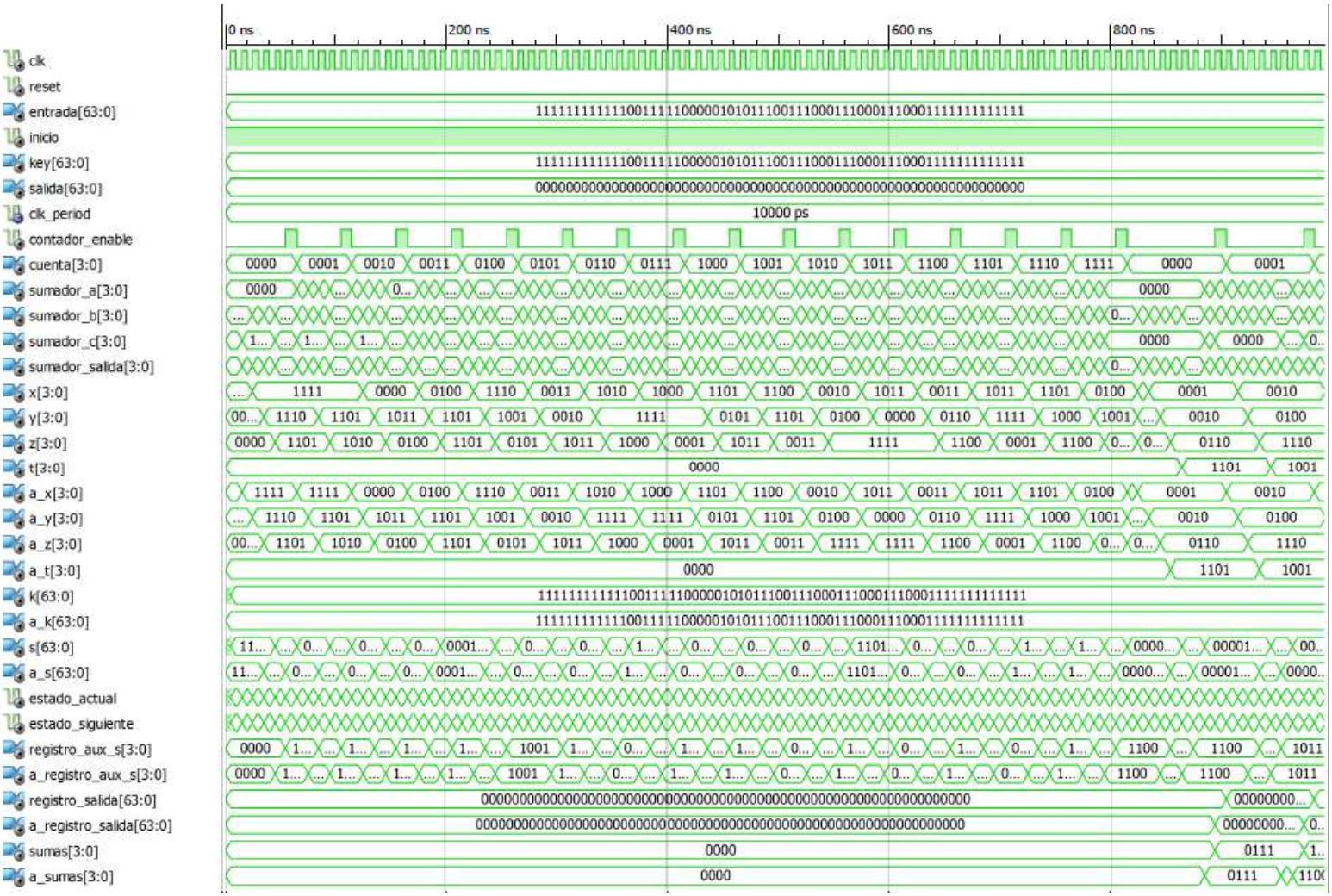
```
The minimum pass rate for the random excursion (variant) test
is approximately = 10 for a sample size = 12 binary sequences.

For further guidelines construct a probability table using the MAPLE program
provided in the addendum section of the documentation.
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

ANEXO E – CÓDIGO MATLAB Y RESULTADO DEL TEST CHI-CUADRADO

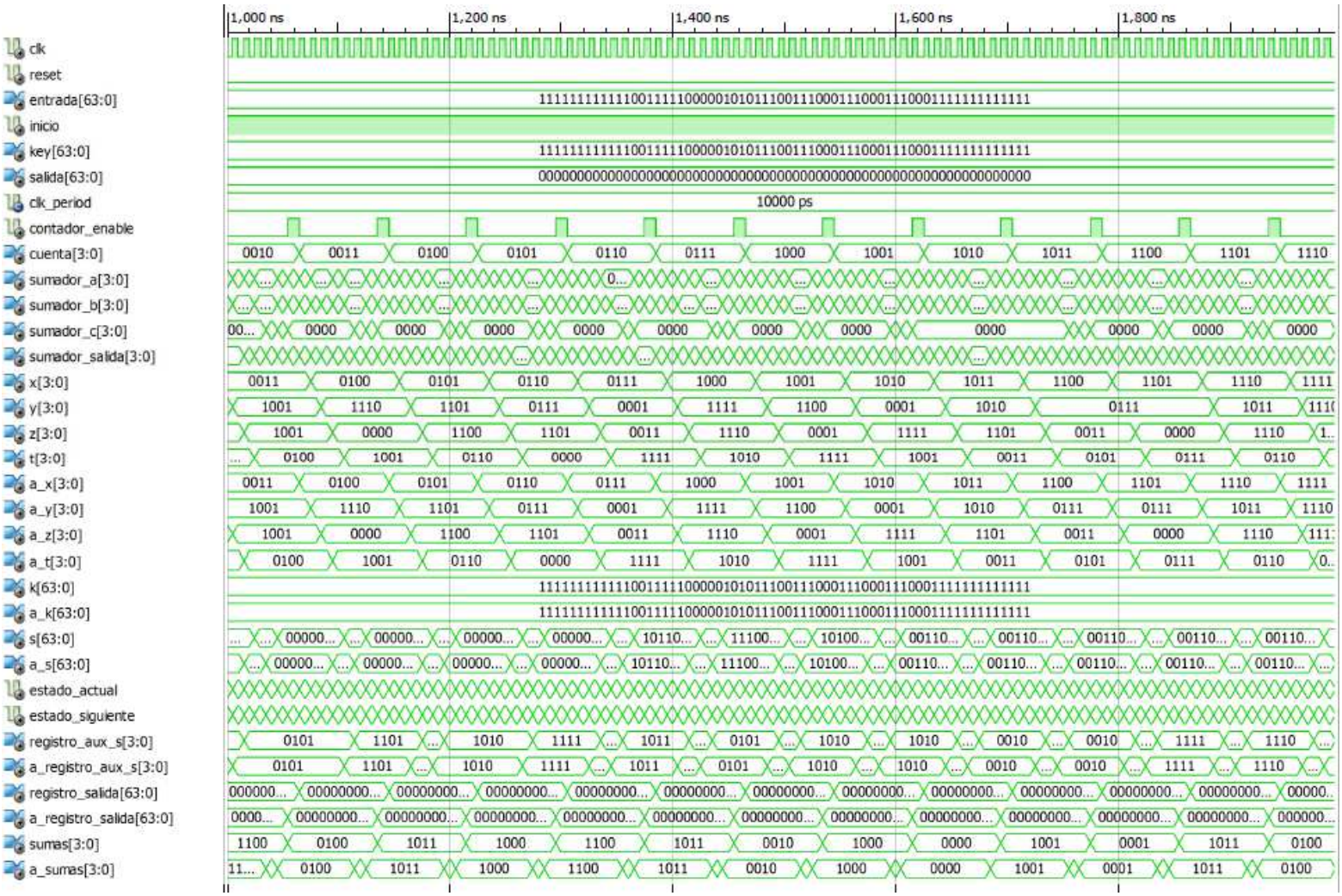
```
bins = 1:5;

obsCounts = [16 100 102 43 22];

n = sum(obsCounts);

expCounts = [sum(n*binopdf(1:28,64,0.5)),sum(n*binopdf(29:30,64,0.5)),sum(n*binopdf(31:33,64,0.5)),sum(n*binopdf(34:35,64,0.5)),sum(n*binopdf(36:64,64,0.5))]

[h,p,st] = chi2gof(bins,'ctrs',bins,...
                   'frequency',obsCounts, ...
                   'expected',expCounts,...
                   'nparams',1)

p = 0,0914
```

ANEXO F – SIMULACIONES ARQUITECTURA DOS SUMADORES Y 64 BITS

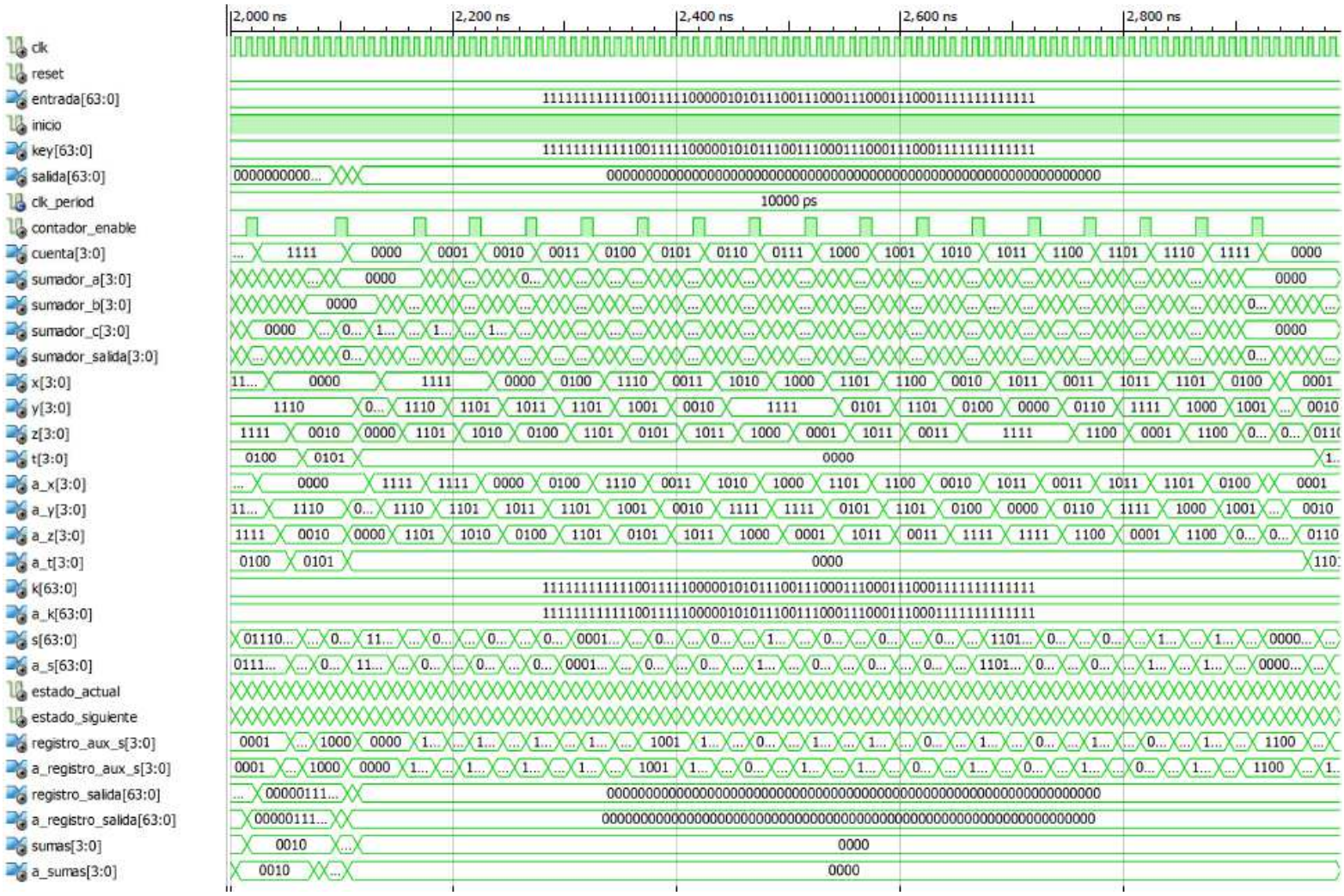
- SIMULACIÓN 0 - 1000 ns



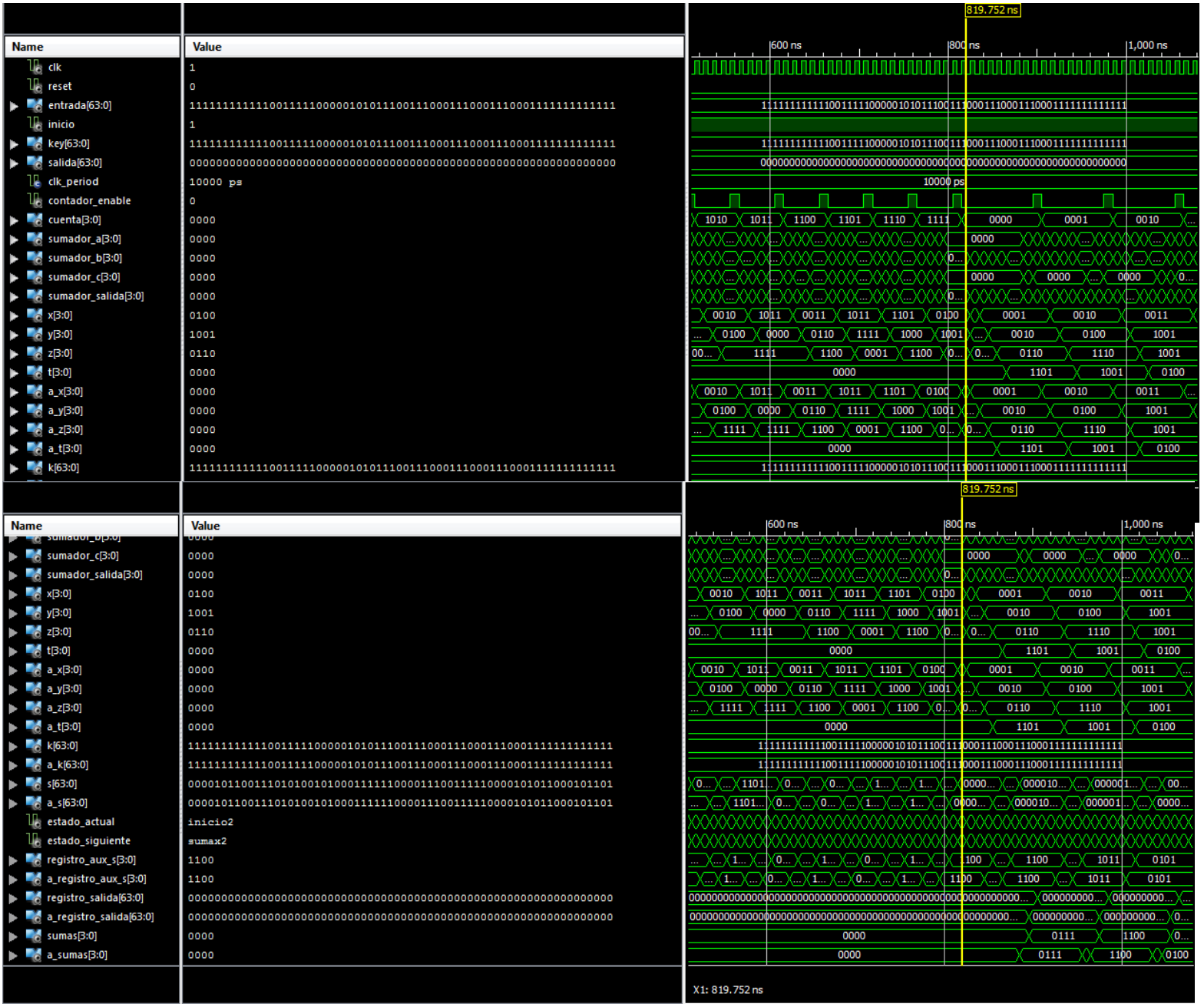
- SIMULACIÓN 1000ns - 2000ns



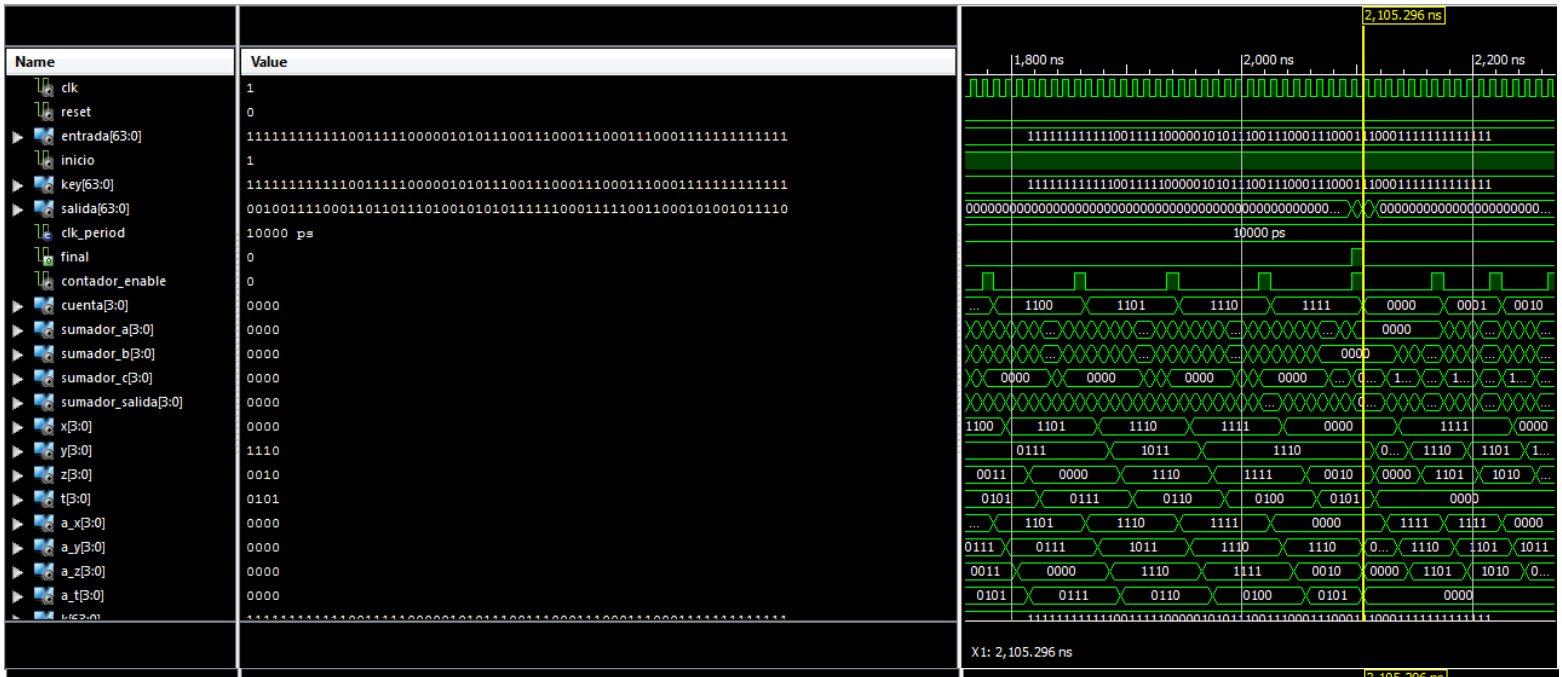
- SIMULACIÓN 2000ns - 3000ns

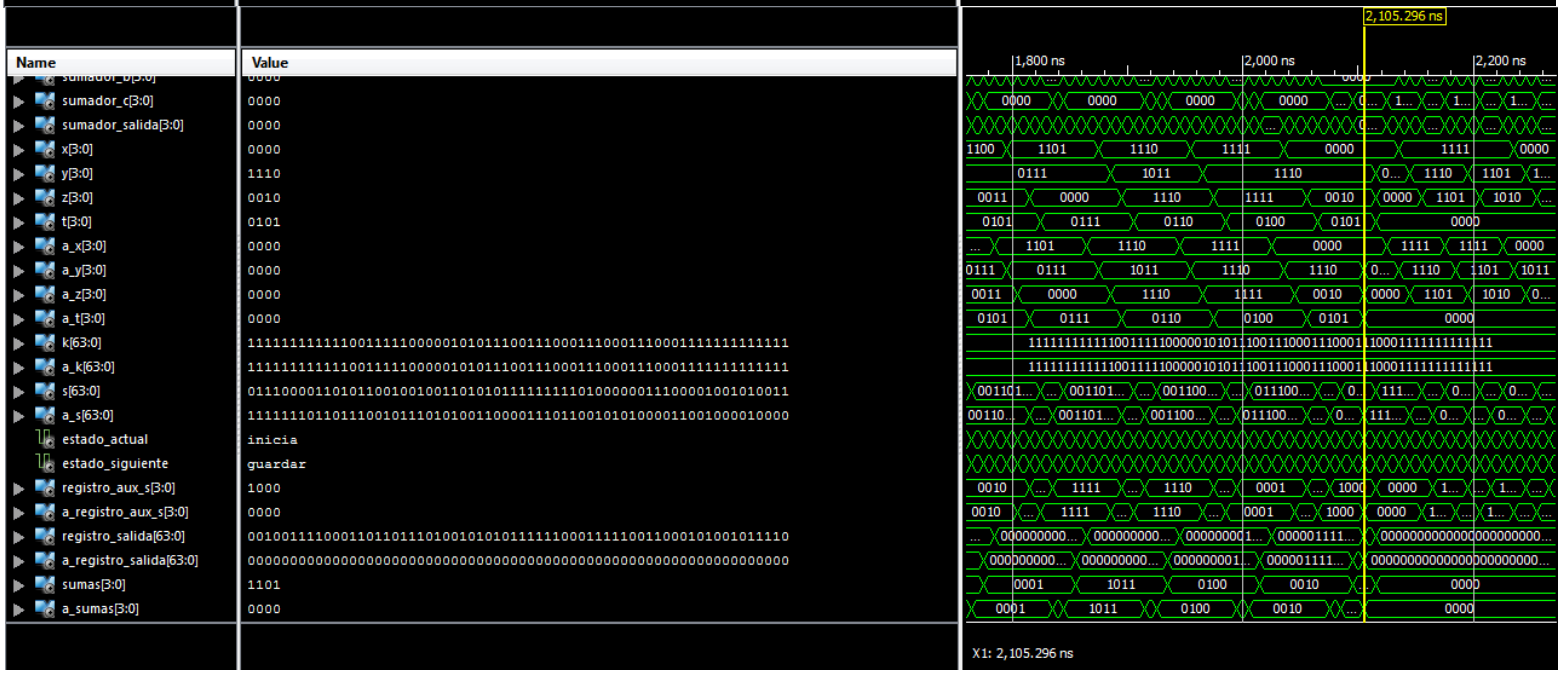


- RESULTADOS TRAS LA PRIMERA PARTE DEL CIRCUITO



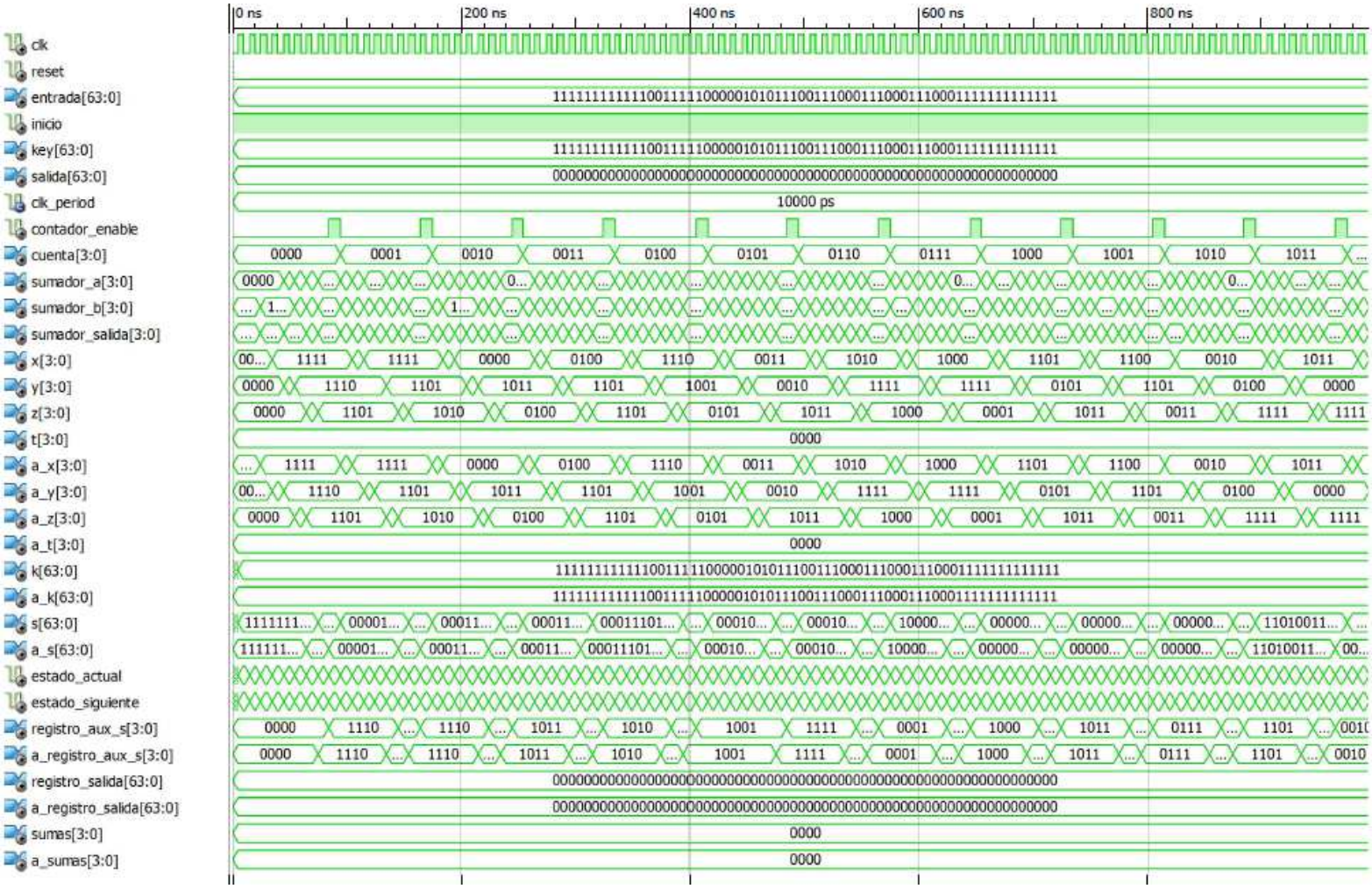
- RESULTADOS FINALES



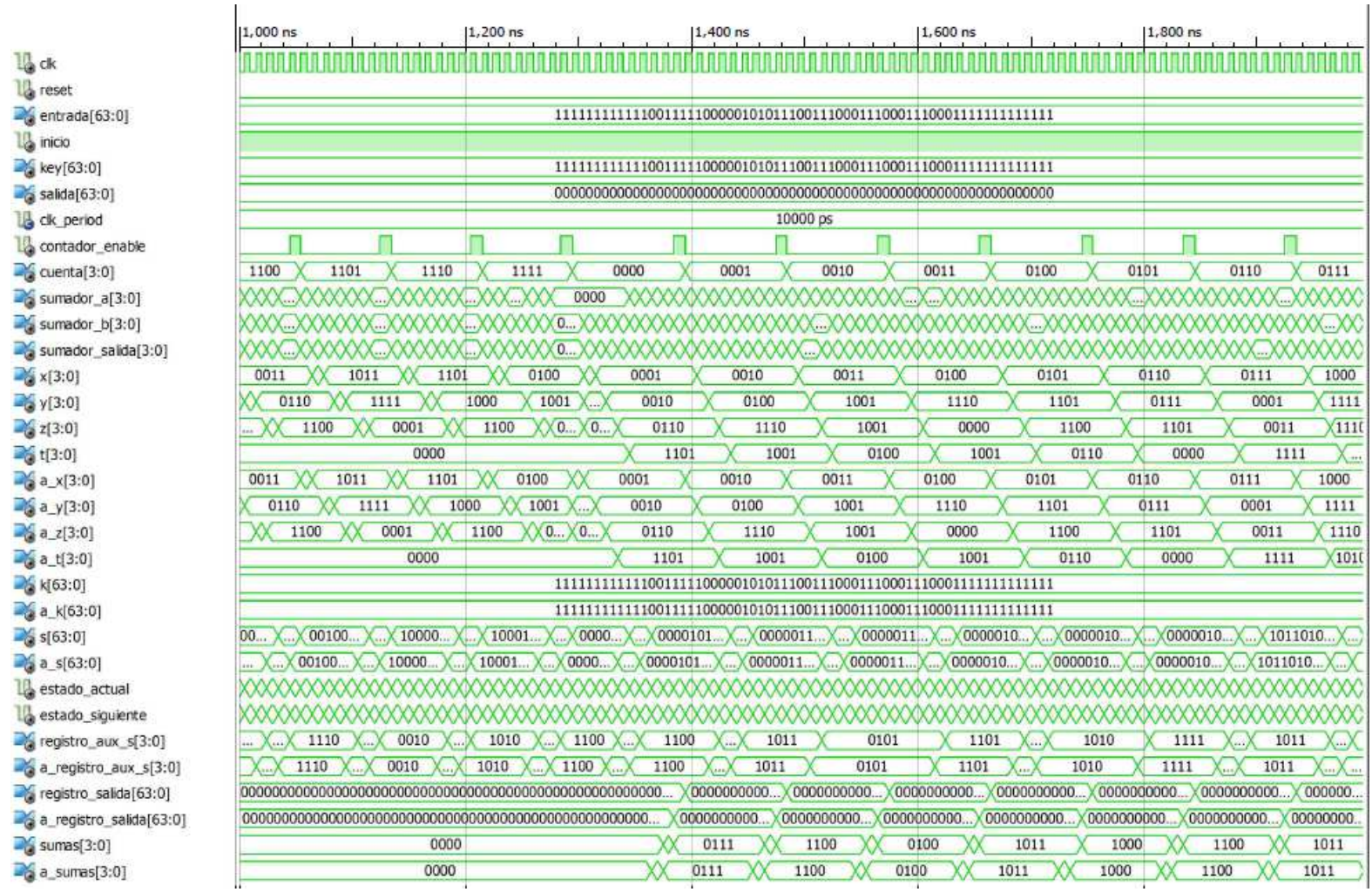


ANEXO G – SIMULACIONES ARQUITECTURA CON UN SUMADOR DE 2 ENTRADAS Y 64 BITS

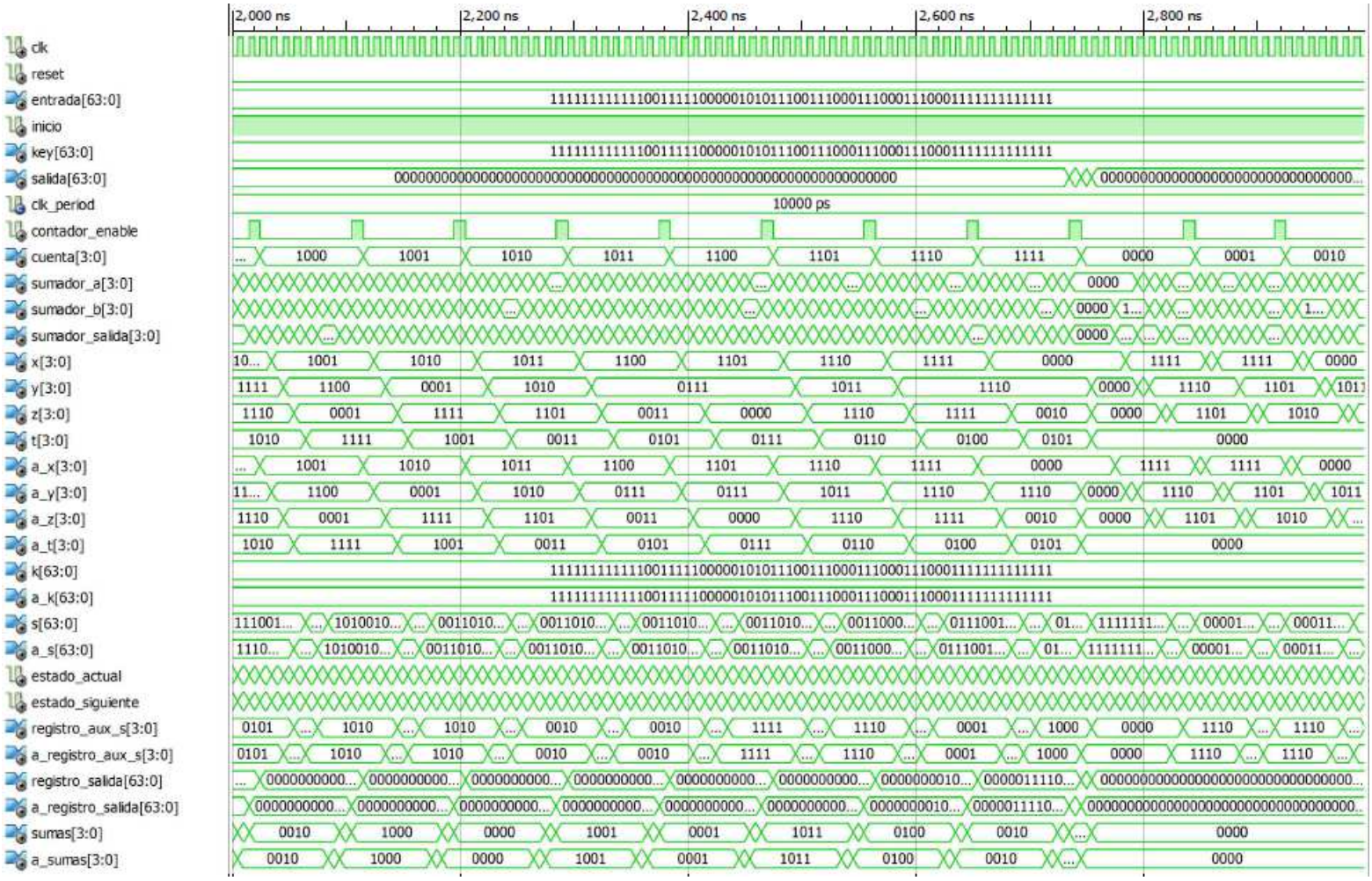
- SIMULACIÓN 0 - 1000 ns



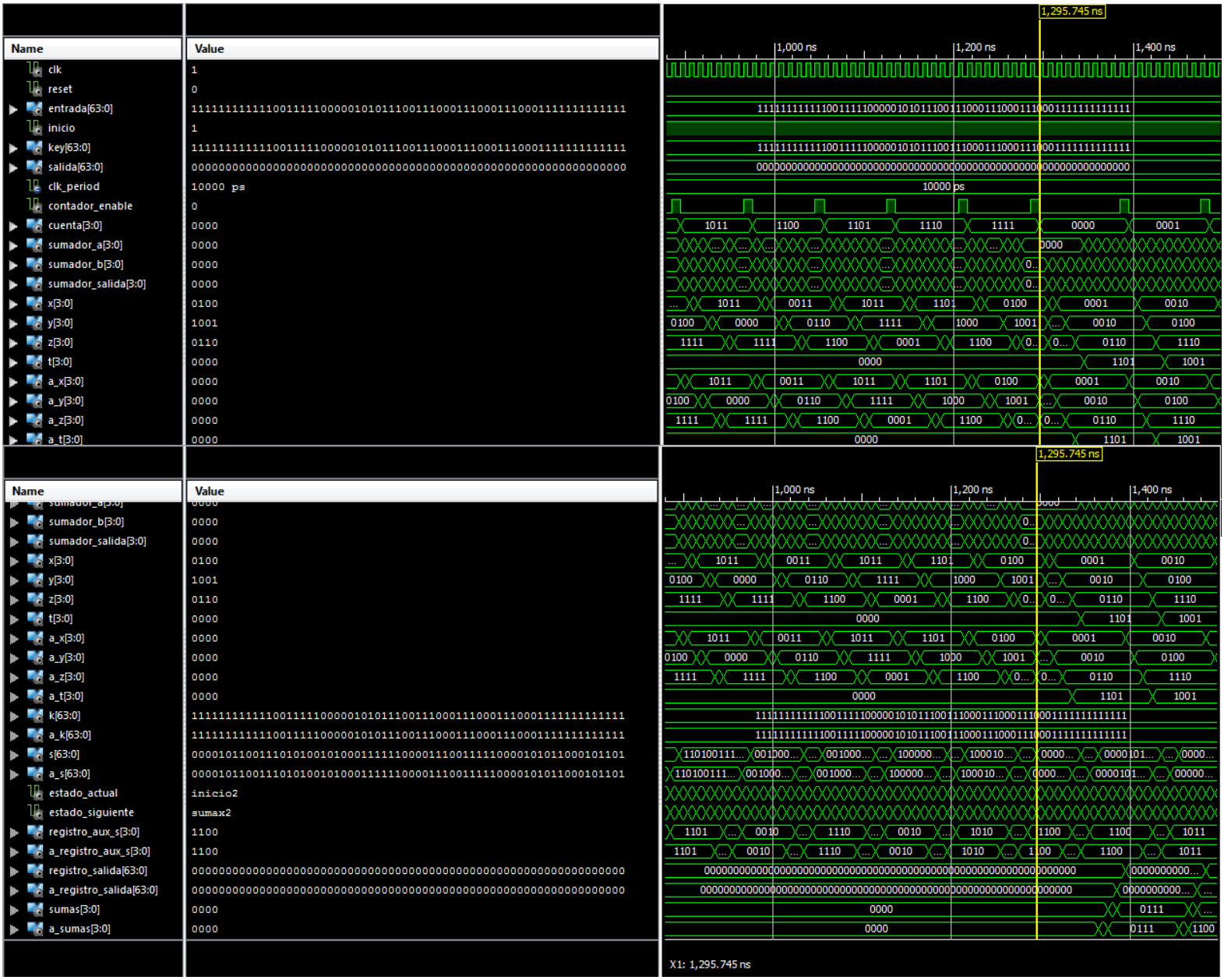
- SIMULACIÓN 1000 - 2000 ns



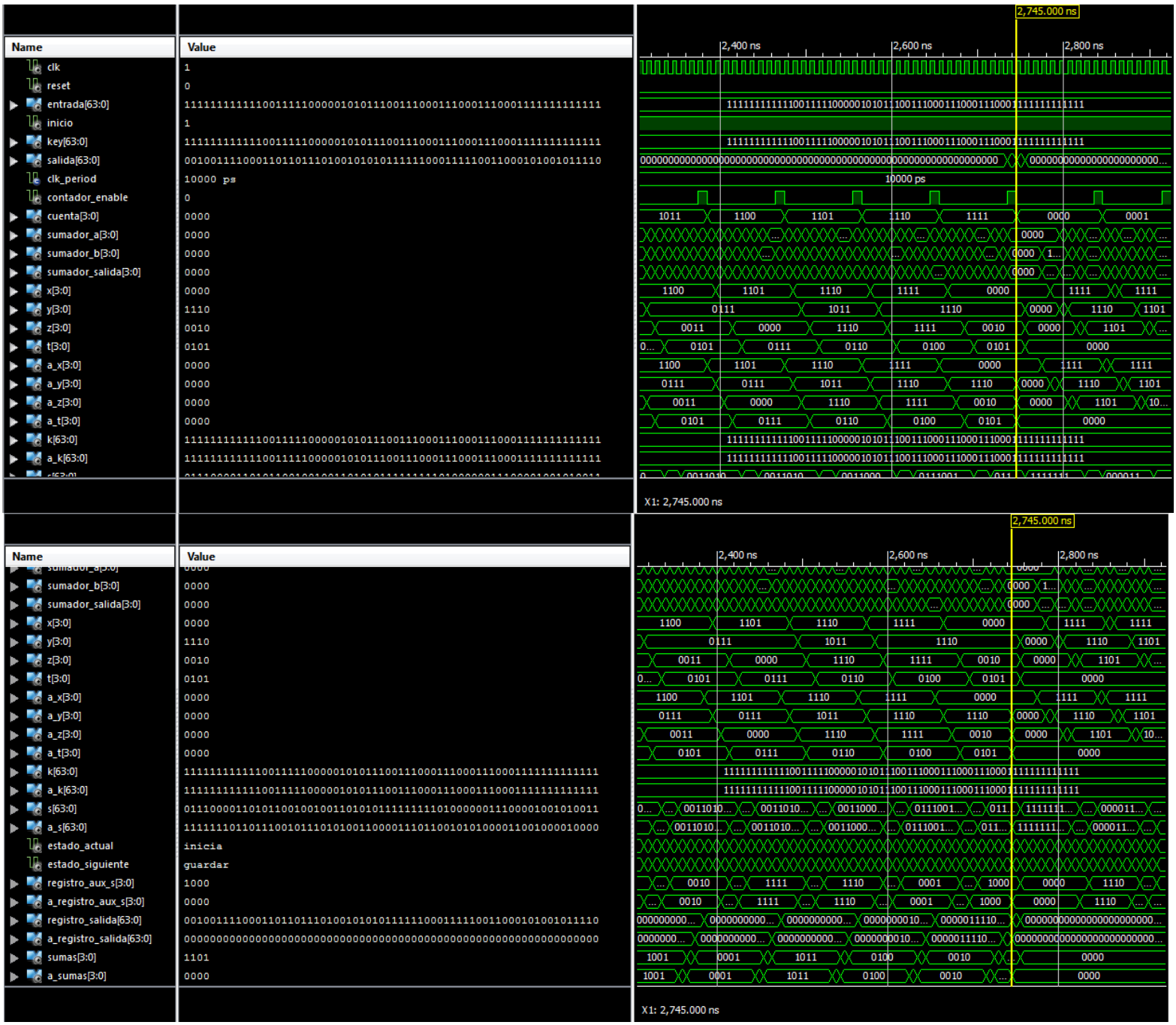
- SIMULACIÓN 2000 - 3000 ns



- RESULTADOS TRAS LA PRIMERA PARTE DEL CIRCUITO



- RESULTADOS FINALES



ANEXO H – CÓDIGO ARQUITECTURA CON DOS SUMADORES DE 2 ENTRADAS y 160 BITS

----------Entidad Control----------

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity Control is
Port ( Clk : in STD_LOGIC;
Reset : in STD_LOGIC;
Entrada : in STD_LOGIC_VECTOR (159 downto 0);
Inicio : in STD_LOGIC;
Key : in STD_LOGIC_VECTOR (159 downto 0);
Final : out STD_LOGIC;
Salida : out STD_LOGIC_VECTOR (159 downto 0));
end Control;

architecture Behavioral of Control is

component Contador is
Port (Clk : in STD_LOGIC;
Enable : in STD_LOGIC;
Reset : in STD_LOGIC;
Cuenta : out STD_LOGIC_VECTOR (4 downto 0));
end component;

component Sumador is
Port (A : in STD_LOGIC_VECTOR (4 downto 0);
B : in STD_LOGIC_VECTOR (4 downto 0);
C : in STD_LOGIC_VECTOR (4 downto 0);
Suma : out STD_LOGIC_VECTOR (4 downto 0));
end component;

Signal Contador_Enable: STD_LOGIC;
Signal Cuenta: STD_LOGIC_VECTOR (4 downto 0);
Signal Sumador_A: STD_LOGIC_VECTOR (4 downto 0);
Signal Sumador_B: STD_LOGIC_VECTOR (4 downto 0);
Signal Sumador_C: STD_LOGIC_VECTOR (4 downto 0);
Signal Sumador_Salida: STD_LOGIC_VECTOR (4 downto 0);
Signal X: STD_LOGIC_VECTOR (4 downto 0);
Signal Y: STD_LOGIC_VECTOR (4 downto 0);
Signal Z: STD_LOGIC_VECTOR (4 downto 0);
Signal T: STD_LOGIC_VECTOR (4 downto 0);
Signal a_x: STD_LOGIC_VECTOR (4 downto 0);
Signal a_y: STD_LOGIC_VECTOR (4 downto 0);
Signal a_z: STD_LOGIC_VECTOR (4 downto 0);
Signal a_t: STD_LOGIC_VECTOR (4 downto 0);
Signal K: STD_LOGIC_VECTOR (159 downto 0);
Signal a_k: STD_LOGIC_VECTOR (159 downto 0);
Signal s: STD_LOGIC_VECTOR (159 downto 0);
Signal a_s: STD_LOGIC_VECTOR (159 downto 0);
type estados is (inicia, guardar, sumax, sumay, sumaz, swap1, swap3, swap4, inicio2,
sumax2, sumay2, sumaz2, sumat, swap2, estado_XOR1, estado_XOR2);
signal estado_actual: estados;
signal estado_siguiente: estados;
```

```
Signal registro_aux_s: STD_LOGIC_VECTOR (4 downto 0);
Signal a_registro_aux_s: STD_LOGIC_VECTOR (4 downto 0);
Signal registro_salida: STD_LOGIC_VECTOR (159 downto 0);
Signal a_registro_salida: STD_LOGIC_VECTOR (159 downto 0);
Signal sumas: STD_LOGIC_VECTOR (4 downto 0);
Signal a_sumas: STD_LOGIC_VECTOR (4 downto 0);

begin

Contador_h: Contador port map (Clk => Clk,
Enable => Contador_Enable,
Reset => Reset,
Cuenta => Cuenta);
Sumador_h: Sumador port map (A => Sumador_A,
B => Sumador_B,
C => Sumador_C,
Suma => Sumador_Salida);

process (Clk, Reset)
begin
if (reset = '1') then
X <= (others => '0');
Y <= (others => '0');
Z <= (others => '0');
T <= (others => '0');
elsif Clk'event and Clk='1' then
X <= a_x;
Y <= a_y;
Z <= a_z;
T <= a_t;
end if;
end process;
process (Clk, Reset)
begin
if (reset = '1') then
s <= (others => '0');
elsif Clk'event and Clk='1' then
s <= a_s;
end if;
end process;

process (Clk, Reset)
begin
if (reset = '1') then
registro_aux_s <= (others => '0');
elsif Clk'event and Clk='1' then
registro_aux_s <= a_registro_aux_s;
end if;
end process;

process (Clk, Reset)
begin
if (reset = '1') then
sumas <= (others => '0');
elsif Clk'event and Clk='1' then
sumas <= a_sumas;
end if;
end process;
```

```
process (Clk, Reset)
begin
if (reset = '1') then
registro_salida <= (others => '0');
elsif Clk'event and Clk='1' then
registro_salida <= a_registro_salida;
end if;
end process;

process (Clk, Reset)
begin
if (reset = '1') then
k <= (others => '0');
elsif Clk'event and Clk='1' then
k <= a_k;
158 end if;
end process;

process (Clk, Reset)
begin
if (Reset='1') then
estado_actual <= inicia;
elsif(Clk'event and Clk='1') then
estado_actual <= estado_siguiente;
end if;
end process;

process (estado_actual, cuenta, Key, k, s, X, Y, Z, T, Sumador_Salida, Entrada, Sumas,
registro_aux_s, Sumas, registro_salida, Inicio)
begin
a_s <= s;
a_registro_salida <= registro_salida;

case estado_actual is
when inicia =>
a_x <= (others => '0');
a_y <= (others => '0');
a_z <= (others => '0');
a_t <= (others => '0');
a_registro_aux_s <= (others => '0');
Contador_Enable <='0';
a_k <= Key;
a_s <=
"11111111101110111100110111101011001110001011110110101011010010011100101000110000011110
11100110101100010110101001001010000011100110001010010000011000100000100000";
a_registro_salida <= (others => '0');
Sumador_A <= (others => '0');
Sumador_B <= (others => '0');
Sumador_C <= (others => '0');
a_sumas <= (others => '0');
Salida <= registro_salida;
Final <= '0';
if Inicio = '1' then
estado_siguiente <= guardar;
else
estado_siguiente <= inicia;
end if;
when guardar =>
Salida <= (others => '0');
```

```
a_registro_aux_s <= registro_aux_s;
Contador_Enable <='0';
a_k <= Key;
a_s <=
"11111111101110111100110111101011001110001011110110101011010010011100101000110000011110
11100110101100010110101001001010000011100110001010010000011000100000100000";
a_registro_salida <= registro_salida;
a_x <= X;
a_y <= Y;
a_z <= Z;
a_t <= T;
Sumador_A <= (others => '0');
Sumador_B <= (others => '0');
Sumador_C <= (others => '0');
a_sumas <= sumas;
Final <= '0';
estado_siguiente <= sumax;
when sumax =>
Salida <= (others => '0');
a_registro_salida <= registro_salida;
a_registro_aux_s <= registro_aux_s;
a_k <= k;
a_s <= s;
Sumador_A <= X;
Sumador_B <= s((((conv_integer (cuenta))*5)+4) downto ((conv_integer (
cuenta))*5));
Sumador_C <= K((((conv_integer (cuenta))*5)+4) downto ((conv_integer (
cuenta))*5));
a_x <= Sumador_Salida;
a_y <= Y;
a_z <= Z;
a_t <= T;
a_sumas <= sumas;
Contador_Enable <='0';
Final <= '0';
estado_siguiente <= sumay;
when sumay =>
Salida <= (others => '0');
a_registro_salida <= registro_salida;
a_registro_aux_s <= s((((conv_integer (cuenta))*5)+4) downto ((
conv_integer (cuenta))*5));
a_k <= k;
a_s <= s;
Sumador_A <= Y;
Sumador_B <= s((((conv_integer (X))*5)+4) downto ((conv_integer (X))*5
));
Sumador_C <= K((((conv_integer (X+cuenta))*5)+4) downto ((conv_integer
(X+cuenta))*5));
a_x <= X;
a_y <= Sumador_Salida;
a_z <= Z;
a_t <= T;
a_sumas <= sumas;
Contador_Enable <='0';
Final <= '0';
estado_siguiente <= sumaz;
when sumaz =>
Salida <= (others => '0');
a_registro_salida <= registro_salida;
```

```
a_registro_aux_s <= registro_aux_s;
a_k <= k;
a_s((((conv_integer (cuenta))*5)+4) downto ((conv_integer (cuenta))*5))
<= s((((conv_integer (X))*5)+4) downto ((conv_integer (X))*5));
a_s((((conv_integer (X))*5)+4) downto ((conv_integer (X))*5)) <=
registro_aux_s;
Sumador_A <= Z;
Sumador_B <= s((((conv_integer (Y))*5)+4) downto ((conv_integer (Y))*5
));
Sumador_C <= K((((conv_integer (Y+cuenta))*5)+4) downto ((conv_integer
(Y+cuenta))*5));
a_x <= X;
a_y <= Y;
a_z <= Sumador_Salida;
a_t <= T;
a_sumas <= sumas;
Contador_Enable <= '0';
Final <='0';
estado_siguiente <= swap1;
when swap1 =>
Salida <= (others => '0');
a_registro_salida <= registro_salida;
a_registro_aux_s <= s((((conv_integer (Y))*5)+4) downto ((conv_integer
(Y))*5));
Contador_Enable <='0';
a_k <= k;
a_s <= s;
Sumador_A <= (others => '0');
Sumador_B <= (others => '0');
275 Sumador_C <= (others => '0');
a_x <= X;
a_y <= Y;
a_z <= Z;
a_t <= T;
a_sumas <= sumas;
Final <= '0';
estado_siguiente <= swap2;


when swap2 =>
Salida <= (others => '0');
a_registro_salida <= registro_salida;
a_registro_aux_s <= registro_aux_s;
Contador_Enable <='1';
a_k <= k;
Sumador_A <= (others => '0');
Sumador_B <= (others => '0');
Sumador_C <= (others => '0');
a_x <= X;
a_y <= Y;
a_z <= Z;
a_t <= T;
a_sumas <= sumas;
a_s((((conv_integer (Y))*5)+4) downto ((conv_integer (Y))*5)) <= s((((
conv_integer (Z))*5)+4) downto ((conv_integer (Z))*5));
a_s((((conv_integer (Z))*5)+4) downto ((conv_integer (Z))*5)) <=
registro_aux_s;
Final <= '0';
if cuenta="11111" then
estado_siguiente <= inicio2;
```

```
else
estado_siguiente <= sumax;
end if;

when inicio2 =>
Salida <= (others => '0');
a_registro_salida <= registro_salida;
a_registro_aux_s <= registro_aux_s;
a_x <= (others => '0');
a_y <= (others => '0');
a_z <= (others => '0');
a_t <= (others => '0');
Contador_Enable <='0';
a_k <= k;
a_s <= s;
a_sumas <= sumas;
Sumador_A <= (others => '0');
Sumador_B <= (others => '0');
Sumador_C <= (others => '0');
Final <= '0';
estado_siguiente <= sumax2;
when sumax2 =>
a_registro_salida <= registro_salida;
Salida <= (others => '0');
a_registro_aux_s <= registro_aux_s;
a_k <= k;
a_s <= s;
Contador_Enable <='0';
Sumador_A <= X;
Sumador_B <= "00001";
Sumador_C <= "00000";
a_sumas <= sumas;
a_x <= Sumador_Salida;
a_y <= Y;
a_z <= Z;
a_t <= T;
Final <= '0';
estado_siguiente <= sumay2;
when sumay2 =>
a_registro_salida <= registro_salida;
Salida <= (others => '0');
a_registro_aux_s <= registro_aux_s;
a_k <= k;
a_s <= s;
Sumador_A <= Y;
Sumador_B <= s((((conv_integer (X))*5)+4) downto ((conv_integer (X))*5
));
Sumador_C <= (others => '0');
a_sumas <= sumas;
a_x <= X;
a_y <= Sumador_Salida;
a_z <= Z;
a_t <= T;
Contador_Enable <='0';
Final <='0';
estado_siguiente <= sumaz2;
when sumaz2 =>
a_registro_salida <= registro_salida;
Salida <= (others => '0');
a_registro_aux_s <= s((((conv_integer (X))*5)+4) downto ((conv_integer
```

```
(X))*5));
a_k <= k;
a_s <= s;
Sumador_A <= Z;
Sumador_B <= s((((conv_integer (Y))*5)+4) downto ((conv_integer (Y))*5
));
Sumador_C <= (others => '0');
a_sumas <= sumas;
a_x <= X;
a_y <= Y;
a_z <= Sumador_Salida;
a_t <= T;
Contador_Enable <= '0';
Final <= '0';
estado_siguiente <= sumat;
when sumat =>
a_registro_salida <= registro_salida;
Salida <= (others => '0');
a_registro_aux_s <= registro_aux_s;
a_k <= k;
a_s((((conv_integer (X))*5)+4) downto ((conv_integer (X))*5)) <= s((((
conv_integer (Y))*5)+4) downto ((conv_integer (Y))*5));
a_s((((conv_integer (Y))*5)+4) downto ((conv_integer (Y))*5)) <=
registro_aux_s;
Sumador_A <= T;
Sumador_B <= s((((conv_integer (T))*5)+4) downto ((conv_integer (T))*5
));
Sumador_C <= (others => '0');
a_sumas <= sumas;
a_x <= X;
a_y <= Y;
a_z <= Z;
a_t <= Sumador_Salida;
Contador_Enable <='0';
Final <='0';
estado_siguiente <= swap3;
when swap3 =>
a_registro_salida <= registro_salida;
Salida <= (others => '0');
a_registro_aux_s <= s((((conv_integer (Z))*5)+4) downto ((conv_integer
(Z))*5));
a_k <= k;
a_s <= s;
Sumador_A <= (others => '0');
Sumador_B <= (others => '0');
402 Sumador_C <= (others => '0');
403 a_sumas <= Sumas;
404 a_x <= X;
a_y <= Y;
a_z <= Z;
a_t <= T;
Contador_Enable <= '0';
Final <= '0';
estado_siguiente <= swap4;
when swap4 =>
a_registro_salida <= registro_salida;
Salida <= (others => '0');
a_registro_aux_s <= registro_aux_s;
a_k <= k;
a_s((((conv_integer (Z))*5)+4) downto ((conv_integer (Z))*5)) <= s((((
```

```
conv_integer (T))*5)+4) downto ((conv_integer (T))*5));
a_s((((conv_integer (T))*5)+4) downto ((conv_integer (T))*5)) <=
registro_aux_s;
Sumador_A <= (others => '0');
Sumador_B <= (others => '0');
Sumador_C <= Sumas;
a_sumas <= Sumador_Salida;
a_x <= X;
a_y <= Y;
a_z <= Z;
a_t <= T;
Contador_Enable <= '0';
Final <= '0';
estado_siguiente <= estado_XOR1;
when estado_XOR1 =>
a_registro_salida <= registro_salida;
Salida <= (others => '0');
a_registro_aux_s <= registro_aux_s;
a_k <= k;
a_s <= s;
Sumador_A <= s((((conv_integer (X))*5)+4) downto ((conv_integer (X))*5
));
Sumador_B <= s((((conv_integer (Y))*5)+4) downto ((conv_integer (Y))*5
));
Sumador_C <= s((((conv_integer (Z))*5)+4) downto ((conv_integer (Z))*5
));
a_sumas <= Sumador_Salida + s((((conv_integer (T))*5)+4) downto ((
conv_integer (T))*5));
a_x <= X;
a_y <= Y;
a_z <= Z;
a_t <= T;
Contador_Enable <= '0';
Final <= '0';
estado_siguiente <= estado_XOR2;

when estado_XOR2 =>
a_registro_aux_s <= registro_aux_s;
a_k <= k;
a_s <= s;
Sumador_A <= (others => '0');
Sumador_B <= (others => '0');
Sumador_C <= (others => '0');
a_x <= X;
a_y <= Y;
a_z <= Z;
a_t <= T;
Contador_Enable <='1';
a_sumas <= sumas;
a_registro_salida(conv_integer (cuenta)*5) <= Entrada(conv_integer (
cuenta)*5) XOR s(conv_integer(sumas)*5);
461 a_registro_salida((conv_integer (cuenta)*5)+1) <= Entrada((conv_integer
(cuenta)*5+1)) XOR s((conv_integer(sumas)*5+1));
462 a_registro_salida((conv_integer (cuenta)*5)+2) <= Entrada((conv_integer
(cuenta)*5+2)) XOR s((conv_integer(sumas)*5+2));
a_registro_salida((conv_integer (cuenta)*5)+3) <= Entrada((conv_integer
(cuenta)*5+3)) XOR s((conv_integer(sumas)*5+3));
a_registro_salida((conv_integer (cuenta)*5)+4) <= Entrada((conv_integer
(cuenta)*5+4)) XOR s((conv_integer(sumas)*5+4));
if cuenta="11111" then
```

```
Final <= '1';
Salida <= registro_salida;
estado_siguiente <= inicia;
else
Final <= '0';
Salida <= (others => '0');
estado_siguiente <= sumax2;
end if;

end case;
end process;
end Behavioral;
```

----------Entidad Contador----------

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity Contador is
Port ( Clk : in STD_LOGIC;
Enable : in STD_LOGIC;
Reset : in STD_LOGIC;
Cuenta : out STD_LOGIC_VECTOR (4 downto 0));
end Contador;

architecture Behavioral of Contador is
signal Count: std_logic_vector (4 downto 0);
begin
process (Clk, Enable, Reset)
begin
if (Reset='1') then
Count <= "00000";
elsif (Clk'event and Clk='1') then
if (Enable='1') then
Count <= Count + '1';
end if;
end if;
end process;
Cuenta <= Count;

end Behavioral;
```

----------Entidad Sumador----------

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

use IEEE.std_logic_arith.all;

use IEEE.std_logic_unsigned.all;


entity Sumador is

Port ( A : in STD_LOGIC_VECTOR (4 downto 0);

B : in STD_LOGIC_VECTOR (4 downto 0);

C : in STD_LOGIC_VECTOR (4 downto 0);
```

```
Suma : out STD_LOGIC_VECTOR (4 downto 0));
end Sumador;

architecture Behavioral of Sumador is

begin

process (A,B,C)
begin
Suma <= A+B+C;
end process;

end Behavioral;
```

ANEXO I – CÓDIGO ARQUITECTURA SUMADOR DE 2 ENTRADAS Y 160 BITS

```
------------------Entidad Control-----------------
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
entity Control is
Port ( Clk : in STD_LOGIC;
Reset : in STD_LOGIC;
Entrada : in STD_LOGIC_VECTOR (159 downto 0);
Inicio : in STD_LOGIC;
Key : in STD_LOGIC_VECTOR (159 downto 0);
Final: out STD_LOGIC;
Salida : out STD_LOGIC_VECTOR (159 downto 0));
end Control;

architecture Behavioral of Control is

component Contador is
Port (Clk : in STD_LOGIC;
Enable : in STD_LOGIC;
Reset : in STD_LOGIC;
Cuenta : out STD_LOGIC_VECTOR (4 downto 0));
end component;

component Sumador is
Port (A : in STD_LOGIC_VECTOR (4 downto 0);
B : in STD_LOGIC_VECTOR (4 downto 0);
Suma : out STD_LOGIC_VECTOR (4 downto 0));
end component;
Signal Contador_Enable: STD_LOGIC;
Signal Cuenta: STD_LOGIC_VECTOR (4 downto 0);
Signal Sumador_A: STD_LOGIC_VECTOR (4 downto 0);
```

```
Signal Sumador_B: STD_LOGIC_VECTOR (4 downto 0);
Signal Sumador_Salida: STD_LOGIC_VECTOR (4 downto 0);
Signal X: STD_LOGIC_VECTOR (4 downto 0);
Signal Y: STD_LOGIC_VECTOR (4 downto 0);
66 Signal Z: STD_LOGIC_VECTOR (4 downto 0);
67 Signal T: STD_LOGIC_VECTOR (4 downto 0);
68 Signal a_x: STD_LOGIC_VECTOR (4 downto 0);
Signal a_y: STD_LOGIC_VECTOR (4 downto 0);
Signal a_z: STD_LOGIC_VECTOR (4 downto 0);
Signal a_t: STD_LOGIC_VECTOR (4 downto 0);
Signal K: STD_LOGIC_VECTOR (159 downto 0);
Signal a_k: STD_LOGIC_VECTOR (159 downto 0);
Signal s: STD_LOGIC_VECTOR (159 downto 0);
Signal a_s: STD_LOGIC_VECTOR (159 downto 0);
type estados is (inicia, guardar, sumax, sumay, sumaz, swap1, swap3, swap4, inicio2,
sumax2, sumay2, sumaz2, sumat, swap2, estado_XOR, suma2x, suma2y, suma2z, sumas1,
sumas2);
signal estado_actual: estados;
signal estado_siguiente: estados;
Signal registro_aux_s: STD_LOGIC_VECTOR (4 downto 0);
Signal a_registro_aux_s: STD_LOGIC_VECTOR (4 downto 0);
Signal registro_salida: STD_LOGIC_VECTOR (159 downto 0);
Signal a_registro_salida: STD_LOGIC_VECTOR (159 downto 0);
Signal sumas: STD_LOGIC_VECTOR (4 downto 0);
Signal a_sumas: STD_LOGIC_VECTOR (4 downto 0);


begin

Contador_h: Contador port map (Clk => Clk,
Enable => Contador_Enable,
Reset => Reset,
Cuenta => Cuenta);

Sumador_h: Sumador port map (A => Sumador_A,
B => Sumador_B,
Suma => Sumador_Salida);

process (Clk, Reset)
begin
if (reset = '1') then
X <= (others => '0');
Y <= (others => '0');
Z <= (others => '0');
T <= (others => '0');
elsif Clk'event and Clk='1' then
X <= a_x;
Y <= a_y;
Z <= a_z;
T <= a_t;
end if;
end process;

process (Clk, Reset)
begin
if (reset = '1') then
s <= (others => '0');
elsif Clk'event and Clk='1' then
s <= a_s;
end if;
```

```
end process;

process (Clk, Reset)
begin
if (reset = '1') then
registro_aux_s <= (others => '0');
elsif Clk'event and Clk='1' then
registro_aux_s <= a_registro_aux_s;
end if;
end process;

process (Clk, Reset)
begin
if (reset = '1') then
sumas <= (others => '0');
elsif Clk'event and Clk='1' then
sumas <= a_sumas;
end if;
end process;


process (Clk, Reset)
begin
if (reset = '1') then
registro_salida <= (others => '0');
elsif Clk'event and Clk='1' then
registro_salida <= a_registro_salida;
end if;
end process;

process (Clk, Reset)
begin
if (reset = '1') then
k <= (others => '0');
elsif Clk'event and Clk='1' then
k <= a_k;
end if;
end process;

process (Clk, Reset)
begin
if (Reset='1') then
estado_actual <= inicia;
elsif(Clk'event and Clk='1') then
estado_actual <= estado_siguiente;
end if;
end process;

process (estado_actual, cuenta, Key, k, s, X, Y, Z, T, Sumador_Salida, Entrada, Sumas,
registro_aux_s, Sumas, registro_salida, Inicio)
begin
a_s <= s;
a_registro_salida <= registro_salida;

case estado_actual is
when inicia =>
a_x <= (others => '0');
a_y <= (others => '0');
a_z <= (others => '0');
a_t <= (others => '0');
```

```
a_registro_aux_s <= (others => '0');
Contador_Enable <='0';
a_k <= Key;
a_s <=
"11111111101110111100110111101011001110001011110110101011010010011100101000110000011110
11100110101100010110101001001010000011100110001010010000011000100000100000";
a_registro_salida <= (OTHERS => '0');
Sumador_A <= (OTHERS => '0');
Sumador_B <= (OTHERS => '0');
a_sumas <= (OTHERS => '0');
Salida <= registro_salida;
Final <= '0';
if Inicio = '1' then
estado_siguiente <= guardar;
else
estado_siguiente <= inicia;
end if;
when guardar =>
Salida <= (OTHERS => '0');
a_registro_aux_s <= registro_aux_s;
Contador_Enable <='0';
a_k <= Key;
a_s <=
"11111111101110111100110111101011001110001011110110101011010010011100101000110000011110
11100110101100010110101001001010000011100110001010010000011000100000100000";
a_registro_salida <= registro_salida;
a_x <= X;
a_y <= Y;
a_z <= Z;
a_t <= T;
Sumador_A <= (OTHERS => '0');
Sumador_B <= (OTHERS => '0');
a_sumas <= sumas;
Final <= '0';
estado_siguiente <= sumax;
when sumax =>
Salida <= (OTHERS => '0');
a_registro_salida <= registro_salida;
a_registro_aux_s <= registro_aux_s;
a_k <= k;
a_s <= s;
Sumador_A <= X;
Sumador_B <= s((((conv_integer (cuenta))*5)+4) downto ((conv_integer (
cuenta))*5));
a_x <= Sumador_Salida;
a_y <= Y;
a_z <= Z;
a_t <= T;
a_sumas <= sumas;
Contador_Enable <='0';
Final <= '0';
estado_siguiente <= suma2x;
when suma2x =>
Salida <= (OTHERS => '0');
a_registro_salida <= registro_salida;
a_registro_aux_s <= registro_aux_s;
a_k <= k;
a_s <= s;
```

```
Sumador_A <= X;
Sumador_B <= K((((conv_integer (cuenta))*5)+4) downto ((conv_integer (
cuenta))*5));
a_x <= Sumador_Salida;
a_y <= Y;
a_z <= Z;
a_t <= T;
a_sumas <= sumas;
Contador_Enable <='0';
Final <= '0';
estado_siguiente <= sumay;

when sumay =>
Salida <= (OTHERS => '0');
a_registro_salida <= registro_salida;
a_registro_aux_s <= registro_aux_s;
a_k <= k;
a_s <= s;
Sumador_A <= Y;
Sumador_B <= s((((conv_integer (X))*5)+4) downto ((conv_integer (X))*5
));
a_x <= X;
a_y <= Sumador_Salida;
a_z <= Z;
a_t <= T;
a_sumas <= sumas;
Contador_Enable <='0';
Final <= '0';
estado_siguiente <= suma2y;
when suma2y =>
Salida <= (OTHERS => '0');
a_registro_salida <= registro_salida;
a_registro_aux_s <= registro_aux_s;
a_k <= k;
a_s <= s;
Sumador_A <= Y;
Sumador_B <= K((((conv_integer (X+cuenta))*5)+4) downto ((conv_integer
(X+cuenta))*5));
a_x <= X;
a_y <= Sumador_Salida;
a_z <= Z;
a_t <= T;
a_sumas <= sumas;
Contador_Enable <='0';
Final <= '0';
estado_siguiente <= sumaz;
when sumaz =>
Salida <= (OTHERS => '0');
a_registro_salida <= registro_salida;
a_registro_aux_s <= s((((conv_integer (cuenta))*5)+4) downto ((
conv_integer (cuenta))*5));
a_k <= k;
a_s <= s;
Sumador_A <= Z;
Sumador_B <= s((((conv_integer (Y))*5)+4) downto ((conv_integer (Y))*5
));
a_x <= X;
a_y <= Y;
a_z <= Sumador_Salida;
a_t <= T;
```

```
a_sumas <= sumas;
Contador_Enable <= '0';
Final <= '0';
estado_siguiente <= suma2z;
when suma2z =>
Salida <= (OTHERS => '0');
a_registro_salida <= registro_salida;
a_registro_aux_s <= registro_aux_s;
a_k <= k;
a_s <= s;
a_s((((conv_integer (cuenta))*5)+4) downto ((conv_integer (cuenta))*5))
<= s((((conv_integer (X))*5)+4) downto ((conv_integer (X))*5));
a_s((((conv_integer (X))*5)+4) downto ((conv_integer (X))*5)) <=
registro_aux_s;
Sumador_A <= Z;
Sumador_B <= K((((conv_integer (Y+cuenta))*5)+4) downto ((conv_integer
(Y+cuenta))*5));
a_x <= X;
a_y <= Y;
a_z <= Sumador_Salida;
a_t <= T;
a_sumas <= sumas;
Contador_Enable <= '0';
Final <= '0';
estado_siguiente <= swap1;
when swap1 =>
Salida <= (OTHERS => '0');
a_registro_salida <= registro_salida;
a_registro_aux_s <= s((((conv_integer (Y))*5)+4) downto ((conv_integer
(Y))*5));
Contador_Enable <='0';
a_k <= k;
a_s <= s;
Sumador_A <= (OTHERS => '0');
Sumador_B <= (OTHERS => '0');
a_x <= X;
a_y <= Y;
a_z <= Z;
a_t <= T;
a_sumas <= sumas;
Final <= '0';
estado_siguiente <= swap3;


when swap3 =>
Salida <= (OTHERS => '0');
a_registro_salida <= registro_salida;
a_registro_aux_s <= registro_aux_s;
Contador_Enable <='1';
a_k <= k;
Sumador_A <= (OTHERS => '0');
Sumador_B <= (OTHERS => '0');
a_x <= X;
a_y <= Y;
a_z <= Z;
a_t <= T;
a_sumas <= sumas;
a_s((((conv_integer (Y))*5)+4) downto ((conv_integer (Y))*5)) <= s((((
conv_integer (Z))*5)+4) downto ((conv_integer (Z))*5));
a_s((((conv_integer (Z))*5)+4) downto ((conv_integer (Z))*5)) <=
```

```
registro_aux_s;
Final <= '0';
if cuenta="11111" then
estado_siguiente <= inicio2;
else
estado_siguiente <= sumax;
end if;

when inicio2 =>
Salida <= (OTHERS => '0');
a_registro_salida <= registro_salida;
a_registro_aux_s <= registro_aux_s;
a_x <= (OTHERS => '0');
a_y <= (OTHERS => '0');
a_z <= (OTHERS => '0');
a_t <= (OTHERS => '0');
Contador_Enable <='0';
a_k <= k;
a_s <= s;
a_sumas <= sumas;
Sumador_A <= (OTHERS => '0');
Sumador_B <= (OTHERS => '0');
Final <= '0';
estado_siguiente <= sumax2;
when sumax2 =>
a_registro_salida <= registro_salida;
Salida <= (OTHERS => '0');
a_registro_aux_s <= registro_aux_s;
a_k <= k;
a_s <= s;
Contador_Enable <='0';
Sumador_A <= X;
Sumador_B <= "00001";
a_sumas <= sumas;
a_x <= Sumador_Salida;
a_y <= Y;
a_z <= Z;
a_t <= T;
Final <= '0';
when sumay2 =>
a_registro_salida <= registro_salida;
Salida <= (OTHERS => '0');
a_registro_aux_s <= registro_aux_s;
a_k <= k;
a_s <= s;
Sumador_A <= Y;
Sumador_B <= s((((conv_integer (X))*5)+4) downto ((conv_integer (X))*5
));
a_sumas <= sumas;
a_x <= X;
a_y <= Sumador_Salida;
a_z <= Z;
a_t <= T;
Contador_Enable <='0';
Final <= '0';
estado_siguiente <= sumaz2;
when sumaz2 =>
a_registro_salida <= registro_salida;
Salida <= (OTHERS => '0');
a_registro_aux_s <= s((((conv_integer (X))*5)+4) downto ((conv_integer
```

```
(X))*5));
a_k <= k;
a_s <= s;
403 Sumador_A <= Z;
Sumador_B <= s((((conv_integer (Y))*5)+4) downto ((conv_integer (Y))*5
));
a_sumas <= sumas;
a_x <= X;
a_y <= Y;
a_z <= Sumador_Salida;
a_t <= T;
Contador_Enable <= '0';
Final <= '0';
estado_siguiente <= sumat;
when sumat =>
a_registro_salida <= registro_salida;
Salida <= (OTHERS => '0');
a_registro_aux_s <= registro_aux_s;
a_k <= k;
a_s((((conv_integer (X))*5)+4) downto ((conv_integer (X))*5)) <= s((((
conv_integer (Y))*5)+4) downto ((conv_integer (Y))*5));
a_s((((conv_integer (Y))*5)+4) downto ((conv_integer (Y))*5)) <=
registro_aux_s;
Sumador_A <= T;
Sumador_B <= s((((conv_integer (T))*5)+4) downto ((conv_integer (T))*5
));
a_sumas <= sumas;
a_x <= X;
a_y <= Y;
a_z <= Z;
a_t <= Sumador_Salida;
Contador_Enable <='0';
Final <= '0';
estado_siguiente <= swap2;
when swap2 =>
a_registro_salida <= registro_salida;
Salida <= (OTHERS => '0');
a_registro_aux_s <= s((((conv_integer (Z))*5)+4) downto ((conv_integer
(Z))*5));
a_k <= k;
a_s <= s;
Sumador_A <= X;
Sumador_B <= Y;
a_sumas <= sumas;
a_x <= X;
a_y <= Y;
a_z <= Z;
a_t <= T;
Contador_Enable <= '0';
Final <= '0';
estado_siguiente <= swap4;
when swap4 =>
a_registro_salida <= registro_salida;
Salida <= (OTHERS => '0');
a_registro_aux_s <= registro_aux_s;
a_k <= k;
a_s((((conv_integer (Z))*5)+4) downto ((conv_integer (Z))*5)) <= s((((
conv_integer (T))*5)+4) downto ((conv_integer (T))*5));
a_s((((conv_integer (T))*5)+4) downto ((conv_integer (T))*5)) <=
registro_aux_s;
```

```
Sumador_A <= (OTHERS => '0');
Sumador_B <= (OTHERS => '0');
a_sumas <=Sumas;
a_x <= X;
a_y <= Y;
a_z <= Z;
a_t <= T;
Contador_Enable <= '0';
Final <= '0';
estado_siguiente <= sumas1;
when sumas1 =>
a_registro_salida <= registro_salida;
Salida <= (OTHERS => '0');
a_registro_aux_s <= registro_aux_s;
a_k <= k;
a_s <= s;
Sumador_A <= s((((conv_integer (X))*5)+4) downto ((conv_integer (X))*5
));
Sumador_B <= s((((conv_integer (Y))*5)+4) downto ((conv_integer (Y))*5
));
a_sumas <= Sumador_Salida;
a_x <= X;
a_y <= Y;
a_z <= Z;
a_t <= T;
Contador_Enable <= '0';
Final <= '0';
estado_siguiente <= sumas2;
when sumas2 =>
a_registro_salida <= registro_salida;
Salida <= (OTHERS => '0');
a_registro_aux_s <= registro_aux_s;
a_k <= k;
a_s <= s;
Sumador_A <= s((((conv_integer (Z))*5)+4) downto ((conv_integer (Z))*5
));
Sumador_B <= s((((conv_integer (T))*5)+4) downto ((conv_integer (T))*5
));
a_sumas <= Sumador_Salida + Sumas;
a_x <= X;
a_y <= Y;
a_z <= Z;
a_t <= T;
Contador_Enable <= '0';
Final <= '0';
estado_siguiente <= estado_XOR;

when estado_XOR =>
a_registro_aux_s <= registro_aux_s;
a_k <= k;
a_s <= s;
Sumador_A <= (OTHERS => '0');
Sumador_B <= (OTHERS => '0');
a_x <= X;
a_y <= Y;
a_z <= Z;
a_t <= T;
Contador_Enable <='1';
a_sumas <= sumas;
a_registro_salida(conv_integer (cuenta)*5) <= Entrada(conv_integer (
```

```
cuenta)*5) XOR s(conv_integer(sumas)*5);
a_registro_salida((conv_integer (cuenta)*5)+1) <= Entrada((conv_integer
(cuenta)*5+1)) XOR s((conv_integer(sumas)*5+1));
a_registro_salida((conv_integer (cuenta)*5)+2) <= Entrada((conv_integer
(cuenta)*5+2)) XOR s((conv_integer(sumas)*5+2));
a_registro_salida((conv_integer (cuenta)*5)+3) <= Entrada((conv_integer
(cuenta)*5+3)) XOR s((conv_integer(sumas)*5+3));
a_registro_salida((conv_integer (cuenta)*5)+4) <= Entrada((conv_integer
(cuenta)*5+4)) XOR s((conv_integer(sumas)*5+4));
if cuenta="11111" then
Final <= '1';
Salida <= registro_salida;
estado_siguiente <= inicia;
else
Final <= '0';
Salida <= (OTHERS => '0');
estado_siguiente <= sumax2;
end if;

end case;
end process;
end Behavioral;
```

----------Entidad Contador----------

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity Contador is
Port ( Clk : in STD_LOGIC;
Enable : in STD_LOGIC;
Reset : in STD_LOGIC;
Cuenta : out STD_LOGIC_VECTOR (4 downto 0));
end Contador;

architecture Behavioral of Contador is
signal Count: std_logic_vector (4 downto 0);
begin
process (Clk, Enable, Reset)
begin
if (Reset='1') then
Count <= "00000";
elsif (Clk'event and Clk='1') then
if (Enable='1') then
Count <= Count + '1';
end if;
end if;
end process;
Cuenta <= Count;

end Behavioral;
```

----------Entidad Sumador----------

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;
```

```
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity Sumador is
Port ( A : in STD_LOGIC_VECTOR (4 downto 0);
B : in STD_LOGIC_VECTOR (4 downto 0);
C : in STD_LOGIC_VECTOR (4 downto 0);
Suma : out STD_LOGIC_VECTOR (4 downto 0));
end Sumador;

architecture Behavioral of Sumador is

begin

process (A,B,C)
begin
Suma <= A+B+C;
end process;

end Behavioral;
```

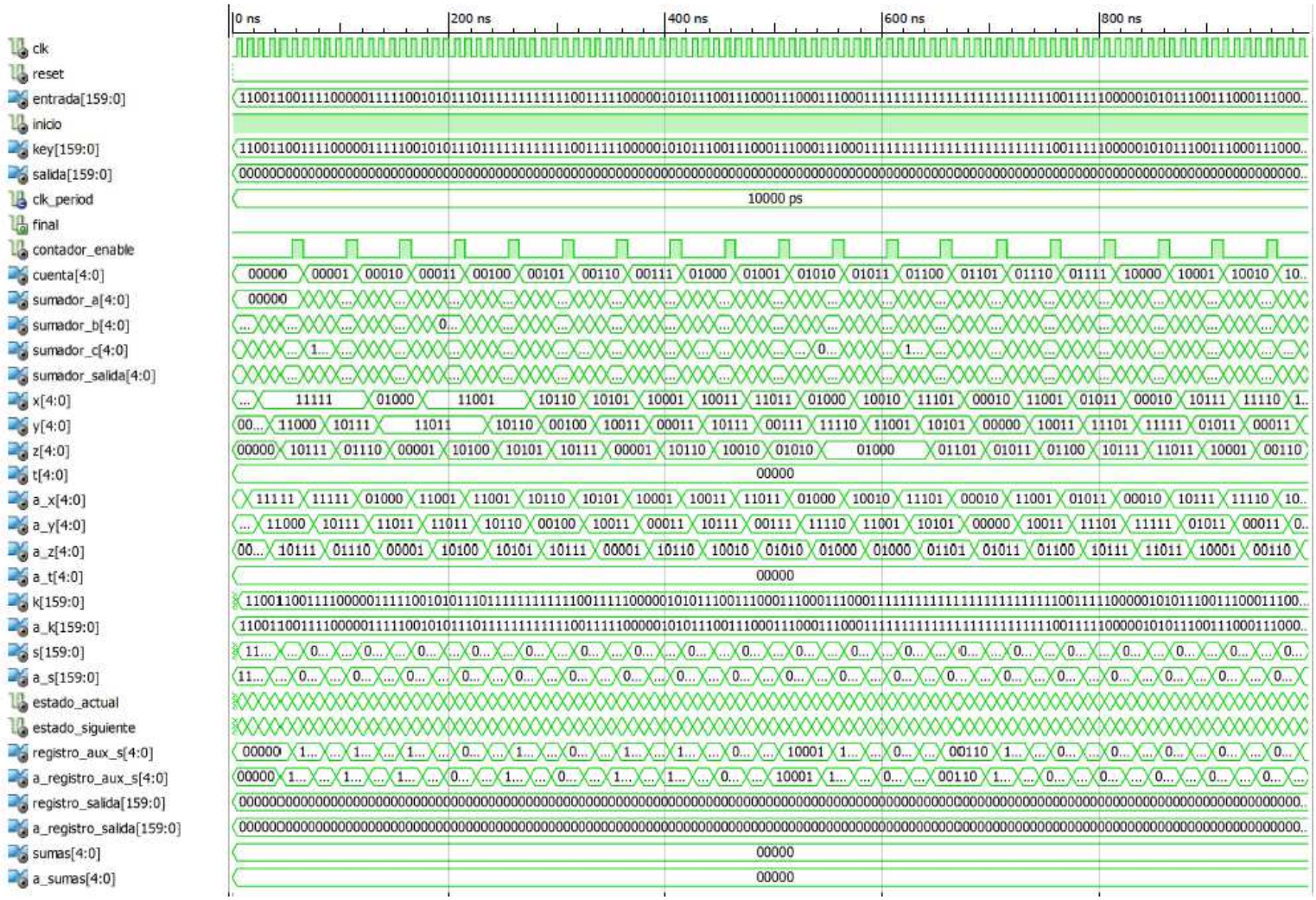
ANEXO J - CÓDIGO BANCO DE PRUEBAS PARA 160 BITS

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY prueba IS
END prueba;

ARCHITECTURE behavior OF prueba IS


COMPONENT Control
PORT(
Clk : IN std_logic;
Reset : IN std_logic;
Entrada : IN std_logic_vector(159 downto 0);
Inicio : IN std_logic;
Key : IN std_logic_vector(159 downto 0);
Salida : OUT std_logic_vector(159 downto 0)
);
END COMPONENT;


--Inputs
signal Clk : std_logic := '0';
```

```
signal Reset : std_logic := '0';
signal Entrada : std_logic_vector(159 downto 0) := (others => '0');
signal Inicio : std_logic := '0';
signal Key : std_logic_vector(159 downto 0) := (others => '0');

--Outputs
signal Salida : std_logic_vector(159 downto 0);

-- Clock period definitions
constant Clk_period : time := 10 ns;

BEGIN

-- Instantiate the Unit Under Test (UUT)
uut: Control PORT MAP (
Clk => Clk,
Reset => Reset,
Entrada => Entrada,
Inicio => Inicio,
Key => Key,
Salida => Salida
);

-- Clock process definitions
Clk_process :process
begin
Clk <= '0';
wait for Clk_period/2;
Clk <= '1';
wait for Clk_period/2;
end process;


-- Stimulus process
stim_proc: process
begin
Reset <= '1';
Inicio <='1';
wait for 100 ps;
Inicio <= '1';
Reset <= '0';
wait for 100 ps;
Inicio <= '1';
Entrada <=
"11001100111100000111110010101110111111111111001111100000101011100111000111000111
11111111111111111111111001111100000101011100111000111000111000111111111111";

Key <=
"11001100111100000111110010101110111111111111001111100000101011100111000111000111
11111111111111111111111001111100000101011100111000111000111000111111111111";


wait;
end process;
END
```

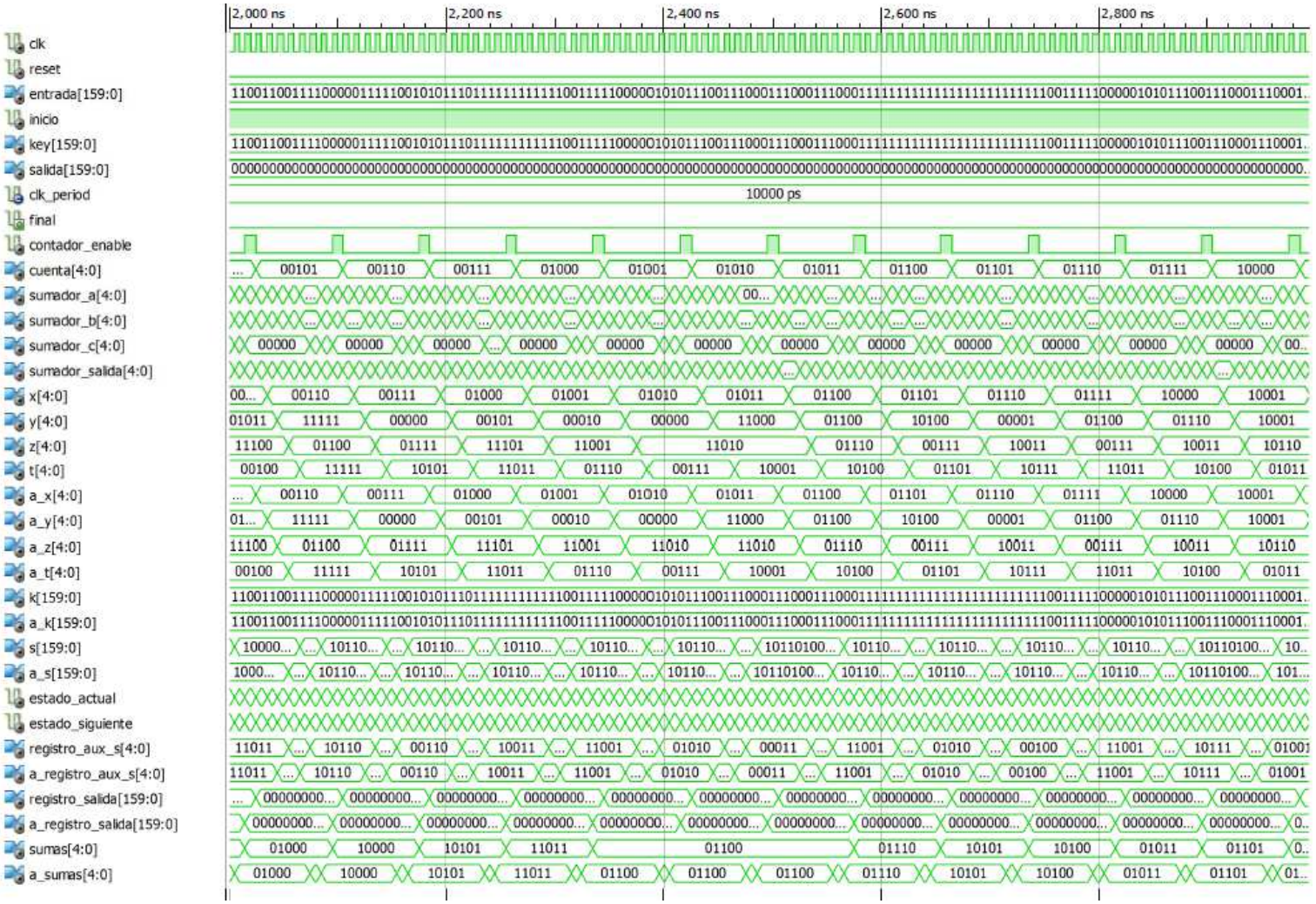
ANEXO K – SIMULACIONES ARQUITECTURA DOS SUMADORES Y 160 BITS

- SIMULACIÓN 0 - 1000 ns

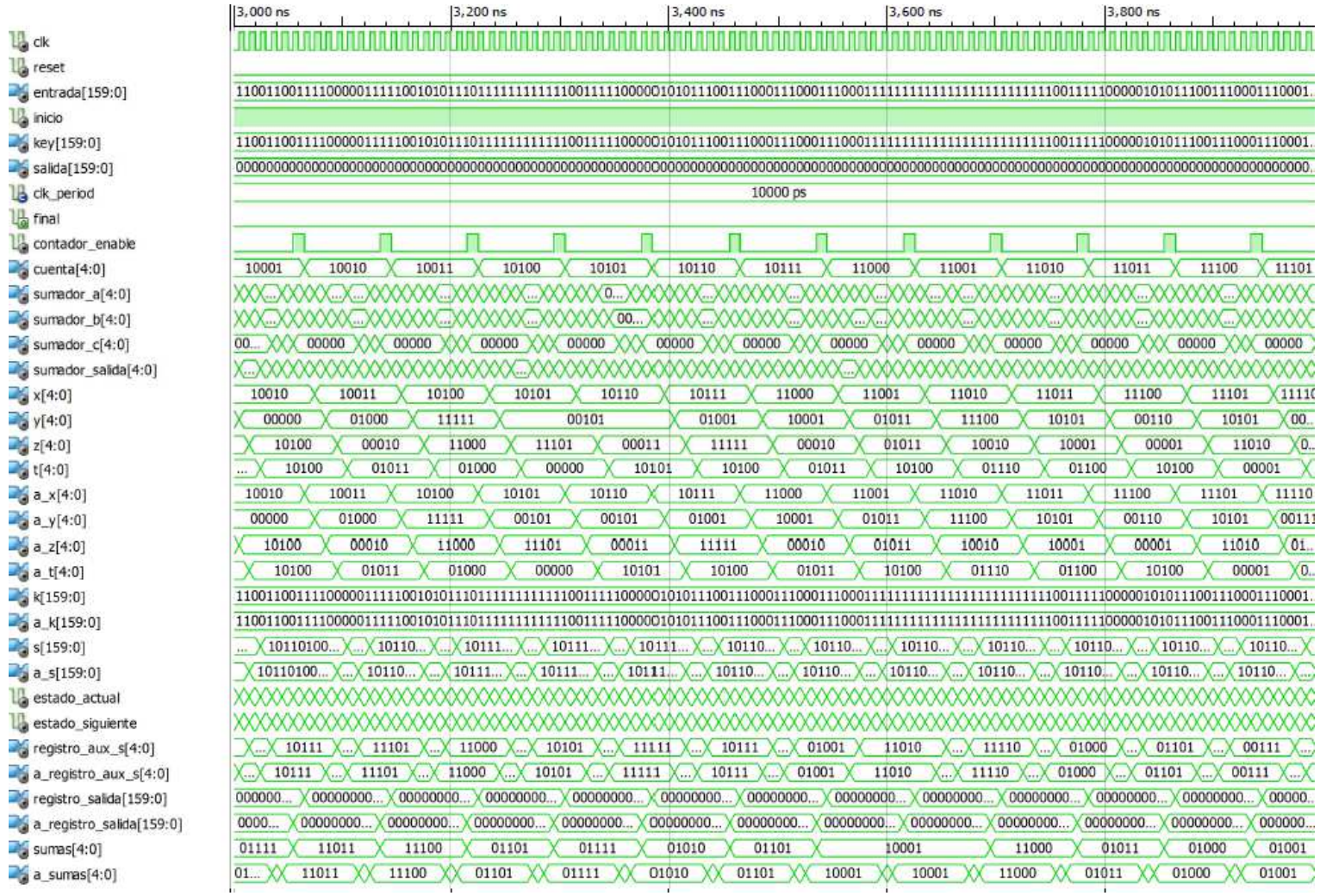


- SIMULACIÓN 1000 - 2000 ns

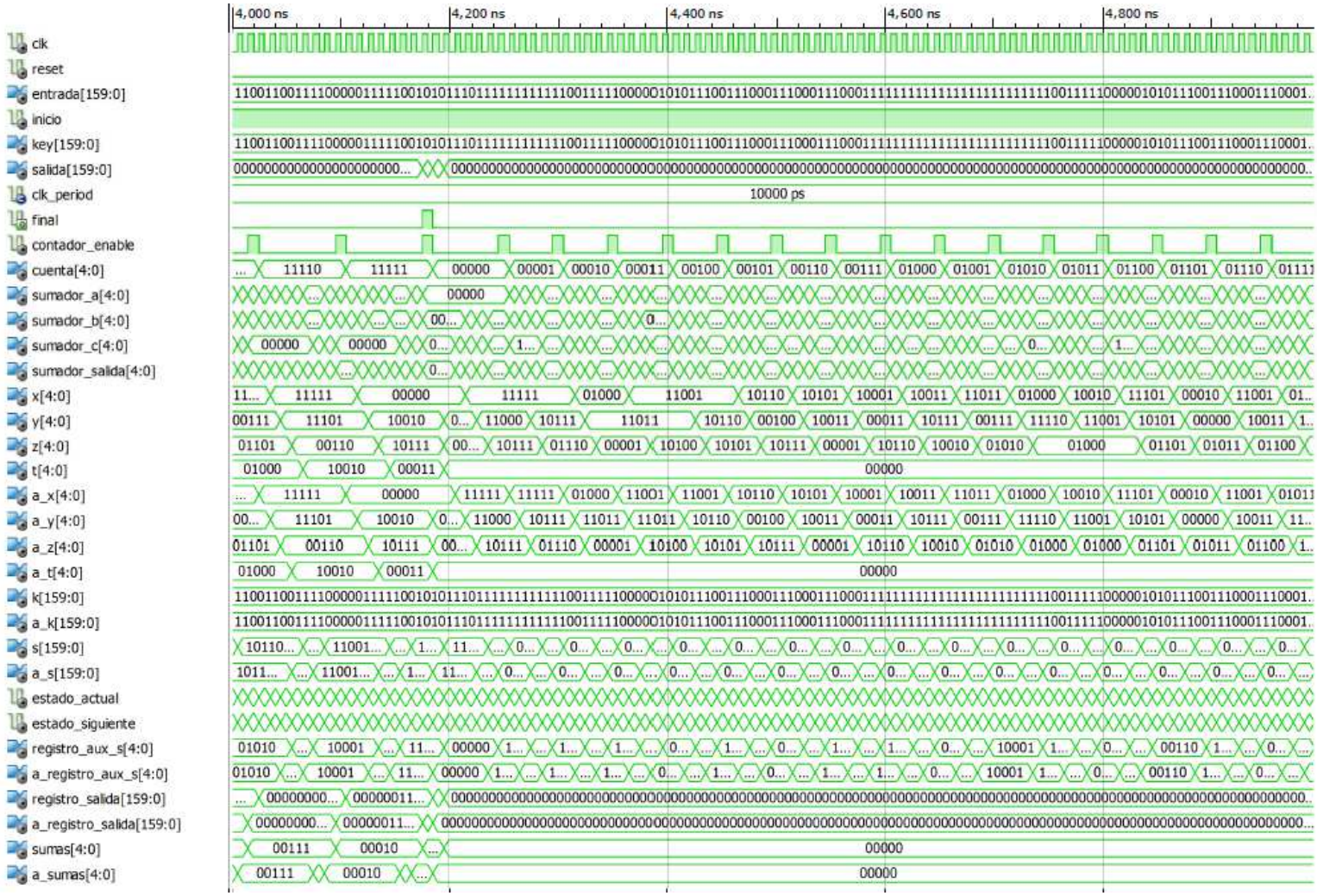
- SIMULACIÓN 2000 - 3000 ns



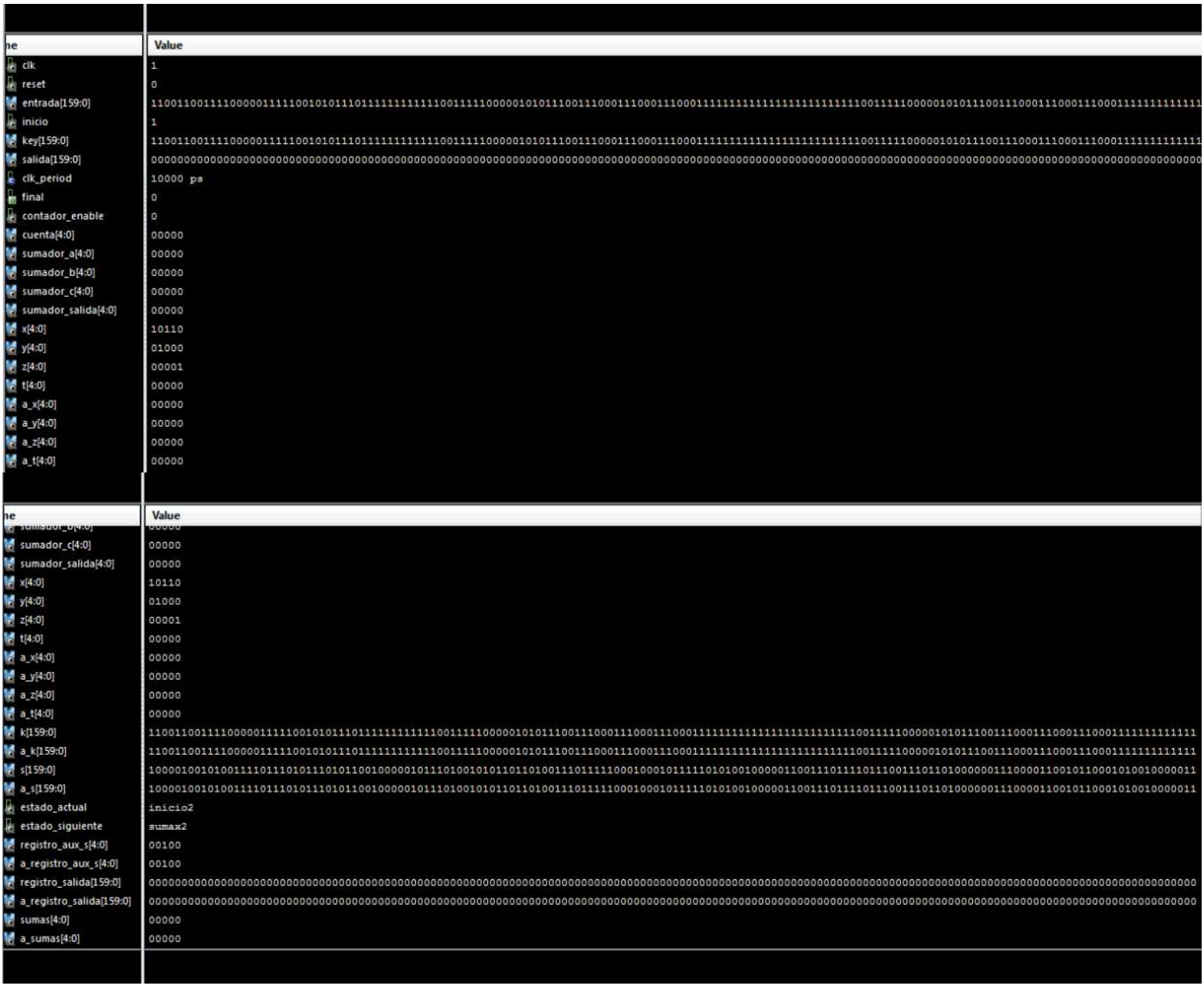
- SIMULACIÓN 3000 - 4000 ns



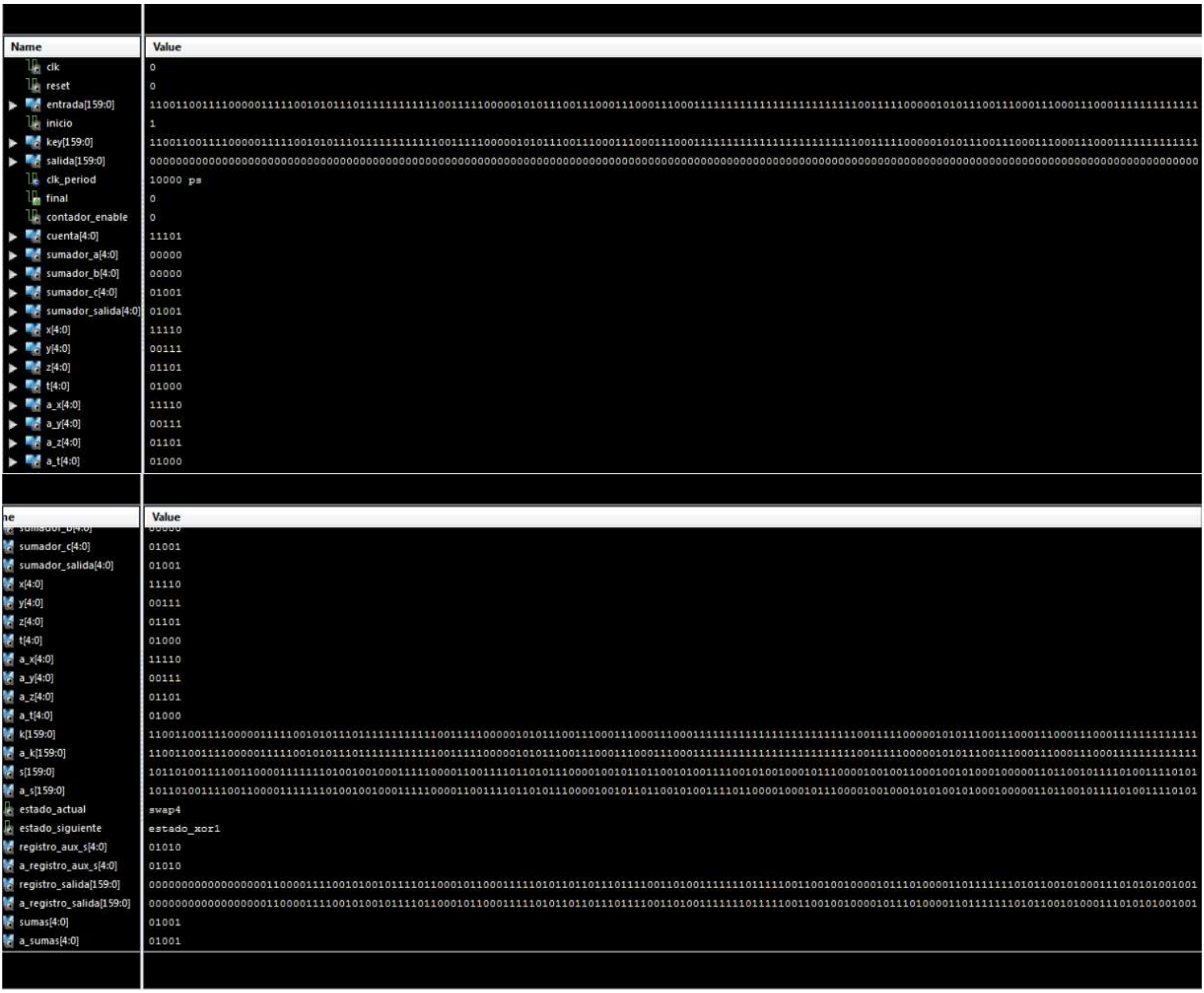
- SIMULACIÓN 4000 - 5000 ns



- RESULTADOS TRAS LA PRIMERA PARTE DEL CIRCUITO



- RESULTADOS FINALES

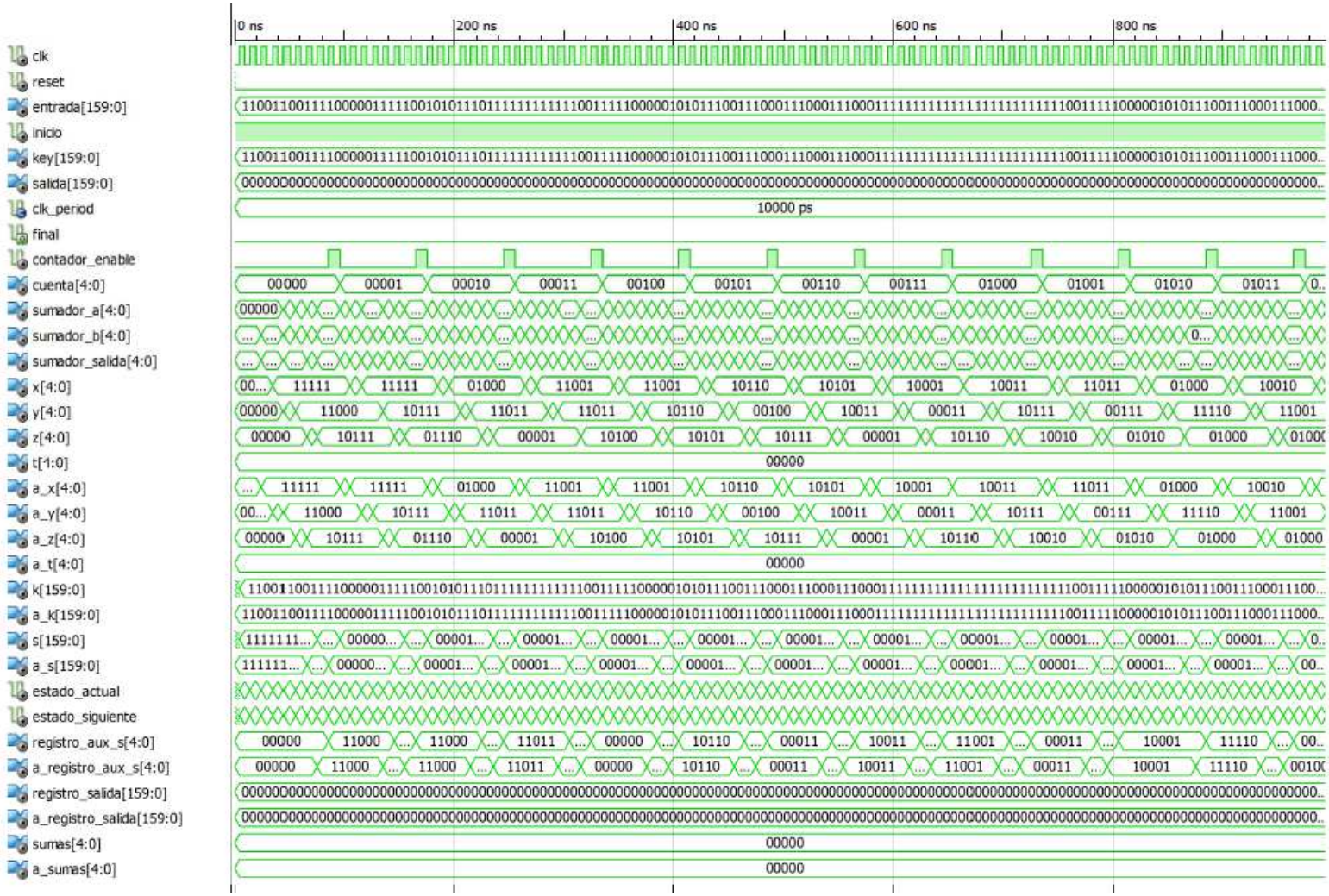


Name	Value
clk	0
reset	0
entrada[159:0]	11001100111100000111110010101110111111111111001111100000101011100111000111000111100011111111111111111111111111001111100000101011100111000111000111100011111111111
inicio	1
key[159:0]	11001100111100000111110010101110111111111111001111100000101011100111000111000111100011111111111111111111111111001111100000101011100111000111000111100011111111111
salida[159:0]	000
clk_period	10000 ps
final	0
contador_enable	0
cuenta[4:0]	11101
sumador_a[4:0]	00000
sumador_b[4:0]	00000
sumador_c[4:0]	01001
sumador_salida[4:0]	01001
x[4:0]	11110
y[4:0]	00111
z[4:0]	01101
t[4:0]	01000
a_x[4:0]	11110
a_y[4:0]	00111
a_z[4:0]	01101
a_t[4:0]	01000

ne	Value
sumador_b[4:0]	00000
sumador_c[4:0]	01001
sumador_salida[4:0]	01001
x[4:0]	11110
y[4:0]	00111
z[4:0]	01101
t[4:0]	01000
a_x[4:0]	11110
a_y[4:0]	00111
a_z[4:0]	01101
a_t[4:0]	01000
k[159:0]	11001100111100000111110010101110111111111111001111100000101011100111000111000111100011111111111111111111111111001111100000101011100111000111000111100011111111111
a_k[159:0]	11001100111100000111110010101110111111111111001111100000101011100111000111000111100011111111111111111111111111001111100000101011100111000111000111100011111111111
s[159:0]	10110100111100110000111111101001001000111111000011001111011010111000010010111011001010011110010100100010111000010010011000100101000010000110110010111101001111010101
a_s[159:0]	10110100111100110000111111101001001000111111000011001111011010111000010010111011001010011110010100100010111000010010011000100101000010000110110010111101001111010101
estado_actual	swap4
estado_siguiente	estado_xor1
registro_aux_s[4:0]	01010
a_registro_aux_s[4:0]	01010
registro_salida[159:0]	00000000000000000001100001111001010010111010001011000111101011011101110011010011111101111001100100100001011010000110111111101011001010001110101010010001
a_registro_salida[159:0]	00000000000000000001100001111001010010111010001011000111101011011101110011010011111101111001100100100001011010000110111111101011001010001110101010010001
sumas[4:0]	01001
a_sumas[4:0]	01001

ANEXO L – SIMULACIONES ARQUITECTURA CON UN SUMADOR DE 2 ENTRADAS Y 160 BITS

- SIMULACIÓN 0 - 1000 ns

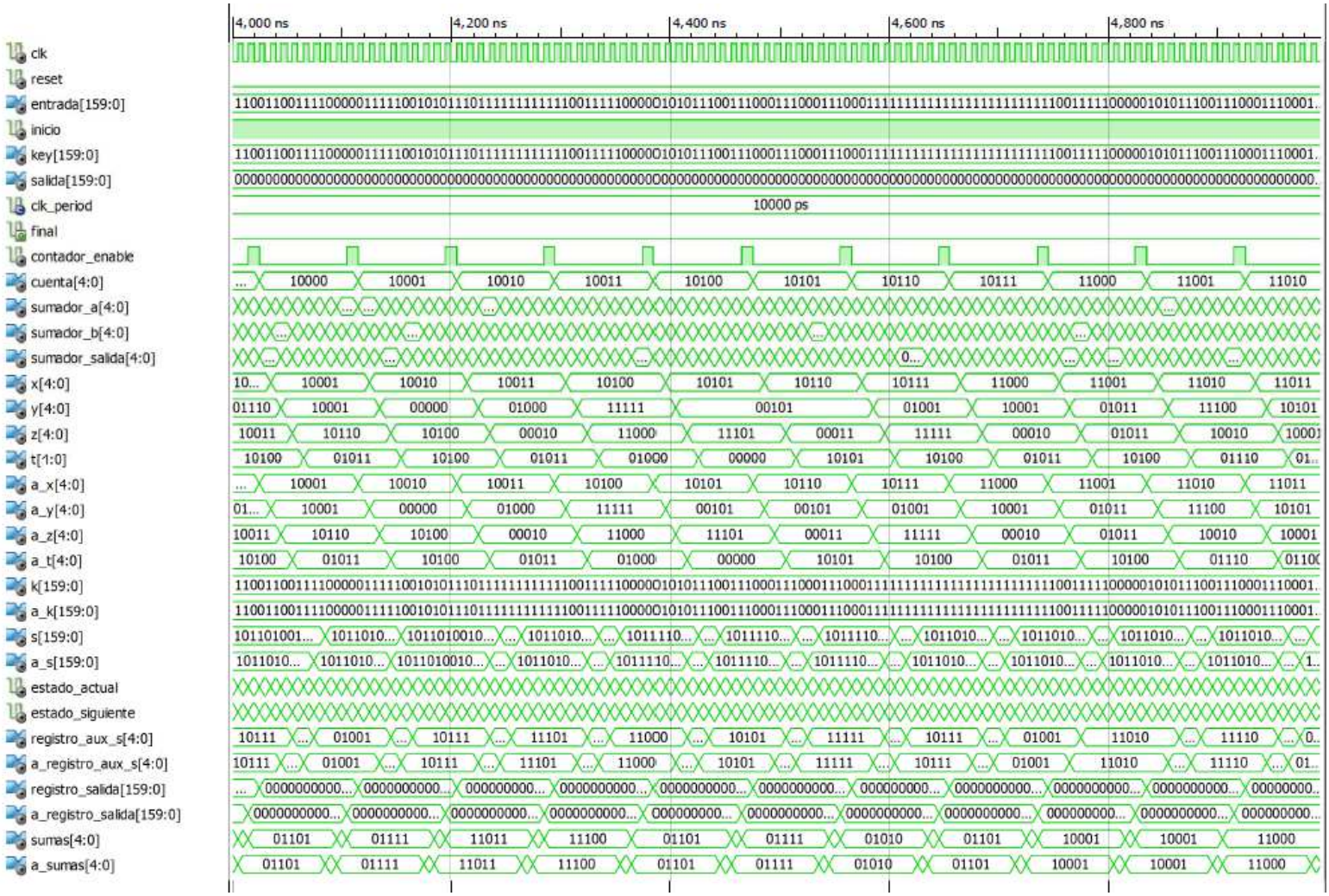


- SIMULACIÓN 1000 - 2000 ns

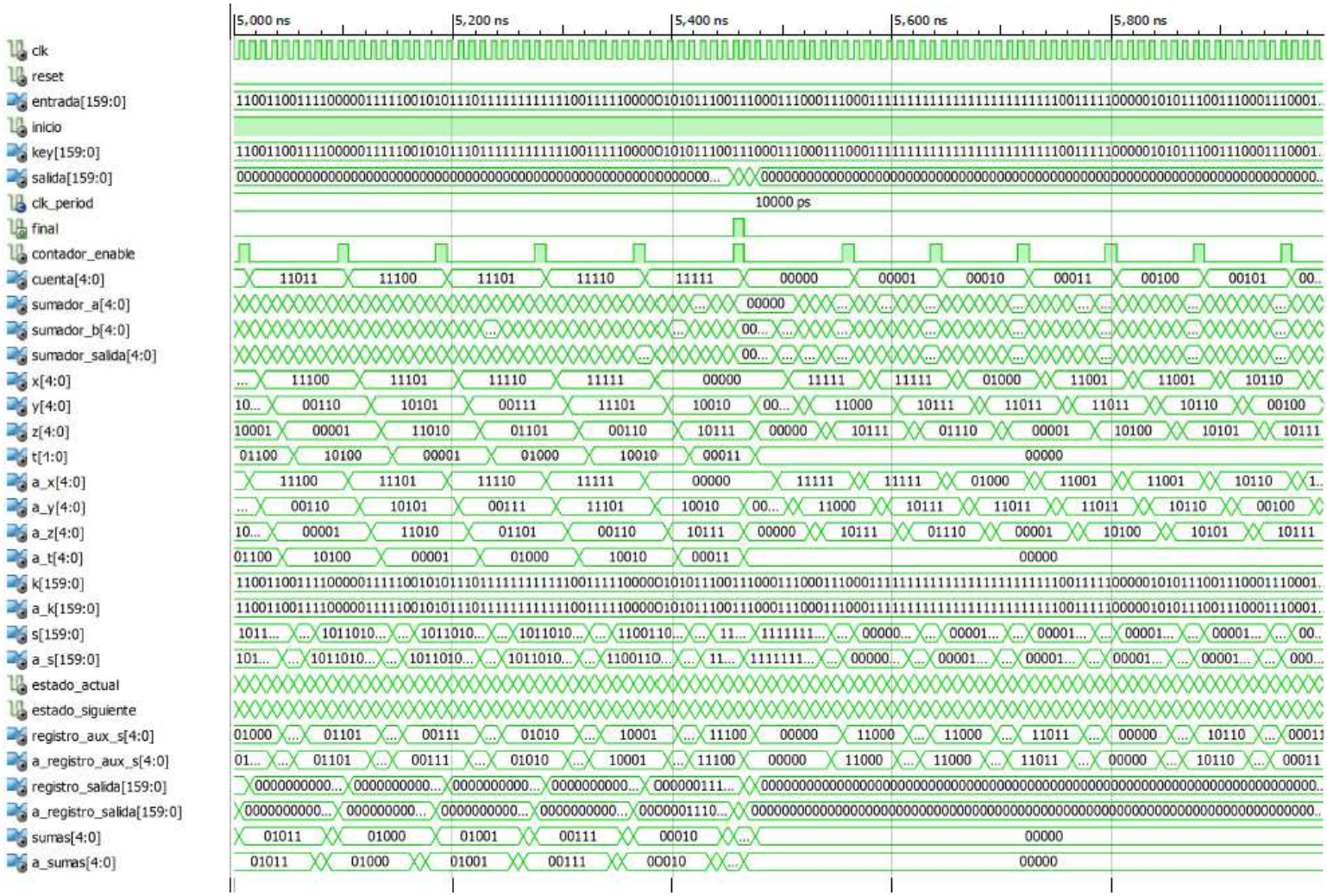
- SIMULACIÓN 2000 - 3000 ns

- SIMULACIÓN 3000 - 4000 ns

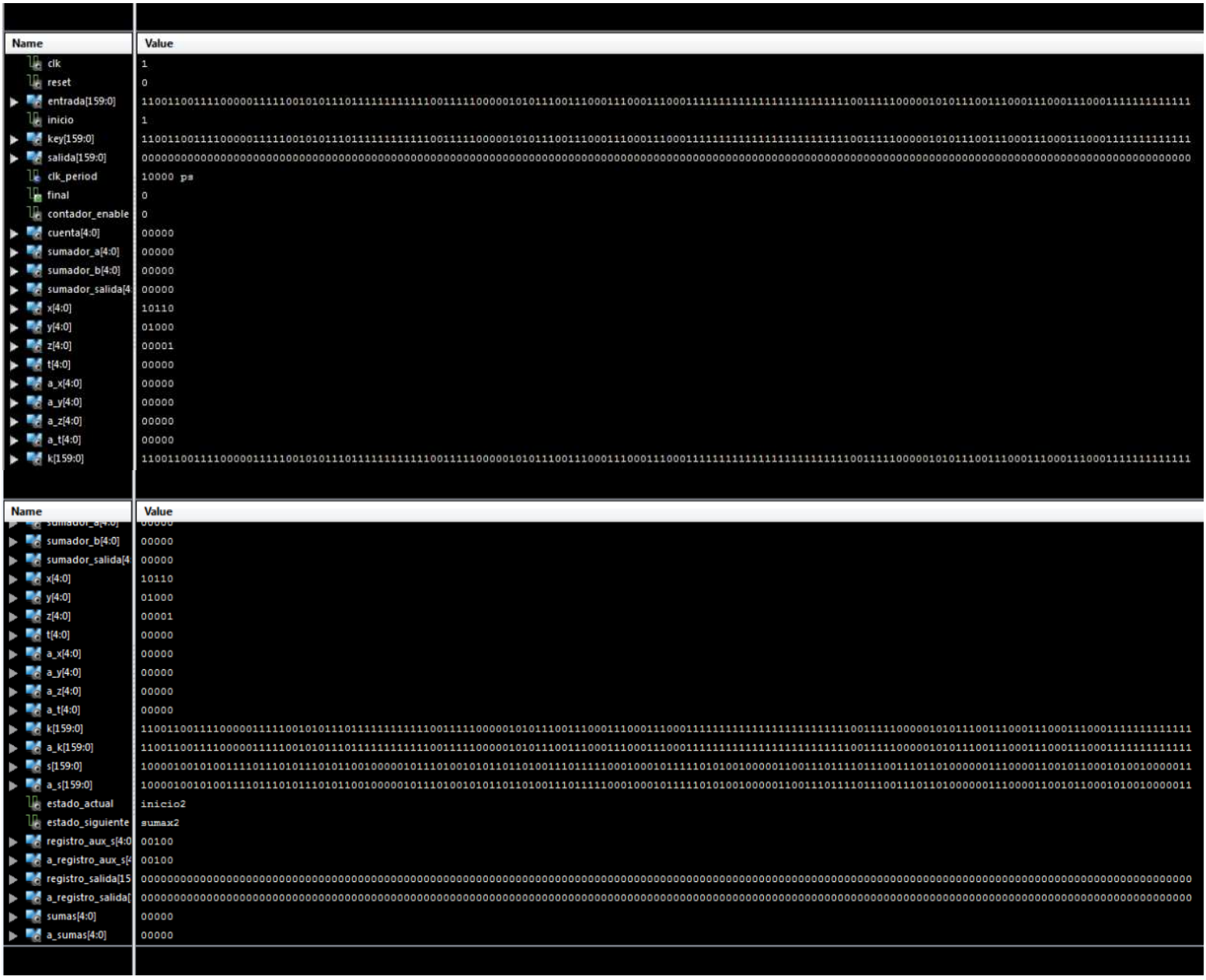
- SIMULACIÓN 4000 - 5000 ns



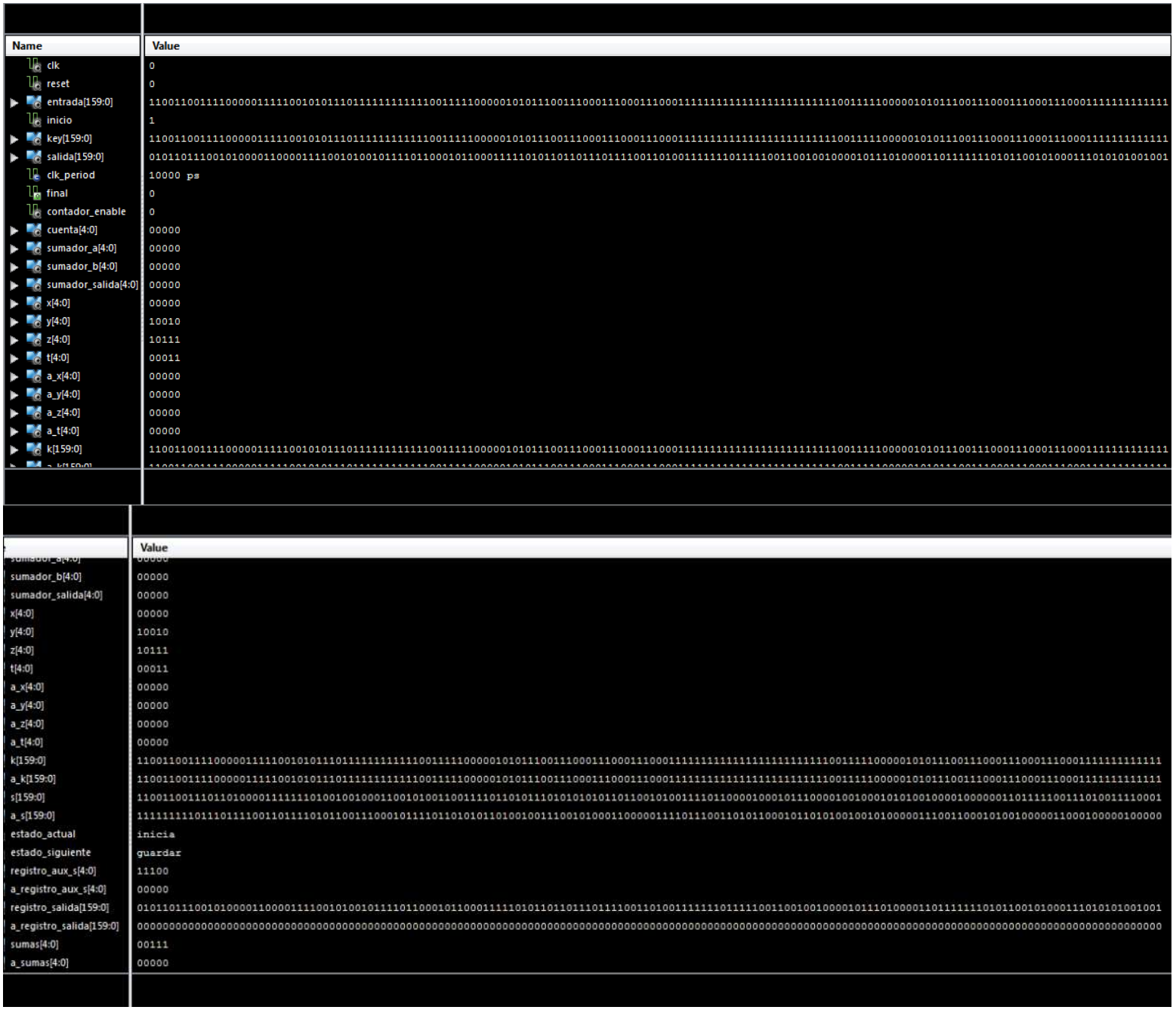
- SIMULACIÓN 5000 - 6000 ns



- RESULTADOS TRAS LA PRIMERA PARTE DEL CIRCUITO



- RESULTADOS FINALES



Name	Value
clk	0
reset	0
entrada[159:0]	1100110011110000011111001010111011111111111100111110000010101110011100011100011100011111111111111111111111100111110000010101110011100011100011100011111111111111
inicio	1
key[159:0]	1100110011110000011111001010111011111111111100111110000010101110011100011100011100011111111111111111111111100111110000010101110011100011100011100011111111111111
salida[159:0]	0101101110010100001100001111001010010111011000101100011111010110110111011110011010011111101111100110010010000101110100001101111111010110010100011101010100100
clk_period	10000 ps
final	0
contador_enable	0
cuenta[4:0]	00000
sumador_a[4:0]	00000
sumador_b[4:0]	00000
sumador_salida[4:0]	00000
x[4:0]	00000
y[4:0]	10010
z[4:0]	10111
t[4:0]	00011
a_x[4:0]	00000
a_y[4:0]	00000
a_z[4:0]	00000
a_t[4:0]	00000
k[159:0]	1100110011110000011111001010111011111111111100111110000010101110011100011100011100011111111111111111111111100111110000010101110011100011100011100011111111111111

	Value
sumador_b[4:0]	00000
sumador_salida[4:0]	00000
x[4:0]	00000
y[4:0]	10010
z[4:0]	10111
t[4:0]	00011
a_x[4:0]	00000
a_y[4:0]	00000
a_z[4:0]	00000
a_t[4:0]	00000
k[159:0]	1100110011110000011111001010111011111111111100111110000010101110011100011100011100011111111111111111111111100111110000010101110011100011100011100011111111111111
a_k[159:0]	1100110011110000011111001010111011111111111100111110000010101110011100011100011100011111111111111111111111100111110000010101110011100011100011100011111111111111
s[159:0]	1100110011101010000111111101001001000110010100110011101101011010101011011001010011101100001000101110000100100010101001000010000001101111001110100111100011
a_s[159:0]	1111111110110111100110111010110011100010111101101010110100100111001010001100000111011100110101100010110101001001010000011100110001010010000011000100000100000
estado_actual	inicia
estado_siguiente	guardar
registro_aux_s[4:0]	11100
a_registro_aux_s[4:0]	00000
registro_salida[159:0]	0101101110010100001100001111001010010111011000101100011111010110110111011110011010011111101111100110010010000101110100001101111111010110010100011101010100100
a_registro_salida[159:0]	0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
sumas[4:0]	00111
a_sumas[4:0]	00000