UNIVERSIDAD CARLOS III DE MADRID

DEPARTAMENTO DE TECNOLOGÍA ELECTRÓNICA

INGENIERÍA TÉCNICA INDUSTRIAL: ELECTRÓNICA INDUSTRIAL

PROYECTO FIN DE CARRERA

"IMPLEMENTACIÓN HARDWARE DE FUNCIÓN RESUMEN PARA TECNOLOGÍA RFID"

AUTOR: JOSE LUIS CHAVARRIAS LÓPEZ
TUTOR: HONORIO MARTÍN GONZÁLEZ

OCTUBRE 2012

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

CAPÍTULO 1
INTRODUCCIÓN

1.1 ANTECEDENTES

La tecnología de identificación por radiofrecuencia RFID, (Radio Frequency Identification) se ha convertido en una de las tecnologías de comunicación con más crecimiento, y es actualmente, cuando su desarrollo está cobrando mayor protagonismo.

RFID no es una tecnología nueva, sino que lleva existiendo desde 1940. Durante la Segunda Guerra Mundial, los militares estadounidenses utilizaban un sistema de identificación por radiofrecuencia para el reconocimiento e identificación a distancia de los aviones.

Las posibilidades que ofrece la lectura a distancia de la información contenida en una etiqueta, sin necesidad de contacto físico, junto con la capacidad para realizar múltiples lecturas (y en su caso, escrituras) simultáneamente, provoca que su aplicación esté presente, en una gran variedad de ámbitos.

Sin embargo esto puede suponer un problema para las empresas, que ven aumentar la vulnerabilidad de sus sistemas de información. Esto se debe a la relativa facilidad con la que terceras personas pueden tener acceso, leer o modificar los datos, poniendo en peligro la integridad del sistema. Con el fin de solucionar estos problemas se emplean protocolos de autenticación, que permiten realizar una identificación segura de los agentes implicados en la comunicación RFID (*tag* y lector).

1.2 OBJETIVOS DEL PROYECTO

El objetivo de este proyecto es la implementación de una función resumen para etiquetas RFID, de bajo coste. Dado los límites recursos disponibles en este tipo de etiquetas, hay que seleccionar cuidadosamente el algoritmo que garantiza la seguridad del sistema. En este proyecto se ha decidido implementar la función Hash Tav-128, ya que está orientada hacia su uso en sistemas RFID de bajo coste.

El lenguaje que se va a emplear para la implementación del protocolo es VHDL, que nos permite el diseño de circuitos digitales. Y por otro lado, para la implementación

de los circuitos desarrollados será la herramienta Synopsys, la que nos permitirá obtener una estimación real del área y consumo de potencia, en la implementación.

Los objetivos que se establecen para el proyecto son:

1) Implementar la función Hash Tav-128, de forma genérica para realizar los correspondientes estudios a la generación del número de bits.
2) Buscar diferentes arquitecturas para la función Hash tratando de mejorar alguno de los parámetros críticos de la tecnología (área y throughput).
3) Síntesis y obtención de los principales parámetros como área, potencia, puertas equivalentes y tiempo, para diferentes arquitecturas y número de bits.
4) Analizar y comparar los diferentes resultados obtenidos y extraer las conclusiones de la mejor opción, para nuestra tecnología.

1.3 ESTRUCTURA DEL CONTENIDO

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

En el segundo capítulo se desarrollan los aspectos más importantes de la tecnología RFID, como el origen y evolución, elementos del sistema, frecuencia de funcionamiento, seguridad, aplicaciones además de otros aspectos destacados.

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

En el capítulo cuatro se muestran las arquitecturas diseñadas para la implementación de la función hash Tav- 128. Además se incluyen los diagramas de flujo de la maquinas de estados que controlan la función hash, y las simulaciones realizadas en Modelsim.

El capítulo cinco contiene los análisis y resultados realizados para la comprobación de las distintas arquitecturas y se incluyen comparaciones entre los distintos resultados obtenidos.

En el sexto capítulo se muestran las conclusiones del proyecto, así como las posibles líneas futuras que pueden mejorar la función presentada.

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

CAPÍTULO 2
LA TECNOLOGÍA RFID

2.1 INTRODUCCIÓN A RFID

RFID (*Radio Frequency Identification* o *Identificación por radiofrecuencia*) es una tecnología que permite el almacenamiento y recuperación de datos sin necesidad de contacto visual. Un sistema RFID está compuesto por las etiquetas o *tags* RFID, en los que previamente se han grabado los datos identificativos del objeto al que están adheridos. Un lector físico se encarga de leer estos datos, transformar y transmitir dicha información a la aplicación informática específica que la gestiona.

2.2 ORIGEN Y EVOLUCIÓN

La tecnología RFID tiene sus orígenes desde comienzos de la década de 1920 y está relacionada con la guerra, concretamente con la II Guerra Mundial, donde utilizaban radares para detectar el acercamiento de aviones. El problema era que no había forma de identificar si los aviones pertenecían al enemigo o si eran pilotos del propio país que regresaban de una misión. El ejército alemán descubrió que si los pilotos balanceaban sus aviones al volver a la base cambiaría la señal de radio reflejada de vuelta, con este método se podía distinguir a los aviones alemanes de los aliados y se convirtió en el primer dispositivo de RFID pasivo.

Fue en la década de los 50 cuando diferentes trabajos como los creados por D.B. Harris “Sistema de Radio Transmisión con Respuesta Modulatoria Pasiva” fueron determinantes para que la tecnología RFID dejase de ser una idea y se convirtiese en una solución.

En la década de los 60 la actividad comercial comenzó a existir en este campo. El primer sistema que fue usado era el EAS *“Electronic Article Surveillance”* (Vigilancia Electrónica de Artículos) para detectar robos en grandes almacenes. El sistema constaba de dos lectores ubicados en la salida del establecimiento, de tal forma que el cliente tenía pasar entre ellos. Si el lector detectaba el paso de una etiqueta o tag no desactivaba, hacía saltar una alarma acústica.

En los 70 se produjeron notables avances, las primeras patentes para dispositivos RFID fueron solicitadas en Estados Unidos (Charles Walton, dispositivo para la apertura de puertas sin llave o Mario W. Cardullo, etiqueta activa con memoria re-escribible).

En la década de los 80 y primeros años de los 90, con la gran cantidad de estudios y desarrollos logrados, se realizaron varias aplicaciones de esta tecnología en sistemas de corto alcance para el control de animales, además de introducirse en sistemas que gestionaba el paso de los vehículos por los pasos de control.

En la actualidad el principal responsable del desarrollo e implantación de esta tecnología es Auto-ID Center, una sociedad constituida en 1999 por un centenar de empresas punteras, universidades y centros de investigación de todo el mundo. El AutoID Center, ahora conocido como AUTOID Labs, está conformado por 6 laboratorios localizados en universidades de prestigio de EE.UU, Reino Unido, Australia, Japón, China y Suiza

2.3 ELEMENTOS DEL SISTEMA RFID

Los sistemas RFID constan, de tres componentes básicos: etiqueta o *tags*, lector y subsistema de procesamiento de datos.

2.3.1 Etiqueta o *tag* RFID

La etiqueta RFID está compuesta por una antena, un transductor de radio y un microchip. La antena es la encargada de transmitir la información que identifica a la etiqueta. El transductor es el que convierte la información que transmite la antena, y el chip posee una memoria interna para almacenar el número de identificación y en algunos casos datos adicionales. La capacidad de esta memoria depende del modelo. En el caso de *tags* sin chip, la información que se puede almacenar es bastante limitada (hasta 24 bits).

Nota: Para más información sobre la tecnología RFID consultar referencias [1] a [8].

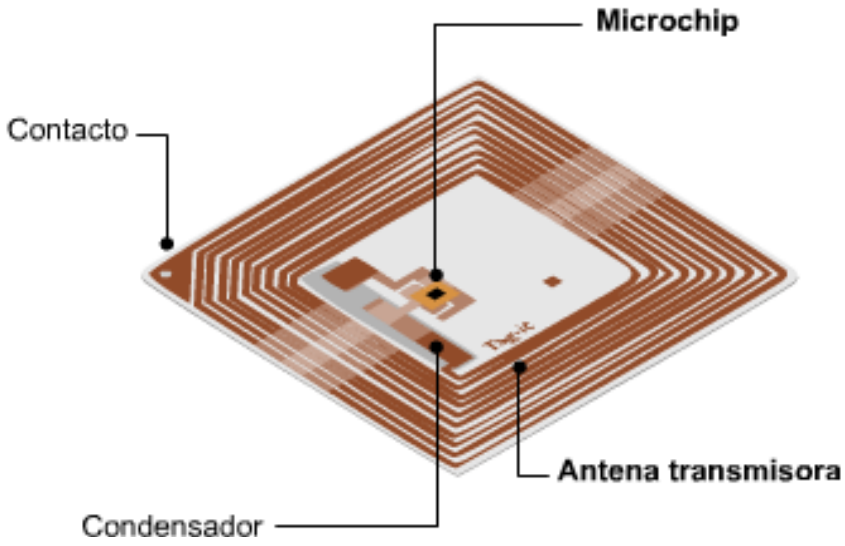

Figura 1: Tag RFID

No existe un único modelo de etiqueta RFID, sino que existen diferentes tipos dependiendo del mecanismo para almacenar datos, o de la comunicación que utiliza para transmitir la información que contiene. Permitiendo por ello la elección de la mejor etiqueta dependiendo de su aplicación.

Clasificación de etiquetas según su origen de la energía, batería interna o fuente de alimentación:

- Etiquetas pasivas: No requieren fuente de alimentación interna, ya que toda la energía que necesita la recoge del campo electromagnético creado por el lector, suficiente para activar el circuito integrado y de este modo generar y transmitir una respuesta.

  Las etiquetas pasivas suelen tener un alcance de unos 10 centímetros y pueden llegar hasta unos pocos metros, que dependerá de la frecuencia de funcionamiento, el diseño y el tamaño de la antena. Son las etiquetas más económicas del mercado, sin embargo tienen el inconveniente en su rango de comunicación.
- Etiquetas activas: Disponen de una batería para el suministro de energía, la cual utilizan para dar corriente a sus circuitos integrados y para propagar la señal al lector, es por ello permiten una mayor cobertura de difusión. Normalmente tienen una mayor capacidad de almacenamiento de información, más que un código de identificación, debido a que disponen de

más energía que permite enviar más información. Gracias a la fuente de energía interna, son capaces de transmitir señales más potentes que las de las etiquetas pasivas, convirtiéndoles por ello en un sistema mucho más eficiente cuando el entorno para la radiofrecuencia es más dificultoso. Son mucho más fiables y tienen menos errores que las pasivas.

Estas etiquetas RFID son las más caras del mercado, tienen un mayor tamaño y por lo general su vida útil es mucho más corta que las pasivas, dependiendo de la vida de su fuente de alimentación.

- Etiquetas semi-activas o semi-pasivas: Estas etiquetas utilizan una batería para activar sus circuitos integrados, como las etiquetas RFID activas, pero la energía para generar la comunicación con el lector es la que recoge de las ondas de radio de éste, como las etiquetas RFID pasivas. Debido a que utiliza una batería, estas son más grandes y más caras que las etiquetas pasivas y más baratas y más pequeñas que las activas, sin embargo consiguen mejor rango de comunicación en comparación con las pasivas.

Clasificación de etiquetas según su tipo de memoria, atendiendo si en su memoria es sólo de lectura, o además se puede escribir en ellas:

- Sólo lectura: Cuando esta se fabrica y solo entonces, se escribe en su memoria un código de identificación para el producto, el cual no se puede modificar en el tiempo de vida de la etiqueta. Suelen tener una capacidad de almacenamiento muy baja, porque es suficiente con unos pocos bits poder almacenar su código de identificación.
- Lectura y escritura: La información puede ser almacena en el proceso de fabricación del chip, pero se puede modificar por el lector, pudiendo almacenar más información en caso necesario. Este tipo de etiquetas suelen tener más capacidad de almacenamiento ya que se suele incluir más información sobre el producto.

2.3.2 El lector o transceptor

Los componentes de un lector son el módulo de radio, el procesador y sus conexiones que pueden ser de varios tipos para conectar con distintos dispositivos. El componente principal de los lectores es la antena.

Los lectores pueden ser fijos o móviles, de lo que dependerá tanto el tipo de comunicación con el subsistema como el tipo de antena necesaria.

- Antenas móviles: Son antenas que se pueden mover para identificar las etiquetas. Normalmente estas antenas se encuentran en los lectores móviles con antenas integradas o son utilizadas manualmente por un operario.
- Antenas fijas: Están conectadas a los lectores mediante cables. Un único lector puede gestionar varias antenas mediante la creación de una zona de interrogación, como por ejemplo Dock Door (2 entradas) para puertas lectoras o de arco (3 entradas) para cintas transportadoras.

Mediante la antena que incluye el lector se permite el envío de información codificada a través de ondas de radiofrecuencia. El circuito receptor que existe en la etiqueta es capaz de detectar el campo modulado, generado por la antena del lector, y posteriormente se decodifica la información, usando la propia antena de la etiqueta o *tag* se envía una señal al lector como modo de respuesta.

La fabricación y el diseño de los lectores RFID varían mucho entre los distintos modelos aunque estos soporten los mismos protocolos. Una de las grandes diferencias existentes entre ellos se encuentra en los rendimiento de lectura, pero también influye el tipo y número de conexiones con etiquetas que son capaces de soportar simultáneamente.

Figura 2: Lector RFID

Existen varias consideraciones para elegir lector más apropiado para cada aplicación, dependiendo de: la frecuencia operativa, multi-protocolo en el caso de diferencia en los protocolos, codificación para escribir, memoria, potencia, etc.

2.3.3 Subsistema de procesamiento de datos, o middleware

El middleware consiste en un sistema de gestión que reside entre la infraestructura de la tecnología RFID y las aplicaciones empresariales. Una de sus funciones es la gestión de los lectores, el filtrado de los datos y el control de la infraestructura.

El middleware tiene incorporadas muchas funcionalidades además de que cada uno de los desarrolladores de software le añade características que hacen que cada uno de ellos tenga su propia identidad. También se pueden desarrollar con funcionalidades adaptadas a una aplicación en concreto gracias al desarrollo de software a medida.

2.4 COMUNICACIÓN ENTRE LOS SISTEMAS RFID

El modo de funcionamiento de los sistemas RFID es simple [5]. La etiqueta RFID, que contiene los datos de identificación del objeto al que se encuentra adherido, genera una señal de radiofrecuencia con dichos datos. Esta señal puede ser captada por un lector RFID, el cual se encarga de leer la información y pasarla en formato digital a la aplicación específica que utiliza RFID.

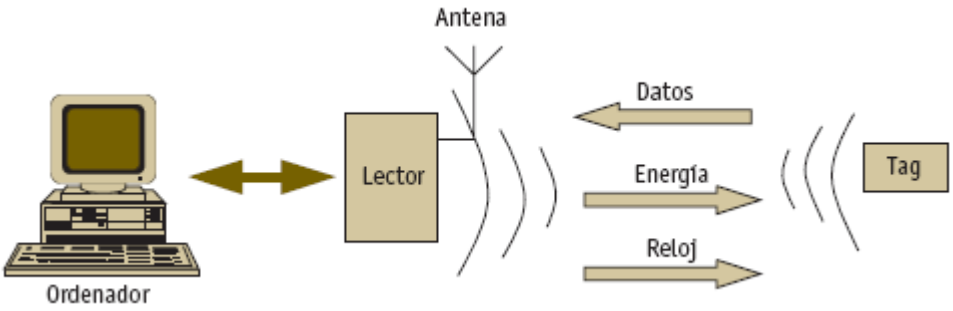

Figura 3: Esquema de funcionamiento de un sistema RFID.

2.4.2 Modos de comunicación

Tiene que ver con la capacidad de transmisión de la red, es decir, si es posible la comunicación bidireccional o no. En la figura 4 se puede ver los diferentes esquemas de comunicación.

- Full-Duplex (FDX): La etiqueta RFID y el lector pueden comunicarse simultáneamente.
- HALF-Duplex (HDX): El lector y la etiqueta no pueden comunicarse simultáneamente sino que cada uno transmitirá en su turno.
- Secuencial (SEQ): La etiqueta se alimenta de forma de forma intermitente. La transferencia entre etiqueta y lector se produce en esos intervalos en los que el lector no se comunica con la etiqueta.

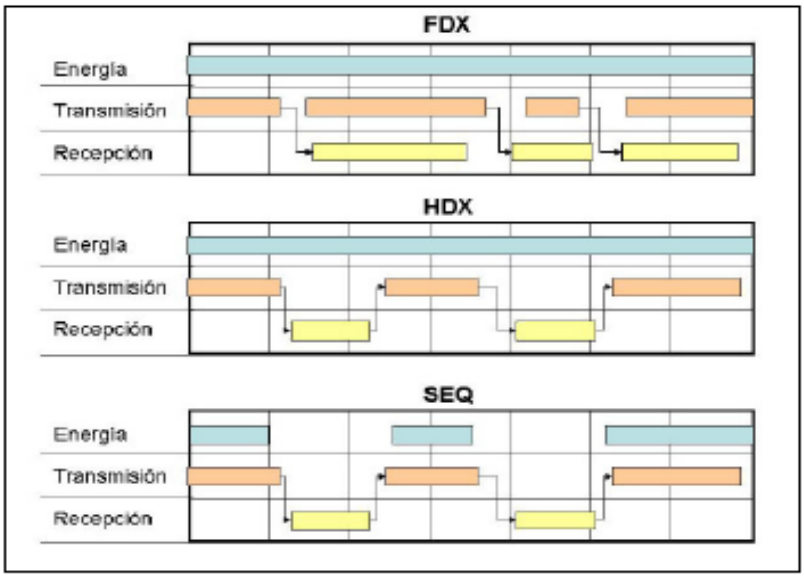

Figura 4: Representación de los diferentes esquemas de comunicación

2.4.3 Métodos de propagación de energía

El tipo de acoplamiento afecta directamente al rango de lectura entre las etiquetas RFID y los lectores. El tipo de acoplamiento afecta directamente al rango de lectura entre las etiquetas RFID y los lectores. Los tipos de acoplamientos son [10]:

- Acoplamiento electromagnético: Este tipo de acoplamientos es el utilizado en los sistemas RFID de frecuencia ultra alta (UHF). Los *tags* reflectan la señal con la misma frecuencia emitida por el lector pero cambiando la información contenida en ella. El acoplamiento consiste en reflejar la señal para enviarla al origen. Como el lector y el *tag* usan la misma frecuencia para comunicarse, utilizan turnos.

- Acoplamiento inductivo: Es el más común, el lector proporciona energía por acoplamiento inductivo a los *tags* mediante antenas en forma de bobina para generar campo magnético.
- Acoplamiento magnético: Es similar al acoplamiento inductivo. El *tag* y el lector forman un par de transformadores mediante bobinas. La antena del lector es una bobina enrollada en una pieza de ferrita con los dos extremos al aire. El sistema está diseñado para unos rangos de lectura entre 0,1 cm y 1cm

2.4.4 Modulación y codificación

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

Tipos de codificación

Un sistema codificador de señal toma el mensaje a transmitir y su representación en forma de señal y la adecua óptimamente a las características del canal de transmisión. Este proceso implica proveer al mensaje con un grado de protección contra interferencias o colisiones y contra modificaciones intencionadas de ciertas características de la señal. Existen diferentes tipos de codificación para un sistema RFID. En la Figura 5 puede observarse una representación de las siguientes codificaciones:

- Código NRZ (No Return to Zero): un ‘1’ binario es representado por una señal ‘alta’ y un ‘0’ binario es representado por una señal ‘baja’. La codificación NRZ se usa, exclusivamente con una modulación FSK o PSK.
- Código Manchester: Un ‘1’ binario es representado por una transición negativa en la mitad del período de bit y un ‘0’ binario es representado por una transición positiva. Basado en una modulación con sub-portadora.
- Código Unipolar RZ: Un ‘1’ binario es representado por una señal ‘alta’ durante la primera mitad del periodo de bit, mientras que un ‘0’ binario es representado por una señal ‘baja’ que dura todo el periodo de bit.

- Código DBP: Un ‘0’ binario es codificado por una transición, de cualquier tipo, en mitad del período de bit. Un ‘1’ es codificado con una ausencia de transición. Además, el nivel de señal es invertido a inicio de cada periodo de bit, de modo que el pulso pueda ser más sencillamente reconstruido en el receptor si es necesario.
- Código Miller: Un ‘1’ es representado por una transición de cualquier tipo en la mitad del período de bit, mientras que el ‘0’ binario es representado con la continuidad del nivel de la señal hasta el próximo periodo de bit. Una secuencia de ceros crea una transición al principio de cada periodo de bit, de modo que el pulso pueda ser más sencillamente reconstruido en el receptor si es necesario.
- Código Miller Modificado: En esta variante del código Miller, cada transición es reemplazada por un pulso ‘negativo’. El código Miller Modificado es altamente recomendable para transmitir del lector al tag en sistemas RFID que usan acoplamiento inductivo. Debido a la tan corta duración del pulso (tpulso << Tbit) es posible asegurar una continua alimentación del transponder debido al campo magnético del lector mientras dura la transferencia de información.
- Código diferencial: En la codificación Diferencial cada ‘1’ binario que se tiene que transmitir causa un cambio en el nivel de la señal, así como para un ‘0’ el nivel permanece invariante. El código diferencial puede ser generado muy simplemente a partir de una señal NRZ usando una compuerta XOR.

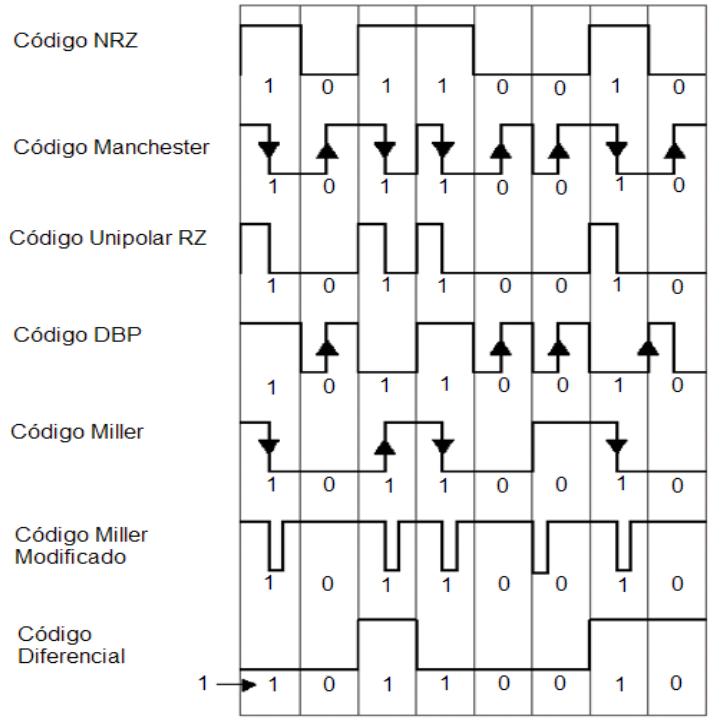

Figura 5: Representación gráfica de las distintas codificaciones.

Tipos de modulaciones

Este proceso consiste en modificar alguna de las variables de dicha señal con el fin de adecuarla a las características tanto del canal de comunicación como de los agentes que intervienen en la misma. Hay tres tipos de codificación en la interfaz aérea, que son: ASK, FSK y PSK. En la Figura 6 puede observarse una representación de las siguientes modulaciones:

- Modulación por amplitud (ASK): Se basa en la modulación de la amplitud y envía los datos digitales sobre una portadora analógica cambiando la amplitud de la onda en el tiempo. Los dos valores binarios se representan con dos amplitudes diferentes; es decir uno de los dígitos binarios se representa mediante la presencia de la portadora a amplitud constante, y el otro dígito se representa mediante la ausencia de la señal portadora.
- Modulación por frecuencia (FSK): Modula basándose con la frecuencia y envía los cambios de señal a través de la modificación de la frecuencia. Los dos valores binarios se representan con dos frecuencias diferentes (f1 y f2) próximas a la frecuencia de la señal portadora fp. Esta modulación aumenta la protección contra el ruido y las interferencias, obteniendo un

comportamiento más eficiente respecto a ASK. La desventaja es que es necesario un mayor ancho de banda

- Modulación por fase (PSK): Consiste en modular la fase y envía los datos mediante cambios en la fase de la señal. Existen dos alternativas de modulación: PSK convencional, donde se tienen en cuenta los desplazamientos de fase y PSK diferencial, en la cual se consideran las transiciones de esta fase.

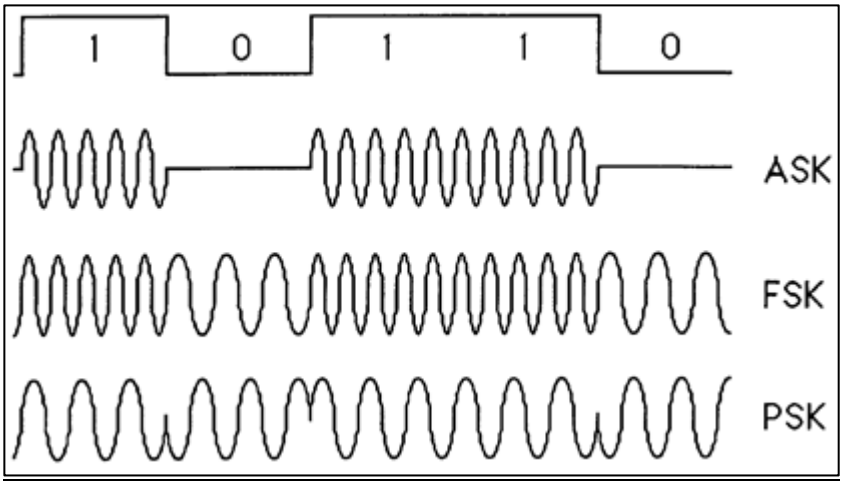

Figura 6: Representación de los principales tipos de modulación.

2.4.5 Rangos de frecuencia

Dependiendo de la frecuencia de operación, las etiquetas se pueden clasificar en baja, alta, ultra alta frecuencia y microondas. La frecuencia de operación determinará aspectos de la etiqueta como la capacidad de los datos, la velocidad y tiempo de lectura de éstos, el radio de cobertura, el coste de la etiqueta y el área de aplicación de ésta.

- Baja frecuencia (LF): Comprende las frecuencias emitidas en un rango de 125Khz y 134Khz.
    - Baja velocidad de lectura.
    - Identificación unitaria (no existe multi-lectura).
    - Cortas distancias de lectura (hasta 45cm).
- Alta frecuencia (HF): Comunicación a 13,56Mhz. Se utiliza normalmente en tarjetas inteligentes.
    - Velocidad media de lectura.

  - Tamaño del *tag* pequeño.
  - Distancia de lectura inferior a 2m.
- Ultra alta frecuencia (UHF): Es la banda comprendida entre 860Mhz y 960Mhz. Se usa cuando debemos leer varios *tags* a alta velocidad.
  - Velocidad de alta lectura.
  - Tamaño del *tag* mediano.
  - Multi-lectura.
  - Grandes distancias de lectura.
- Microondas: (~2.45GHz), ofrecen grandes alcances (por encima de los 30 metros)

En la figura 7 [5] se realiza una comparativa de las características de las etiquetas, dependiendo del intervalo de frecuencia de trabajo.

Parámetros	*Baja frecuencia (<135 KHz)*	*Alta frecuencia (13,56 MHz)*	*Ultra alta Frecuencia (433 MHz, 860 MHz, 928 MHz)*	*Frecuencia microondas (2,45 GHz, 5,8 GHz)*
Cobertura	Menor	←	→	Mayor
Tamaño de la etiqueta	Mayor	←	→	Menor
Velocidad de lectura de datos	Menor	←	→	Mayor
Lectura en presencia de líquidos o metales	Mejor	←	→	Peor
Lectura en presencia de interferencias EM	Peor	←	→	Mejor

Figura 7: Representación comparativa de las etiqueta según frecuencia de trabajo.

2.5 VENTAJAS DE LA TECNOLOGÍA RFID

La tecnología RFID es muy versátil y puede ser aplicada en diferentes sectores y diferentes procesos de negocio. Para poder obtener el máximo rendimiento de las soluciones basadas en RFID, es necesario analizar cada proyecto de forma personalizada, identificando momentos y aspectos del flujo de trabajo en los que se puede alcanzar mayor beneficio.

A continuación analizamos las ventajas e inconvenientes de la tecnología RFID, nombrando y representando algunas de sus características respecto de otras tecnologías. En tabla 1[3] se muestra una comparativa con las distintas alternativas de identificación

de objetos, y en la tabla 2[3] se muestra una comparativa con otras tecnologías de comunicación inalámbricas.

	Código de Barras	Memorias de Contacto	RFID Pasivo	RFID Activo
Modificación de datos	No modificable	Modificable	Modificable	Modificable
Seguridad de datos	Seguridad mínima	Altamente seguro	Rango de baja a alta seguridad	Alta seguridad
Almacenamiento	Lineal: 8-30 caracteres 2D: 7200 dígitos	De 8Mb en adelante	Alrededor de 64Kbytes	Alrededor de 8Mbytes
Coste	Bajo	Alto	Medio	Muy alto
Estándares	Estable e implantado	Propietario, sin estándar	Estándares en fase de implantación	Propietario y estándares abiertos
Tiempo de vida	Bajo por deterioro	Largo	Indefinido	3-5 años de vida de batería
Distancia de lectura	Pocos centímetros	Contacto necesario	Del orden de 1m	Del orden de 100m
Interfaz	Lectura óptica directa	Contacto	Sin barreras aunque con interferencias	Sin barreras aunque con interferencias

Tabla 1: Comparativa de tecnologías de identificación de objeto.

Tecnología	Bluetooth	WIFI	RFID
Transmisión	Voz y datos	Voz y datos	Datos (código EPC)
Equipos	Hasta 8 equipos (Piconets)	Equipos con configuración compatible	Miles de etiquetas RFID con 1 o varios lectores
Comunicación	Síncronos, Bidireccional y asíncronos	Varios estándares IEEE 802.11	Síncrono y Bidireccional Varios estándares
Velocidad	Síncronos: 432 Kbps y Bidireccional: 721 kbps en un sentido y 57.6 en el otro	Según el estándar	
Encriptación		WEP, WPA	Código EPC
Utilización	Señales de radiofrecuencia	Señales de radiofrecuencia	Señales de radiofrecuencia
Distancia	10 m a 100 m	30 m	Pasivas: 10 mm a 6m Activas: varios km
Red	Inalámbrica	Inalámbrica conocida como 802.11	Transmisión por campos electromagnéticos e identificación

Tabla 2: Comparativa entre tecnologías.

2.5.1 Ventajas en comparación con su antecesor, el código de barras

- A diferencia del código de barras, las etiquetas no necesitan contacto visual con el módulo lector para que éste pueda leerlas. Estas pueden ser detectadas desde distancias de más de 15 metros
- Mientras el código de barras identifica un tipo de producto, las etiquetas electrónicas identifican el producto a nivel individual.
- La tecnología RFID permite leer múltiples etiquetas electrónicas simultáneamente. Los códigos de barras, por lo contrario, tienen que ser leídos secuencialmente.
- Las etiquetas electrónicas pueden almacenar mucha más información sobre un producto que el código de barras, que solamente puede contener un código y, en algunos casos, un precio o cantidad.
- Mientras que sobre el código de barras se puede escribir solo una vez, sobre las etiquetas electrónicas puede re-escritas se puede actualizar, modificar o borrar información. Esto permite su reutilización posterior con otro código identificativo, o incorporarlo posteriormente más información importante de la vida de un producto.
- La tecnología RFID evita falsificaciones. Con una simple fotocopia se puede reproducir un código de barras. Las etiquetas electrónicas, en cambio, no se pueden copiar.
- Un código de barras se estropea o se rompe fácilmente, mientras que una etiqueta electrónica es más resistente porque, normalmente, forma parte del producto o se coloca bajo una superficie protectora y soporta mejor la humedad y la temperatura.

2.5.2 Ventajas en el ámbito empresarial

- Lectura más rápida y más precisa
- Niveles más bajos en el inventario
- Reducción de las roturas de stock
- Disminución de las perdidas desconocidas
- Mejor utilización de los activos

- Lucha contra la falsificación
- Retirada del mercado de productos concretos

2.6 INCONVENIENTES DE LA TECNOLOGÍA RFID

Las ventajas que ofrece la tecnología RFID son muchas, sin embargo, existen demasiadas desventajas como para causar un frenado en el crecimiento masivo de esta tecnología.

A continuación se enumeran alguna de las desventajas más importantes de la tecnología de identificación por radiofrecuencia:

- Interceptación fraudulenta de datos y su posterior uso. Privacidad y falsificación.
- Interferencias en las señales de radiofrecuencia que puedan generar algunos elementos.
- Implantación de sistemas para gestionar la información acumulada
- El coste de la tecnología RFID activa provoca la no utilización en productos de bajo valor.

2.7 APLICACIONES DE RFID

Las aplicaciones de esta tecnología son amplias y variadas, se ha aplicado para mejorar procesos o para cubrir distintas necesidades existentes. A continuación se enumeran algunos de los ámbitos donde se aplica la tecnología RFID:

- Control de distribución de fármacos e identificación de pacientes en hospitales: Control de medicamentos en pacientes, pretendiendo evitar los riesgos de errores. También se aplica para la gestión hospitalaria, realizando una identificación y seguimiento del paciente por las diferentes áreas del centro.
- Logística: Actualmente es la aplicación a la que mayor valor añadido aportan los sistemas RFID. La tecnología de identificación por radiofrecuencia se puede emplear para la gestión del inventario de modo que se gestionen los activos y se realice la trazabilidad de éstos. De esta forma será posible la localización de un activo en el inventario de manera fácil al instante o trazar

los movimientos que sigue un producto a lo largo de su vida. Gracias a la tecnología RFID la realización de inventarios es mucho más ágil, y no es susceptible de errores, a diferencia con el sistema que predomina actualmente para la identificación, el código de barras.

Además ayuda a los fabricantes y distribuciones (ver figura 10) a obtener una visibilidad mejorada de suministros mediante el incremento de la cantidad de puntos de captura de datos.

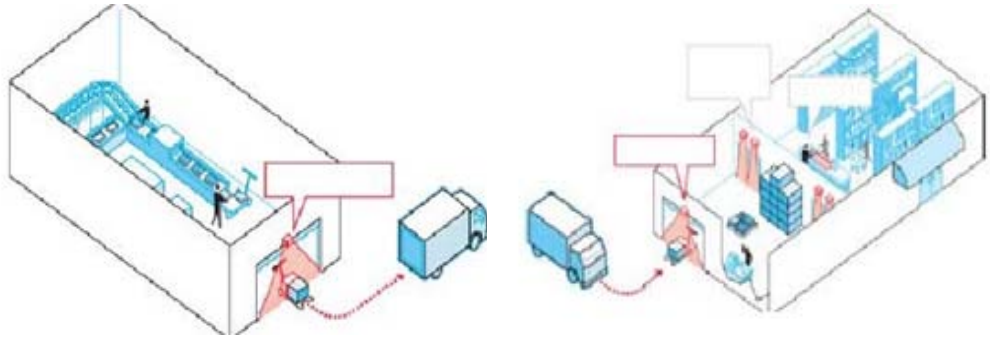

Figura 8: Representación de entrada/ salida de mercancías.

- Identificación de animales: Es un sistema de identificación de animales basado en el empleo de un microchip inyectable que se implanta bajo la piel de la mascota, y el cual contiene la información necesaria para su identificación.
- Control de vehículos (telepeajes): Utilización de la tecnología para el cobro automático en redes de autopistas de peajes, de forma que se puede abandonar el peaje sin necesidad de detener el vehículo.

Figura 9: Cobro automático de peaje

Consiste en un dispositivo electrónico que se debe colocar en el parabrisas del vehículo, el cual consiste en una etiqueta RFID, denominado OBE. AL aproximarse a un peaje, se debe colocar en la vía indicada para el cobro automático y pasar a una velocidad máxima de 40km/h.

2.8 ESTANDARES

La estandarización de esta tecnología posibilita que exista interoperabilidad entre aplicaciones y ayuda a que los diferentes productos no interfieran, independientemente del fabricante.

La estandarización RFID comenzó por la competencia de dos grupos competidores: *ISO* y *EPC Global*. Los estándares RFID tratan los siguientes temas:

- Protocolo de interfaz: Forma en la que las etiquetas y los lectores se pueden comunicar.
- Contenido de datos: Organización de los datos que se intercambian.
- Conformidad: Pruebas de los productos que deben cumplir para reunir los requisitos del estándar.
- Aplicaciones: Como se puede utilizar las aplicaciones con RFID.

2.8.1 Estándares ISO

La International Organization for Standardization (ISO) es una organización internacional no gubernamental integrada por una red de institutos nacionales, cuya aportación es igualitaria. Su función principal es buscar la estandarización de normas de productos y seguridad nivel internacional. En la tabla 3 y 4 se muestran las normas ISO relativas a RFID, y la ISO englobada bajo la serie 18000, respectivamente

ISO/IEC 11784-11785, ISO 10536, ISO 18000	Sobre privacidad y seguridad de datos.
ISO 14223/1	Identificación por radiofrecuencia de animales, transpondedores avanzados e interfaz radio.
ISO 14443	Orientados a los sistemas de pago electrónico y documentación personal. Estandar HF.
ISO 15693	Se utiliza en tarjetas sin contacto de credito y devito. Estandar HF.
ISO 18000-7	Para todos los productos en RFID activa. Estandar industrial para UHF.
ISO 18185	Estandar industrial para el seguimiento de contenedores a frecuencias de 433MHz y 2,4GHz.
ISO/IEC 15961	Se encaraga del protocolo de datos e interfaz de aplicación.
ISO/IEC 15962	Protocolo de codificación de datos y funcionalidades de la memoria de las etiquetas.
ISO/IEC 15963	Sobre el sistema de trazado y monitorización que afecta a la etiqueta RFID.
ISO 19762-3	Establece los terminos y definiciones únicas de identifficación por radiofrecuencia (RFID) en el campo de la identificación automática y captura de datos técnicos.
ISO 23389	Estandar para los contenedores (normas de lectura/ escritura).
ISO 24710	Técnicas AIDC para gestión de objetos con interfaz ISO 18000.

Tabla 3: Descripción de normas ISO relativas a RFID.

ISO 1800-2	Parámetros para comunicaciones por debajos de 135MHz.
ISO 1800-3	Parámetros para comunicaciones a 13,56MHz.
ISO 1800-4	Parámetros para comunicaciones a 2,45GHz.
ISO 1800-5	Parámetros para comunicaciones a 5,8GHz.
ISO 1800-6	Parámetros para comunicaciones desde 860 a 960MHz.
ISO 1800-7	Parámetros para comunicaciones a 433MHz.

Tabla 4: Serie ISO 18000, frecuencias empleadas

2.8.2 Estándar EPCGlobal

EPCglobal es una organización sin ánimo de lucro que se creó, apoyada por la industria, para establecer y mantener la red EPCglobal ("EPC Global Network") como el estándar mundial para la identificación automática de objetos en la cadena de suministros. La investigación de la organización se ha centrado en el desarrollo de estos cinco elementos esenciales para la red EPC global [2]:

- Código de Producto (EPC): Código numérico estandarizado de 96 bits que identifica de forma unívoca un objeto. La figura 10 [3] se muestra el formato lógico de los datos contenidos en una etiqueta EPC.

Figura 10: Formato de EPC.

- El protocolo de comunicación: EPCglobal clasifica los *tags* teniendo en cuenta los diferentes niveles de sofisticación y capacidad, ver figura 11 [4]

EPC Class	Definición	Programación
Class 0	Tags pasivos sólo de lectura	Programado como parte del semiconductor en el proceso de fabricación
Class 0+	Una sóla escritura y múltiples lecturas Versión de EPC Class 0	Programado una vez y posteriormente bloqueado
Class 1	Tags pasivos de una sóla escritura y múltiples lecturas	
Class 1 - Gen2	Múltiple lectura y escritura, incrementa la velocidad en la transmisión de datos, interoperabilidad global	Puede ser reprogramado varias veces
Class 2	Tags reescribibles	
Class 3	Tags semi-pasivos	
Class 4	Tags activos	
Class 5	Lectores	N/A

Figura 11: *Tipos de etiquetas definidos por EPCglobal.*

- Milddleware EPC: Recogida y tratamiento de datos ("*Savant*"), incluyendo la comunicación con los equipos lectores/escritures. El middleware debe filtrar y consolidar adecuadamente los datos antes de enviarlos a los sistemas corporativos.
- Servicio de información EPC (EPCIS): Define el formato de la información de intercambio entre componentes dentro de la red EPC.

- La especificación ONS: Su función dentro de la red EPCglobal es identificar la localización del servidor que contiene la información que necesita un aplicación.

La estándar EPC Clase 1 Generación 2 (C1G2) se ha publicado como estándar ISO 18000-6C reconocido internacionalmente para gestión y control de cadenas de suministros, por tanto es el punto de unión entre los estándares ISO y EPC.

La EPCglobal Network es un medio para usar la tecnología RFID en la cadena global de suministros y compartir la información obtenida entre usuarios autorizados a través de internet.

La EPCGlobal Network tiene varios componentes los cuales, en coordinación, proporcionan la habilidad de capturar y compartir información dentro de la red. Los objetos se etiquetan con *tags* de bajo coste que contienen su código de identificación (EPC). En cada agente de la red, se coloca un sistema de identificación ("ID System") formado por lectores y antenas, que recogen los códigos EPC de todos los objetos. Estos datos son gestionados por el Middleware EPC, el cual los entrega a los sistemas de información EPC. Para obtener información adicional sobre un EPC, el agente autorizado, a través de sus Sistemas de Información EPC ("EPCIS"), acudirá a los servicios de información centralizados, entre los que destaca el servicio de nombres de objetos ("Objets Naming Service").

2.8.3 Estándar de comunicación EPCglobal Class1 Gen2

El estándar EPCglobal Class1 Gen2 está implementado con un mecanismo similar al del protocolo Aloha Ranurado. Se centra en el estándar para sistemas RFID pasivos Class-1 Gen-2, en la banda UHF (660MHz- 930MHz). Incluye un conjunto de especificaciones hardware para tags pasivos y el hardware y software del sistema lector en el cual reside toda la complejidad del sistema [3] [12]. Las razones que han llevado a los autores a elegir de este protocolo son [12]:

- La complejidad del protocolo reside en el *reader*.
- Los *tags* que se comercializan hoy en día cumplen los requisitos mínimos para su implementación. No supone un coste adicional hardware.

- Está basado en Aloha Ranurado. Ideal para sistemas RFID donde el *reader* no tiene un conocimiento a priori de los *tags* que hay en cobertura.
- El *reader* puede adaptar el tamaño de ciclo en cada ciclo mediante un algoritmo sencillo, obteniendo mejores resultados en tiempo medio de identificación, utilización del canal y probabilidad de colisión.

El proceso de identificación que define el estándar EPCglobal Class-1 Gen-2 para un sistema RFID pasivo se inicia en el instante en que el lector monitoriza el entorno, con el fin de detectar los *tags* en su rango de cobertura. Los *tags* que se activan por la señal electromagnética del paquete *Broadcast* [3], responde provocando una colisión múltiple. Esta colisión la detecta el lector y es entonces cuando comienza un ciclo de identificación. En cada proceso de identificación el tiempo se distribuye en ciclos y a su vez estos en "ranuras temporales", llamadas *slots*. La identificación comienza cuando el lector envía un paquete tipo *Query*, incluyendo en uno de sus campos cuatro bits, que indican el valor de *Q*, este indica el tamaño del ciclo que será 2^Q *slots*. Los *tags* que deben identificarse generan un numero aleatorio comprendido o dentro del intervalo $[0, 2^Q-1]$. El valor de *r* representa el *slot* en el cual el *tag* trasmitirá su identificador *ID*.
En cada ciclo, el comienzo de un nuevo *slot* lo indica el lector por medio del paquete *QueryRep*, menos el *slot* 0, que se indica automáticamente tras el envió del paquete *Query*. Los *tags* que deben identificarse utilizan un contador (*counter=r*) de tal modo que contabilizan los *slots* que les quedan para enviar su identificador. El contador se decrementa cada vez que reciben un paquete *QueryRep*, si este alcanza el valor *0*, envían su *ID*, que corresponde al slot elegido en el ciclo. Pueden ocurrir 2 cosas después de que un *tag* envié su identificador:

(1) Más de un *tag* ha elegido el mismo slot para transmitir su identificador. El lector detecta esta colisión y envía un nuevo paquete *QueryRep*, provocando que los *tags* actualicen su $counter=2^Q-1$, ver figura (12)[11].

(2) La transmisión del *ID* ha sido correcta. El lector envía un paquete respuesta *ASK*. Todos los *tags* lo reciben pero solo el que ha enviado su *ID* es el que debe contesta enviando su paquete *DATA*, como pudiera ser su paquete *EPC*. Tras esta transmisión si pueden ocurrir 2 cosas:

(2.1) El lector recibe correctamente el paquete *EPC*. Envía un nuevo *QueryRep* comenzando de este modo un nuevo *slot*, ver figura (12)[11].

(2.2) El lector no recibe el paquete *EPC*. Envía un paquete *Nack*, todos los *tags* lo reciben pero solo el *tag* que envió el paquete *DATA* será el que establezca su contador *counter* = 2^{Q}*-1*. Tras el paquete *Nack* el lector vuelve a manda un paquete *QueryRep*, indicando el inicio de un nuevo *slot*, ver figura 12[11].

Cuando un ciclo finaliza el lector vuelve a enviar un paquete *Query*, para que aquellos *tags* que no hayan logrado identificarse vuelvan a competir de nuevo.

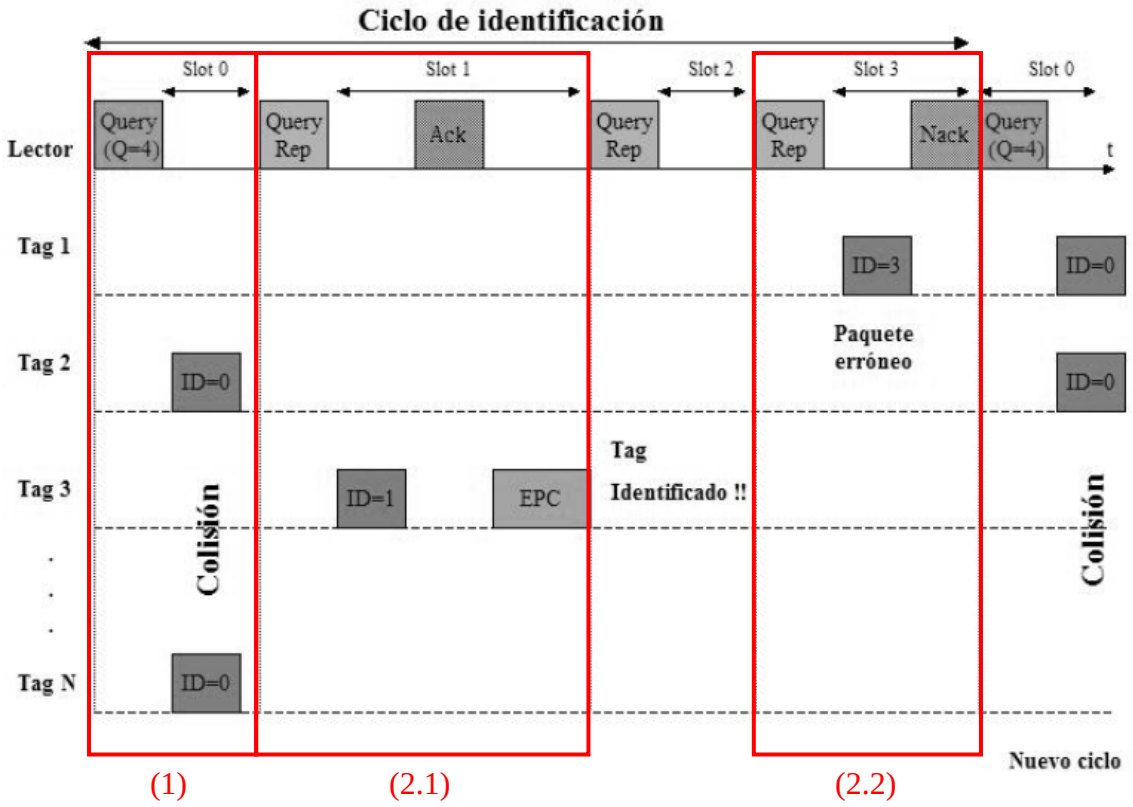

Figura 12: Algoritmo anticolisión EPC global Clss-1 Gen2

2.9 RIESGOS EN LA SEGURIDAD RFID

Los riesgos para la seguridad de la tecnología RFID son acciones encaminadas a deteriorar, interrumpir o aprovecharse del servicio. Los riesgos se concretan en ataques que pueden sufrir las instalaciones. Algunos de los ataques más habituales en un sistema RFID es evitar la comunicación entre el lector y la etiqueta, además de ataques en las comunicaciones por radiofrecuencia [13] [14].

- Aislamiento de etiquetas: Se puede conseguir introduciendo la etiqueta en una “Jaula de Faraday” o creando un campo electromagnético que interfiere con el creado por el lector.
- Suplantación: Envió de información falsa como pudría ser un código del producto (EPC).
- Inserción: Este ataque consiste en la inserción de comandos ejecutables en la memoria de datos de una etiqueta. Estos comandos pueden inhabilitar lectores y otros elementos del sistema. “Desactivación del sistema”.
- Repetición: Consiste en enviar al lector RFID una señal que reproduce la de una etiqueta válida. Permite suplantar la identidad que representa una etiqueta RFID.
- Riesgos de ataque mediante inyección de lenguaje de consultas SQL**:** Por medio de la comunicación entre la etiqueta y el lector, se pasa lenguaje SQL hacia el soporte físico que lee la etiqueta, el cual, debido a dicho ataque, ejecuta las órdenes incluidas en la etiqueta y esto puede ser introducido en una base de datos.
- Ataques Man in the Middle (MIN): Vulnera la confianza mutua en los procesos de comunicación y reemplaza una de las entidades. Ya que la tecnología RFID se basa en la interoperabilidad entre lectores y etiquetas es vulnerable a este tipo de ataques.

2.10 PROTOCOLOS ANTICOLISIÓN

Existen problemas en los procesos de comunicación de los sistemas RFID, cuando varios *tags* intenta identificarse en el mismo instante temporal, los mensajes de respuesta de estos pueden colisionar, impidiendo una correcta identificación. Es por ello que es necesario algún tipo de mecanismo anticolisión. Este mecanismo debe garantizar la correcta y rápida identificación de todos los *tags* que circulan por la zona de cobertura.

2.10.2 Algoritmo determinista de segmentación (Splitting)

El conjunto de *tags* a identificar se descompone en pequeños subgrupos mediante técnicas de *splitting* hasta que el número de tags por subgrupo sea de uno. Para llevarlo a cabo los *tags* seleccionan un número aleatorio, o bienel *Reader* envía un número de serie correspondiente a un único subgrupo de *tags* (ID). Estos algoritmos se conocen también como algoritmos de búsqueda en árbol o búsqueda binaria [15].

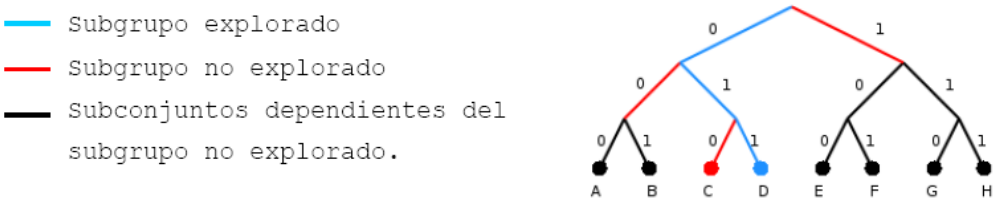

Figura 13: Algoritmo de búsqueda en árbol.

2.10.3 Algoritmo probabilísticos

Se basa en el protocolo *FSA*. En este caso, tras recibir una señal del *Reader*, los *tags* arbitrariamente seleccionan una ruptura de entre la longitud de la trama y envían su *ID* en la ranura elegida. Este mecanismo es muy simple, pero cuando el número de *tags* aumenta, es necesario algún mecanismo de adaptación de trama, para recibir el valor de la trama [16].

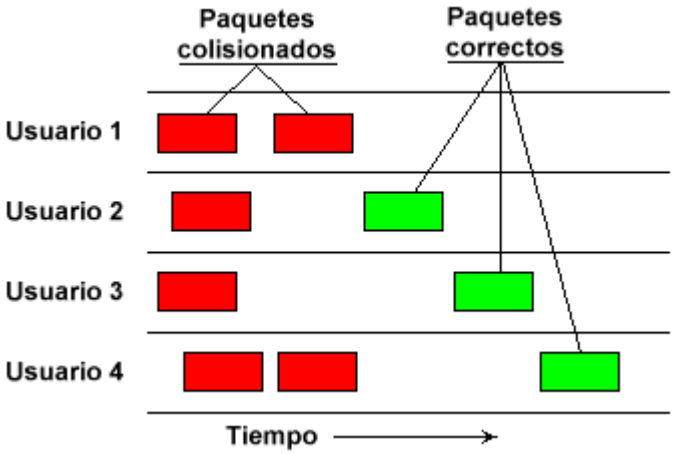

Figura 14: Flujo de paquetes en un canal ALOHA.

CAPÍTULO 3
METODOLOGÍA

En este capítulo se describe la forma de trabajo que se ha llevado, para la realización del proyecto, así como las herramientas necesarias para su ejecución, y los pasos a seguir hasta llegar a los resultados que queremos.

3.1 FLUJO DE TRABAJO

En primer lugar se ha realizado una búsqueda del algoritmo, que tuviera ciertas características y que nos pudiera servir para el alcanzar los objetivos de este proyecto.

Una vez seleccionado el algoritmo que se desarrollará en este trabajo, se lleva a cabo el diseño del mismo en un lenguaje de descripción hardware. Para conseguir un diseño optimo, se realizarán distintas arquitecturas, para posteriormente demostrar y argumentar cuál de estas será mejor para su implantación.

El programa que se utilizará para la comprobación y simulación de los diseños será Modelsim. Una vez comprobadas las simulaciones y por lo tanto conseguidos los objetivos, se sintetizará en el ISE los diseños generados, para asegurarse de que todas las arquitecturas, están libres de *warning o errores* que el Modelsim no detecta.

Posteriormente los diseños serán sintetizados empleando la herramienta de síntesis Synopsys, que nos permitirá obtener los recursos en área y potencia, además podemos conocer el número de puertas lógicas equivalentes que necesita nuestras distintas arquitecturas propuestas, y por lo tanto nos facilitará la elección de la mejor, según las necesidades y limitaciones que presentas los *tags* RFID. Incluimos el análisis de cada una de las arquitecturas para distintos bits de salida, comprobando de este modo las variaciones que se generan en consumo.

En la figura 15 se muestra el diagrama de flujo que se ha seguido para la realización del proyecto, y que acabamos de explicar:

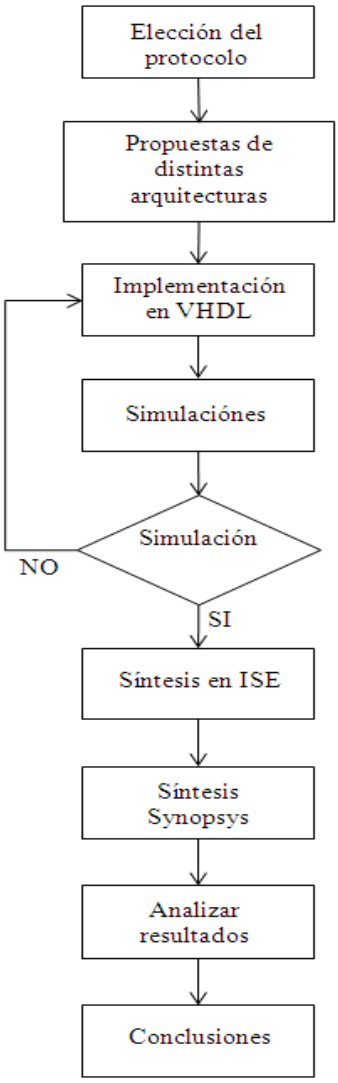

Figura 15: Diagrama de flujo de la implementación.

3.2 LENGUAJE VHDL

El lenguaje VHDL ha sido empleado para la realización de este proyecto, por lo que explicaremos brevemente en que se basa, así como su procedencia en una pequeña introducción.

Para empezar comentar que VHDL es el acrónimo que representa la combinación de VHSIC y HDL, donde VHSIC es el acrónimo de *Very High Speed Integrated Circuit* y HDL es a su vez el acrónimo de *Hardware Description Language*. Por tanto la primera idea que debemos abstraer es que el VHDL no es un lenguaje de programación, es una forma para describir formalmente circuitos electrónicos.

El primer borrador vio la luz en agosto de 1985 diseñado por *Intermetrics*, *IBM* y *Texas Instruments* e impulsado por el Departamento de Defensa de los Estados Unidos. En diciembre de 1987 fue aprobado como estándar del IEEE y posteriormente, en 1993, fue revisado y registrado como norma IEEE Std 1076-1993.

VHDL es un lenguaje de semántica orientada a la simulación y por ello su principal dominio de aplicación es el modelado de dispositivos *hardware* para comprobar su correcto funcionamiento. De todos modos tiene otras áreas de aplicación tales como: síntesis automática, verificación formal, modelado de rendimiento, diagnosis de fallos y documentación.

3.3 HERRAMIENTAS UTILIZADAS

Los programas que se han utilizado para realizar este proyecto fin de carrera son ModelSim de Menthors Graphics, ISE de Xilinx, VMware Player de VMware Inc. y el sintetizador de Synopsys Inc. Describiremos brevemente cada uno de ellos.

El primer programa, Modelsim [17] se ha utilizado para la implementación del código en el lenguaje de descripción hardware VHDL y su posterior simulación.

Con el ISE *(Integrate Software Environment)* [18][19][20][21], hemos realizado una síntesis de lo implementado algo más exhaustiva, consiguiendo de este modo eliminar los posibles warning que no detectan el Modelsim y evitando así, posibles errores a la hora de trabajar con Synopsys.

El programa Synopsys [22] que trabaja sobre Linux, motivo por el cual se ha utilizado el programa VMware Player para la utilización de una máquina virtual del sistema operativo Red Hat Linux Fedora. Se ha utilizado para la síntesis de los algoritmos diseñados y poder llevar un estudio adicional de área y consumo, este estudio de área sería diferente al de ISE puesto que el ISE se utiliza para implementar en FPGA´s que son más genéricas y se miden en LUTs mientras que el Synopsys se usa para implementar en ASIC´s que son circuitos integrados hechos a la medida para un uso en particular y su diseño está basado en celdas estándares (standard cells).

3.4 TEST- BENCH

Para la realización de la simulación de los diferentes códigos implementados se han realizado un banco de pruebas, donde se inicializamos las señales de entrada con

ciertos valores, además de estimular algunas entradas como puede ser la señal *start*, que inicializa el inicio del Tav-128. La señales generales, *reset* y *clk* también están estimuladas en el test-bench.

CAPÍTULO 4

TAV-128

En este proyecto se implementará una función hash de 128 bit, como es el Tav-128 orientado hacia etiquetas RFID de bajo coste y que proporciona un nivel de seguridad conveniente para su mayoría de usos.

4.1 FUNCIÓN HASH

La función hash hace referencia a un tipo de algoritmo que permite resumir y posteriormente identificar de manera íntegra la información contenida en un mensaje, texto, etc. evitando que la información pueda modificarse. Las características principales de este algoritmo son las siguientes [23]:

- Es unidireccional, ya que sólo funciona en la dirección empleada inicialmente. Esto implica que una vez aplicado el algoritmo a la información, y obtenido el resumen; la persona que tenga en su poder este resumen no va a poder obtener la información inicial, aunque conozca la función hash que se utilizó para obtenerla.
- Los caracteres de la información original sobre la que se aplica el algoritmo hash, son determinantes para crear el resumen. Si varía cualquier carácter, varía el resumen final.
- Es de tamaño estándar, independientemente de la información sobre la que es aplicado.

Tradicionalmente se exceden en las capacidades de los *tags* de bajo coste. La complejidad requerida del hardware de estos dispositivos esta en el área del circuito, o en el número de puertas lógicas equivalentes. Alrededor de 4000 puertas lógicas equivalentes (GE) son dedicadas a tareas relacionadas con la seguridad. La mejor implementación de SHA-256 [24] requiere alrededor de 11000 puertas y 1120 ciclos de reloj, para realizar un cálculo de hash en un bloque de datos de 512 bit. El número de recurso que se necesitan es demasiado para este tipo de *tags* y por eso se propone el uso de otra función. Otras funciones Hash muy extendidas son las funciones SHA-1 [25]

(8.100 puertas y 1228 ciclos) y MD5 [3] (8.400 y 612 ciclos), como podemos observar, exceden de las capacidades de las etiquetas de bajo coste.

4.1.1 Función Tav-128

En la figura 16 se muestra el código C de la función hash Tav-128 [33]. Analizando el código podemos observar en la inicialización, como las variables *h0* y *h1* obtienen el valor del acumulador *a0*. En las funciones A y B se realizan diferentes operaciones simples como sumas y desplazamientos de bits, repitiéndose estas sentencias hasta 32 veces, valor que fija por la variable de control *r1*. El resultado de estas funciones genera un primer valor a los vectores *h0* y *h1*.

Seguidamente el Tav-128 entra en las funciones C y D, donde *h0* y *h1* vuelven a obtener un nuevo valor. En estas funciones se realizan diferentes operaciones de suma y desplazamiento de bits, dentro de un bucle anidado. El bucle interior se genera 8 veces, fijado por la variable *r2*, y que provoca a su salida el incremento de la variable de control *i*, y la ejecución de las sentencias correspondientes al vector *state*, que usan los valores de *h0* y *h1* a su salida del mismo. Este bucle interior se vuelve a realizar hasta 4 veces, valor que fija la variable *nstate* correspondiente al bucle exterior.

Cuando las variables de control *j* alcanzan el valor *nstate*, la función sale del bucle anidado, y actualiza el valor del acumulador *a0*, sumando los valores que se han generado en *h0* y *h1*.

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

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

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

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

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

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

}
```

Figura 16: Función hash Tav-128

4.1.2 Seguridad del Tav-128

Muchos de los ataque en la funcionas Hash más importante son debido a que el algoritmo generalmente es muy lineal. Para evitar esto, las funciones C y D del código son altamente no lineales. Además tiene una fase de filtro correspondiente a las funciones A y B, con el fin de evitar el acceso a cualquier bit. La salida de 128 bit proporciona una velocidad razonable además de robustez a ataques realistas. Además el algoritmo contiene el uso de ocho rondas en el bucle interior, (parámetro r2, de la función C y D) que proporciona suficiente margen de seguridad. Las tablas 5 y 6 [33], reflejan los resultados que se obtuvieron en el análisis de los test de aleatoriedad (ENT y DIEHARD). Esta función resumen también ha obtenido resultados positivos en el test NIST [15].

	Tav-128
Test	**p-value**
Birthday Spacings	0.725
	0.868
GCD	0.229
	0.138
Gorilla	0.779
Overlapping Permutations	0.823
	0.849
	0.349
	0.897
	0.939
Ranks of 31×31 and 32×32 Matrices	0.556
	0.241
Ranks of 6×8 Matrices	0.315
Monkey Tests on 20-bit Words	0.312
Monkey Test OPSO, OQSO, DNA	OK
Count the 1's in a Stream of Bytes	0.473
Count the 1's in Specific Bytes	OK
Parking Lot Test	0.235
Minimum Distance Test	0.580
Random Spheres Test	0.912
The Squeeze Test	0.487
Overlapping Sums Test	0.106
Runs Up and Down Test	0.147
The Craps Test	0.3211
	0.067
	0.775
	0.261
Overall KS p-value	**0.826**

Tabla 5: Resultados obtenidos con Diehard

Test	**Tav-128**
Entropy	7.999999 bits/byte
Compression Rate	0%
χ^2 Statistic	269.73 (50%)
Arithmetic Mean	127.4993
Monte Carlo π estimation	3.14178848 (0.01%)
Serial correlation coefficient	-0.000073

Tabla 6: Resultados obtenidos con ENT

4.2 ARQUITECTURAS PROPUESTAS DEL TAV-128

En este proyecto final de carrera se ha implementado la función hash *Tav-128* donde se han tenido en cuenta los limitados recursos que presentan los *tags* RFID. Con estas especificaciones se han realizado 3 implementaciones diferentes (A, B y C) y se han analizado por separado cada uno de los resultados, para considerar cuál de ellas es la más apropiada para las necesidades requeridas.

Normalmente el numero de puertas equivalentes destinados a la seguridad en las etiquetas de bajo coste, no pueden exceder de 4000 EG [26][27][28]. Por ello, uno de los objetivos que se persigue es el diseño de esta función es no moverse en un rango superior a 4000 puertas equivalentes (Equivalent Gates, EG).

Otro requisito importante es el consumo de potencia debido a la naturaleza de bajo coste de los *tags*, esta no debe exceder de los 10μW [29]. Un parámetro influyente en este aspecto es la frecuencia de reloj [30]. La frecuencia de operación estándar para este tipo de etiquetas es 100KHz, que es bastante diferente de las que se utiliza comúnmente es otras aplicaciones. El número de ciclos de reloj estimados para la generación de esta función es de aproximadamente 500 o 600 ciclos.

4.2.1 Implementación, caso A

La arquitectura A ha sido diseñada para consumir el mínimo número de ciclos de reloj en la ejecución. Para ello, no se ha tenido en cuenta el número de sumadores a utilizar, como podemos observar en la figura 17.

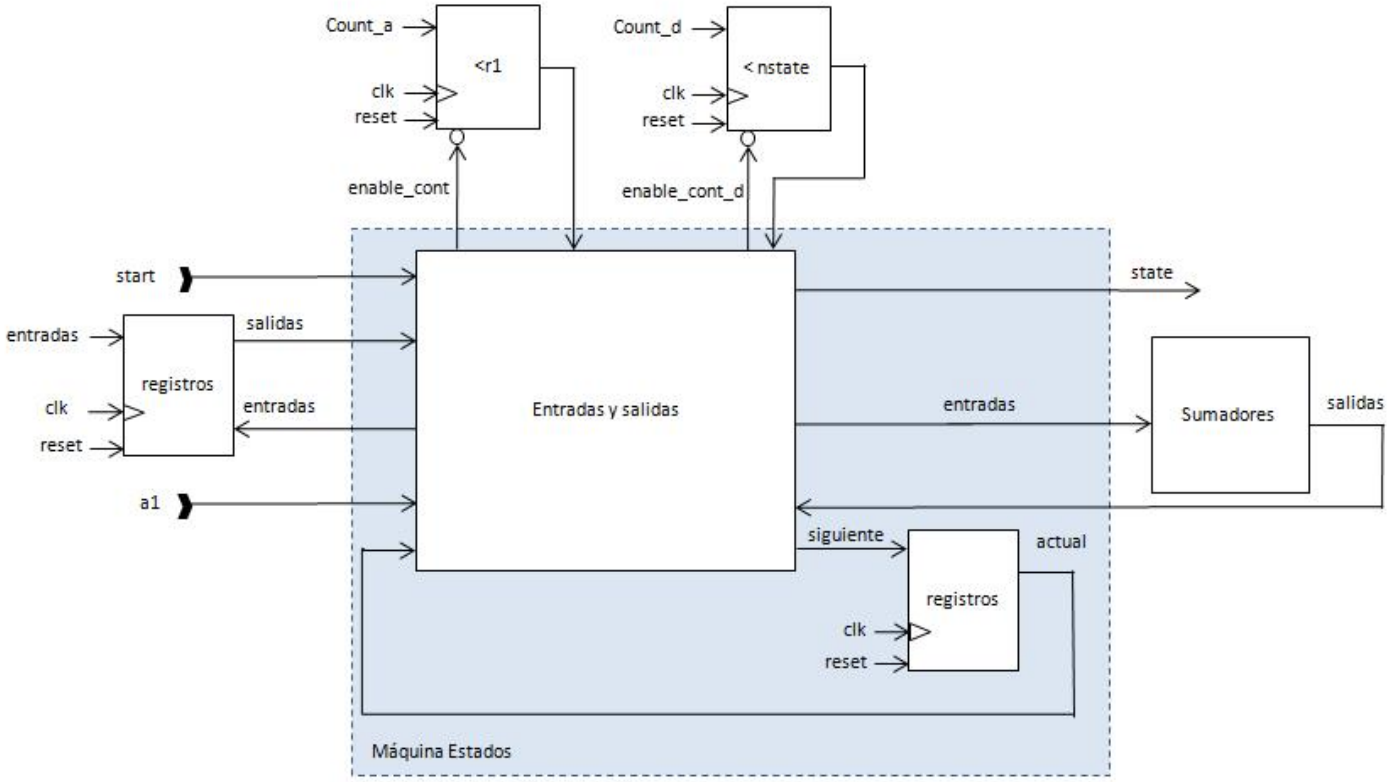

Figura 17: Arquitectura B de la función hash Tav-128

Con el uso de varios sumadores conseguimos realizar las funciones A y B en paralelo, además de conseguir realizar sentencias de las funciones C y D en un mismo ciclo de reloj. Por el contrario esta arquitectura necesita de bastante área, detalle que intentamos mejorar en los siguientes casos propuestos del proyecto. En la figura 18 podemos observar el diagrama de bloques de la máquina de estados.

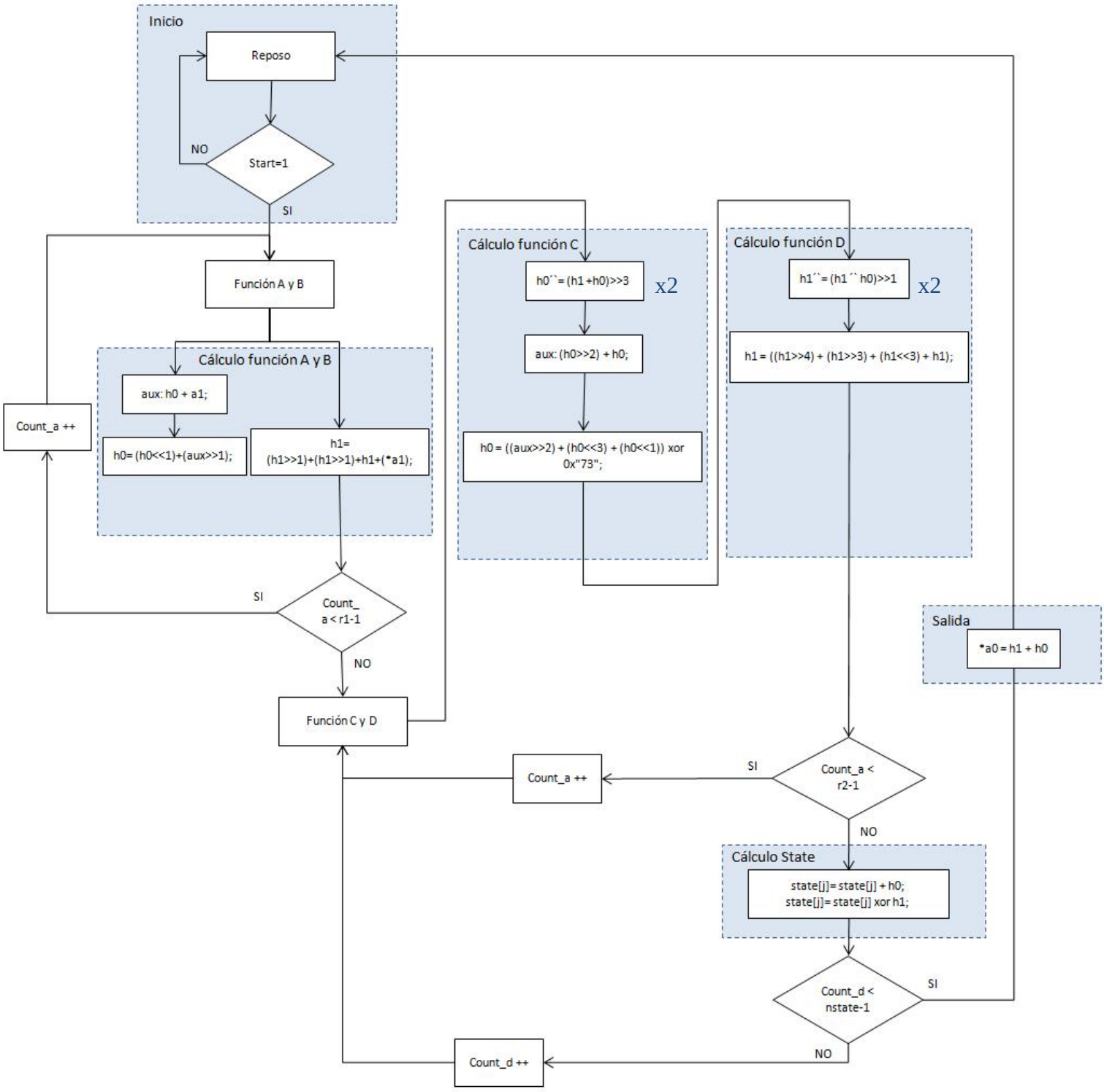

Figura 18: Máquina de estados implementada para el caso A.

En la simulación de la función mediante el programa ModelSim (ver figura 19) representamos el funcionamiento de esta arquitectura con 5 sumadores, donde reflejamos el comportamiento de la función A en rojo y B en azul. Tras la inicialización de la señal *start* la función hash empieza a ejecutarse. Simultáneamente se ejecutan 32 veces (r1=32) la funciones A y B correspondientes a los vectores *h0* y *h1* respectivamente, realizándose las distintas sentencias de suma y desplazamiento de bits, fijadas por el Tav-128.

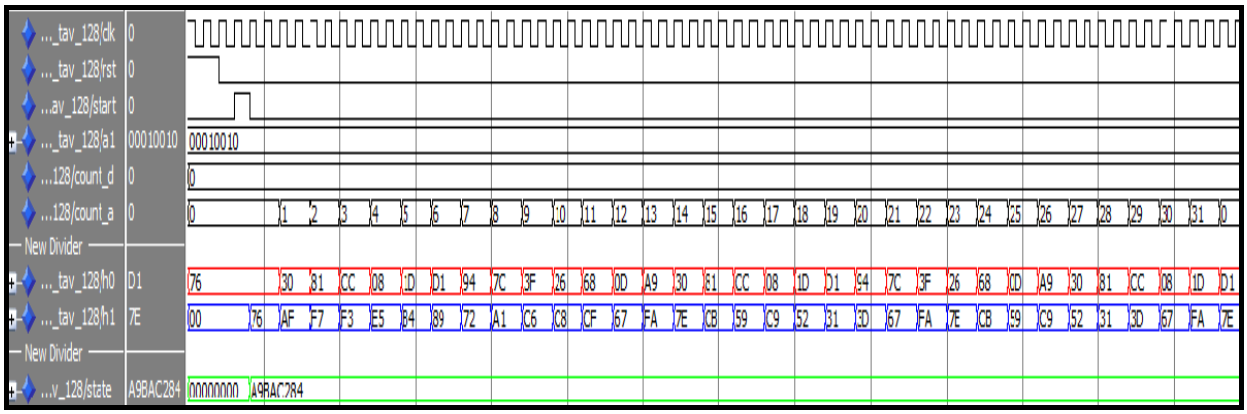

Figura 19: Simulación función A y B de la arquitectura B.

En la figura 20 se representa la otra parte de la simulación correspondientes a las funciones C y D, y por lo tanto el comportamiento de *h0* y *h1*. Además se representa en color verde el vector *state* que cambia al salir del bucle *for* anidado, ejecutándose las sentencias correspondientes al Tav-128.

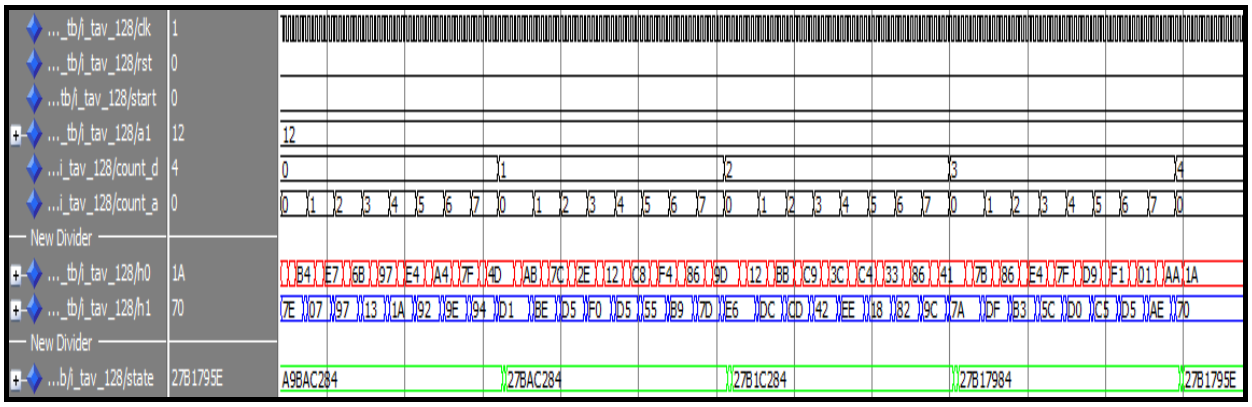

Figura 20: Simulación función A y B de la arquitectura B

4.2.2 Implementación, caso B

La arquitectura B ha sido diseñada para consumir el mínimo número de puertas equivalentes. Para ello, se ha decidido utilizar un solo sumador de n-bits, como podemos observar en la figura 21.

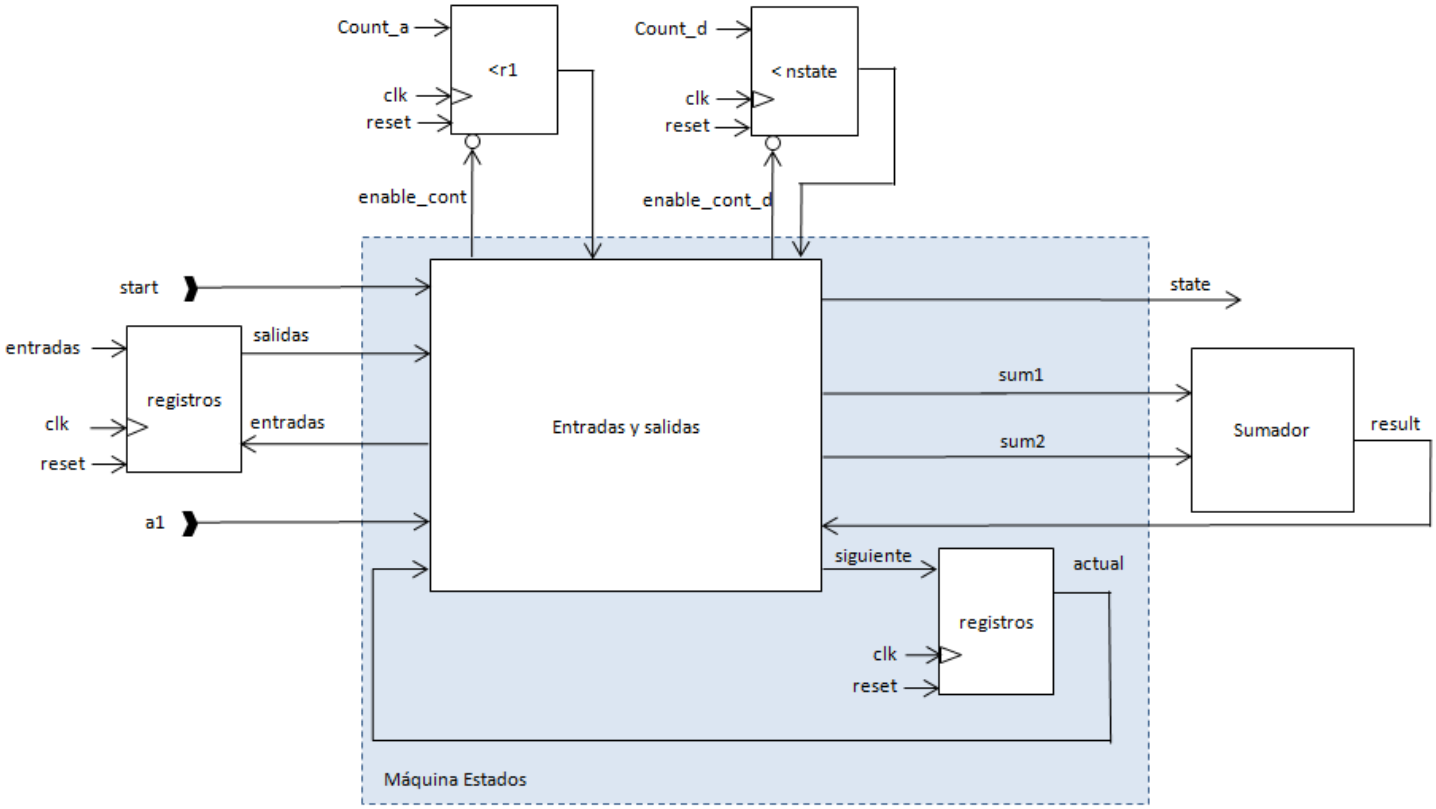

Figura 21: Arquitectura B de la función hash Tav-128

Con el uso de un solo sumador conseguimos mejoras en el área, pero nos limita en la realización de la función A y B en paralelo, (a diferencia de la arquitectura A) penalizando el uso de ciclos de reloj. En la figura 22 podemos observar el diagrama de bloques de la máquina de estados.

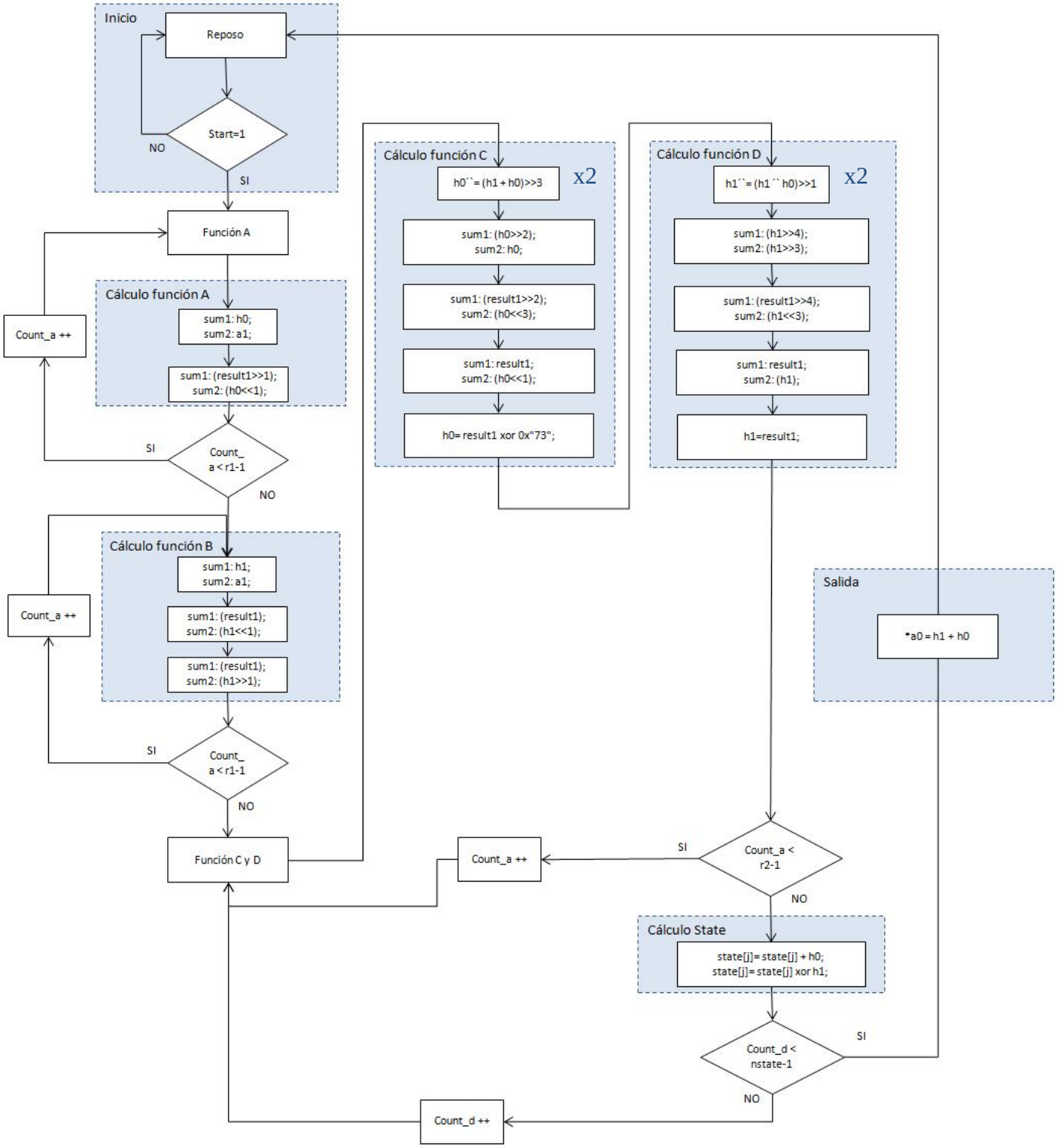

Figura 22: Máquina de estados implementada para el caso B.

En la simulación de la función mediante el programa ModelSim (ver figura 23) representamos el funcionamiento de esta arquitectura con un solo sumador, donde reflejamos el comportamiento de la función A en rojo y B en azul. Tras la inicialización de la señal *start* la función hash empieza a ejecutarse. Primeramente se ejecuta 32 veces (r1=32) la función A correspondiente al vector *h0*, realizando distintas sentencias de suma y desplazamiento de bits. Tras la finalización de las distintas operaciones se ejecuta la función B correspondiente al vector *h1*, del mismo modo que la anterior pero con diferentes sentencias, fijadas por el Tav-128.

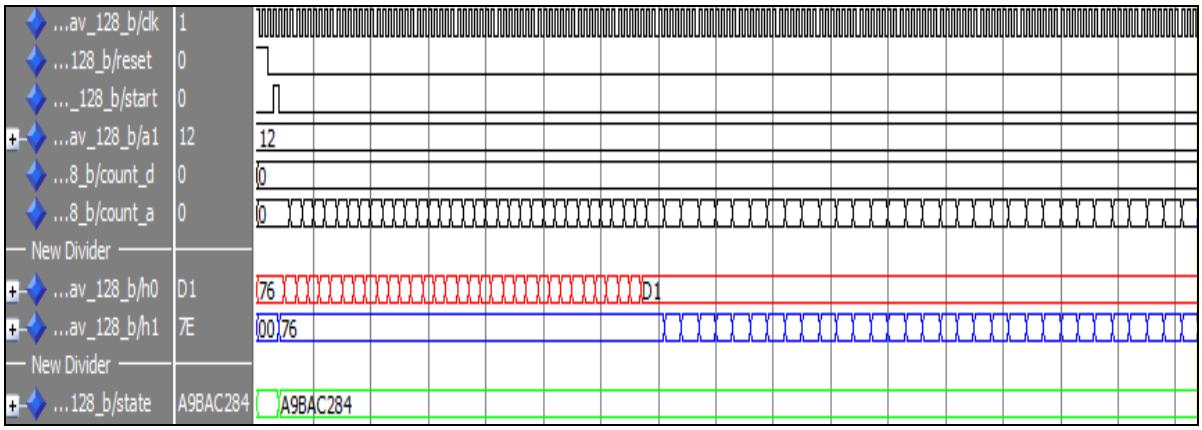

Figura 23: Simulación función A y B de la arquitectura B.

Debido a la longitud de la simulación, ésta se ha dividido en dos partes para poder representar las funciones C y D, y por lo tanto el comportamiento de *h0* y *h1* (ver figura 24). Además se representa en color verde el vector *state* que cambia al salir del bucle *for* anidado, ejecutándose las sentencias correspondientes al Tav-128.

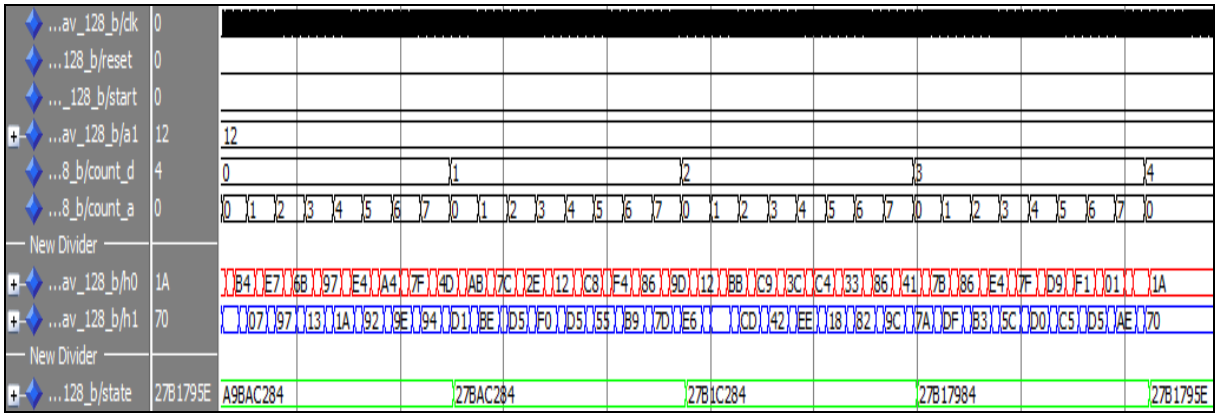

Figura 24: Simulación función C y D de la arquitectura B.

4.2.3 Implementación, caso C

La arquitectura C se ha diseñado para obtener una mejora de área con respecto a la arquitectura A, pero con una menor penalización en el uso de ciclos de reloj que la arquitectura B. Para ello se ha utilizado 2 sumadores de n-bits, en la figura 25 podemos ver el diagrama de bloques de la arquitectura C.

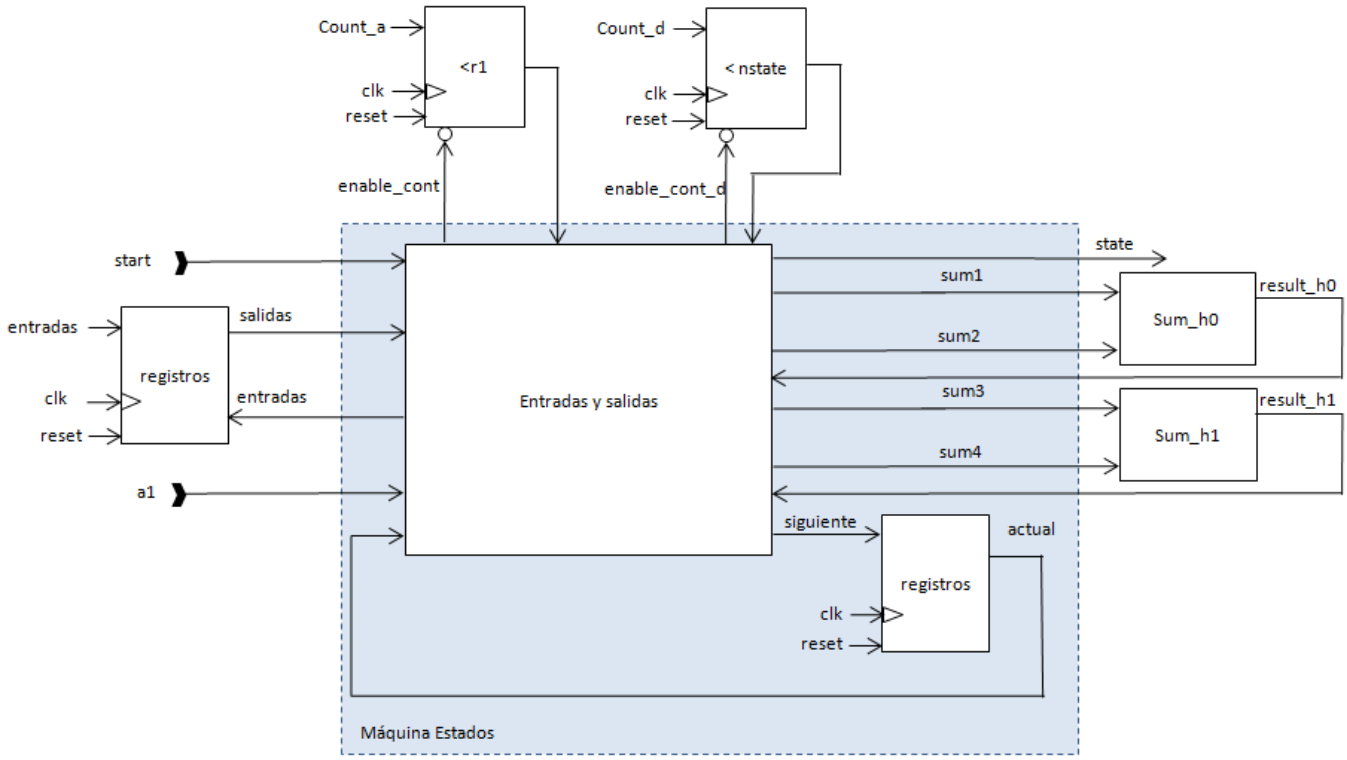

Figura 25: Arquitectura C de la función hash Tav-128

En este caso la máquina de estados controla los dos sumadores que se han utilizado con el fin de conseguir una mayor rapidez en la ejecución del Tav-128. Esto nos permite realizar sentencias en paralelo, como son las funciones A y B. También se han añadido otras sentencias correspondientes a las funciones C y D, que no alteran el funcionamiento de Tav-128 y que nos permite debido a esta arquitectura realizar algunas de sus sumas en el mismo ciclo de reloj, ver figura 26.

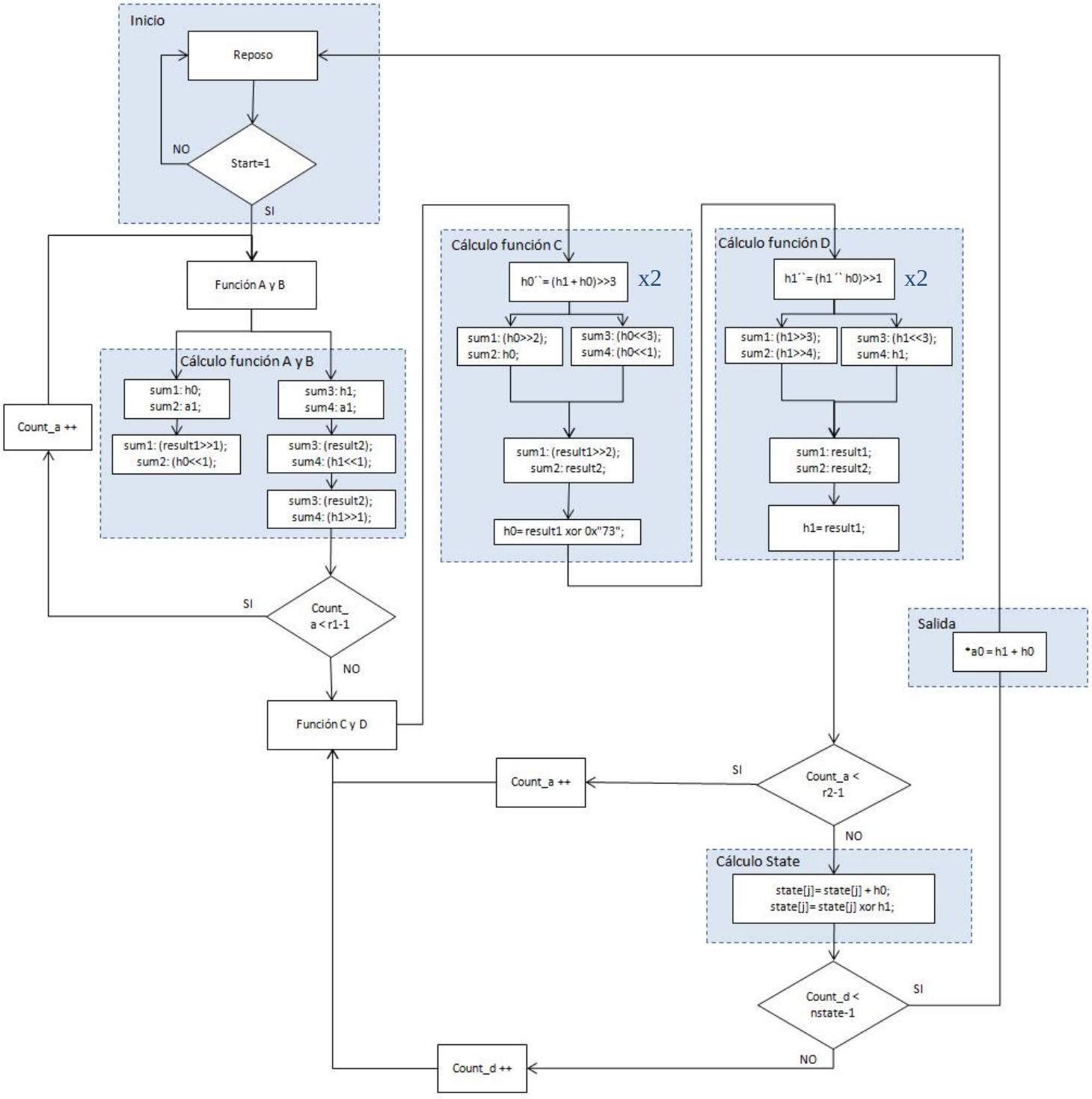

Figura 26: Máquina de estados implementada para el caso C.

En la simulación de la función para este caso C (ver figura 27) representamos el funcionamiento de esta arquitectura con dos sumadores, donde reflejamos el comportamiento de la función A en rojo y B en azul. Tras la inicialización de la señal *start* la función hash empieza a ejecutarse. Simultáneamente se ejecutan 32 veces (r1=32) la funciones A y B correspondientes a los vectores *h0* y *h1* respectivamente, realizándose las distintas sentencias de suma y desplazamiento de bits, fijadas por el Tav-128.

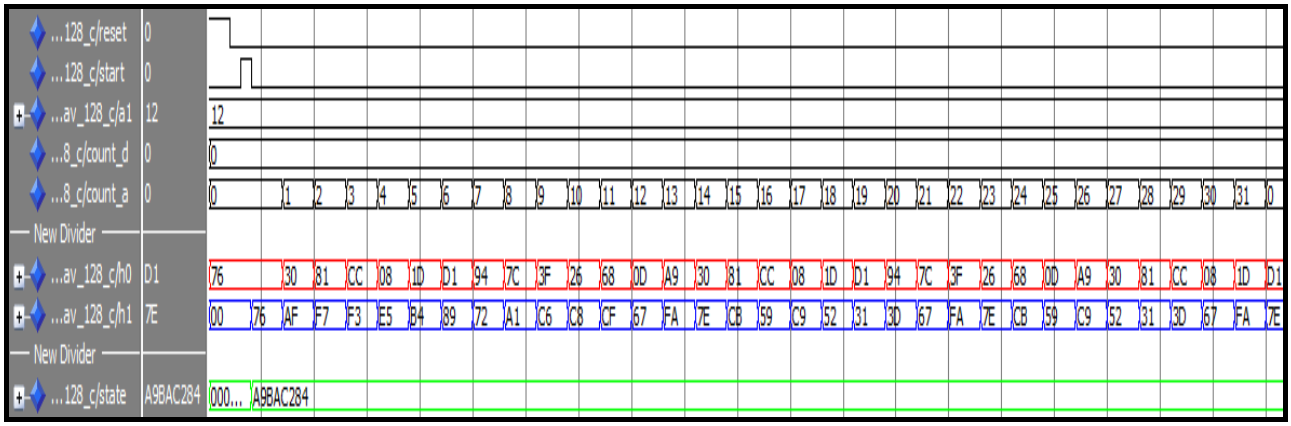

Figura 27: Simulación función A y B de la arquitectura C.

En la figura 28 se representa la otra parte de la simulación correspondientes a las funciones C y D, y por lo tanto el comportamiento de *h0* y *h1*. Además se representa en color verde el vector *state* que cambia al salir del bucle *for* anidado, ejecutándose las sentencias correspondientes al Tav-128.

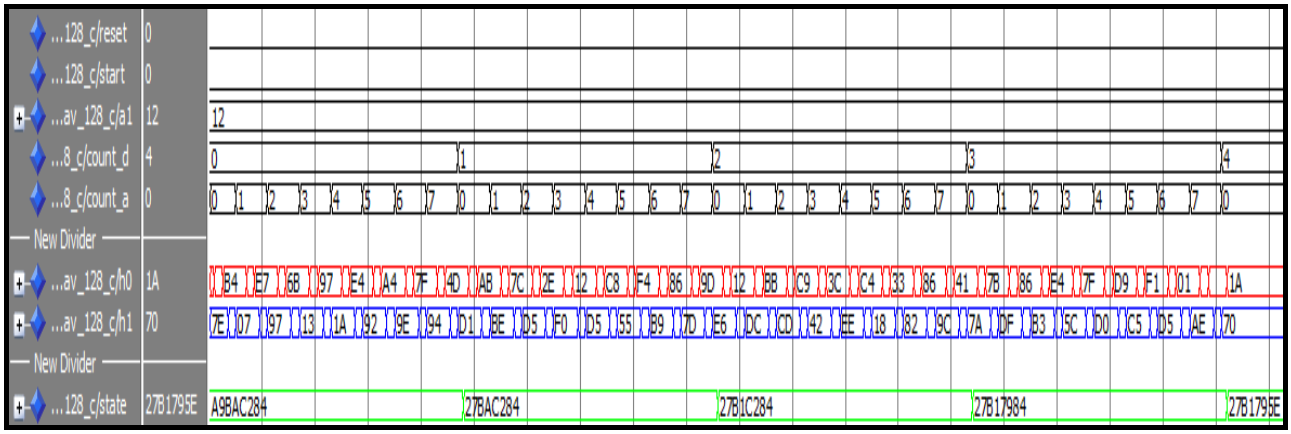

Figura 28: Simulación función C y D de la arquitectura B.

CAPÍTULO 5
RESULTADOS Y ANÁLISIS

En este capítulo se van a mostrar los resultados obtenidos en la síntesis realizada en Synopsys, de las tres arquitecturas diseñadas (A, B y C), modificando además el número de bits de salida en la función Hash.

5.1 CONSIDEREACIONES PREVIAS

Obtenidos los resultados de las simulaciones y una vez comprobado el correcto funcionamiento de la función resumen Tav-128 en Modelsim, se han realizado las síntesis de los diseños empleando la herramienta Design Vision del programa Sinopsys. Con esta herramienta se han obtenido las variables de área y potencia. El número de ciclos consumidos en cada diseño se han obtenido a través de las simulaciones.

Para la obtención de resultados en la herramienta Sinopsys se ha utilizado la librería comercial Faraday 90 nm del fabricante UMC. Esta librería contiene el *layout* de las celdas básicas utilizadas. Por lo que los resultados obtenidos, tanto en área como en consumo de potencia, son una muy buena aproximación a los resultados que se obtendrían una vez fabricado el circuito.

Los resultados que se muestran a continuación han sido obtenidos con un esfuerzo medio en área y mapeado, ya que con estas opciones se han obtenido los mejores resultados. Para el cálculo de la potencia consumida se ha utilizado en esfuerzo bajo, para que Synopsys muestre el consumo más crítico. En la síntesis se ha usado una frecuencia de 100 KHz, que es la frecuencia estándar para las etiquetas RFID de bajo coste. La alimentación utilizada ha sido de 1,25 V.

En la presentación de resultados, se muestra el área medido en puertas equivalentes. Una puerta equivalente representa una unidad de medida que permite especificar la complejidad de fabricación independientemente de la tecnología que se emplee. Está basado en el número de puertas lógicas individuales que tendrían que estar interconectadas para realizar la misma función que el circuito implementado. En el

diseño de circuitos digitales, para cada tecnología de fabricación, se utiliza una biblioteca de celdas estándar, la cual contiene una gran variedad de ellas.

En nuestro caso la tecnología empleada es de 90nm y el área de una puerta NAND es de 3,136 μm^2. Con este valor y una vez obtenido los resultados de área en los diseños, podremos saber el número de puertas equivalentes dividiendo el área del diseño entre el valor de NAND.

En la fase de síntesis de la que se van a extraer los datos de consumo de área y potencia de los diseños A, B y C, se han realizado para los casos de generar a la salida de la función resumen 8, 32, 64 y 128 bits. Con estas pruebas se pretende determinar qué nivel de seguridad que puede alcanzarse, cumpliendo siempre las restricciones impuestas en este proyecto.

5.2 RESULTADOS DE SINTESÍS, ARQUITECTURA A

Como se ha nombrado en el capitulo anterior con esta arquitectura queremos conseguir el menor número de ciclos de reloj, para la ejecución de Tav-128. Para ello se han utilizado 5 sumadores, que permite la realización de distintas sentencias en paralelo.

De los datos obtenidos de estas síntesis se obtiene la siguiente tabla resumen con los resultados del diseño A, sintetizados en función del número de bits de salida.

	WIDTH_state = 16 BITS	WIDTH_state = 32 BITS	WIDTH_state = 64 BITS	WIDTH_state = 128 BITS
Número de puertos	23	43	83	163
Número de nodos	429	770	1417	2785
Número de celdas	378	692	1249	2490
Área Combinacional (µm²)	1787,519	3749,952	7423,696	14542,416
Área no combinacional (µm²)	698,544	1055,264	1865,136	3462,928
Área total (µm²)	2486,063	4801,216	9288,832	18005,344
Potencia interna (nW)	41,174 (72%)	74,022 (70%)	138,789 (69%)	258,549 (68%)
Potencia conmutación (nW)	16,226 (28%)	32,324 (30%)	61,210 (31%)	122,3095 (32%)
Potencia dinámica total (nW)	57,409(100%)	106,346 (100%)	199,999 (100%)	380,859(100%)
Perdidas de potencia (nW)	3,841	7,986	15,479	29,45
Puertas equivalentes	792,7496811	1531	2962	5741,5
Tiempo de ejecución	268	268	268	268
Throughput (Kbps)	2,985	11,94	23,88	47,761

Tabla 7: Resultado de la síntesis A, según el número de bits de salida.

En primer lugar el tiempo de ejecución empleado para realizar este diseño A (ver tabla 7), es de 268 ciclos de reloj. Este resultado lo podemos desglosar en 2 partes,

concretamente: número de ciclos empleados para ejecutar las funciones A y B (en paralelo), y número de ciclos empleados para la funciones C y D (ver tabla 8).

Haciendo una comparativa con los resultados que nos encontraremos correspondientes a ciclos de reloj en los otros dos diseños (B y C), este caso A, es el más rápido en ejecutar la función resumen.

	Ciclos de reloj
Funciones A y B	64
Funciones C y D	204

Tabla 8: Desglose de ciclos consumidos, caso A.

En la figura 29 se reflejan los datos obtenidos de potencia consumida para cada una de las salidas, y donde cumplimos con los requisitos de no exceder los 10µW.

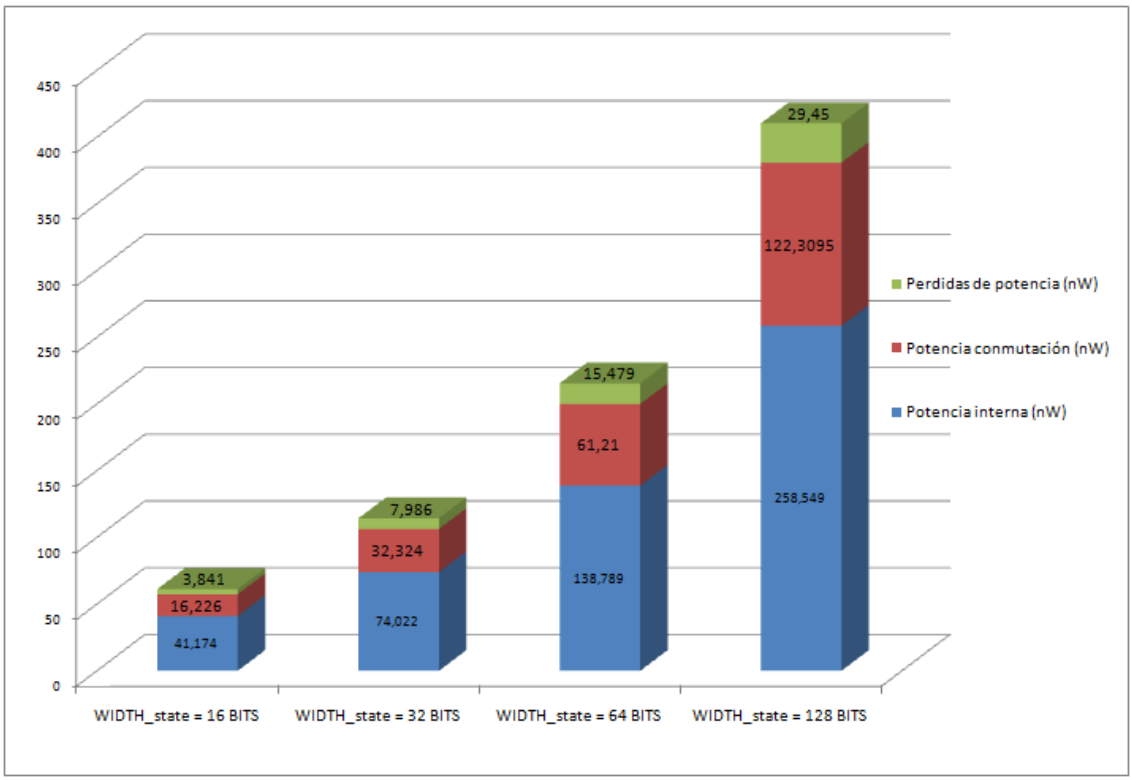

Figura 29: Distribución de potencia en arquitectura A, según sus bits de salida.

En la gráfica que aparece a continuación (ver figura 30), se muestra una comparativa del área empleada para los distintos números de bits de salida. Se puede observar como el área aumenta, a medida que el número de bits de salida es mayor.

En comparación con las arquitecturas B y C, este diseño es el que más área necesita, y es debido en gran parte al número de sumadores utilizados (5).

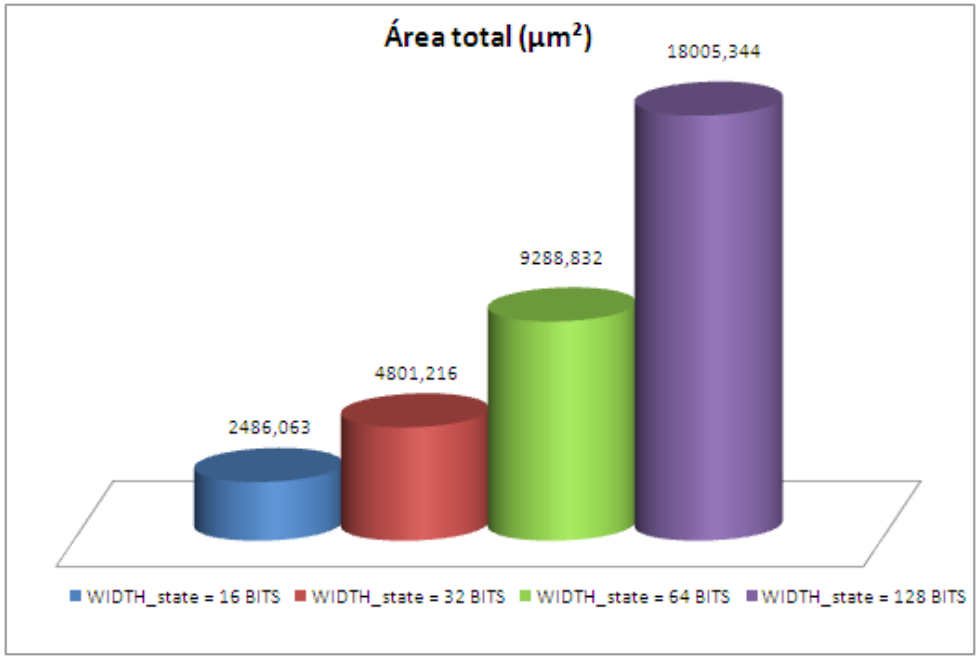

Figura 30: Área del diseño A según el número de bits.

A continuación mostramos el número de puertas lógicas equivalentes necesarias para los distintos números de bits de salida (ver figura 31). Se puede observar que para todos los casos exceptuando para 128 bits, se cumplen las restricciones de área (4000 E.G.). Por lo tanto esta arquitectura reduce el nivel de seguridad de la función Tav-128, para una salida de 128 bits.

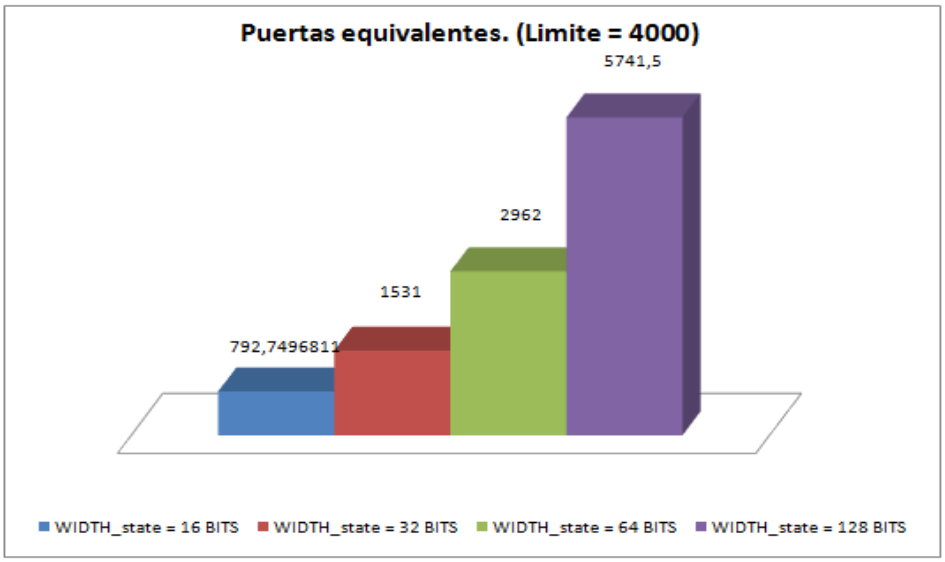

Figura 31: Número de E.G del diseño A según el número de bits.

Para esta arquitectura cumplimos con los requisitos mínimos para una salida de hasta 64 bits. En la tabla 9 reflejamos los datos finales para esta salida, entre los que está el numero de bits por segundo que es capaz de generar nuestra función hash.

	Potencia dinámica total (nW)	Puertas equivalentes	Throughput (Kbps)
WIDTH_state = 64 BITS	199,999	2962	23,88

Tabla 9: Parámetros para 64 bits, en arquitectura A.

5.3 RESULTADOS DE SINTESÍS, ARQUITECTURA B

El propósito de esta arquitectura B, ha sido reducir el área, penalizando el numero de ciclos de reloj, como ya se mencionó en el capítulo 4, para ello se han implementado un sumadores de n bits.

De los datos obtenidos de estas síntesis se obtiene la siguiente tabla resumen con los resultados del diseño B, sintetizados en función del número de bits de salida.

	WIDTH_state = 16 BITS	WIDTH_state = 32 BITS	WIDTH_state = 64 BITS	WIDTH_state = 128 BITS
Número de puertos	23	43	83	163
Número de nodos	353	537	869	1570
Número de celdas	325	491	782	1400
Área Combinacional (µm²)	1308,495	2135,615	3618,943	6701,631
Área no combinacional (µm²)	689,92	1100,736	1931,776	3590,72
Área total (µm²)	1998,415	3236,352	5550,72	10292,352
Potencia interna (nW)	41,426(72%)	68,565 (72%)	122,082 (72%)	229,339 (72%)
Potencia conmutación (nW)	16,209(28%)	26,596 (28%)	46,534 (28%)	88,993 (28%)
Potencia dinámica total (nW)	57,636(100%)	95,162 (100%)	168,616 (100%)	318,33 (100%)
Perdidas de potencia (nW)	3,344	5,247	8,929	16,066
Puertas equivalentes	637,2496811	1032	1770	3282
Tiempo de ejecución	504	504	504	504
Throughput (Kbps)	1,587	6,349	12,698	25,396

Tabla 10: Resultado de la síntesis B, según el número de bits de salida.

Como en la anterior síntesis primeramente analizamos los resultados de los ciclos de reloj empleados para realizar este diseño B (ver tabla 9), un total de 504 ciclos. Estos resultados lo podemos desglosar en 3 partes, concretamente: número de ciclos empleados para ejecutar las funciones A, número de ciclos empleados para ejecutar las funciones B, y número de ciclos empleados para la funciones C y D (ver tabla 10).

En esta arquitectura B, se emplean mas ciclos de reloj en comparación con la arquitectura A y C, concretamente 236 y 120 respectivamente.

	Ciclos de reloj
Función A	64
Función B	96
Funciones C y D	344

Tabla 11: Desglose de ciclos consumidos, caso B.

Este diseño como el anterior cumple con los requisitos de no exceder los 10µW. En la figura 32 se reflejan los datos obtenidos de potencia consumida para cada uno de los bits de salidas.

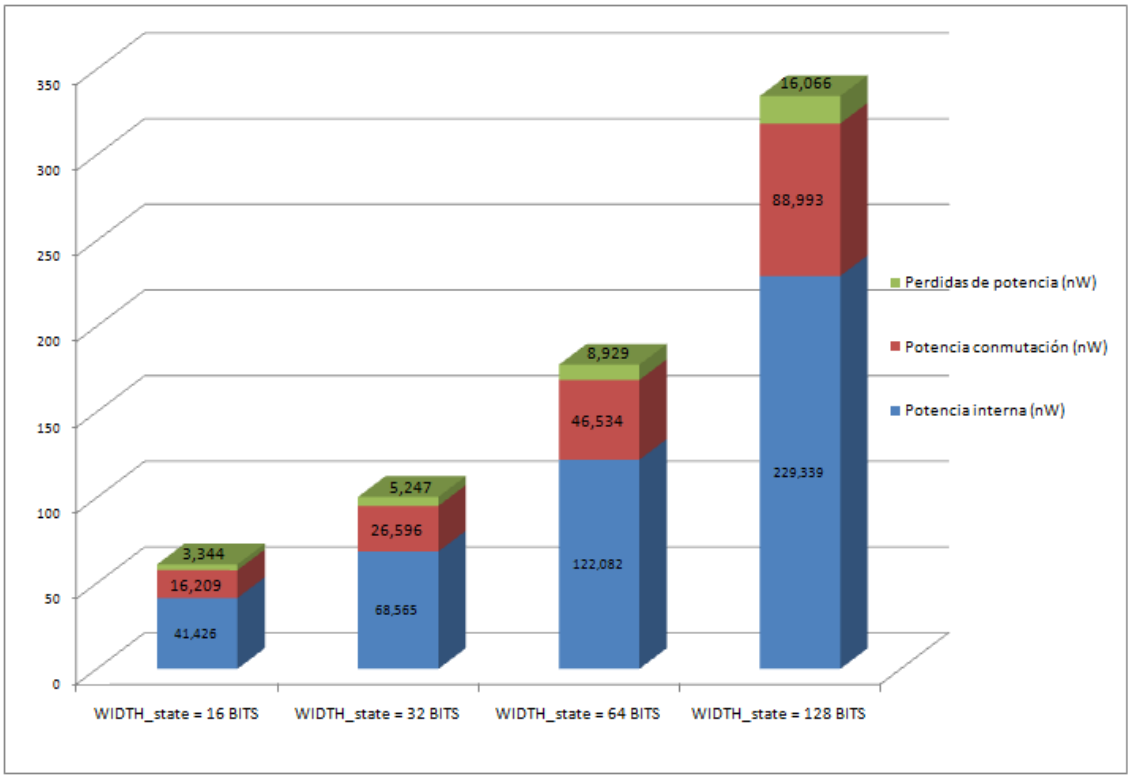

Figura 32: Distribución de potencia en arquitectura B, según sus bits de salida.

En la siguiente gráfica (ver figura 33), observamos el área empleada para los distintos números de bits de salida.

En comparación con las otras dos arquitecturas, este diseño es el que menos área necesita, y es debido al uso de un solo sumador de nº bits. La reducción que se consigue con respecto al caso A, es de un 42,83%, y un 15,31% menos que el caso C.

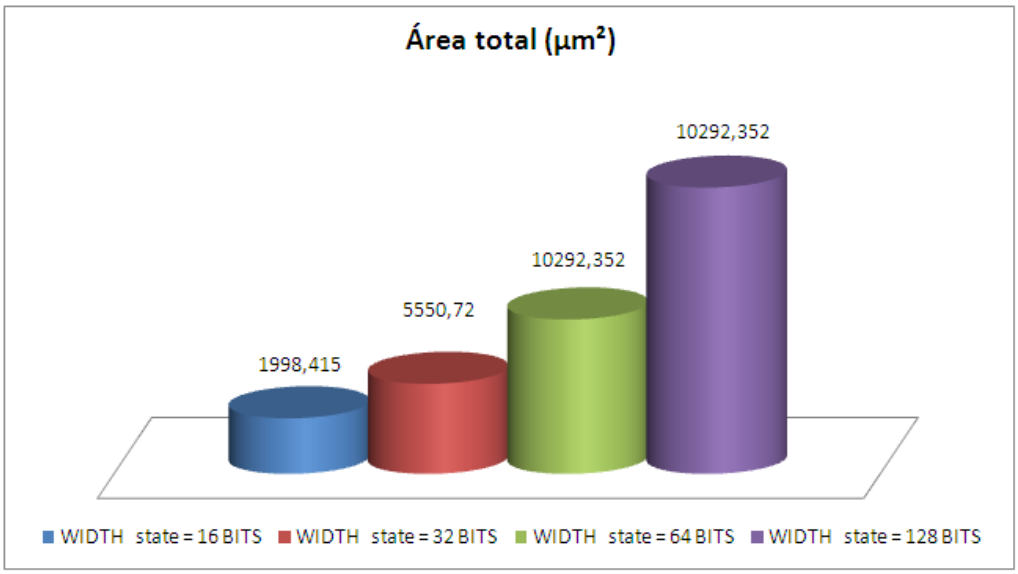

Figura 33: *Área del diseño B según el número de bits.*

Se puede observar en la figura 34 que para todos los casos de nº de bits de salida, se cumplen las restricciones de área impuestas (4000 E.G.). Por lo tanto esta arquitectura para cualquiera de los casos cumple el nivel de seguridad de la función Tav-128, a diferencia de la arquitectura A, donde se exceptuaba para una salida 128 bits.

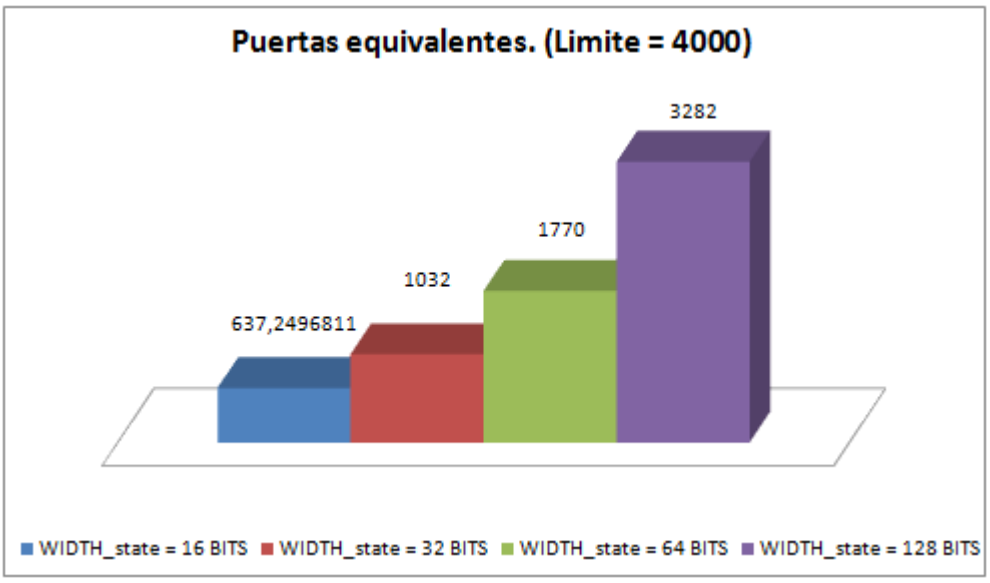

Figura 34: *Número de E.G del diseño B según el número de bits.*

La arquitectura B (ver tabla 12) cumple con los todos los requisitos mínimos para una salida de hasta 128 bits. Como diferencia con la arquitectura A, el throughput es menor, debido a un mayor tiempo de ejecución del mismo.

	Potencia dinámica total (nW)	Puertas equivalentes	Throughput (Kbps)
WIDTH_state = 128 BITS	318,33	3282	25,396

*Tabla 12***:** *Parámetros para 128 bits, en arquitectura B.*

5.4 RESULTADOS DE SINTESÍS, ARQUITECTURA C

El objetivo de esta arquitectura C, ha sido reducir el número de ciclos de reloj empleados para ejecutar la función hash, penalizando con respecto al diseño B, el área consumida. Como ya se mencionó en el capítulo 4 en el diseño se han implementado dos sumadores de n bits.

De los datos obtenidos de estas síntesis se obtiene la siguiente tabla resumen con los resultados del diseño C, sintetizados en función del número de bits de salida.

	WIDTH_state = 16 BITS	WIDTH_state = 32 BITS	WIDTH_state = 64 BITS	WIDTH_state = 128 BITS
Número de puertos	23	43	83	163
Número de nodos	369	592	987	1805
Número de celdas	336	536	883	1605
Área Combinacional (µm²)	1460,591	2449,215	4338,655	8114,399
Área no combinacional (µm²)	747,936	1218,336	2157,568	4019,168
Área total (µm²)	2208,52	3667,552	6496,224	12153,568
Potencia interna (nW)	45,643 (72%)	78,150 (73%)	142,888 (73%)	272,339 (70%)
Potencia conmutación (nW)	17,910 (28%)	29,014 (27%)	54,127 (27%)	114,658 (10%)
Potencia dinámica total (nW)	63,553 (100%)	107,165 (100%)	197,016 (100%)	387,058 (100%)
Perdidas de potencia (nW)	3,6583	6,083	10,631	19,651
Puertas equivalentes	704,247449	1169,5	2071,5	3875,5
Tiempo de ejecución	384	384	384	384
Throughput (Kbps)	2,083	8,333	16,666	33,333

Tabla 13: Resultado de la síntesis C, según el número de bits de salida.

Los resultados obtenidos en ciclos de reloj empleados para realizar este diseño C (ver tabla 11), han sido 384 ciclos. Desglosando estos resultados en 2 partes (ver tabla 12): número de ciclos empleados para ejecutar las funciones A y B (en paralelo), y número de ciclos empleados para la funciones C y D, donde el uso de dos sumadores permite realizar algunas sentencias en paralelo en estas últimas funciones.

En esta arquitectura C, se reduce el número de ciclos de reloj con respecto a la arquitectura B, concretamente 120 ciclos. Sin embargo en comparación con la arquitectura A se aumenta en, 116 ciclos de reloj.

	Ciclos de reloj
Funciones A y B	96
Funciones C y D	288

*Tabla 14***:** *Desglose de ciclos consumidos, caso C.*

En la figura 35 se reflejan la distribución de las potencias para cada una de las salidas, y donde cumplimos en todos los casos, los requisitos de no exceder los 10µW.

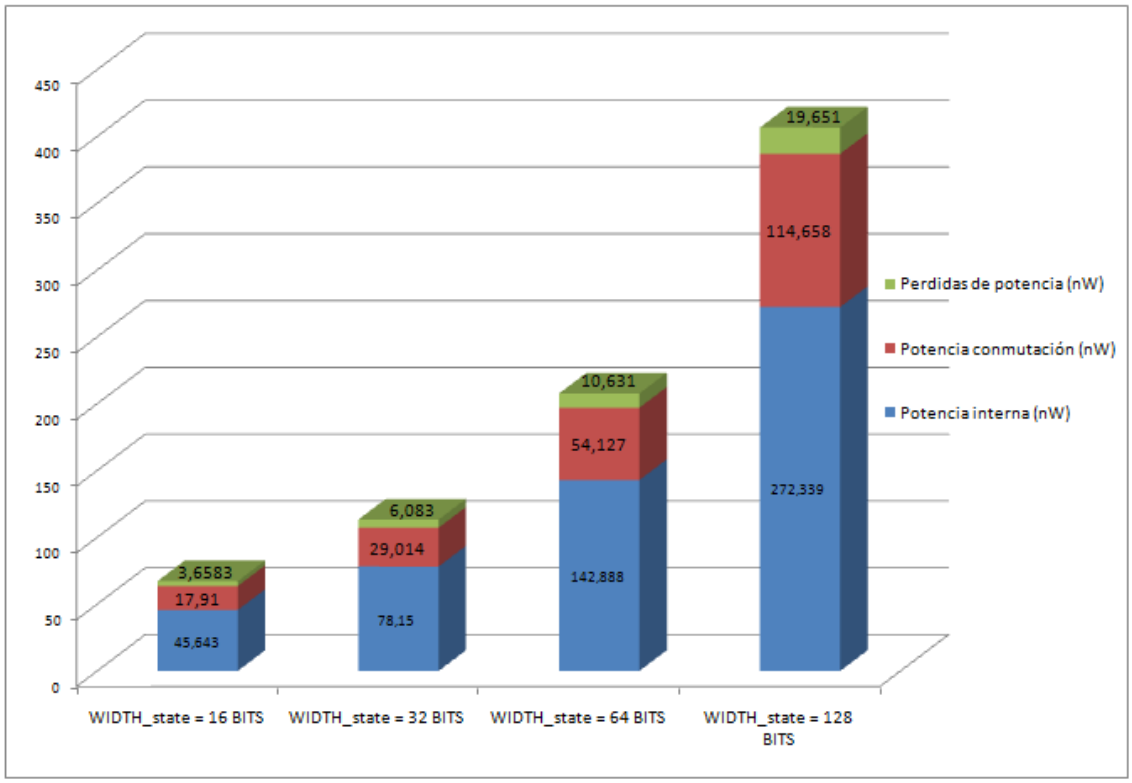

Figura 35: Distribución de potencia en arquitectura C, según sus bits de salida

A continuación se representa el área empleada para los distintos números de bits de salida, ver figura 36.

En comparación con las otras dos arquitecturas, este diseño no es el que menos área necesita, pero tampoco el que más. La reducción que se consigue con respecto al caso A, es de un 32,50%, y un 15,31% más que el caso C.

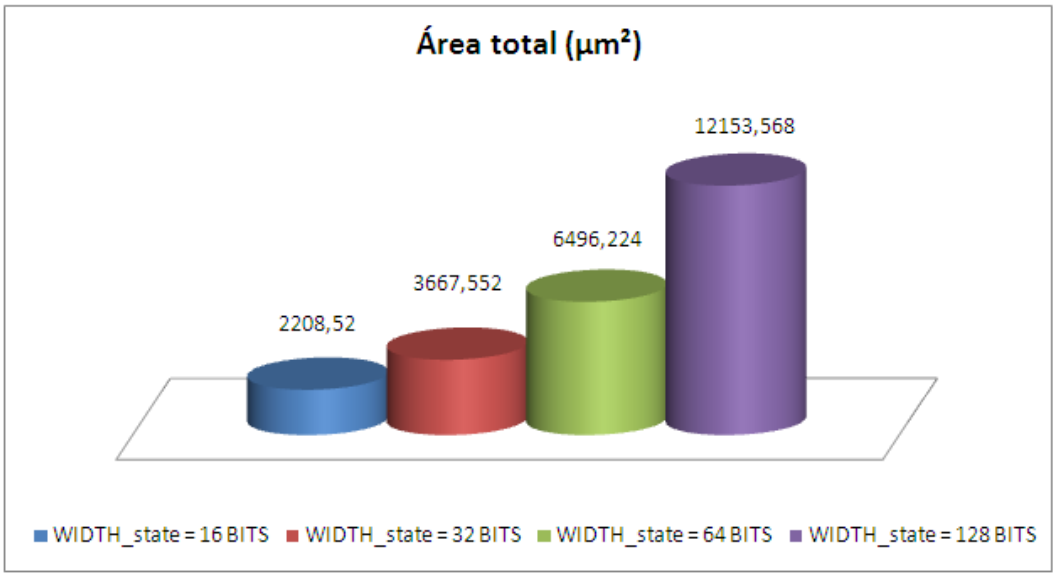

Figura 36: *Área del diseño C según el número de bits.*

El número de puerta lógicas equivalentes de esta arquitectura C se observa en siguiente gráfico (ver figura 37).

Para todos los casos de N de bits de salida, se cumplen las restricciones de área impuestas (4000 E.G.). Por lo tanto esta arquitectura para cualquiera de los casos cumple el nivel de seguridad de la función Tav-128.

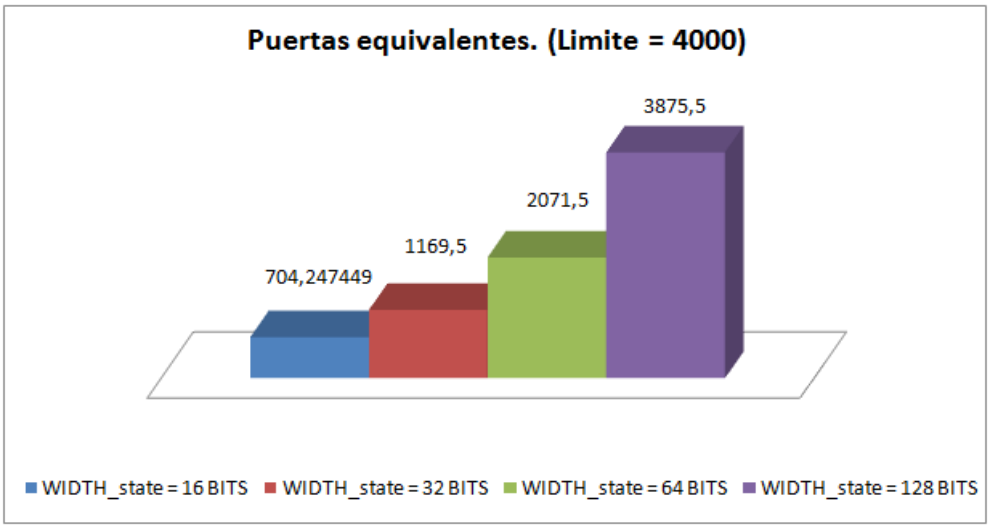

Figura 37: *Número de E.G del diseño C según el número de bits.*

En la tabla 15 se muestra un resumen de los resultados para la arquitectura C. En comparación con la arquitectura A el throughput es mayor, consecuencia de los 128 bits de salida, y del número de ciclos de reloj empleados, 384.

	Potencia dinámica total (nW)	Puertas equivalentes	Throughput (Kbps)
WIDTH_state = 128 BITS	387,058	3875	33,33

Tabla 15: Parámetros para 128 bits, en arquitectura C.

5.4 COMPARACIÓN DE ARQUITECTURAS

En este apartado vamos a comparar los resultados de los diferentes parámetros para las tres arquitecturas diseñadas, con el máximo número de bits, que cumplen los distintos requerimientos. La comparación entre las arquitecturas y el número de bits de salida son:

- Arquitectura A, para 64 bits.
- Arquitectura B, para 128 bits.
- Arquitectura C, para 128 bits.

5.4.1 Ciclos de reloj/throughput

En la figura 38, se ha realizado una comparativa para las tres arquitecturas diseñadas, con el número de bits de salida correspondiente al cumplimiento de lo que se ha requerido.

Se ha relacionado el número de ciclos de reloj con el throughput y se puede comprobar cómo en la Arquitectura A, el menor empleo de ciclos de reloj, no provoca un mayor valor del throughtput, que el resto de arquitecturas. Esto es consecuencia de otro parámetro influyente como es el número de bits de salida, en este caso 64 bits que es menor que el resto, debido al no cumplimiento del diseño para una salida de 128 bits.

Por otro lado, en los otros dos diseños se puede comprobar la diferencia de resultados, relacionándolo con la diferencia de añadir un sumador de n bits, o no. El tener dos sumadores de n bits provoca que en el diseño C, se puedan realizar varias sentencias de la función hash Tav-128 en paralelo, y en consecuencia menor número de ciclos de reloj empleados para la ejecución.

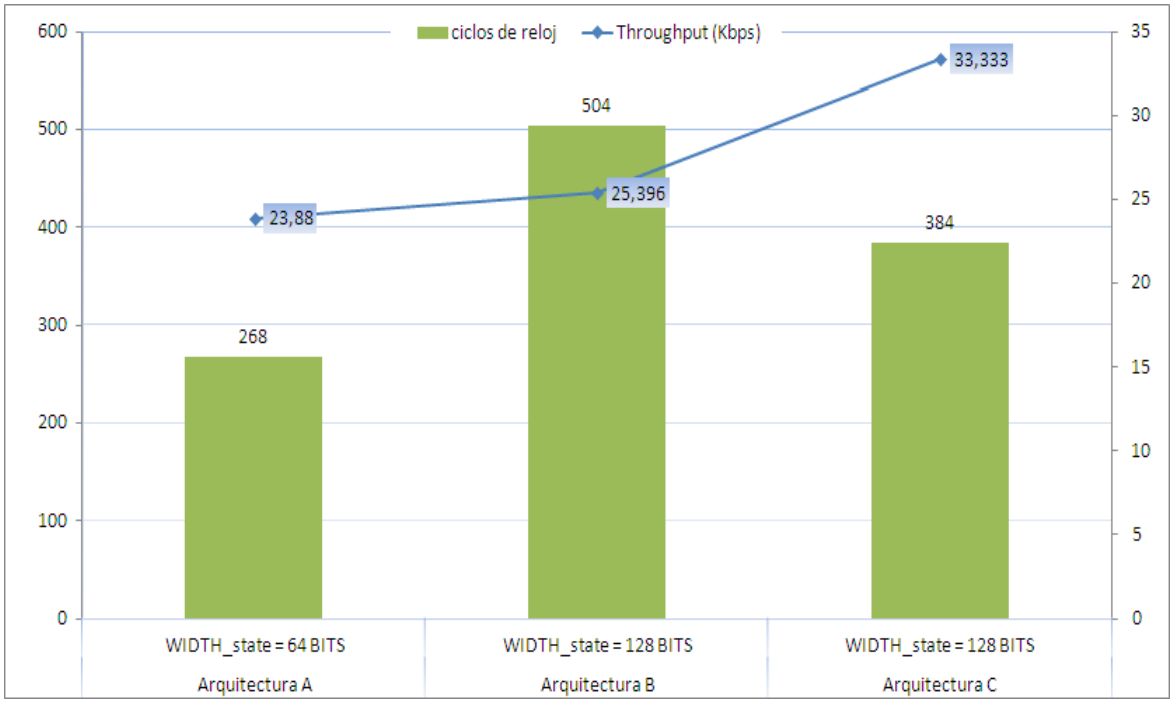

Figura 38: Comparación ciclos de reloj/throughput.

5.4.2 Consumo de potencia

Tanto la arquitectura A para 64 bits de salida, como las arquitecturas B y C con 128 bits de salida, cumplen los requisitos de no exceder los 10µW (ver figura 39).

En la arquitectura A la potencia interna de conmutación, como la interna es mucho más baja que las del resto, debido al no cumplimiento de este diseño para 128 bit de salida.

La potencia interna y de conmutación de la arquitectura B es un 15,78% y un 22,38% menor respectivamente que la C.

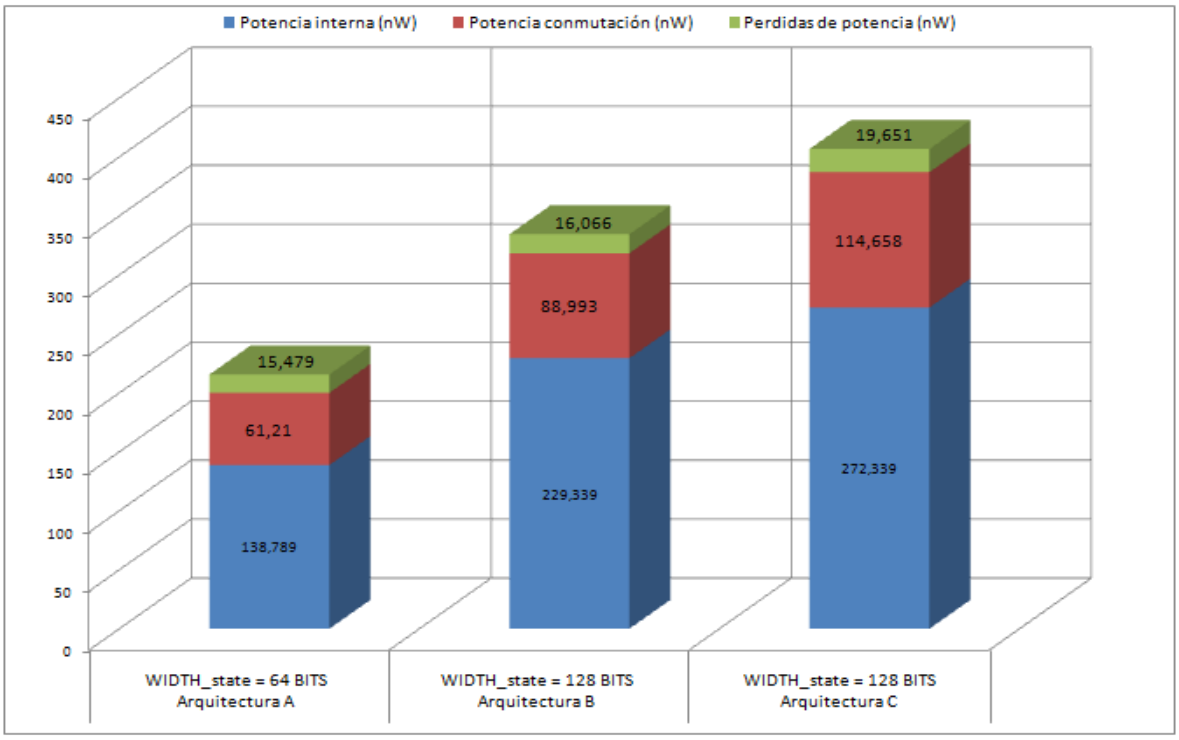

Figura 39: Comparación consumo de potencia.

5.4.3 Puertas lógicas equivalentes

En este apartado se hace una comparativa de las puertas lógicas equivalentes empleadas para cada una de las arquitecturas.

La arquitectura A es la que menos E.G emplea para la ejecución, sin embargo el número de bits de salida es menor (64 bits), reduciendo el nivel de seguridad de la función hash.

Las arquitecturas B y C cumplen con las restricciones de 4000 E.G. que está directamente relacionado con el área (ver figura 40). Comparando estas dos arquitecturas con el mismo número de bits de salida (128 bits), la arquitectura B utiliza un 15,30% menos de puertas lógicas equivalentes que la C.

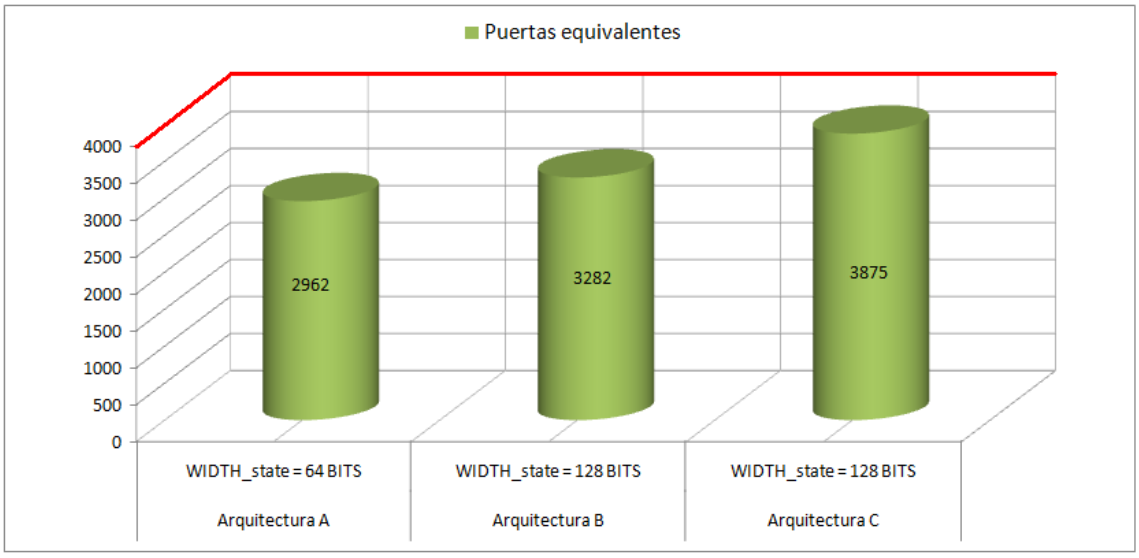

Figura 40: Comparación en puertas equivalentes.

Como resumen de las comparaciones que acabamos de realizar en este apartado, tendremos varias opciones de diseño, dependiendo de las exigencias de la aplicación.

Si lo que queremos es un nivel de alto en tiempo de ejecución, debemos escoger la arquitectura A, que ejecuta el Tav-128 en 268 ciclos de reloj, consecuencia de la realización de varias sentencias en paralelo y por lo tanto el uso de varios sumadores. Además se consiguen el mayor número de etiquetas/segundos leídas, que en el resto de diseños. Sin embargo con este diseño penalizamos en nivel de seguridad al reducirse el número de bits de salida.

Para un menor consumo de área y mayor seguridad, la arquitectura B es la idónea, consecuencia del numero de bits de salida, penalizando sin ser límite, en tiempo de ejecución (504 ciclos).

Si nuestra aplicación exige un alto throughtput elegimos la arquitectura C manteniendo además el nivel de seguridad como el diseño B, consecuencia del tiempo de ejecución empleado, para la ejecución de la función hash (384 ciclos). En esta arquitectura se leen más etiquetas/segundos que en la arquitectura B.

CAPÍTULO 6
CONCLUSIONES Y LÍNEAS FUTURAS

6.1 CONCLUSIONES

Tal y como se indicó en el capítulo de introducción de este proyecto, el objetivo presente es la implementación de la función Hash Tav-128, de forma genérica para realizar los correspondientes estudios a la generación del número de bits. Este objetivo se ha alcanzado con éxito, como se muestran los resultados en el capítulo 5.

Se ha realizado el diseño de las tres arquitecturas diferentes mediante lenguaje de descripción de hardware, con el objetivo de mejorar distintos aspectos en la generación de la función hash. Esta parte del diseño están reflejados en el capítulo 4 donde explicamos genéricamente las diferentes mejoras que se han conseguido con respecto a las otras.

Teniendo en cuenta las variables que hemos tenido como restricciones en la realización de la función hash para el cumplimiento con el nivel de seguridad, se pueden sacar varias conclusiones dependiendo de las necesidades de nuestra aplicación.

- Si lo que queremos es un nivel de alto en tiempo de ejecución, debemos escoger la arquitectura A
- Por otro lado si la necesidad viene limitada por el consumo de área, la opción correcta sería la arquitectura B.
- Si nuestra aplicación demanda un nivel de lectura alto lo mejor es elegir el diseño C.

Los resultados que se han obtenido y que se han reflejado en el capítulo anterior, son una buena aproximación a los resultados que se obtendrían en el circuito fabricado gracias a la librería utilizada que contiene el *layout* de las celdas básicas utilizadas.

6.2 LÍNEAS FUTURAS

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

a) Implementar la función hash Tav-128 dentro de protocolos que lo requieran, por ejemplo el incluido [33].
b) Estudiar la influencia del parámetro R2 tanto en seguridad como en mejora de uso de ciclos de reloj.
c) Proponer alguna arquitectura que reduzca el consumo de la función hash implementada.
d) Hacer un estudio sobre la seguridad e implementaciones en el que se compare la función Tav-128 con otras funciones Hash.
e) Búsqueda e implementación de nuevas funciones que puedan ser aplicados a las tarjetas RFID de bajo coste.

CAPÍTULO 7
PRESUPUESTO

En este capítulo se presenta los costes que se han generado para este proyecto. En él se incluye tanto los costes materiales como los personales [31] [32]. Para los costes personales se toma el coste de ingeniero mes en 1700 euros, como la dedicación no ha sido de 8 horas por día se considera que el coste por mes trabajado será un 30% aproximadamente. En cuanto a los costes materiales se tomarán un ordenador y los programas usados. Para obtener los costes imputables al proyecto se aplicará la siguiente fórmula de amortización de los mismos.

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

A = número de meses desde la fecha de facturación desde que el equipo es usado.
B = periodo de depreciación (meses).
C = coste del equipo (sin IVA).
D = % del uso que se dedica al proyecto.

Se ha tomado como el período de depreciación de los equipos informáticos 60 meses, en cuanto a las licencias de los programas se ha tomado 12 meses. Los costes indirectos se consideran un 20% de los costes totales del proyecto.

PRESUPUESTO DEL PROYECTO					
Autor:		José Luis Chavarrías López			
Descripción del proyecto:		Implementación Hardware de función resumen para			
GASTOS PERSONALES					
Personal	Categoria	Dedicación (hombre/mes)	Costes/mes	Porcentaje dedicación	Total (euros)
José Luis Chavarrías López	Ingeniero	6	1700	40%	4080
GASTOS MATERIALES					
Equipo	Costes (euros)	% Uso dedicado al proyeco	Dedicación (meses)	Periodo de depreciación	Coste imputables
PC	500	60	6	60	15
Licencia Modelsim	100	100	6	12	500
Licencia Synopsys	1800	100	6	12	900
Costes Materiales	1415				

Resumen de costes	**Euros**
Personal	4080
Amortización	1415
Costes indirectos	1099
Total	**6594**

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

CAPÍTULO 8
BIBLIOGRAFÍA

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

[2] Libera. Whitepaper series, "RFID: tecnología, aplicaciones y perspectivas" http://www.libera.net/uploads/documents/whitepaper_rfid.pdf

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

[4] Jose Manuel Jimenes Payá: "Diseño y construcción del subsistema RFID para un expositor inteligente, Proyecto Fin de Carrera, Universidad de Cataluña, diciembre 2008. http://upcommons.upc.edu/pfc/bitstream/2099.1/6086/1/memoria.pdf

[5] Informe de vigilancia tecnológica. "Tecnología de identificación por radiofrecuencia (RFID): aplicaciones en el ámbito de la salud. Javier I. Portillo García, Ana Belén Bermejo Nieto, Ana M. Bernardos Barbolla. www.madrimasd.org/tic/Informes/Downloads_GetFile.aspx?id=7913

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

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

[8] Guía sobre seguridad y privacidad de la tecnología RFID. "Instituto nacional de tecnologías de la comunicación". http://www.agpd.es/portalwebAGPD/revista_prensa/revista_prensa/2010/notas_prensa/common/julio/Guia_RFID.pdf

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

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

[11] "Estudio de la configuración óptima de longitud de ciclo en sistemas RFID", Mª Victoria Bueno Delgado, Javier Vales Alonso, Esteban Egea López, Joan García Haro. http://ait.upct.es/~eegea/pub/jitel08.pdf.

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

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

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

[15] "Evaluación de CSMA No Persistente como protocolo anticolisión en sistemas RFID activos.", Javier Vales Alonso[1], Fco. Javier González Castaño[2], Esteban Egea López[1], Mª Victoria Bueno Delgado[1], Alejandro Martínez Sala[1], Joan García Haro. 23 de noviembre de 2007.
http://ait.upct.es/~eegea/pub/jorRFID.pdf

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

[17] "Manual básico de uso de la herramienta ModelSim", Departamento de Tecnología Electrónica de la escuela politécnica superior de la UC3M, de Casado Ortiz, Fernando y Mengibar Pozo, Luis.

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

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

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

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

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

[23] www.inteco.es/wikiAction/Seguridad/.../area...1/Funcion_Hash

[24] Feldhofer, M., Rechberger, C.: A case against currently used hash functions in RFID protocols. In: Proc. of RFIDSec 2006 (2006). 19 de Octubre del 2012.

[25] SECURE HASH STANDARD http://csrc.nist.gov/publications/fips/fips180-2/fips180-2.pdf. Ultima fecha de visita el 15/10/2012

[26] S. Karthikeyan and M. Nesterenko. RFID security without extensive cryptography. In Proc. of SASN'05, 2005. Ultima fecha de visita el 18/10/2012

[27] A. Juels and S. Weis. Authenticating Pervasive Devices with Human Protocols.In V. Shoup, editor, Advances in Cryptology - Crypto 05, LNCS 3126, 293-198, Springer- Verlag, 2005. Ultima fecha de visita el 18/10/2012

[28] P. Peris-Lopez, J.C. Hernandez-Castro, J.M. Estevez-Tapiador, A. Ribagorda, ' RFID Systems: A Survey on Security Threats and Proposed Solutions', International Conference on Personal Wireless Communications - PWCA'06, LNCS 4217, 2006.

[29]: M. O'Neill, "Low-Cost SHA-1 Hash Function Architecture for RFID Tags". *Hand . of conference on RFID Security, 2008*

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

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

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

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

ANEXOS

CÓDIGO VHDL AQUITECTURA A

```
library IEEE;
  use IEEE.std_logic_1164.all;
  use IEEE.std_logic_arith.all;
  use IEEE.std_logic_unsigned.all;
------------------------------------------------------------------
-- ENTITY                                                       --
------------------------------------------------------------------
entity tav_128 is
  generic(
      r1          : natural:= 32;
      r2          : natural:= 8;
      nstate      : natural:= 4;
      H_WIDTH     : natural:= 8;
      WIDTH_state : natural:= 32
  );

  port(
    clk     : in std_logic;
    rst     : in std_logic;
    start   : in std_logic;
    a1      : in std_logic_vector (H_WIDTH-1 downto 0);
    state   : out std_logic_vector (WIDTH_state-1 downto 0)
  );
end entity;

architecture arq_tav_128 of tav_128 is

signal enable_cont         : std_logic:='0';
signal enable_cont_d       : std_logic:='0';
signal count_a             : INTEGER range r1 downto 0;
signal count_d             : INTEGER range nstate downto 0;
signal h0,h1,a_h0,a_h1     : std_logic_vector (H_WIDTH-1 downto 0);
signal aux,a_aux           : std_logic_vector (H_WIDTH-1 downto 0);
signal a_state_0           : std_logic_vector (H_WIDTH-1 downto 0);
signal a_state_1           : std_logic_vector (H_WIDTH-1 downto 0);
signal a_state_2           : std_logic_vector (H_WIDTH-1 downto 0);
signal a_state_3           : std_logic_vector (H_WIDTH-1 downto 0);
signal state_0             : std_logic_vector (H_WIDTH-1 downto 0);
signal state_1             : std_logic_vector (H_WIDTH-1 downto 0);
signal state_2             : std_logic_vector (H_WIDTH-1 downto 0);
signal state_3             : std_logic_vector (H_WIDTH-1 downto 0);
------------------------------------------------------------------
-- MÁQUINA DE ESTADOS                                           --
------------------------------------------------------------------
TYPE estados IS (reposo,funcion_a0,funcion_a,funcion_c_aux,
funcion_c_desp,funcion_auxh0,funcion_c,funcion_d_aux,
funcion_d_desp,funcion_d,state_h0,state_h1);

SIGNAL estado_actual,siguiente : estados;

begin


------------------------------------------------------------------
--             INICIO: MÁQUINA DE ESTADOS                       --
------------------------------------------------------------------
```

```
process(clk,rst)
    begin
        if rst='1' then
            estado_actual<=reposo;
      elsif clk'event and clk='1' then
            estado_actual<=siguiente;
      end if;
end process;

process(estado_actual,start,enable_cont,enable_cont_d,count_a,a_h0,
a_h1,a_aux,h0,h1,aux,count_d,a1,state_0,state_1,state_2,state_3,
a_state_0,a_state_1,a_state_2,a_state_3)

  begin

  a_state_0 <= state_0;
  a_state_1 <= state_1;
  a_state_2 <= state_2;
  a_state_3 <= state_3;
  a_aux <= aux;
  a_h0  <= h0;
  a_h1  <= h1;
  enable_cont_d <= '0';
  enable_cont <= '0';
case estado_actual is
-----------------------------------------------------------------
--             INICIO: ESTADOS FUNCION A Y B                   --
-----------------------------------------------------------------
   when reposo =>
    if start = '1' then           --- SEÑAL DE INICIO DE ALGORITMO
        a_h1 <= h0;
        a_state_0 <= x"a9";
        a_state_1 <= x"ba";
        a_state_2 <= x"c2";
        a_state_3 <= x"84";
        siguiente <= funcion_a0;
    else
        siguiente <= reposo;
    end if;

   when funcion_a0 =>             --- (h0+(*a1))
        a_aux     <= h0 + a1;
        siguiente <= funcion_a;

   when funcion_a =>
        enable_cont <= '1';
        a_h0 <= (h0(H_WIDTH-2 downto 0) & h0(H_WIDTH-1)) +
                (aux(0)& aux(H_WIDTH-1 downto 1));
        a_h1 <= (h1(0) & h1(H_WIDTH-1 downto 1)) +
                (h1(H_WIDTH-2 downto 0)& h1(H_WIDTH-1)) + h1 + a1;

    if count_a < r1-1 then
        siguiente <= funcion_a0;
    else
        siguiente <= funcion_c_aux;
    end if;
```

```
-------------------------------------------------------------------
--             FIN: ESTADOS FUNCION A Y B                        --
-------------------------------------------------------------------
-------------------------------------------------------------------
--             INICIO: ESTADOS FUNCION C                         --
-------------------------------------------------------------------

when funcion_c_aux =>     --- (h0+h1)
  a_aux <= h1 + h0;
  siguiente <= funcion_c_desp;

when funcion_c_desp =>     --- h0´`= (h0+h1)>>3
  a_h0 <= (aux(H_WIDTH-6 downto 0) & aux(H_WIDTH-1 downto H_WIDTH-5))
           xor h0 ;
  siguiente <= funcion_auxh0;

when funcion_auxh0 =>     --- ((h0>>2)+h0)
  a_aux <= ((h0(H_WIDTH-7 downto 0) & h0(H_WIDTH-1 downto H_WIDTH-6))
           + h0);
  siguiente <= funcion_c;

when funcion_c =>  -- ((((h0>>2)+h0)>>2) + (h0<<3) + (h0<<1))xor 0x73
  a_h0 <= ((aux(1) & aux(0) & aux(H_WIDTH-1 downto 2)) +
          (h0(H_WIDTH-4 downto 0) & h0(H_WIDTH-1 downto H_WIDTH-3)) +
          (h0(H_WIDTH-2 downto 0) & h0(H_WIDTH-1)) xor x"76");
  siguiente <= funcion_d_aux;

-------------------------------------------------------------------
--             INICIO: ESTADOS FUNCION D                         --
-------------------------------------------------------------------

when funcion_d_aux =>     --- (h0´`h1)
  a_aux <= h1 xor h0;
  siguiente <= funcion_d_desp;

when funcion_d_desp =>    --- h1´`= (h0´`h1)>>1
  a_h1 <= (aux(0) & aux(H_WIDTH-1 downto 1)) xor h1 ;
  siguiente <= funcion_d;

when funcion_d =>     --- ((h1>>4)+ (h1>>3)+ (h1<<3)+ h1)
  enable_cont_d <= '1';
  a_h1 <= ((h1(H_WIDTH-5 downto 0) & h1(H_WIDTH-1 downto H_WIDTH-4))
          + (h1(H_WIDTH-6 downto 0) & h1(H_WIDTH-1 downto H_WIDTH-5))
          + (h1(H_WIDTH-4 downto 0) & h1(H_WIDTH-1 downto H_WIDTH-3))
          + h1);

    if count_a < r2-1 then
        siguiente <= funcion_c_aux;
    else
        siguiente <= state_h0;
    end if;
-------------------------------------------------------------------
--             FIN: ESTADOS FUNCION C Y D                        --
-------------------------------------------------------------------
-------------------------------------------------------------------
-- INICIO ESTADO: STATE = STATE + H0 y  STATE = STATE Xor H1     --
-------------------------------------------------------------------
```

```
when state_h0 =>
      if count_d = 1 and count_a = 0 then
        a_state_0 <=  state_0 + h0;
      elsif count_d = 2 and count_a = 0 then
        a_state_1 <=  state_1 + h0;
      elsif count_d = 3 and count_a = 0 then
        a_state_2 <=  state_2 + h0;
      elsif count_d = 4 and count_a = 0 then
        a_state_3 <=  state_3 + h0;
      end if;
      siguiente <= state_h1;

when state_h1 =>
      if count_d = 1 and count_a = 0 then
        a_state_0 <=  state_0 xor h1;
      elsif count_d = 2 and count_a = 0 then
        a_state_1 <=  state_1 xor h1;
      elsif count_d = 3 and count_a = 0 then
        a_state_2 <=  state_2 xor h1;
      elsif count_d = 4 and count_a = 0 then
        a_state_3 <=  state_3 xor h1;
      end if;

      if count_d =4 and count_a = 0 then
        siguiente <= reposo;
        a_h0<=h1 + h0;
      else
        siguiente <= funcion_c_aux;
      end if;


-------------------------------------------------------------------------
-                FIN ESTADO: STATE = STATE + H0                        --
--                           STATE = STATE Xor H1                       --
-------------------------------------------------------------------------
when others =>

end case;
end process;

state <= state_0 & state_1 & state_2 & state_3;
-------------------------------------------------------------------------
--                FIN: MÁQUINA DE ESTADOS                              --
-------------------------------------------------------------------------
-------------------------------------------------------------------------
-- CONTADOR FUNCIÓN A y B                                               --
-------------------------------------------------------------------------

process(clk,rst)
  begin
    if rst='1' then
      count_a <= 0;
    elsif clk'event and clk='1' then
        if enable_cont ='1' then
            if count_a < r1-1 then
              count_a <= count_a + 1;
            else
```

```
              count_a <= 0;
       end if;
        elsif enable_cont_d ='1' then
            if count_a < r2-1 then
              count_a <= count_a + 1;
            else
              count_a <= 0;
            end if;
        end if;
    end if;
end process;

-------------------------------------------------------------------
-- CONTADOR FUNCIÓN C y D                                        --
-------------------------------------------------------------------
process(clk,rst)
  begin
    if (rst='1')  then
      count_d <= 0;
    elsif clk'event and clk='1' then
       if enable_cont_d = '1' then
          if count_d < nstate  then
              if count_a = r2-1 then
                 count_d <=  count_d + 1;
              end if;
          end if;
        end if;
    end if;
 end process;
-------------------------------------------------------------------
--             REGISTRO h0 y h1                                  --
-------------------------------------------------------------------

process(clk,rst)
  begin
    if rst='1'then
      h0      <= x"76";
      h1      <= (others =>'0');
      aux     <= (others =>'0');
      state_0 <= (others =>'0');
      state_1 <= (others =>'0');
      state_2 <= (others =>'0');
      state_3 <= (others =>'0');
    elsif clk' event and clk='1' then
      h0      <= a_h0;
      h1      <= a_h1;
      aux     <= a_aux;
      state_0 <= a_state_0;
      state_1 <= a_state_1;
      state_2 <= a_state_2;
      state_3 <= a_state_3;
    end if;
  end process;

end arq_tav_128;
```

CÓDIGO VHDL

AQUITECTURA B

```
library IEEE;
  use IEEE.std_logic_1164.all;
  use IEEE.std_logic_arith.all;
  use IEEE.std_logic_unsigned.all;
--------------------------------------------------------------------
--                          ENTITY                                --
--------------------------------------------------------------------
entity tav_128_b is
  generic(
      r1          : natural:= 32;
      r2          : natural:= 8;
      nstate      : natural:= 4;
      H_WIDTH     : natural:= 8;
      WIDTH_state : natural:= 32
  );

  port(
    clk     : in std_logic;
    reset   : in std_logic;
    start   : in std_logic;
    a1      : in std_logic_vector (H_WIDTH-1 downto 0);
    state   : out std_logic_vector (WIDTH_state-1 downto 0)
  );
end entity;

architecture arq_tav_128_b of tav_128_b is

signal h0,h1,a_h1,a_h0,reg,a_reg
       : std_logic_vector (H_WIDTH-1 downto 0);
signal a_state_0,a_state_1,a_state_2,a_state_3,
       state_0,state_1,state_2,state_3
       : std_logic_vector (H_WIDTH-1 downto 0);
signal enable_cont,enable_cont_d
       : std_logic:='0';
--------------------------------------------------------------------
--                        COMPONENTE SUMADOR                      --
--------------------------------------------------------------------
component sumador
    Port(termino1 :in std_logic_vector (H_WIDTH-1 downto 0);
    termino2 :in std_logic_vector (H_WIDTH-1 downto 0);
    salida: out std_logic_vector (H_WIDTH-1 downto 0));
end component;

TYPE estados IS (reposo,funcion_a0,funcion_A,funcion_b0,funcion_b1,
funcion_B,funcion_c0,funcion_c1,funcion_c2,funcion_c3,funcion_c4,
funcion_C,funcion_d0,funcion_d1,funcion_d2,funcion_d3,funcion_d4,
funcion_D,state_h0,state_h1);

SIGNAL actual,siguiente:estados;
signal sum1,sum2,result    : std_logic_vector (H_WIDTH-1 downto 0);
signal count_a             : INTEGER range r1 downto 0;
signal count_d             : INTEGER range nstate downto 0;
begin
--------------------------------------------------------------------
--                               MAP                              --
--------------------------------------------------------------------
sum_h0 : sumador PORT MAP(sum1,sum2,result);
```

```
-------------------------------------------------------------------
--             INICIO: MÁQUINA DE ESTADOS                        --
-------------------------------------------------------------------
process(clk,reset)          a_reg <= result;
    begin
        if reset='1' then
            actual<=reposo;
        elsif clk'event and clk='1' then
            actual<=siguiente;
        end if;
end process;

process(actual,start,a1,h1,h0,a_h0,a_reg,reg,result,count_a,count_d,
enable_cont,a_state_0,a_state_1,a_state_2,a_state_3,state_0,state_1,
state_2,state_3)
  begin
  a_state_0 <= state_0;
  a_state_1 <= state_1;
  a_state_2 <= state_2;
  a_state_3 <= state_3;
  sum1 <= (others =>'0');
  sum2 <= (others =>'0');
  a_reg <= reg;
  a_h0  <= h0;
  a_h1  <= h1;
  enable_cont <= '0';
  enable_cont_d <= '0';
  case actual is
-------------------------------------------------------------------
--             INICIO: ESTADOS FUNCION A Y B                     --
-------------------------------------------------------------------
   when reposo =>
    if start = '1' then          --- SEÑAL DE INICIO DE ALGORITMO
        a_h1 <= h0;
        a_reg <= result;
        sum1      <= h0;
        sum2      <= a1;
        a_state_0 <= x"a9";
        a_state_1 <= x"ba";
        a_state_2 <= x"c2";
        a_state_3 <= x"84";
        siguiente <= funcion_a0;
    else
        siguiente <= reposo;
    end if;

    when funcion_a0 =>
        a_reg <= result;
        sum1 <= (reg(0)& reg(H_WIDTH-1 downto 1));
        sum2 <= (h0(H_WIDTH-2 downto 0) & h0(H_WIDTH-1));
        a_h0 <= result;
    siguiente <= funcion_A;

    when funcion_A =>
        enable_cont <= '1';
    if count_a < r1-1 then
```

```
        sum1 <= h0;
        sum2 <= a1;
        siguiente <= funcion_a0;
    else
        siguiente <= funcion_b0;
    end if;

    when funcion_b0 =>
        a_reg <= result;
        sum1      <= h1;
        sum2      <= a1;
        siguiente <= funcion_b1;

     when funcion_b1 =>
        a_reg <= result;
        sum1 <= reg;
        sum2 <= (h1(H_WIDTH-2 downto 0)& h1(H_WIDTH-1));
        siguiente <= funcion_B;

    when funcion_B =>
        a_reg <= result;
        sum1 <= reg;
        sum2 <= (h1(0) & h1(H_WIDTH-1 downto 1));
        a_h1 <= result;
        enable_cont <= '1';
    if count_a < r1-1 then
        siguiente <= funcion_b0;
    else
        siguiente <= funcion_c0;
    end if;

-------------------------------------------------------------------------
--             FIN: ESTADOS FUNCION A Y B                              --
-------------------------------------------------------------------------
-------------------------------------------------------------------------
--             INICIO: ESTADOS FUNCION C                               --
-------------------------------------------------------------------------
    when funcion_c0 =>    --- (h0+h1)
        a_reg <= result;
        sum1 <= h0;
        sum2 <= h1;
    siguiente <= funcion_c1;

    when funcion_c1 =>    --- h0´`= (h0+h1)>>3
        a_h0 <= (reg(H_WIDTH-6 downto 0) &
                 reg(H_WIDTH-1 downto H_WIDTH-5))xor h0;
    siguiente <= funcion_c2;

    when funcion_c2 =>    --- ((h0>>2)+h0)
        a_reg <= result;
        sum1 <= (h0(H_WIDTH-7 downto 0) &
                 h0(H_WIDTH-1 downto H_WIDTH-6));
        sum2 <=  h0;
    siguiente <= funcion_c3;

    when funcion_c3 =>    --- ((((h0>>2)+h0)>>2) + (h0<<3)
        a_reg <= result;
```

```
        sum1 <= (reg(1) & reg(0) & reg(H_WIDTH-1 downto 2));
        sum2 <= (h0(H_WIDTH-4 downto 0) &
                 h0(H_WIDTH-1 downto H_WIDTH-3));
    siguiente <= funcion_c4;

    when funcion_c4 => -- ((((h0>>2)+h0)>>2) + (h0<<3) + (h0<<1))
        a_reg <= result;
        sum1 <= reg;
        sum2 <= (h0(H_WIDTH-2 downto 0) & h0(H_WIDTH-1));
    siguiente <= funcion_C;

    when funcion_C => --((((h0>>2)+h0)>>2)+(h0<<3)+(h0<<1))xor 0x"76"
        a_h0 <= reg xor x"76";
    siguiente <= funcion_d0;
-------------------------------------------------------------------
----              INICIO: ESTADOS FUNCION D                      --
-------------------------------------------------------------------
    when funcion_d0 =>    --- (h0´`h1)
        a_reg <= h0 xor h1;
    siguiente <= funcion_d1;

    when funcion_d1 =>    --- h1´`= (h0´`h1)>>1
        a_h1 <= (reg(0) & reg(H_WIDTH-1 downto 1)) xor h1;
    siguiente <= funcion_d2;

    when funcion_d2 =>    --- (h1>>4) + (h1>>3)
        a_reg <= result;
        sum1 <= (h1(H_WIDTH-5 downto 0) &
                 h1(H_WIDTH-1 downto H_WIDTH-4));
        sum2 <= (h1(H_WIDTH-6 downto 0) &
                 h1(H_WIDTH-1 downto H_WIDTH-5));
    siguiente <= funcion_d3;

    when funcion_d3 =>    --- ((h1>>4)+ (h1>>3) + (h1<<3)
        a_reg <= result;
        sum1 <=  reg;
        sum2 <= (h1(H_WIDTH-4 downto 0) &
                 h1(H_WIDTH-1 downto H_WIDTH-3));
    siguiente <= funcion_d4;

    when funcion_d4 =>    --- ((h1>>4)+ (h1>>3) + (h1<<3) + h1
        a_reg <= result;
        sum1 <=  reg;
        sum2 <=  h1;
    siguiente <= funcion_D;

    when funcion_D =>
        enable_cont_d <= '1';
        a_h1 <= reg;

    if count_a < r2-1 then
        siguiente <= funcion_c0;
    else
        siguiente <= state_h0;
    end if;
```

```
-------------------------------------------------------------------
--             FIN: ESTADOS FUNCION C Y D                        --
-------------------------------------------------------------------
-------------------------------------------------------------------
--  INICIO ESTADO: STATE = STATE + H0   y STATE = STATE Xor H1    --
-------------------------------------------------------------------
    when state_h0 =>
        if count_d = 1 and count_a = 0 then
          a_reg <= result;
          sum1 <= state_0;
          sum2 <= h0;
        elsif count_d = 2 and count_a = 0 then
          a_reg <= result;
          sum1 <= state_1;
          sum2 <= h0;
        elsif count_d = 3 and count_a = 0 then
          a_reg <= result;
          sum1 <= state_2;
          sum2 <= h0;
        elsif count_d = 4 and count_a = 0 then
          a_reg <= result;
          sum1 <= state_3;
          sum2 <= h0;
        end if;
    siguiente <= state_h1;

    when state_h1 =>
        if count_d = 1 and count_a = 0 then
          a_state_0 <=  reg xor h1;
        elsif count_d = 2 and count_a = 0 then
          a_state_1 <=  reg xor h1;
        elsif count_d = 3 and count_a = 0 then
          a_state_2 <=  reg xor h1;
        elsif count_d = 4 and count_a = 0 then
          a_state_3 <=  reg xor h1;
        end if;
      if count_d =4 and count_a = 0 then
        siguiente <= reposo;
        sum1 <= h1;
        sum2 <= h0;
        a_h0<=result;
      else
        siguiente <= funcion_c0;
      end if;

    when others =>

end case;
end process;
state <= state_0&state_1&state_2&state_3;
-------------------------------------------------------------------
--             REGISTROS                                         --
-------------------------------------------------------------------

process(clk,reset)
  begin
```

```
    if reset='1'then
      h0       <= x"76";
      h1       <= (others =>'0');
      reg      <= (others =>'0');
      state_0  <= (others =>'0');
      state_1  <= (others =>'0');
      state_2  <= (others =>'0');
      state_3  <= (others =>'0');
    elsif clk' event and clk='1' then
      h0       <= a_h0;
      h1       <= a_h1;
      reg      <= a_reg;
      state_0 <= a_state_0;
      state_1 <= a_state_1;
      state_2 <= a_state_2;
      state_3 <= a_state_3;
    end if;
  end process;


-------------------------------------------------------------------------
-- CONTADOR FUNCIÓN A y B                                               --
-------------------------------------------------------------------------

process(clk,reset)
  begin
    if reset='1' then
      count_a <= 0;
    elsif clk'event and clk='1' then
        if enable_cont ='1' then
            if count_a < r1-1 then
              count_a <= count_a + 1;
            else
              count_a <= 0;
            end if;
        elsif enable_cont_d ='1' then
            if count_a < r2-1 then
              count_a <= count_a + 1;
            else
              count_a <= 0;
            end if;
        end if;
    end if;
end process;

-------------------------------------------------------------------------
-- CONTADOR FUNCIÓN C y D                                               --
-------------------------------------------------------------------------
process(clk,reset)
  begin
    if (reset='1')  then
      count_d <= 0;
    elsif clk'event and clk='1' then
       if enable_cont_d = '1' then
          if count_d < nstate  then
              if count_a = r2-1 then
                 count_d <=  count_d + 1;
```

```
                else
                   count_d <=  count_d;
                end if;
            end if;
          end if;
      end if;
  end process;
end arq_tav_128_b;
```

CÓDIGO VHDL AQUITECTURA C

```
library IEEE;
  use IEEE.std_logic_1164.all;
  use IEEE.std_logic_arith.all;
  use IEEE.std_logic_unsigned.all;
--------------------------------------------------------------------
-- ENTITY                                                         --
--------------------------------------------------------------------
entity tav_128_c is
  generic(
      r1          : natural:= 32;
      r2          : natural:= 8;
      nstate      : natural:= 4;
      H_WIDTH     : natural:= 8;
      WIDTH_state : natural:= 32
  );

  port(
    clk     : in std_logic;
    reset   : in std_logic;
    start   : in std_logic;
    a1      : in std_logic_vector (H_WIDTH-1 downto 0);
    state   : out std_logic_vector (WIDTH_state-1 downto 0)
  );
end entity;

architecture arq_tav_128_c of tav_128_c is
signal h0,h1,a_h1,a_h0,a_reg1,reg1,a_reg2,reg2
  : std_logic_vector (H_WIDTH-1 downto 0);
signal a_state_0,a_state_1,a_state_2,a_state_3,
  state_0,state_1,state_2,state_3
  : std_logic_vector (H_WIDTH-1 downto 0);
signal enable_cont,enable_cont_d
       : std_logic:='0';
--------------------------------------------------------------------
--                       COMPONENTE SUMADOR                       --
--------------------------------------------------------------------
component sumador
    Port(termino1 :in std_logic_vector (H_WIDTH-1 downto 0);
    termino2 :in std_logic_vector (H_WIDTH-1 downto 0);
    salida: out std_logic_vector (H_WIDTH-1 downto 0));
end component;

TYPE estados IS (reposo,funcion_a1,funcion_a2,funcion_AB,
funcion_c0,funcion_c1,funcion_c2,funcion_c3,funcion_c4,
funcion_C,funcion_d0,funcion_d1,funcion_d2,funcion_d3,
funcion_d4,funcion_D,state_h0,state_h1);

SIGNAL actual,siguiente:estados;
signal sum1,sum2,sum3,sum4,result_h0,result_h1
       : std_logic_vector (H_WIDTH-1 downto 0);
signal count_a             : INTEGER range r1 downto 0;
signal count_d             : INTEGER range nstate downto 0;
begin
--------------------------------------------------------------------
--                                  MAP                           --
--------------------------------------------------------------------
sum_h0 : sumador PORT MAP(sum1,sum2,result_h0);
```

```
(sum_h1 : sumador PORT MAP(sum3,sum4,result_h1);
-------------------------------------------------------------------
--             INICIO: MÁQUINA DE ESTADOS                        --
-------------------------------------------------------------------
process(clk,reset)
    begin
        if reset='1' then
            actual<=reposo;
        elsif clk'event and clk='1' then
            actual<=siguiente;
        end if;
end process;

process(actual,start,a1,h0,a_h0,result_h0,h1,a_h1,result_h1,count_a,
count_d,enable_cont,a_state_0,a_state_1,a_state_2,a_state_3,state_0,
state_1,state_2,state_3,reg1,reg2,a_reg1,a_reg2)

  begin
  sum1 <= (others =>'0');
  sum2 <= (others =>'0');
  sum3 <= (others =>'0');
  sum4 <= (others =>'0');
  a_reg1 <= reg1;
  a_reg2 <= reg2;
  a_state_0 <= state_0;
  a_state_1 <= state_1;
  a_state_2 <= state_2;
  a_state_3 <= state_3;
  a_h0  <= h0;
  a_h1  <= h1;
  enable_cont <= '0';
  enable_cont_d <= '0';
  case actual is
-------------------------------------------------------------------
--             INICIO: ESTADOS FUNCION A Y B                     --
-------------------------------------------------------------------
   when reposo =>
    if start = '1' then       --- SEÑAL DE INICIO DE ALGORITMO
       a_state_0 <= x"a9";
       a_state_1 <= x"ba";
       a_state_2 <= x"c2";
       a_state_3 <= x"84";
       a_h1      <= h0;
       sum1      <= h0;        ---sum_h0 = h0+a1
       sum2      <= a1;
       sum3      <= h0;        ---sum_h1 = h1+a1
       sum4      <= a1;
       a_reg1    <= result_h0;
       a_reg2    <= result_h1;
       siguiente <= funcion_a1;
    else
        siguiente <= reposo;
    end if;

    when funcion_a1 =>
      a_reg1 <= result_h0;
      a_reg2 <= result_h1;
```

```
        --sum_h0 = (h0<<1)+((h0+a1)>>1)--
        sum1 <= (reg1(0)& reg1(H_WIDTH-1 downto 1));
        sum2 <= (h0(H_WIDTH-2 downto 0) & h0(H_WIDTH-1));
        --sum_h1 = (h1<<1)+h1+a1--
        sum3 <= reg2;
        sum4 <= (h1(H_WIDTH-2 downto 0)& h1(H_WIDTH-1));
    siguiente <= funcion_a2;

    when funcion_a2 =>
       a_reg2 <= result_h1;
       sum3   <= reg2;          -- sum_h1 = (h1>>1)+(h1<<1)+h1+a1
       sum4   <= (h1(0) & h1(H_WIDTH-1 downto 1));
    siguiente <= funcion_AB;

    when funcion_AB =>
       a_reg1 <= result_h0;
       a_reg2 <= result_h1;
       enable_cont <= '1';
       a_h0 <= reg1;
       a_h1 <= reg2;
       sum1 <= reg1;
       sum2 <= a1;
       sum3 <= reg2;
       sum4 <= a1;
    if count_a < r1-1 then
        siguiente <= funcion_a1;
    else
        siguiente <= funcion_c0;
    end if;
-------------------------------------------------------------------------
--              FIN: ESTADOS FUNCION A Y B                             --
-------------------------------------------------------------------------
-------------------------------------------------------------------------
--              INICIO: ESTADOS FUNCION C                              --
-------------------------------------------------------------------------
    when funcion_c0 =>    --- (h0+h1)
      a_reg1  <= result_h0;
      sum1 <= h0;
      sum2 <= h1;
    siguiente <= funcion_c1;

    when funcion_c1 =>    --- h0´`= (h0+h1)>>3
      a_h0 <= (reg1(H_WIDTH-6 downto 0) &
      reg1(H_WIDTH-1 downto H_WIDTH-5))xor h0;
    siguiente <= funcion_c2;

    when funcion_c2 =>
      a_reg1  <= result_h0;
      a_reg2  <= result_h1;
      sum1 <= (h0(H_WIDTH-7 downto 0) &
               h0(H_WIDTH-1 downto H_WIDTH-6));  --- (h0>>2)
      sum2 <=  h0;                               ---  h0
      sum3 <= (h0(H_WIDTH-4 downto 0) &
               h0(H_WIDTH-1 downto H_WIDTH-3));  --- (h0<<3)
      sum4 <= (h0(H_WIDTH-2 downto 0) & h0(H_WIDTH-1)); ---(h0<<1)
    siguiente <= funcion_c3;
```

```
    when funcion_c3 =>
       a_reg1 <= result_h0;
       sum1   <= (reg1(1) & reg1(0) &
                  reg1(H_WIDTH-1 downto 2)); --- ((h0>>2)+h0)>>2)
       sum2   <= reg2;                        --- ((h0<<3) + (h0<<1))
    siguiente <= funcion_c4;  ---------------------------------------
                              ---------------------------
    when funcion_c4 =>
       ----((((h0>>2)+h0)>>2) + (h0<<3) + (h0<<1)) xor 0x"76"----
      a_h0 <= reg1 xor x"76";
    siguiente <= funcion_d0;

-----------------------------------------------------------------------
--             INICIO: ESTADOS FUNCION D                             --
-----------------------------------------------------------------------
    when funcion_d0 =>
       a_reg1 <= h0 xor h1; --- (h0´`h1)
    siguiente <= funcion_d1;

    when funcion_d1 =>
       ---- h1´`= (h0´`h1)>>1 ----
       a_h1 <= (reg1(0) & reg1(H_WIDTH-1 downto 1)) xor h1;
    siguiente <= funcion_d2;

    when funcion_d2 =>
       a_reg1  <= result_h0;
       a_reg2  <= result_h1;
       sum1 <= (h1(H_WIDTH-5 downto 0) &
                h1(H_WIDTH-1 downto H_WIDTH-4)); --- (h1>>3)
       sum2 <= (h1(H_WIDTH-6 downto 0) &
                h1(H_WIDTH-1 downto H_WIDTH-5)); --- (h1>>4)
       sum3 <= (h1(H_WIDTH-4 downto 0) &
                h1(H_WIDTH-1 downto H_WIDTH-3)); --- (h1<<3)
       sum4 <= h1;                                 ---  h1
    siguiente <= funcion_d3;

    when funcion_d3 =>
       a_reg1 <= result_h0;
       sum1 <=  reg1;                          --- (h1>>4)+ (h1>>3)
       sum2 <=  reg2;                          --- (h1<<3) + h1
    siguiente <= funcion_d4;

    when funcion_d4 =>
       enable_cont_d <= '1';
       a_h1 <= reg1;    --- resultado h1
    if count_a < r2-1 then
        siguiente <= funcion_c0;
    else
        siguiente <= state_h0;
    end if;

-----------------------------------------------------------------------
--             FIN: ESTADOS FUNCION C Y D                            --
-----------------------------------------------------------------------
-----------------------------------------------------------------------
-- INICIO ESTADO: STATE = STATE + H0  y  STATE = STATE Xor H1    --
```

```
    when state_h0 =>
        if count_d = 1 and count_a = 0 then
          a_reg1  <= result_h0;
          sum1 <= state_0;
          sum2 <= h0;
        elsif count_d = 2 and count_a = 0 then
          a_reg1  <= result_h0;
          sum1 <= state_1;
          sum2 <= h0;
        elsif count_d = 3 and count_a = 0 then
          a_reg1  <= result_h0;
          sum1 <= state_2;
          sum2 <= h0;
        elsif count_d = 4 and count_a = 0 then
          a_reg1  <= result_h0;
          sum1 <= state_3;
          sum2 <= h0;
        end if;
    siguiente <= state_h1;

    when state_h1 =>
        if count_d = 1 and count_a = 0 then
          a_state_0 <=  reg1 xor h1;
        elsif count_d = 2 and count_a = 0 then
          a_state_1 <=  reg1 xor h1;
        elsif count_d = 3 and count_a = 0 then
          a_state_2 <=  reg1 xor h1;
        elsif count_d = 4 and count_a = 0 then
          a_state_3 <=  reg1 xor h1;
        end if;
      if count_d =4 and count_a = 0 then
        a_reg1  <= result_h0;
        siguiente <= reposo;
        sum1 <= h1;
        sum2 <= h0;
        a_h0 <= result_h0;
      else
        siguiente <= funcion_c0;
      end if;


    when others =>

end case;
end process;

state <= state_0 & state_1 & state_2 & state_3;
-------------------------------------------------------------------
--             REGISTROS                                         --
-------------------------------------------------------------------

process(clk,reset)
  begin
    if reset='1'then
      h0       <= x"76";
      h1       <= (others =>'0');
      state_0  <= (others =>'0');
```

```
      state_1  <= (others =>'0');
      state_2  <= (others =>'0');
      state_3  <= (others =>'0');
      reg1     <= (others =>'0');
      reg2     <= (others =>'0');
    elsif clk' event and clk='1' then
      state_0 <= a_state_0;
      state_1 <= a_state_1;
      state_2 <= a_state_2;
      state_3 <= a_state_3;
      h0      <= a_h0;
      h1      <= a_h1;
      reg1    <= a_reg1;
      reg2    <= a_reg2;
    end if;
  end process;


-------------------------------------------------------------------------
-- CONTADOR FUNCIÓN A y B                                               --
-------------------------------------------------------------------------

process(clk,reset)
  begin
    if reset='1' then
      count_a <= 0;
    elsif clk'event and clk='1' then
        if enable_cont ='1' then
            if count_a < r1-1 then
              count_a <= count_a + 1;
            else
              count_a <= 0;
            end if;
        elsif enable_cont_d ='1' then
            if count_a < r2-1 then
              count_a <= count_a + 1;
            else
              count_a <= 0;
            end if;
        end if;
    end if;
end process;

-------------------------------------------------------------------------
-- CONTADOR FUNCIÓN C y D                                               --
-------------------------------------------------------------------------
process(clk,reset)
  begin
    if (reset='1')  then
      count_d <= 0;
    elsif clk'event and clk='1' then
       if enable_cont_d = '1' then
          if count_d < nstate  then
              if count_a = r2-1 then
                 count_d <=  count_d + 1;
              else
                 count_d <=  count_d;
```

```
              end if;


          end if;
        end if;
    end if;
 end process;
end arq_tav_128_c;
```

CÓDIGO VHDL

SUMADOR

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

ENTITY sumador IS
    GENERIC (H_WIDTH: natural:= 8);

Port(
        termino1 :in std_logic_vector (H_WIDTH-1 downto 0);
        termino2 :in std_logic_vector (H_WIDTH-1 downto 0);
        salida: out std_logic_vector (H_WIDTH-1 downto 0)
);
END sumador;
ARCHITECTURE f OF sumador is
    begin
  process(termino1,termino2)
        begin
                salida<=termino1 + termino2;
  end process;
end f;
```

CÓDIGO VHDL

TEST-BENCH

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


entity tav_128_tb is
end entity;

architecture arq_tav_128_tb of tav_128_tb is

constant r1            : natural := 32;
constant r2            : natural := 8;
constant nstate        : natural := 4;
constant H_WIDTH       : natural := 8;
constant WIDTH_state   : natural := 32;

component tav_128
      generic(
          r1          : natural:= 32;
          r2          : natural:= 8;
          nstate      : natural:= 4;
          H_WIDTH     : natural:= 8;
          WIDTH_state : natural:= 32
      );

      port(
          clk     : in std_logic;
          rst     : in std_logic;
          start   : in std_logic;
          a1      : in std_logic_vector (H_WIDTH-1 downto 0):= x"12";
          state   : out std_logic_vector (WIDTH_state-1 downto 0)
      );
    end component;

signal clk     : std_logic:= '1';
signal rst     : std_logic;
signal a1      : std_logic_vector (H_WIDTH-1 downto 0):= x"12";
signal h0      : std_logic_vector (H_WIDTH-1 downto 0);
signal state   : std_logic_vector (WIDTH_state-1 downto 0);
signal start   : std_logic;

begin

rst      <= '1', '0' after 20 us;
clk      <= not clk after 5 us;


process
    begin
        start <='0';
        wait for 30 us;
        start <= '1';
        wait for 10 us;
        start <= '0';
        wait for 50 ms;
end process;
```