IMPLEMENTACIÓN SOBRE FPGA DE UN ALGORITMO DE INTERCAMBIO DE LLAVE SIMÉTRICA BASADO EN REDES NEURONALES

JAIRO ALBERTO BAUTISTA MARTÍNEZ

UNIVERSIDAD INDUSTRIAL DE SANTANDER

FACULTAD DE INGENIERÍAS FÍSICO MECÁNICAS

ESCUELA DE INGENIERÍA ELÉCTRICA, ELECTRÓNICA Y DE TELECOMUNICACIONES

BUCARAMANGA

2015

IMPLEMENTACIÓN SOBRE FPGA DE UN ALGORITMO DE INTERCAMBIO DE LLAVE SIMÉTRICA BASADO EN REDES NEURONALES

JAIRO ALBERTO BAUTISTA MARTÍNEZ

Trabajo de Grado para optar al título de

Ingeniero Electrónico

Director

Óscar Mauricio Reyes Torres

Ingeniero Electrónico, Ph.D.

Co-Director

HÉCTOR IVÁN GÓMEZ ORTIZ

Ingeniero Electrónico, M.Sc.

UNIVERSIDAD INDUSTRIAL DE SANTANDER

FACULTAD DE INGENIERÍAS FÍSICO MECÁNICAS

ESCUELA DE INGENIERÍA ELÉCTRICA, ELECTRÓNICA Y DE TELECOMUNICACIONES

BUCARAMANGA

2015

AGRADECIMIENTOS

El autor expresa sus agradecimientos a:

A mis padres por darme la vida y apoyarme en todo momento de mi formación profesional.

Al Doctor Óscar Reyes y el codirector Héctor Gómez por su invaluable guía y la confianza depositada en mí.

A los profesores William Salamanca, Carlos Angulo y Sergio Abreo del grupo de investigación CPS por sus enseñanzas en el campo de los sistemas digitales.

A Jesús Sarmiento e Hilda Velásquez por sus consejos y colaboración que hicieron posible este logro.

A la Universidad Industrial de Santander por brindarme una formación profesional.

DEDICATORIA

A mis padres Carmen Martínez y Jairo Bautista por su amor incondicional, consuelo y ánimo en los momentos que más lo necesitaba.

A mi compañera de vida Livy Colmenares por ser mi soporte y compartir mis sueños.

A mis amigos Jesús Sarmiento, Fabio Hernández, José Caballero y Julio Flórez por ser modelo y apoyo en mi carrera profesional.

A mis amigos y compañeros que fueron los que hicieron agradable este camino.

Jairo Alberto Bautista Martínez

CONTENIDO

Pág.

LISTA DE FIGURAS

Pág.

LISTA DE TABLAS

Pág.

LISTA DE ANEXOS

Pág.

LISTA DE ABREVIATURAS

ANN	Artificial neuronal network
ASIC	Application specific integrated circuit
BEL	Basic element of logic
BRAM	Block of random access memory
CLB	Configurable logic block
D-FF	Data-type flip flop
FF	Flip flop
FPGA	Field programmable gate array
FSM	Finite state machine
ISE	Integrated software environment
LC	Logic cell
LFSR	Linear feedback shift register
LUT	Look-Up table
MTU	Maximum transfer unit
PPM	Permutation parity machine
PRBS	Pseudo random binary sequence
RAM	Random access memory
TPM	Tree parity machine
VHDL	VHSIC hardware description language
VHSIC	Very high speed integrated circuit

RESUMEN

TÍTULO: IMPLEMENTACIÓN SOBRE FPGA DE UN ALGORITMO DE INTERCAMBIO DE LLAVE SIMÉTRICA BASADO EN REDES NEURONALES[1]

AUTOR: JAIRO ALBERTO BAUTISTA MARTÍNEZ[2]

PALABRAS CLAVE: Redes neuronales artificiales, máquina de árbol de paridad, FPGA, neurocriptografía, llave simétrica, sincronización mutua.

DESCRIPCIÓN: En la actualidad el diseño de circuitos electrónicos se está enfocando en nuevas ciencias de la ingeniería como las basadas en la inteligencia artificial, en particular, el campo dedicado a las redes neuronales artificiales, el cual ha dado una alternativa para el análisis y modelado de sistemas complejos. Este trabajo aborda la aplicación en criptografía simétrica de una topología de red neuronal conocida como máquina de paridad de árbol y su capacidad de sincronización mutua. Este proceso permite a dos dispositivos basados en redes neuronales alinear sus pesos sinápticos que pueden ser usados como un proceso para establecer una llave privada.

En este trabajo, se revisan los principales antecedentes de neurocriptografía, y las características del proceso de sincronización mutua de las máquinas de paridad de árbol para el establecimiento de llave simétrica en aplicaciones criptográficas de peso ligero. Se hace un análisis de la influencia de los parámetros de red en el proceso de sincronización. Se discuten las características principales a considerar para implementaciones por transferencia bit a bit y por paquetes de bits.

Finalmente, se describe mediante VHDL el diseño de un circuito digital que emula el sistema de establecimiento de llave entre dos máquinas de paridad de árbol. Se verifica el comportamiento la implementación funcional sobre una FPGA. A partir de los resultados, se derivan los parámetros de rendimiento tanto en tiempo como en recursos, con el fin de concluir acerca de las propiedades de una implementación práctica de los sistemas criptográficos basados en máquinas de árbol de paridad.

[1] Trabajo de grado.

[2] Facultad de ingenierías Físico Mecánicas. Escuela de Ingeniería Eléctrica, Electrónica y de Telecomunicaciones. Director Dr.–Ing. Óscar Mauricio Reyes Torres. Codirector M. Sc. Héctor Iván Gómez Ortiz.

SUMMARY

TITLE: IMPLEMENTATION ON FPGA OF A SYMMETRICAL KEY ESTABLISHED ALGORITHM BASED IN NEURAL NETWORKS[3]

AUTHOR: JAIRO ALBERTO BAUTISTA MARTÍNEZ[4]

KEY WORDS: Artificial neural networks, tree parity machine, FPGA, neurocryptography, symmetrical key, mutual synchronization.

DESCRIPTION: Nowadays the design of electronic circuits focuses in new sciences of engineering as those based on artificial intelligence, in particular, the field dedicated to artificial neural networks, which provides an alternative to analyze and model complex systems. This work is about the application in symmetrical cryptography of a neural network topology known as tree parity machine and its capability of mutual synchronization. This process allows two devices based in neural networks to align their synaptic weights which can be used as a process to establish a private key.

In this work, it is reviewed the main background of neural cryptography, and the characteristics of the mutual synchronization process of tree parity machines for a key establishment in light weight cryptography applications. The influence of network parameters on the synchronization process is analyzed. Key features to consider for bit-by-bit or bit packaging implementations are discussed.

Finally, the design of a digital circuit that emulates the establishment of key system between two tree parity machines is described in VHDL. The behavior of a functional implementation is verified on a FPGA. From the results, performance parameters both in time and resources are derived, in order to conclude about the properties of a practical implementation of encryption systems based on tree parity machines.

[3] Degree Project.

[4] Physics Mechanical Engineering Faculty. Electrical, Electronic and Telecommunicatios Engineering School. Advisor Dr.-Ing. Óscar Mauricio Reyes Torres. Co-advisor Héctor Iván Gómez Ortiz.

INTRODUCCIÓN

Actualmente la protección de la información que circula a través de los sistemas informáticos de entidades bancarias, centros de investigación, corporaciones comerciales, entidades gubernamentales, entre otros; ha sido prioridad en las investigaciones hechas por la Ingeniería Electrónica y principalmente en el campo de investigación como la seguridad informática, que busca brindar apoyo para hacer confiables los medios de comunicación entre usuarios y proveedores de servicios [1]. Nuevas ciencias como la criptografía proporcionan herramientas para cifrar información importante y sensible, por lo cual validar la autenticidad de una conexión a través de un medio vulnerable a un ataque externo se torna un objetivo importante, debido al crecimiento del número de ataques de fuerza bruta por parte de desarrolladores de programas malintencionados.

En sistemas de comunicación donde se comparten información privada la criptografía provee herramientas para cifrar los datos a transmitir y así proteger de algún modo que un ente no autorizado intervenga la comunicación y se apodere de información importante para el usuario [2]. Los archivos digitales intercambiados representan una señal, texto o imagen que se encriptan para que bajo las normas estándar de interpretación no pueda ser leído sin la llave con la cual se cifró. Esta llave sirve como un filtro por el que pasa el archivo que va a ser compartido por un canal que es potencialmente sensible a intervención y alteración; al llegar al destino el receptor usa la misma llave para hacer la operación inversa hecha en el transmisor y hacer inteligible el archivo original y así poder ser procesado. Al revisar las características de este proceso se deriva la cuestión de cómo compartir también la llave de cifrado de forma que sólo sea

conocida por el transmisor y el receptor, ya que en ella reside la seguridad del método anteriormente descrito.

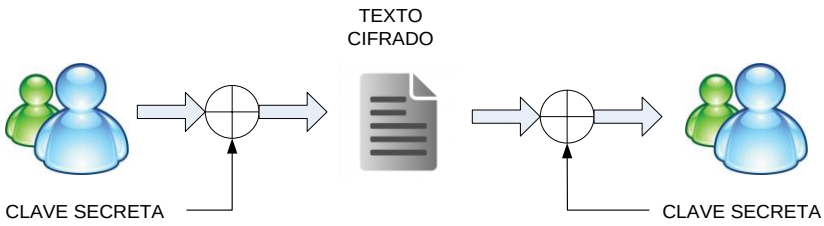
En el entrenamiento de redes neuronales artificiales se ha evidenciado el fenómeno de sincronización mutua [4], donde dos redes conectadas entre sí aprenden entre ellas sin la necesidad de supervisión externa y de forma más rápida que en el entrenamiento mediante ejemplos, por lo cual este proceso puede ser empleado en el protocolo de establecimiento de llave simétrica. El objetivo de este trabajo es comparar implementaciones realizadas sobre FPGA con los estudios más significativos en el uso de las redes neuronales artificiales y su proceso de sincronización mutua para el establecimiento de llave simétrica [11]. Para ello, en el primer capítulo se pone al lector en contexto acerca los métodos criptográficos para el establecimiento de llave, se resumen las principales características del proceso de sincronización mutua de las máquinas de paridad de árbol descritas en los antecedentes publicados acerca de neurocriptografía y se muestra de forma general la estructura interna de las FPGAs, circuitos digitales usados en campo. En el siguiente capítulo se estudia el proceso de sincronización mutua a cambios en los parámetros internos de las redes y el impacto de la elección de la regla de aprendizaje para llegar al estado paralelo de las redes. Luego se muestra el diseño de un procesador digital de propósito específico para realizar el establecimiento de los pesos sinápticos de dos redes comunicadas entre si y las propiedades de la implementación por transferencia de bit a bit o por la variante de paquetes de bits de salida. En el último capítulo se analizan los resultados de las implementaciones realizadas sobre FPGA y se comparan los indicadores de tiempos y recursos de las fuentes sintetizadas con intercambios entre bits y entre paquetes, para evaluar el costo computacional de usar una arquitectura respecto a la otra. Se finaliza este trabajo con las conclusiones derivadas del análisis de los resultados obtenidos de los procesadores y se dan algunas recomendaciones para trabajos futuros.

1. MARCO CONCEPTUAL

1.1. ESTABLECIMIENTO DE LLAVE SIMÉTRICA

En el área de la seguridad informática se definen los protocolos para los procesos de autenticación, establecimiento de sesión o cifrado de información como la criptografía asimétrica y la criptografía simétrica [2]. La criptografía asimétrica utiliza una llave pública y una privada, las dos relacionadas entre sí, con las cuales se puede determinar si el mensaje ha sido modificado durante su transmisión, si el destinatario es el original y con las que se hace el encriptado. En la criptografía simétrica solo es requerida una sola llave de tipo privado, con la cual se realiza el cifrado de la información a proteger, como se ver en la Figura 1 ambos agentes deben conocer la clave secreta para poder interpretar la información cifrada.

Figura 1. Sistema de encriptación de texto utilizando llave simétrica



Al igual que la protección de la información a compartir es importante, de la criptografía simétrica se deriva el problema de cómo compartir también la llave secreta en un canal que puede ser interferido, para esto se define el protocolo de establecimiento de llave con el cual se determina el método de intercambio de

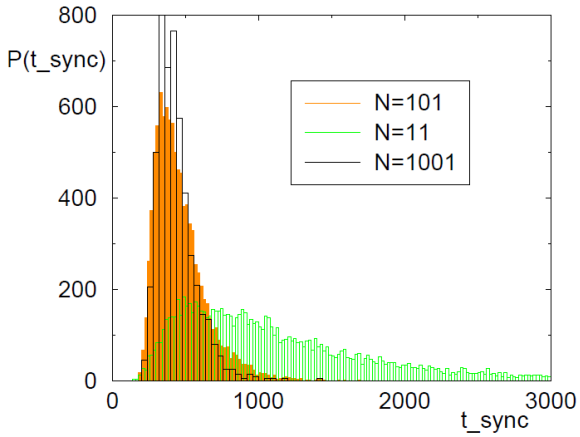
dicha clave. Los sistemas basados en criptografía asimétrica son más robustos que los basados en criptografía simétrica, pero implican también una complejidad en los algoritmos que los ejecutan. Para los sistemas donde se dispone de una cantidad limitada de recursos se usa la criptografía ligera, la cual busca llevar algoritmos de poca complejidad como los usados en la criptografía simétrica a aplicaciones con la misma funcionalidad de la criptografía robusta. Para el establecimiento de llave simétrica se pueden emplear algoritmos de permutación para establecer esta clave secreta y única, pero es posible el uso de las redes neuronales artifíciales, que ofrecen estructuras sencillas cuya implementación en circuitos electrónicos son de poca carga para el sistema.

1.2. ANTECEDENTES DE LA NEUROCRIPTOGRAFÍA

Este trabajo de investigación está basado en las publicaciones sobre seguridad informática en el intercambio de información usando redes neuronales [7] [8] [9] [10] hechas por Markus Volkmer y Sebastian Wallner de la Universidad de Bielefeld de Alemania; el trabajo previo sobre implementación de redes neuronales para el establecimiento de llave escrito por Ido Kanter, Eran Kanter, y Wolfgang Kinzel [4] y en el trabajo de Doctorado de Andreas Ruttor en donde se estudia la dinámica de sincronización de las máquinas de paridad de árbol [11]. El interés del grupo de investigación CPS[5] de la Universidad Industrial de Santander y de este trabajo de pregrado es mostrar estos resultados y comparar estas arquitecturas en sistemas de desarrollo de procesadores de propósito específico sobre FPGA en la línea de sistemas digitales.

[5] Grupo de investigación en Conectividad y Procesado de Señales de la Universidad Industrial de Santander.

Figura 2. Distribución de probabilidad de sincronización contra tiempos de sincronización

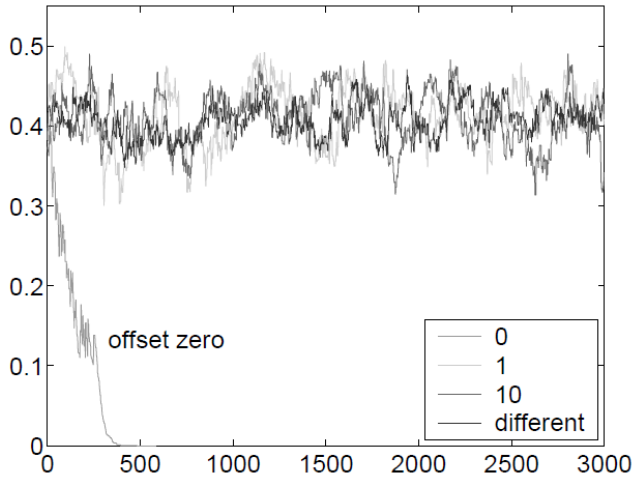


Fuente: Tomado de KANTER I., KANTER E., KINZEL W., "*Secure Exchange of information by synchronitation of neuronal networks"*. Europhys, 2002. p. 6.

El primer artículo del cual está basado este trabajo y es referencia para las publicaciones siguientes es el documento de Kinzel, Kanter y Kanter [4] acerca del uso del proceso de sincronización mutua, propio de las máquinas de árbol de paridad para el establecimiento de llave y hacer cifrado de información. La idea es aprovechar el fenómeno del aprendizaje de las redes *Tree Parity Machine* para que el sistema digital establezca una llave junto con su contraparte sin la necesidad de supervisión y hacer el intercambio de información de forma segura sin llegar a compartir la llave del sistema. Para realizar este estudio se usó programas de cómputo y métodos numéricos para caracterizar las propiedades más relevantes de las redes como algoritmo de intercambio de llave. En sus pruebas mostradas en la Figura 2 se desplazan los pesos al estado anti paralelo $(w^A = -w^B)$ con parámetros fijos de red de $K = 3$ y $L = 3$, y para variaciones en las neuronas de entradas de $N = \{11, 101, 1001\}$, obteniendo un tiempo de sincronización promedio de 410 iteraciones donde hubo actualización de los pesos sinápticos en ambas redes neuronales.

También discuten el caso donde un intruso que interfiere en el canal (con otra red igual no autorizada) busca imitar el proceso de intercambio para conocer la llave privada, se expone entonces que éste no puede conocer el estado de los campos de las neuronas intermedias (y_j) cuyos valores están ocultos y con los cuales se realiza la actualización de pesos en las redes según regla de aprendizaje, por ende esta red intrusa no puede seguir la actualización de sus campos internos correctamente, ya que existen diversas combinaciones dependiendo del valor de K para la misma salida de la red; además, si no logra la sincronización antes que las redes autorizadas lo hagan, la salida de cualquiera de ellas no brindara información útil para el atacante; lo cual hace esta propuesta como un método seguro de intercambio. Además hace introducción al método de variante de paquete de bits como protocolo de comunicación de b salidas por transferencia, como alternativa de red para aprovechar las tramas transmitidas por ciertas comunicaciones dentro de un canal.

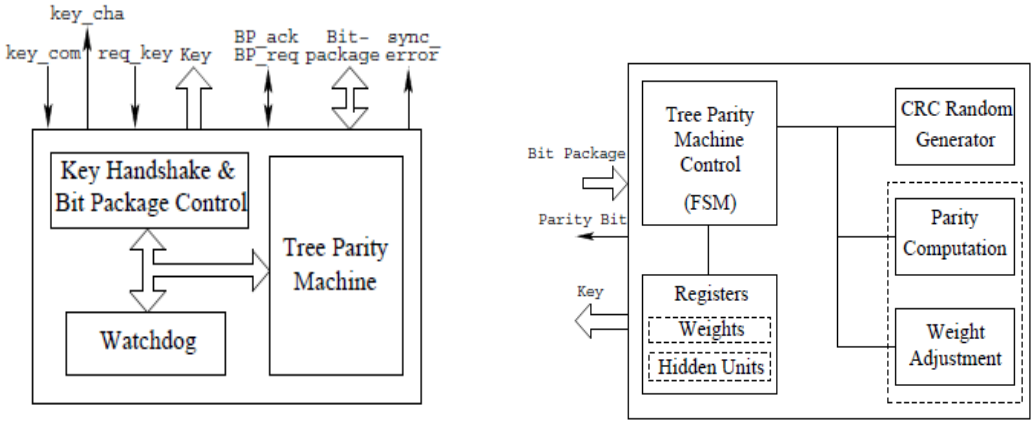
Figura 3. Distancia de los pesos contra cantidad de iteraciones



Fuente: Tomado de VOLKMER M., SCHAUMBURG A., *Authenticated tree parity machine key exchange*. Europhysics Letters, 2004.p. 4.

En los artículos [6] [7] [8] [9] [10] muestran los parámetros internos de las topologías TPM y las reglas de aprendizaje asociadas al proceso de sincronización mutua, propio de este tipo de redes. En la Figura 3 se verifica que a partir de entradas iguales la distancia entre los pesos w^A y w^B tiende a cero en un tiempo finito alrededor de 400 pasos de aprendizaje [11], por lo cual en ambas redes existe la probabilidad de sincronización si se entrenan con un vector igual que alimentan las neuronas de entrada en ambas redes, de otro modo no puede alcanzarse la sincronización. Además se propone el protocolo de autenticación de cero conocimiento (*Zero-KnowIedge Protocol - ZKP*) al proceso para aumentar la seguridad de la comunicación y finalizan reafirmando que un atacante que no logra sincronizar su red antes que los agentes autorizados lo hagan, no podrá aprender de la información compartida en el canal de comunicación.

Figura 4. Arquitectura interna de la implementación de una TPM serial



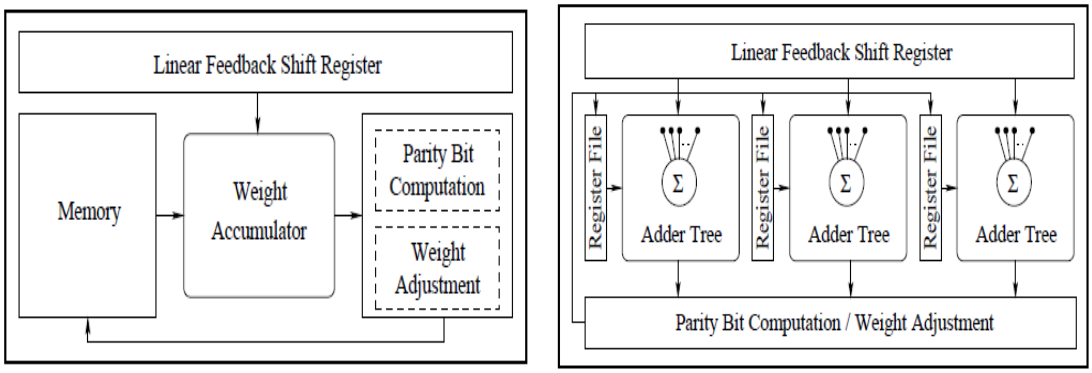
Fuente: Tomado de VOLKMER M, WALLNER S., *Tree parity machine rekeying architectures*. IEEE Transactions on Computers, 2004.p.11-12.

En el documento [7] los autores discuten la seguridad del método de sincronización mutua de las TPM para el establecimiento de llave, mostrando que la probabilidad de éxito de un atacante externo mediante métodos como ataques

genéticos, o mayoritarios disminuye exponencialmente conforme aumenta el parámetro L de la red. También muestran la arquitectura de las red TPM para establecimiento de llave y la variante de paquetes de bits para realizar una implementación funcional de una red en un sistema basado en ASIC con parámetros $K = 3,\ L = 4$ y $N = 49$. En la Figura 4 se ilustra la composición interna de una red funcional mediante diagrama de bloques y las señales de los componentes para una implementación, donde el bloque TPM controla mediante una máquina de estados los registros, el generador de números pseudo aleatorios y el ajuste de los pesos, para ser enviados a otra red mediante el bloque de control de paquetes y acuerdo de saludo (*Key handshake and bit package control)* bajo un protocolo de petición y acuerdo (*Request/acknowledge protocol*). Finalizan analizando arquitecturas de tipo serial y semiparalela implementadas en dispositivos ASIC comparando el área y la velocidad de intercambio contra el tamaño del paquete en la variante y concluyen que si se busca una implementación en hardware donde se dispone de un área muy pequeña y tiempos de sincronización de milisegundos, las TPM son una opción eficiente para el uso en criptografía simétrica.

En el documento [8] como en los anteriores, tratan la seguridad de la redes TPM para el establecimiento de llave en dispositivos embebidos y además muestra la arquitectura serial y semiparalela (Figura 5) de una TPM para su implementación con parámetros neuronales de $K = 3,\ L = 4$ y $N = \{11, 88\}$; al analizar las dos arquitecturas de redes la frecuencia de intercambio de llave promedio es mayor para arquitectura semiparalela que para la serial pero para tamaños de paquetes de $b > 200$ se vuelve menor para arquitecturas de tipo serial; por lo tanto, basar el sistema en una arquitectura semiparalela implica un incremento en el área requerida que favorece la velocidad del proceso (uso en ciclos de operación) para paquetes pequeños.

Figura 5. Arquitectura de TPM de tipo serial y de tipo semiparalela



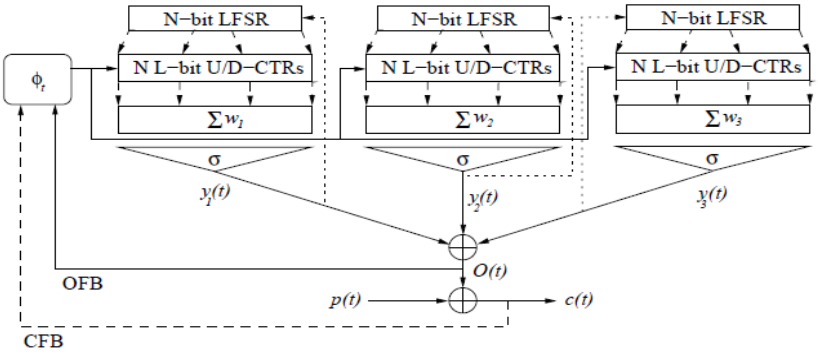
Fuente: Tomado de VOLKMER M, WALLNER S., *Tree parity machine rekeying architectures for embedded security*, 2005. p.7-8.

En la publicación [9] se exponen las redes TPM y su aplicación como un núcleo para comunicaciones seguras, repiten las pruebas realizadas en otros documentos con parámetros neuronales de $K = 3$, $L = 4$ y $N = 49$ obteniendo tiempos de sincronización de 400 iteraciones. Exponen la variante de bits como solución en dispositivos donde se dispone de un gran ancho de banda. En el documento [10] usan las redes neuronales TPM para realizar cifrado de archivos mediante la llave obtenida del proceso de sincronización mutuo y establecimiento de llave para procesos de comunicación inalámbrica. También incluye la variante de bits y la unidad serial para la implementación de la TPM con parámetros de red $K = 3$, $L = 4$ y $N = 88$ y publica los resultados obtenidos para los protocolos de comunicación RFID y NFC[6]. Se incluye brevemente el efecto de múltiples redes sincronizándose y como la profundidad sináptica L influye en la seguridad ante atacantes externos. Además muestra el uso de la llave establecida para cifrar texto mediante la salida CFB. En el esquema de la red en la Figura 6 propone un generador de números pseudo aleatorios de un registro realimentado de desplazamiento (LFSR) y

[6] Identificación de objetos mediante ondas de radio para las comunicaciones de Identificación por Radiofrecuencia (Radio Frequency IDentification – RFID) y Comunicación de Campo Cercano (Near Field Communication – NFC).

contadores ascendentes y descendentes (U/D CTRs) para la actualización de los pesos almacenados, ya que cada vez que se hace una actualización de los pesos sinápticos por la regla de aprendizaje, los pesos sólo aumentan o disminuyen su valor en la unidad.

Figura 6. Configuración de una TPM con salida para cifrar texto



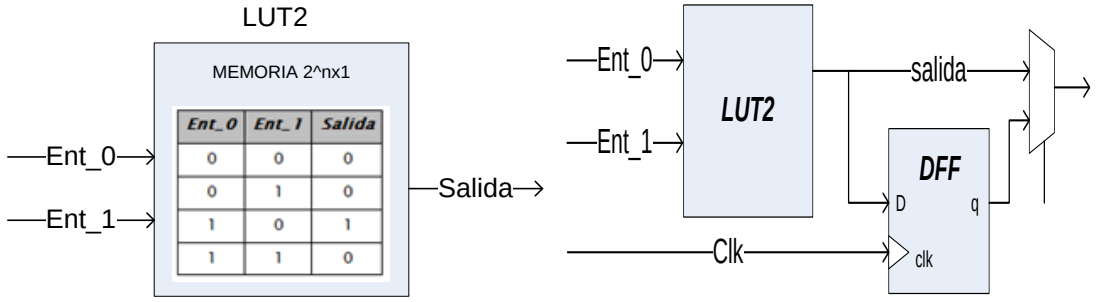
Fuente. Tomado de VOLKMER M, WALLNER S., *Lightweight key exchange and stream cipher based solely on tree parity machine,* 2005. p.7.

En la arquitectura semiparalela se incrementa la cantidad de módulos dependiendo de la cantidad de neuronas ocultas K a fin de aumentar la velocidad de procesamiento de la red y en la arquitectura serial existe un único bloque que contiene toda la información de la red, por lo cual el procesador debe analizar capa intermedia por intervalos regulares de NK, esto resulta en una disminución en el tiempo de operación para que una iteración pueda ser culminada. El procesador de la red TPM descrito en este trabajo es de tipo serial, esto para obtener una implementación pequeña desde el punto de vista de los recursos disponibles en la FPGA siendo referente base para algún trabajo posterior de una arquitectura paralela.

1.3. RECURSOS DE LOS DISPOSITIVOS FPGA

Un dispositivo FPGA es un circuito integrado que posee un arreglo bidimensional de recursos lógicos y de rutas de interconexiones configurables por el usuario. Estos elementos se utilizan para hacer implementaciones digitales en el campo donde se encuentra el proceso y donde el diseño puede ser verificado en la marcha. El bloque básico con el cual una FPGA está diseñada se denomina Tabla de búsqueda (*Look-Up Table* – LUT) y consiste en una memoria de n entradas, en la cual se configura la tabla de verdad de cualquier circuito combinacional [12] Dependiendo de la familia y la tecnología la FPGA puede contener LUTs de diferentes estructuras según la cantidad de entradas y salidas que ésta disponga. Como se ilustra en la Figura 7 éstas unidades se vinculan con un FF y un multiplexor formando un equivalente al circuito secuencial básico de la FPGA denominado celdas lógica (LC). Estas celdas pueden seguir agrupando para formar bloques aún más grandes y versátiles con funciones especializadas.

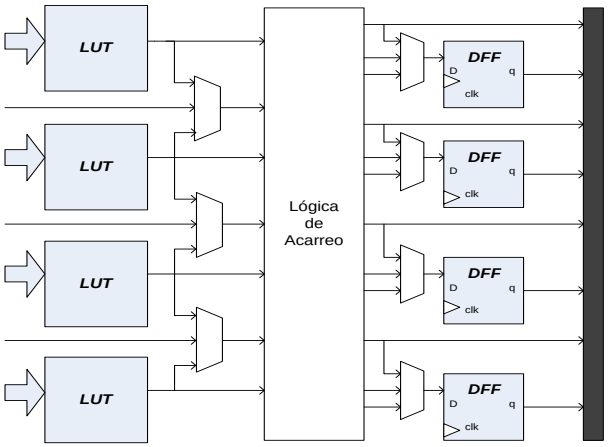
Figura 7. Unidades básicas combinacional y secuencial



Siguiendo en grado de complejidad, una agrupación de LCs forman los bloques llamados *slices* de la Figura 8, compuestos generalmente por cuatro LUTs, cuatro FFs y algunos bloques lógicos adicionales como multiplexores y módulos de

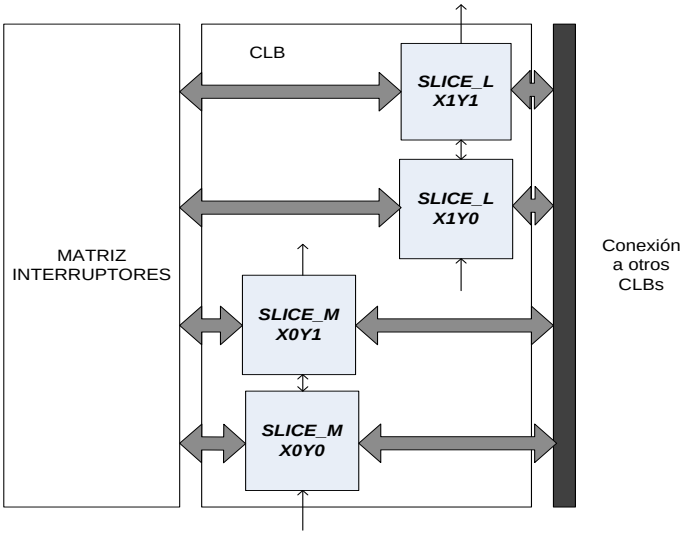
acarreos [13]. Estos bloques junto con las LUTs están descritos principalmente en las hojas de datos de los circuitos integrados como información útil para que el desarrollador conozca la capacidad de cada familia de FPGA. De estos bloques hay diferentes tipos según cada familia, por ejemplo: el *SliceM,* el *SliceL* y el *SliceX*. El *sliceM* es el *slice* convencional de propósito general. El *sliceL* difiere del *sliceM* en su funcionalidad, dado que con el *sliceL* no se implementan memoria distribuida o registros de desplazamiento. El *sliceX* posee la misma estructura del *sliceL* sin los módulos de acarreo para realizar operaciones aritméticas.

Figura 8. Esquema interno de un *Slice* de cuatro LUTs



A un nivel superior de integración de los elementos anteriormente descritos está el bloque lógico combinacional (CLB) de la Figura 9, que integra para familia 6 de Xilinx, cuatro *slices* (dos *slicesM* y dos *slicesL*) y una matriz de interruptores para hacer interconexión con otros CLBs. La FPGA es un circuito integrado con un arreglo bidimensional de estos elementos, alrededor del *chip* se encuentran los buffers de entradas y de salidas para la comunicación externa, dentro del mismo también se disponen en filas y columnas los bloques CLBs, donde el programa habilita y conecta cada entidad para implementar el circuito descrito por el usuario.

Figura 9. Configuración interna de CLB



El fabricante de FPGAs hace una descripción en la hoja de datos para que el usuario conozca la cantidad de LCs, LUTs, *slices* y CLBs disponibles en el circuito integrado, además de módulos avanzados de propósito específico [7] como *buffers* de entrada y salida, multiplicadores, RAM de tipo estática y gestor de reloj digital (DCM). En la Tabla 1 se muestra una hoja de datos con los recursos disponibles para cada circuito integrado FPGA de la familia 3.

Cada familia de FPGA que sale al mercado viene dada con unas características según la tecnología en desarrollo para la fecha. El fabricante ofrece una variedad de modelos de FPGA con diferentes tipos de recursos disponibles, como vemos al comparar la Tabla 1 y la Tabla 2, las LUTs de la *Spartan*® familia 6 son LUTs de sólo de 6 entradas configurables a estructuras de $6x1$ y $5x2$ entradas por salidas, mientras que la familia 3 dispone de LUTs de una, dos, tres y hasta cuatro entradas por salida. Otra diferencia en las hojas de datos es que la familia 6 incorpora el *sliceX* que no está integrado en la familia 3; y que en esta familia la constitución del *slice* es generalmente por dos LUTs, dos RAM, o dos registros en

vez de cuatro como lo hace en la familia 6. Por lo dicho anteriormente, no es posible hacer una comparación como iguales de cada unidad que posee la FPGA a diferentes familias o marcas de fabricantes.

Tabla 1. Recursos de la FPGA *Spartan* Familia 3 de Xilinx®

Dispositivo	LCs	CLBs	Slices	LUTs	RAM
Familia *Spartan*® 3A					
XC3SD1800A	37.440	4.160	16.640	33.280	266.240
XC3SD3400A	53.712	5.968	23.872	47.744	381.952
Familia *Spartan*® 3A/3AN					
XC3S50A/AN	1.584	176	704	1.408	11.264
XC3S200A/AN	4.032	448	1.792	3.584	28.672
XC3S400A/AN	8.064	896	3.584	7.168	57.344
XC3S700A/AN	13.248	1.472	5.888	11.776	94.208
XC3S1400A/AN	25.344	2.816	11.264	22.528	180.224
Familia *Spartan*® 3E					
XC3S100E	2.160	240	960	1.920	15.360
XC3S250E	5.508	612	2.448	4.896	39.168
XC3S500E	10.476	1.164	4.656	9.312	74.496
XC3S1200E	19.512	2.168	8.672	17.344	138.752
XC3S1600E	33.192	3.688	14.752	29.504	236.032
Familia *Spartan*® 3					
XC3S50	1.728	192	768	1.536	12.288
XC3S200	4.320	480	1.920	3.840	30.720
XC3S400	8.064	896	3.584	7.168	57.344
XC3S1000	17.280	1.920	7.680	15.360	122.880
XC3S1500	29.952	3.328	13.312	26.624	212.992
XC3S2000	46.080	5.120	20.480	40.960	327.680
XC3S4000	62.208	6.912	27.648	55.296	442.368
XC3S5000	74.880	8.320	33.280	66.560	532.480

El procesador de la red TPM se implementa en la FPGA XC3S700A/AN de la familia *Spartan*® 3AN. Al sintetizar una descripción en VHDL en este tipo de circuito integrado el archivo resultado hecho por el programa es de extensión .BIT, el cual contiene la información del trazado de las conexiones requeridas para

hacer el diseño descrito en VHDL con los recursos disponibles. El reporte final de la descripción se da en cantidad de recursos usados correspondientes a la FPGA en la cual se sintetizo el diseño.

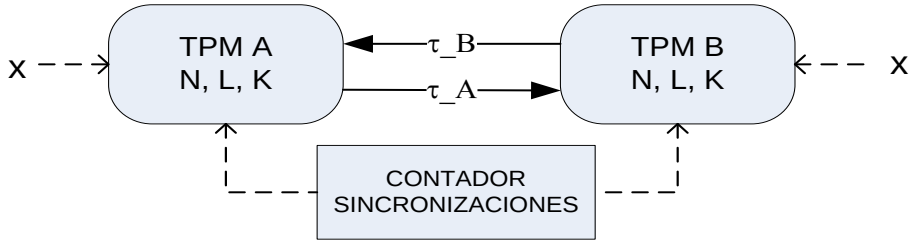
Tabla 2. Recursos del FPGA *Spartan* Familia 6 de Xilinx®

Dispositivo	LCs	SliceMs	SliceLs	SliceXs	LUT_6s	RAM
Familia *Spartan*® 6 LX						
XC6SLX4	3.840	300	0	300	2.400	75.000
XC6SLX9	9.152	360	355	715	5.720	90.000
XC6SLX16	14.579	544	595	1.139	9.112	136.000
XC6SLX25	24.051	916	963	1.879	15.032	229.000
XC6SLX45	43.661	1.602	1.809	3.411	27.288	401.000
XC6SLX75	74.637	2.768	3.063	5.831	46.648	692.000
XC6SLX100	101.261	3.904	4.007	7.911	63.288	976.000
XC6SLX150	147.443	5.420	6.099	11.519	92.152	1.355.000
Familia *Spartan*® 6 LXT						
XC6SLX25T	24.051	916	963	1.879	15.032	229.000
XC6SLX45T	43.661	1.602	1.809	3.411	27.288	401.000
XC6SLX75T	74.637	2.768	3.063	5.831	46.648	692.000
XC6SLX100T	101.261	3.904	4.007	7.911	63.288	976.000
XC6SLX150T	147.443	5.420	6.099	11.519	92.152	1.355.000

2. PROCESO DE SINCRONIZACIÓN MUTUA

Las TPM son ANN multicapa de flujo de datos hacia adelante cuyo entrenamiento se puede realizar en conjunto a otra red (ver Figura 10) en el proceso denominado "sincronización mutua". Por este proceso las redes intercambian salidas y si se cumple una condición se aplica una regla de aprendizaje establecida para así actualizar los valores de los pesos sinápticos internos. Esta actualización provoca que los pesos se desplacen en un intervalo acotado, hasta que en un tiempo finito pero indeterminado se logra que las redes estén sincronizadas. Esta característica se aprovecha para el establecimiento de llave en el campo de la neurocriptografía aportando grandes ventajas al proceso de establecimiento de llave simétrica [4].

Figura 10. Esquema de aprendizaje mutuo entre TPMs



Una TPM como la ilustrada en la Figura 11 está compuesta por tres capas neuronales, una capa de NK neuronas de entradas $x_{ij}(t)$ que toman valores binarios pseudo aleatorios de $\{-1,1\}$. Estas neuronas se ponderan de acuerdo a unos pesos sinápticos que varían en pasos enteros en un intervalo de $[-L,\dots,L]$. Las entradas ponderadas son suministradas la capa intermedia de K neuronas cuyas salidas corresponden a la función signo de la sumatoria de las neuronas de entrada ponderadas, que corresponden al valor de los campos intermedios $y_j(t)$ (ecuación 1) cuyos valores están entre los valores $\{-1,1\}$. La capa final la

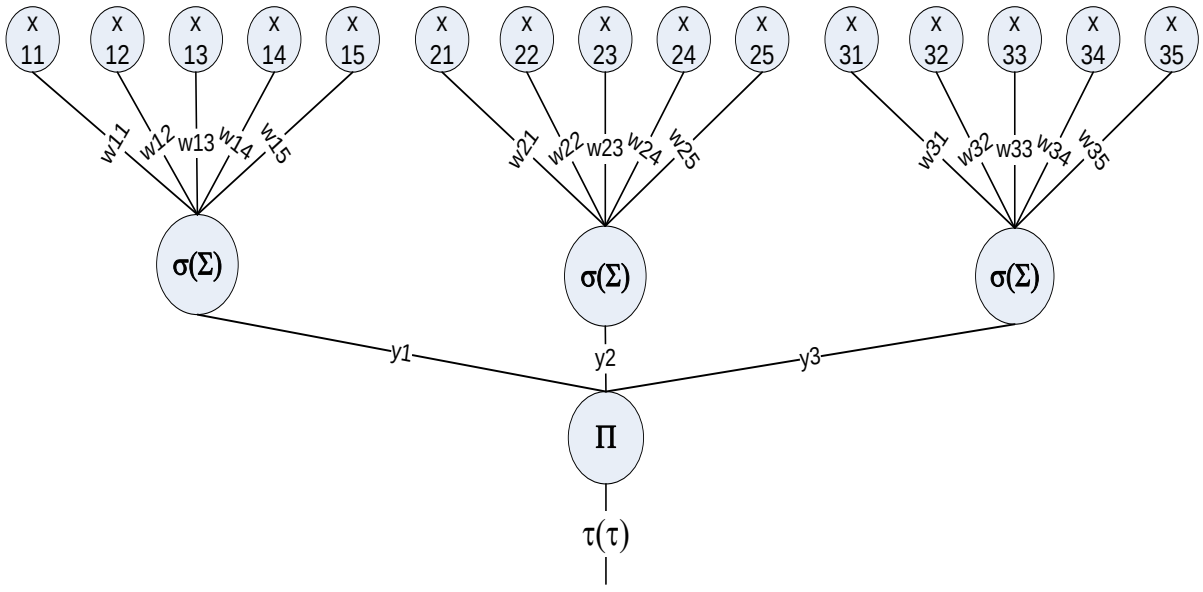
conforma una sola neurona cuyo valor también binario $\tau(t)$ (ecuación 2) verifica la paridad de los campos intermedios, siendo la salida el producto de dichos campos.

$$y_j = sgn\left(\sum_{i=1}^{N} W_{ij} * X_{ij}\right) \tag{1}$$

$$\tau(t) = \prod_{j=1}^{K} y_j \tag{2}$$

De acuerdo al algoritmo que describe el proceso de sincronización mutua entre TPMs, el proceso requiere de la precarga de los pesos sinápticos de las redes por un generador de números aleatorios.

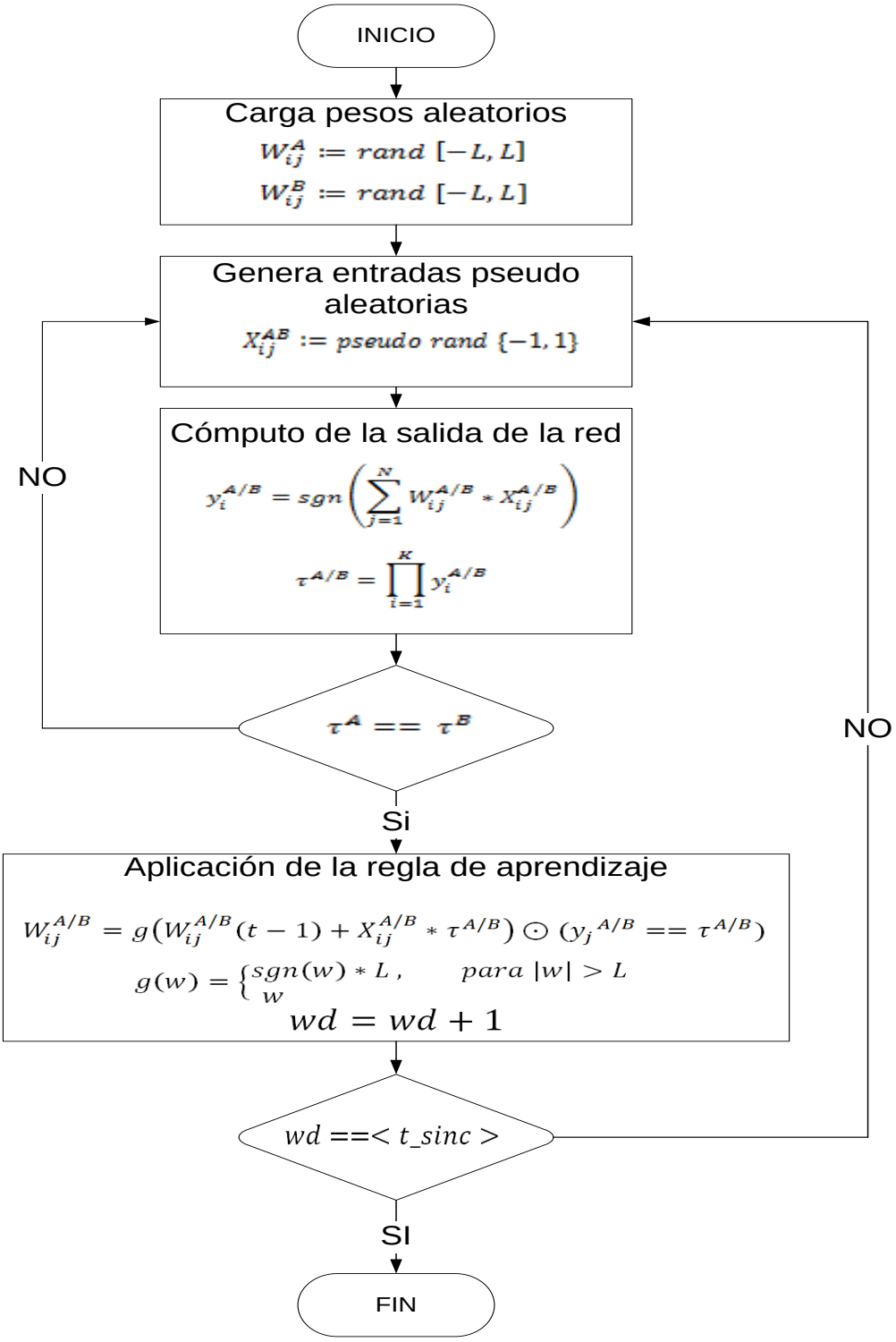
Figura 11. Red TPM con los parámetros definidos



Para iniciar el entrenamiento del sistema por el algoritmo en la Figura 12 se generan de forma pseudo aleatoria unos valores binarios iguales para las neuronas de entrada en ambas redes neuronales y se computa el resultado de la salida de cada red para estas entradas. Entonces se comparan las salidas de cada red y según una condición de aprendizaje se decide si se descartan dichas entradas y se genera una nueva combinación o si se aplica alguna de las reglas de aprendizaje que actualizan los pesos según las leyes de atracción o repulsión, estas fuerzas provocan el desplazamiento discreto de los pesos en un intervalo definido por L y en caso de exceder los limites de operación se aplica una reflexión de limites alrededor de la frontera.

El movimiento generado del entrenamiento provoca que en un tiempo finito los valores de los pesos sean iguales en ambas ANN. Cada vez que se produce un aprendizaje se considera un paso de sincronización y según los parámetros de la red se puede tener un promedio de las iteraciones necesarias para que se llegue al consenso. Si se utilizan las TPMs para el establecimiento de llave simétrica, se requiere que los pesos sinápticos o estados de ambas redes sean iguales al finalizar el entrenamiento simultáneo, por lo cual se deben alinear los estados de la red de forma paralela, para ello la condición de aprendizaje se cumple cuando las salidas de las redes son iguales $(\tau^A = \tau^B)$. También podría ser necesario establecer los estados de la red en antiparalelo [6], es decir con signo contrario, por lo que la condición de aprendizaje se define contraria al estado paralelo $(\tau^A \neq \tau^B)$.

Figura 12. Algoritmo de sincronización mutua



2.1. REGLAS DE APRENDIZAJE PARA ENTRENAMIENTO DE REDES

En el caso de las TPM de parámetros idénticos se pueden aplicar tres reglas de aprendizaje establecidas para alcanzar la sincronización mutua:

Aprendizaje hebbiano mediante fuerzas de atracción entre los pesos:

$$W_{ij}^{A/B}(t) = g\left(W_{ij}^{A/B}(t-1) + X_{ij}^{A/B} * \tau^{A/B}\right) \odot \left(y_i^{A/B} == \tau^{A/B}\right) \quad (3)$$

Aprendizaje anti-hebbiano, mediante fuerzas repulsivas entre los pesos:

$$W_{ij}^{A/B}(t) = g\left(W_{ij}^{A/B}(t-1) - X_{ij}^{A/B} * \tau^{A/B}\right) \odot \left(y_i^{A/B} == \tau^{A/B}\right) \quad (4)$$

Aprendizaje de pasos aleatorios o *random-walk:*

$$W_{ij}^{A/B}(t) = g\left(W_{ij}^{A/B}(t-1) + X_{ij}^{A/B}\right) \odot \left(y_i^{A/B} == \tau^{A/B}\right) \quad (5)$$

Donde la función $g(u)$ se define como:

$$g(u) = \begin{cases} sgn(u) * L, & para\ |u| > L \\ u \end{cases} \quad (6)$$

Cuando se aplica la regla de aprendizaje se desplaza el valor de los pesos sinápticos $W_{ij}^{A/B}(t)$ en pasos enteros y únicamente a aquellos pesos que pertenecen al campo intermedio cuyo valor es igual a la salida en ese instante $\left(W_{ij}^{A/B}(t) \in y_i^{A/B} \mid y_i^{A/B} == \tau^{A/B}\right)$. Esto se hace entre los limites positivos y negativos de L, mediante la aplicación de la ecuación (6) que hace la reflexión de limites alrededor de $\{-L, L\}$. Los pesos oscilan en dicho intervalo y se establece en estado paralelo cuando la distancia entre los pesos se hace cero, esto es $d^{A/B} = |w^A - w^B| = 0$. Una de las características de una TPM es que el tiempo de sincronización mutuo es finito pero no determinístico, por lo cual se debe configurar un contador de sincronías o *watchdog*[7] del sistema con un módulo igual al número promedio de iteraciones para detener el proceso de aprendizaje y de esta forma establecer los pesos en ambas entidades.

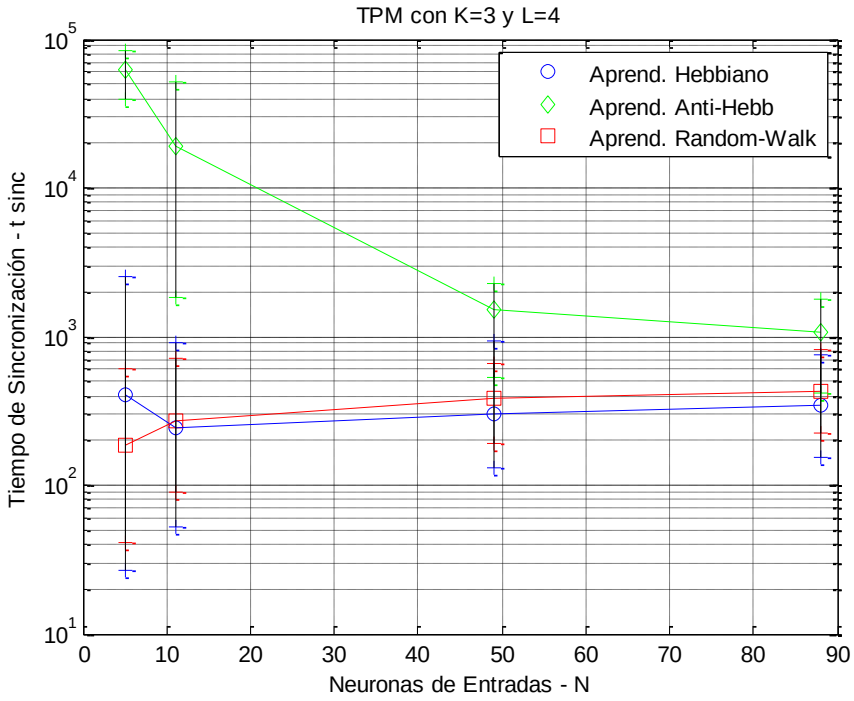
2.2. TIEMPOS DE SINCRONIZACIÓN PARA VARIACIONES DE LA RED

En esta sección se presentan los resultados luego de realizar cien experimentos en el programa de simulación MATLAB® donde se varían los parámetros de las redes para conocer el número de pasos de sincronizaciones necesarias para lograr la sincronización mutua. Se mide la distancia de los pesos de cada red después de cada ciclo de aprendizaje hasta hacerse cero mediante un *script* que contiene las funciones correspondientes a los bloques de proceso en el algoritmo de sincronización de la Figura 12, donde una función inicializa la estructura de la red, otra función genera el valor de las entradas, otra función que evalúa el valor

[7] Un contador *Watchdog* es un mecanismo de seguridad que verifica que un sistema no esté en un estado de bloqueo o que no realice ninguna función en un tiempo determinado.

de salida de la red y una última que adapta los pesos según la regla de aprendizaje.

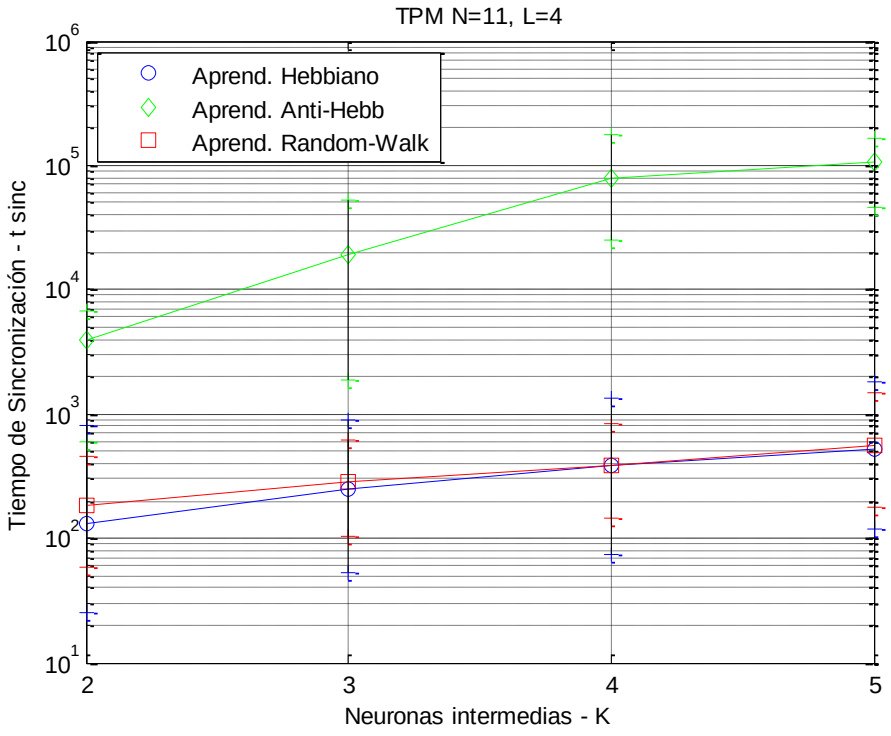
Figura 13. Tiempos de sincronización contra neuronas de entradas



Los resultados de las pruebas mostrados en detalle en el Anexo 1 se graficán para ver la dependencia de los parámetros de la red y de la regla de aprendizaje para llegar a la primera sincronización de las redes, siendo el tiempo medido la cantidad de pasos promedio donde se aplico la regla de aprendizaje a los pesos sinápticos de la red. En el caso de la Figura 13 donde se varia el parámetro N y se mantienen fijos los parametros $K = 3$ y $L = 4$, el aprendizaje hebbiano y el de pasos aleatorios poseen comportamientos similares para valores de $N \geq 11$, punto en el cual se requiere alrededor de 400 iteraciones para alcanzar la sincronización. El aprendizaje anti-hebbiano tarda más en llegar a la sincronía que los otros dos aprendizajes y además posee un comportamiento inversamente proporcional al

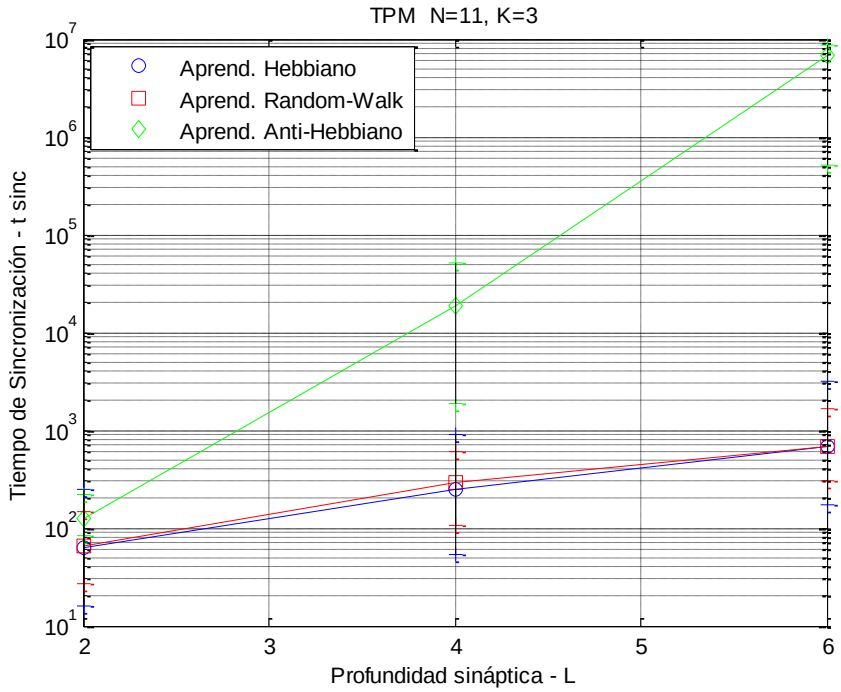
incremento de neuronas de entradas, requiriendo un promedio de 100 pasos de iteraciones más cuando se mira en el limite de las neuronas de entradas estudiadas $N = 88$. En [4] [11] para una regla de aprendizaje hebbiano analizán el cambio del parámetro N para las TPMs que buscán establecer los pesos en estado antiparalelo y muestrán que a mayor número de neuronas de entrada menor tiempo de sincronización es requerido, lo que ocurre en el caso paralelo mencionado anteriormente para el aprendizaje hebbiano y en mayor medida para el anti-hebbiano. Por lo anterior no se recomienda el uso de la regla de aprendizaje anti-hebbiana debido al mayor esfuerzo para alcanzar la sincronización que las otras alternativas de entrenamiento.

Figura 14. Tiempos de sincronización contra neuronas intermedias



Cuando se varía el parametro K que define la cantidad de neuronas de la capa intermedia de la red y se dejan fijos los parametros $N = 11$ y $L = 4$, se observa un incremento proporcional en los tiempos de sincronización a medida que se incrementa las neuronas intermedias. Según [3] el costo de sincronizar la red cuando $K \geq 3$ se incrementa exponencialmente y de la Figura 14 en el caso del aprendizaje anti-hebbiano resulta poco beneficioso en comparación con los aprendizajes hebbiano y de pasos aleatorios que tienen un promedio de 300 iteraciones necesarias para alcanzar la sincronización. Se aprecia como en el caso anterior que el aprendizaje anti-hebbiano no mejora los tiempos de sincronización, sigue siendo muy superior el número de iteraciones necesarias para sincronizar una red que al utilizar los otros metodos. Además se aprecia que al incrementar la cantidad de neuronas ocultas es más dificil llevar la red a la sincronización que al aumentar el número de neuronas de entradas.

Figura 15. Tiempos de sincronización contra profundidad de los pesos sinapticos



La gráfica expuesta en la Figura 15 corresponde al caso donde se mantienen constantes los parámetros N y K, y se varia el parámetro L que define la profundidad de los pesos sinápticos de la red. Se aprecia una dependencia proporcional de la variable L a la capacidad de sincronización de la red, siendo nuevamente el aprendizaje anti-hebbiano con el cual es más dificili sincronizar la red; tendiendo a pasos de sincronía del orden de 10^7 cuando se acerca a valores de $L = 6$. En el trabajo [11] se estudian las características de las redes a cambios del tamaño de los pesos y demuestran un costo mayor al sincronizar redes con el aprendizaje anti-hebbiano con respecto a los otros dos, esto se verifica en esta sección.

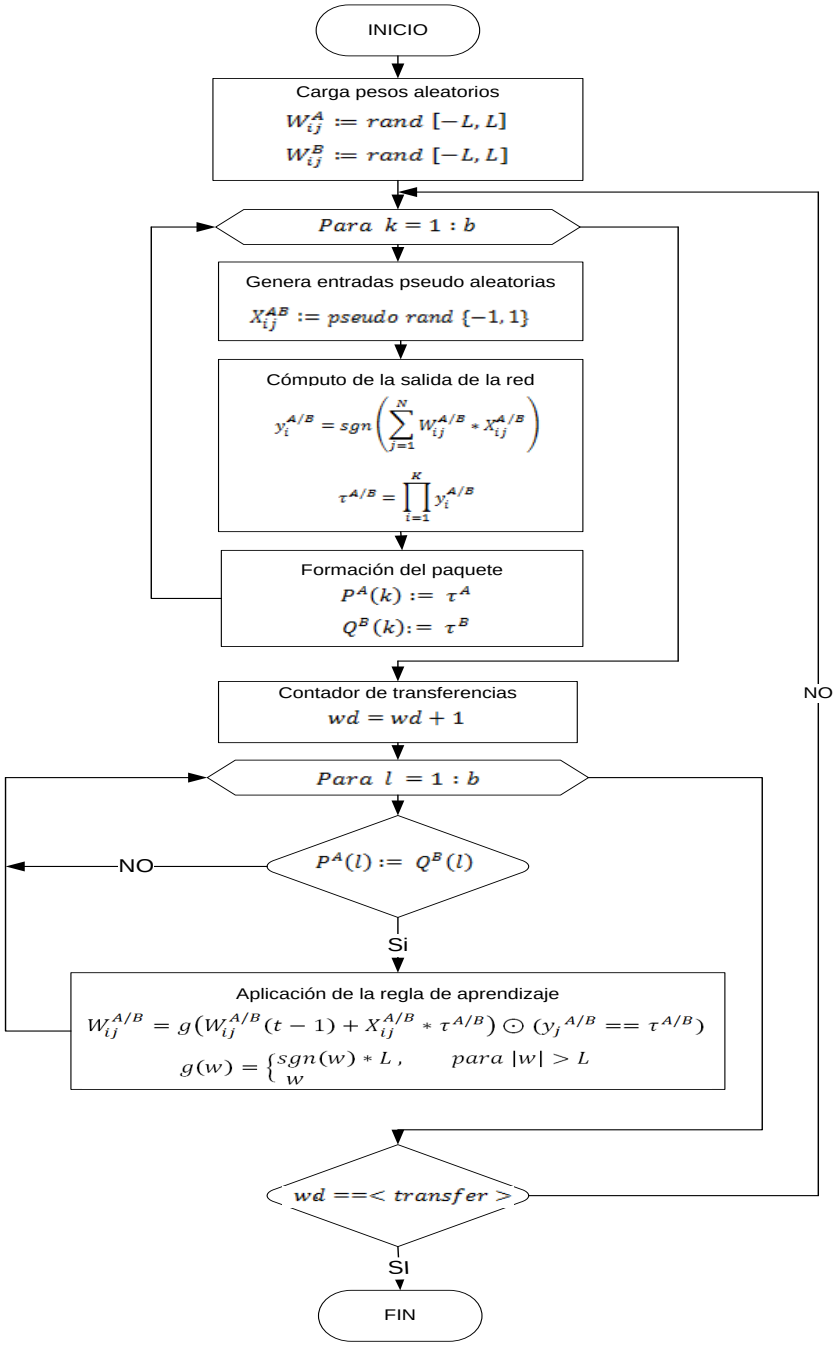
Al final del proceso de sincronización la llave establecida y secreta puede ser derivada de los estados de los pesos sinapticos almacenados en la red, siendo el tamaño máximo de la llave en bits de $N * K * L$, y si se desea tener una llave mas extensa se debe incrementar el parámetro L o incrementar el valor de $N * K$ para agregar más pesos a la red. Se concluye finalmente que para todos los casos el método de aprendizaje anti-hebbiano no es tan eficiente como los otros métodos, por lo cual no se recomienda para implementaciones digitales. Además, se resalta la importancia de la elección de un valor adeacuado para el contador *watchdog* dependiendo de los parámetros característicos de la red, con el cual se puede determinar una mayor probabilidad de éxito en que el sistema se encuentre sincronizado al final del proceso de entrenamiento.

2.3. SINCRONIZACIÓN MUTUA POR VARIANTE DE PAQUETES

En la Figura 16 se muestra el algoritmo para entrenar el sistema compuesto por TPMs mediante la variante de paquete de bits de salidas, entrenamiento por el cual en vez de intercambiar bit a bit las salidas entre las redes, se agrupa en cada red un paquete de tamaño b, que contiene las salidas computadas de la red a diferentes valores pseudo aleatorios de entrada sin que se aplique la regla de aprendizaje; luego se intercambian paquetes entre las entidades para analizar bit a bit la información del paquete y así aplicar internamente la regla de aprendizaje cuando las salidas cumplen la condición de aprendizaje. Esta variante permite mediante un cambio en la dinámica de sincronización mutua aumentar la taza de transferencias de bits entre las dos entidades y aprovechar la unidad máxima de transferencia de las las comunicaciones basadas en modelos de capas[8]. Complemento del análisis hecho para redes que transfieren salidas bit a bit $(b = 1)$, se ilustra el costo en el número de pasos de sincronización promedio cuando se realiza transferencia de paquetes de bits de salida para el caso del aprendizaje hebbiano. Se evidencia que al aumentar cualquiera de los parámetros de la red N, K y L se aumenta el esfuerzo para llegar al estado de sincronización del sistema.

[8] Modelo de capas como OSI y TCP/IP empaquetan en tramas la información que se va a transmitir por los canales de transmición y MTU puede variar dependiendo del tráfico y el ancho de banda.

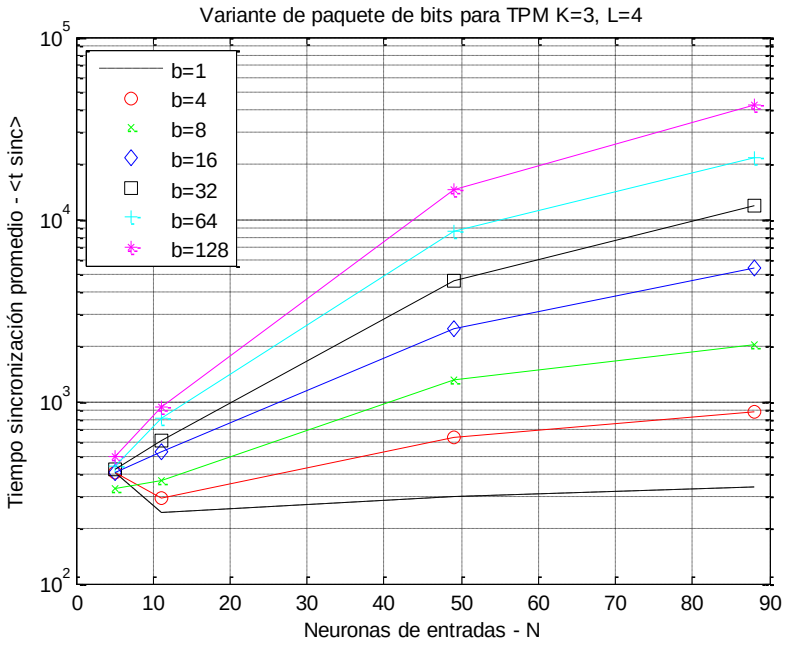
Figura 16. Algoritmo de sincronización mutua por variante de paquete



Al analizar la Figura 17 se puede apreciar que cuando se intercambian paquetes de salidas con $b < 8$ se pueden obtener tiempos de sincronización promedio

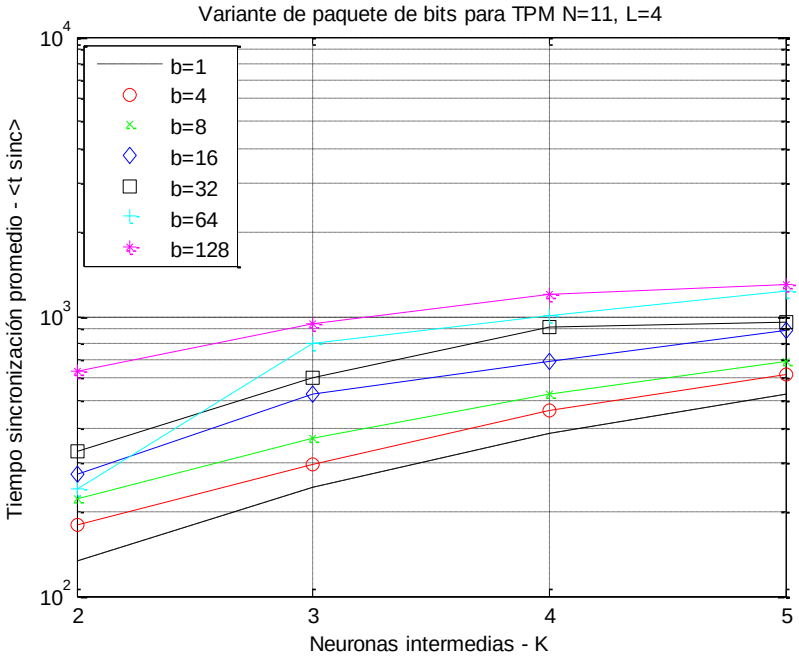
cercanos a la transferencia de bit a bit, y mientras se incrementa el tamaño del paquete aumenta exponencialmente el esfuerzo para llegar a la sincronizacion mutua. El parámetro N es importante al momento de intercambiar paquetes ya que para valores superiores de $N \geq 11$ los tiempos de sincronización aumentan significativamente respecto a valores pequeños del mismo, demostrando que la eficiencia disminuye para cantidades grandes de neuronas de entrada.

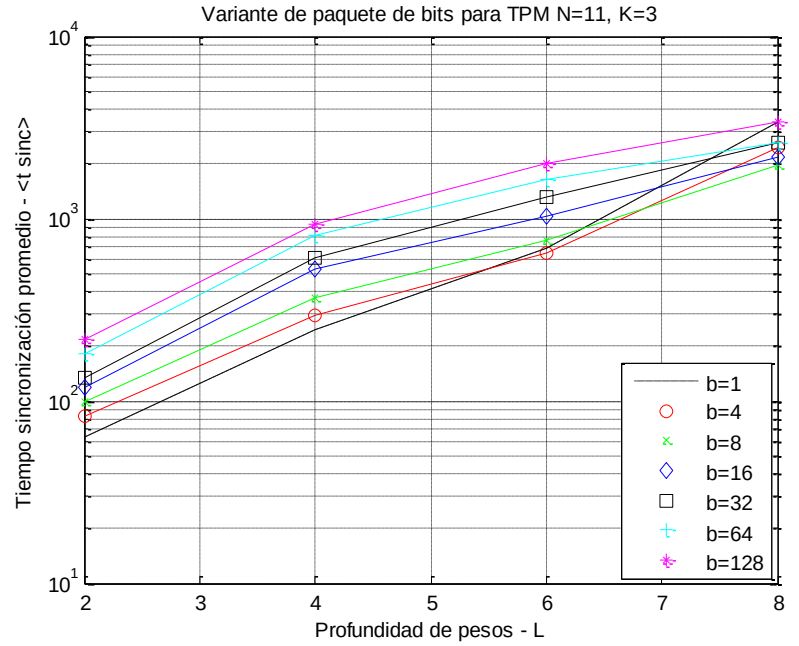
Figura 17. Tiempos de sincronización promedio contra neuronas de entrada a diferentes tamaño de paquete



En los casos de la Figura 18 donde se varian los parámetros de red K y L se aumenta el tiempo promedio de sincronización de forma proporcional al tamaño del paquete, tendiendo a tiempos similares al llegar a los limites superiores de los parámetros de la red, por lo cual a paquetes muy grandes se puede llegar a tiempos de sincronización parecidos entre la variante de paquetes y la transferencia de bit a bit de salida.

Figura 18. Tiempos de sincronización promedio contra neuronas intermedias y proufundidad sináptica a diferentes tamaño de paquete



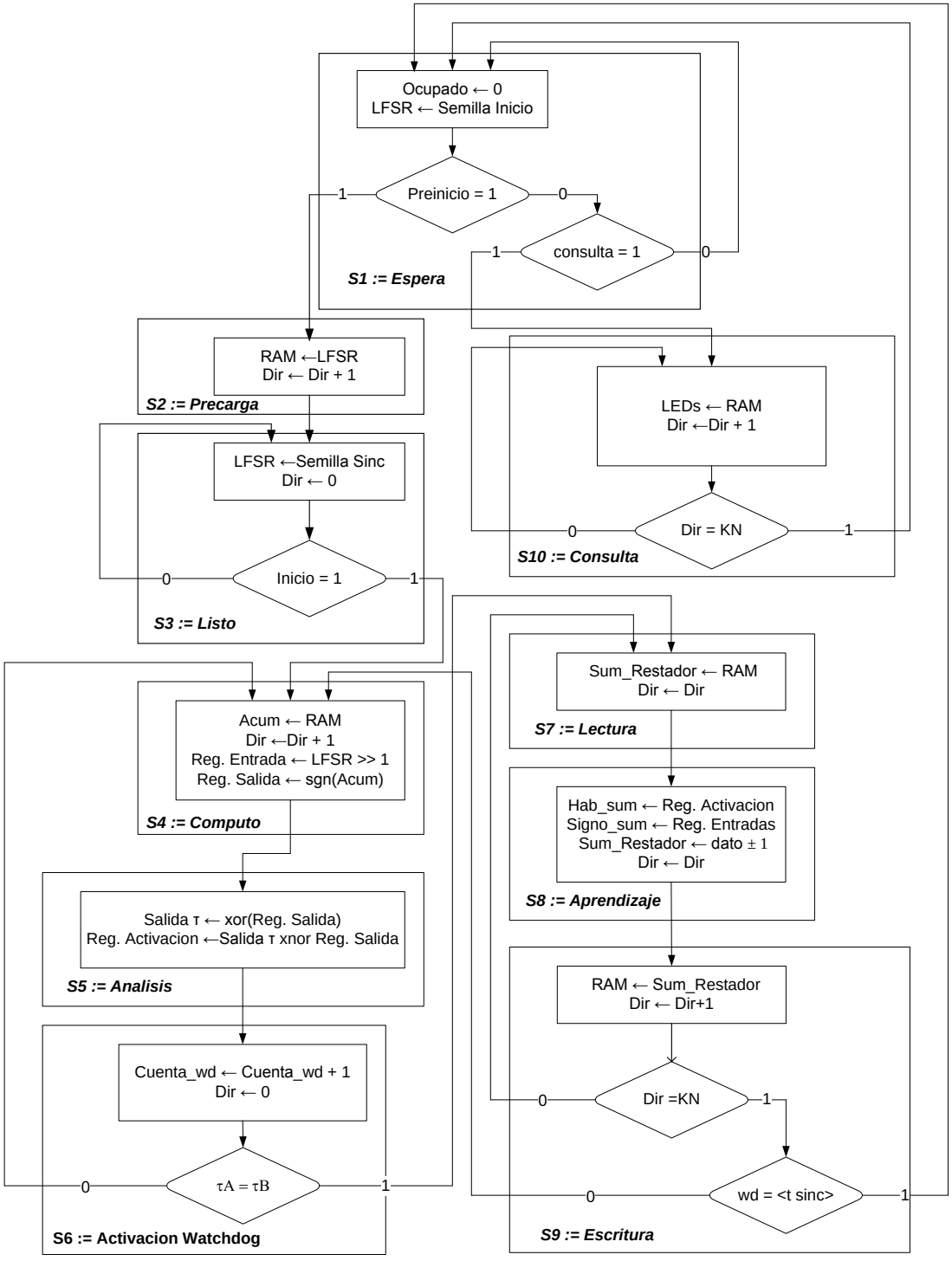


3. DISEÑO DE PROCESADOR TPM EN VHDL

A partir de la literatura recopilada y de los métodos de diseño de circuitos digitales [17] [18] se describen en el programa ISE *Design Suit* de la empresa Xilinx® las fuentes en lenguaje VHDL que constituyen los procesadores de las redes para hacer el establecimiento de llave simétrica por intercambias salidas bit a bit o por variante de paquetes; estos procesadores son configurables a partir de los parámetros N, K y L definidos por el usuario. La descripción de la red se basa la arquitectura de tipo serial formulada en [8] debido a que requiere la menor carga de recursos disponibles al FPGA. Del diagrama de flujo de la Figura 12 se definen las estructuras funcionales de la red que permiten la sincronización mutua entre redes. Este algoritmo se pasa a un diagrama de estados y de transferencias necesarias para realizar el proceso de sincronización mutua de un circuito digital. Con base al nuevo diagrama ASM mostrado en la Figura 19 se elabora un *datapath* que realiza las operaciones requeridas y una máquina de estado que controla el recorrido de los datos en el *datapath.*

En una memoria RAM se almacenan los pesos sinápticos generados en el proceso de sincronización, la cual es inicializada por medio de un generador de números aleatorios externo o de un registro de desplazamiento con realimentación lineal LFSR como generador interno de números pseudo aleatorios para la carga inicial de la memoria y para la generación de los valores de las neuronas de entrada para cada paso de sincronización, esto se hace mediante diferentes semillas que se cargan a los registros del generador en dos diferentes estados. Para leer y escribir la memoria se describe un contador de direcciones cuyo módulo es KN.

Figura 19. Diagrama de transición para sincronización mutua entre TPMs



El primer paso para evaluar la salida de la red es ponderar las entradas binarias $\{-1, 1\}$ por los pesos sinápticos, esto significa que los pesos pueden conservar su signo o invertirlo, por tal motivo un módulo niega el peso según la entrada correspondiente dependiendo del formato del mismo. Estos datos son operados por un bloque acumulador que suma los pesos complementados por las entradas de cada neurona oculta de la red. Los valores de las entradas, los campos intermedios y las salidas son guardados en registros de desplazamiento para ser consultados en los diferentes estados del entrenamiento de la red. Para aplicar la regla de aprendizaje se utiliza un módulo sumador restador que mueve los pesos según el signo de la regla de aprendizaje y luego los escribe sobre la RAM para volver a realizar el proceso de entrenamiento. El signo de la regla de aprendizaje depende de la regla escogida según la salida actual, el valor del campo intermedio y la entrada correspondiente almacenada en bloques de registros.

3.1. FORMATO DE LOS PESOS SINÁPTICOS

El formato empleado para los pesos sinápticos es complemento a dos, debido a que es sencillo realizar cálculos con ellos al momento de usar circuitos digitales. Un solo bloque sumador puede operar números positivos o negativos en complemento a dos sin depender que el número sea mayor o menor que el otro, caso del formato binario. Este formato posee asimetría para los valores positivos y negativos, y sólo posee un valor para el cero [19].

- **Máximo valor positivo:** $2^{m-1} - 1$
- **Mínimo valor negativo:** -2^{m-1}

Los valores binarios que puede tomar las entradas, los campos intermedios y la salida se codifican de la manera expuesta en la Tabla 3 de acuerdo al bit de signo del complemento a dos. Es decir que con un sólo bit podemos representar los dos valores $\{-1, 1\}$ y pueden ser tomados de bit más significativo de cualquier número representado en complemento a dos correspondiendo al signo del número en decimal.

Tabla 3. Codificación de los valores de entrada según el bit signo del complemento a dos

Decimal	Binario
1	0
-1	1

Si se desea conocer la cantidad de bits necesarios para representar un número decimal $D_{10} \in \mathbb{Z}$ en el complemento a dos, se aplica la función inversa al máximo valor positivo posible en este formato para obtener la ecuación (7), la cual nos permite saber la longitud de un registro para almacenar dicho número y que incluye el bit de signo del complemento.

$$m[bits] = \log_2(D_{10}) + 1 \qquad (7)$$

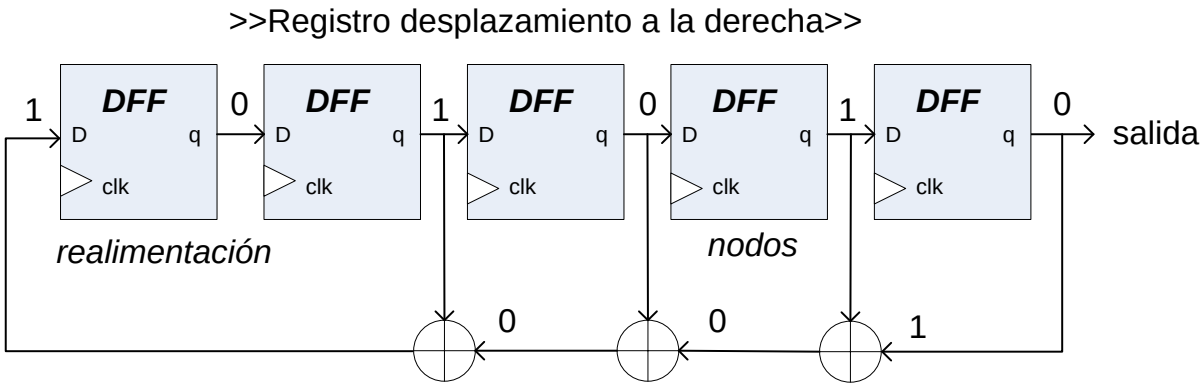
En la descripción de las fuentes en VHDL se definen los tamaños de los vectores en número de bits mediante la ecuación (7) obteniendo la cantidad de bits requeridos para representar los parámetros de la red que están en formato entero decimal.

3.2. GENERACIÓN DE NÚMEROS PSEUDO ALEATORIOS

Los valores de las neuronas de entrada son alimentadas por un generador de secuencia binaria pseudo aleatoria (PRBS). Este elemento genera valores de $\{0, 1\}$ de forma serial cada vez que la Red TPM computa sus parámetros internos para obtener el valor de la salida $\tau(t)$ al inicio del proceso de sincronización o luego de cada actualización.

Para la implementación se utiliza un registro de desplazamiento con realimentación lineal (LFSR) como generador de números pseudo aleatorios de longitud p número de FFs. Este es un registro de desplazamiento a la derecha, al cual se le ha cargado un valor inicial o semilla y que se realimenta de forma serial con unos elementos propios del registros (o *taps)* que han pasado por una operación booleana exclusiva como se muestra en la Figura 20.

Figura 20. Diagrama de funcionamiento de un LFSR



Este registro genera una secuencia de números binarios de m bits que se repite cada $2^p - 1$ ciclos de reloj, por lo que los datos de salida poseen cierta repetitividad. Para hacer que el periodo de la secuencia sea máximo se describe por la teoría de campos finitos[9] los segmentos a realimentar y las características del circuito. Los nodos o *taps* realimentados forman un polinomio módulo dos cuyas potencias dependen de las posiciones relativas de los lazos realimentados y cuyas características para que el periodo sea máximo. Siendo el polinomio de realimentación:

$$a_{p-1}x^{p-1} + a_{p-2}x^{p-2} + \cdots + a_1x^1 + a_0x^0$$

El periodo del registro LFSR es máximo cuando:

- Si y solo si, el polinomio es primitivo
- Los valores de los taps son coprimos
- El número de nodos es par

Existe una secuencia espejo también máxima igual a:

$$[p-1, t_1, t_2, t_3 \ldots t_k] = [p-1, p-1-t_1, p-1-\ t_2, p-\ 1-t_3 \ldots p-1-t_k]$$

Los fabricantes de circuitos digitales han definido una lista de *taps* [22] que pueden ser usados para tener un LFSR en configuración estándar donde se puede tener un periodo máximo del generador (ver Tabla 4).

[9] La teoría de campos finitos fue desarrollada por el matemático francés Evariste Galois (1811-1832), utilizada en teoría de números, geometría algebraica y criptografía.

Tabla 4. Nodos de realimentación recomendados para el uso de un LFSR

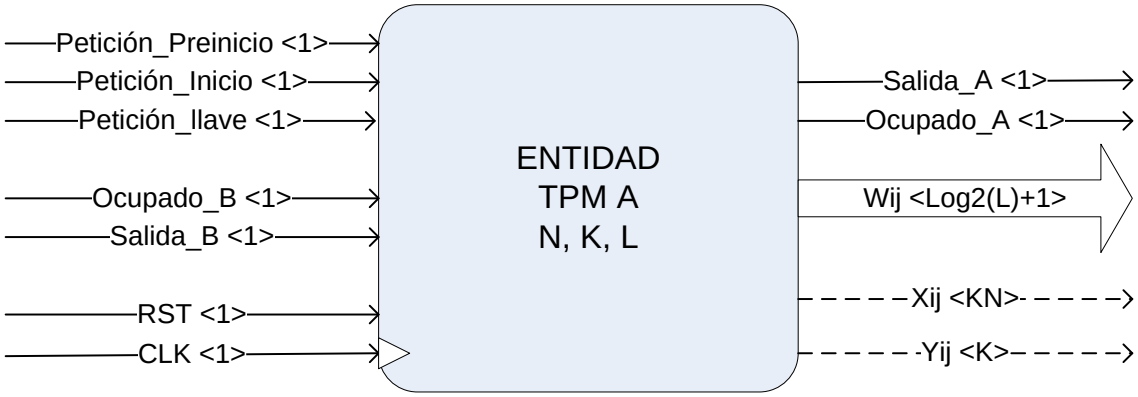
Bits p	**Polinomio de realimentación** $\mathbf{t_k(taps)}$	**Periodo** $[\mathbf{2^p - 1}]$
2	1, 2	3
3	3, 2	7
4	4, 3	15
5	5, 3	31
6	6, 5	63
7	7, 6	127
8	8, 6, 5, 4	255
12	12, 6, 4, 1	4.095
16	16, 15, 13, 4	65.535
24	24, 23, 22, 17	16.777.215
32	32, 22, 2, 1	4.294.967.295
64	64, 63, 61, 60	18.446.744.073.709.551.615

Para la implementación en VHDL se utiliza el generador de secuencia pseudo aleatorio para producir de forma paralela los pesos sinápticos para la carga inicial de la memoria RAM y para generar de forma serial los valores de las entradas de entrenamiento correspondientes para cada neurona de la capa de entradas. Estos valores son generados por diferentes semillas cargadas al registro interno en los estados antes y después del inicio de la sincronización entre las redes.

3.3. DECLARACIÓN DE LA ENTIDAD TPM

La entidad de la red neuronal de la Figura 21 posee un protocolo de inicio ocupado para coordinar la comunicación con cualquier otra entidad del mismo tipo. Las señales de entrada al procesador son el reloj del sistema, el *reset* asíncrono y las entradas de inicio de procesos: una petición de carga inicial, una petición de inicio del proceso y una petición de consulta de los pesos de la red. Además de recibir la información de salida de la red externa: su salida y la señal de ocupado. Y la red envía su salida y su ocupado a la otra red para establecer el intercambio.

Figura 21. Entidad del procesador TPM



El procesador tiene una máquina de estados de arquitectura *Mealy* y es de tipo síncrono, es decir que con un flanco de activación del reloj del sistema se coordina las transferencias entre los bloques del *datapath.* La comunicación entre las redes es síncrona pero puede haber retardos en el canal o demoras internas propias del proceso de sincronización de TPM en cada red, por lo cual mediante el protocolo de inicio ocupado la red espera la señal de disponible de su contraparte para continuar con el proceso.

3.3.1. *Datapath* del procesador TPM. El *datapath* de la Figura 22 está compuesto por doce módulos descritos en VHDL necesarios para realizar el establecimiento de llave mediante el proceso de sincronización mutua entre redes TPM. Se incluye una fuente adicional de tipo *package* donde se definen el valor de las constantes correspondientes a los parámetros de red, el límite del contador *watchdog* y las semillas del generador LFSR.

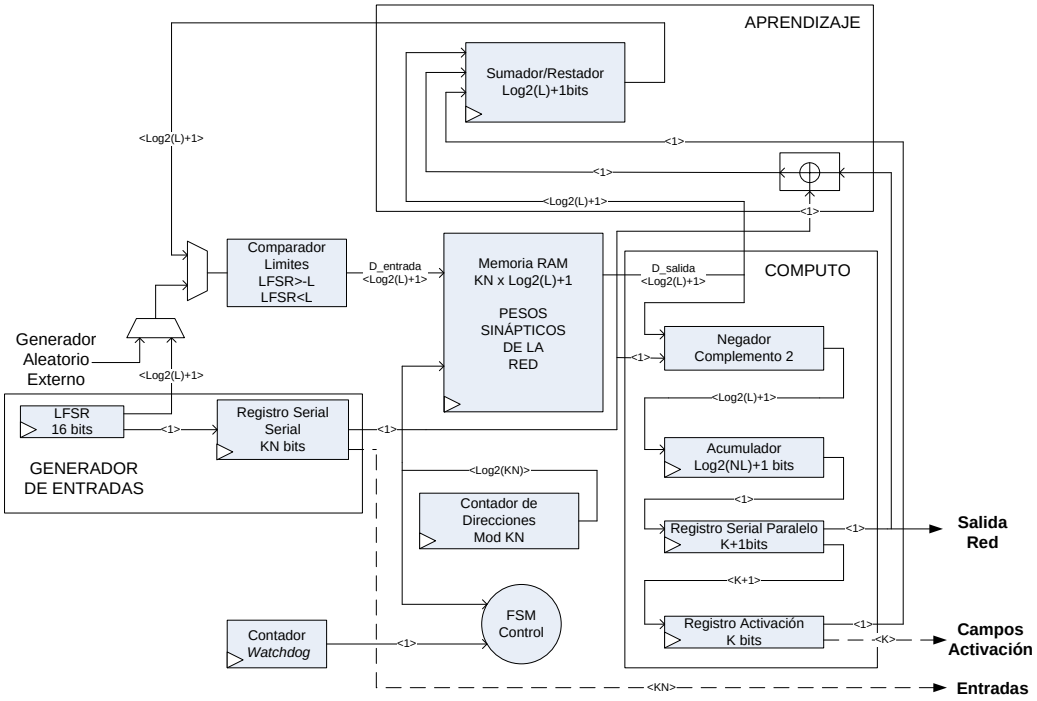
Cada bloque funcional se describe bajo el diseño de transferencia de registros o lógica RTL, por lo que los módulos tienen a la salida un registro sincronizado con el reloj del sistema y se direccionan las señales mediante multiplexores que son controlados por la máquina de estados mediante las palabras de control definidas para cada bloque. Al sincronizar los registros con el reloj del sistema permite evitar violaciones en los tiempos de transferencias permitidos por los FFs y coordinar que cada bloque realice su operación y transmita el dato al siguiente bloque para completar el proceso con cada flanco de ascenso del reloj del sistema.

La memoria RAM es un arreglo bidimensional de NK filas correspondiente a los pesos sinápticos que dependen del parámetro L según la ecuación (7) en formato complemento a dos. Para recorrer la memoria se implementa un contador de direcciones que es controlado por la máquina de estados y que cada N direcciones recorridas corresponden a una neurona intermedia de la red.

La salida de la memoria RAM va a un módulo que aplica el complemento a dos para invertir el signo del dato dependiendo del bit de entrada para ese peso, luego el dato es llevado a un acumulador que suma la información de cada neurona intermedia recibida del negador para hallar el campo correspondiente. Internamente el módulo acumulador puede sumar hasta el máximo límite posible de pesos almacenados por neurona oculta, por lo cual el sumador se dimensiona

para acumular hasta el valor NL, aunque el dato de salida del bloque es de un bit correspondiente al signo de la sumatoria. El signo se guarda en un registro de entrada serial de tamaño $K+1$ para que el procesador obtenga los siguientes campos de la red; en la posición del bit más significativo se va calculando la salida de la red mientras se van obteniendo los datos de las neuronas intermedias mediante compuertas exclusivas.

Figura 22. *Datapath* y FSM con parámetros variables de TPM



El módulo de activación recibe la información del registro de salidas y crea un vector de tamaño K con la información de activación donde el signo de las neuronas intermedias coincide con la salida de la red $(y_j == \tau_{(t)}^{A})$, este registro tiene los bits de habilitación del bloque de actualización que se encarga de aplicar la regla de aprendizaje en el caso de las salidas de las redes sean iguales. El proceso de aprendizaje de la red lo lleva a cabo un módulo sumador restador que

recibe la lectura de la RAM y de acuerdo al valor almacenado en el registro de entrada y la señal de habilitación del registro de activación incrementa o decrementa el peso, que luego es almacenado de nuevo en la memoria.

3.3.2. Máquina de estados finita del procesador TPM. El diagrama de transición de estados de la Figura 23 muestra las transiciones de cada estado del procesador en todo momento del funcionamiento de la red. Se configuran diez estados necesarios para que el sistema pueda establecer un intercambio con otra entidad para realizar el establecimiento de llave. El primer estado llamado de espera es donde la red se encuentra en estado de reposo con condiciones iniciales iguales a cero y donde llega siempre después de cualquier *reset* de tipo asíncrono hecho por el usuario, además carga una semilla diferente en el LFSR de cada red. El segundo estado denominado de precarga es cuando se escribe en la memoria RAM los pesos generados por el generador pseudo aleatorio y acotado por el módulo limitador para preparar el procesador para iniciar al entrenamiento. Al terminar este proceso se pasa a un estado donde la red está lista a recibir la señal de inicio y comenzar el proceso de sincronización mutua y se carga la semilla igual en los generadores. En este estado también se puede conocer el estado de la memoria RAM con los pesos almacenados mediante un estado de consulta que extrae la información contenida en la memoria al exterior. Cuando se recibe la señal de inicio de sincronización se pasa al estado de cómputo, donde se halla el valor de la salida de la red por medio del inversor, el acumulador y los registros de entrada y salida.

Después del cómputo se pasa al estado de análisis el cual tiene la función de esperar a que la otra red termine de operar sus estados internos mediante la señal de ocupado. En el estado de activación de *watchdog* se compara las salidas de las

redes ya obtenidas y envía una señal a la máquina de estado para saber si se pasa a la etapa de aprendizaje y se aumenta el contador de sincronías o se repite el estado de cómputo con nueva información para las entradas; este estado tarda un solo ciclo de reloj del sistema. En todos los estados de la FSM es prioridad en la lógica de transiciones de la red que si se recibe una señal de *reset* del exterior se pasa la máquina al estado de espera, donde se reinicia el proceso de establecimiento. El proceso de sincronización mutua se da por terminado en el momento en que el contador *watchdog* llegue al valor límite definido por el usuario. En la sección 2.2 se da un criterio para la selección del tiempo de sincronización promedio para cada configuración de los parámetros de red para tener una alta probabilidad de éxito en la sincronización de las redes.

Figura 23. Diagrama de estados y transiciones de la FSM

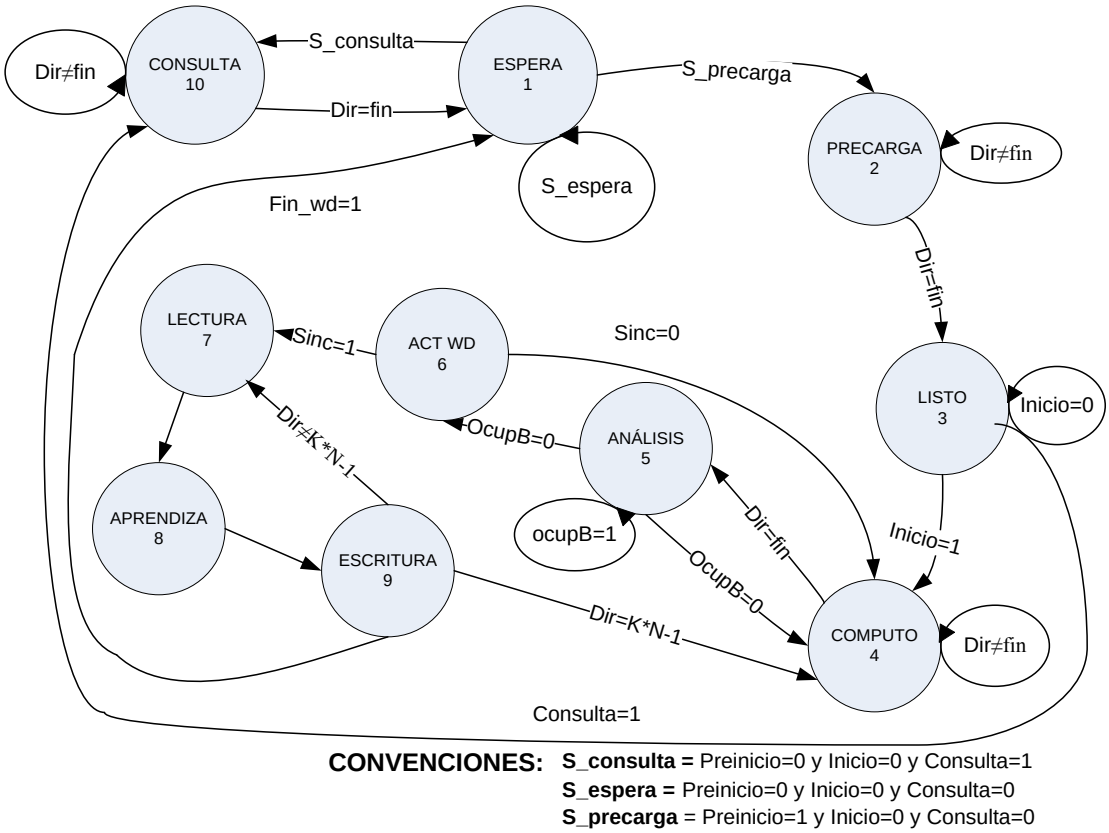


Tabla 5. Señales de control de las fuentes del *datapath*

Control del Contador de Direcciones			**Control de la RAM**		
HAB	**fin**	**Fsm_Contador**	**Escribir**	**HAB**	**Fsm_RAM**
0	0	Dirección←Dirección	0	0	Salida ← 0
0	1	Dirección ←Dirección	0	1	Salida ← RAM
1	0	Dirección←Dirección+1	1	0	RAM ← Entrada
1	1	Dirección ← 0	1	1	RAM ← Entrada
Control de Generador LFSR			**Control del Acumulador**		
HAB	**Sinc**	**Fsm_Generador**	**Inicio**	**HAB**	**FSM_Acum**
0	0	Salida ← Salida	0	0	Salida ← Salida
0	1	Salida ← Semilla Proceso	0	1	Salida ← Salida + Entrada
1	0	Salida ← Salida >> 1	1	0	Salida ← 0
1	1	Salida ← Semilla Proceso	1	1	Salida ← Entrada
Control Registro Entradas			**Control Registro Salidas**		
HAB		**Fsm_Reg_Serial**	**HAB**		**Fsm_Reg_Paralelo**
0		Registro ← Registro	0		Registro ← Registro
1		Registro ← Registro >> 1	1		Registro ← Registro >> 1
Control Registro Activación Intermedia			**Control del Sumador-Restador**		
Desp	**Opere**	**Fsm_Act → fsm_Sum**	**Activa**		**Fsm_Sum**
0	0	Registro ← Registro	0		Salida ← Entrada
0	1	Registro ← Entrada	1		Salida ← Entrada ± 1
1	0	Registro ← Registro >> 1			
1	1	Registro ← Entrada			

Para realizar el aprendizaje de la red se definen tres estados: un estado de lectura, uno de aprendizaje y otro de escritura. En el primer estado se hace una lectura de la memoria RAM y se transfiere al bloque actualizador, en el segundo estado se habilita el sumador restador y se obtiene el valor del incremento según la regla de aprendizaje y en el último estado se escribe en la memoria RAM dando por terminado el proceso de aprendizaje y pasando al estado de cómputo donde se continua el proceso de sincronización. Para realizar el control del *datapath* desde la máquina de estado se definen unas palabras de control mostradas en la

Tabla 5 para cada módulo funcional que contiene la información para ordenar a cada módulo como actuar en cada estado de la FSM.

Tabla 6. Tabla de transición de registros RTL

Tabla RTL		Palabras de Control			
Estado	**Instrucción RTL**	**Contador**	**RAM**	**Generador**	**Acumulador**
Espera 1	Dir ← 0 LFSR←Semilla1 Acum ← 0 R. Ent ← 0 R. Sal ← 0 Act ← 0	“11”	“00”	“00”	“10”
Precarga 2	Dir ← Dir + 1 RAM ← LFSR LFSR←LFSR>>1	“10”	“11”	“10”	“00”
Listo 3	Dir ← 0 LFSR←Semilla2	“11”	“00”	“01”	“00”
Computo 4	Dir ← Dir + 1 Acum ← RAM R.Ent ← LFSR LFSR←LFSR>1 R. Sal ← Acum	“10”	“01”	“10”	“10”[10] “11”[11] “01”[12]
Análisis 5	Dir ← Dir + 1 Act ← R. Sal	“10”	“01”	“00”	“01”
Act_WD 6	Dir ← 0 Acum ← 0	“11”	“00”	“00”	“10”
Lectura 7	Dir ← Dir Sum ← RAM	“01”	“01”	“00”	“10”
Aprend 8	Dir ← Dir Sum ← Entrad±1 Act ← Ac >>1	“01”	“01”	“00”	“10”
Escritura 9	Dir ← Dir + 1 RAM ← Sum R.Ent←R.Ent>>1	“10”	“11”	“00”	“10”
Consulta 10	Dir ← Dir + 1 Salida ← RAM	“10”	“01”	“00”	“01”

[10] Para la salida del contador de direcciones igual a cero.

[11] Para la salida del contador de direcciones igual a múltiplos de K*N.

[12] Para la salida del contador de direcciones diferente de cero o múltiplos de K*N.

Tabla 6 (continuación). Tabla de transición de registros RTL

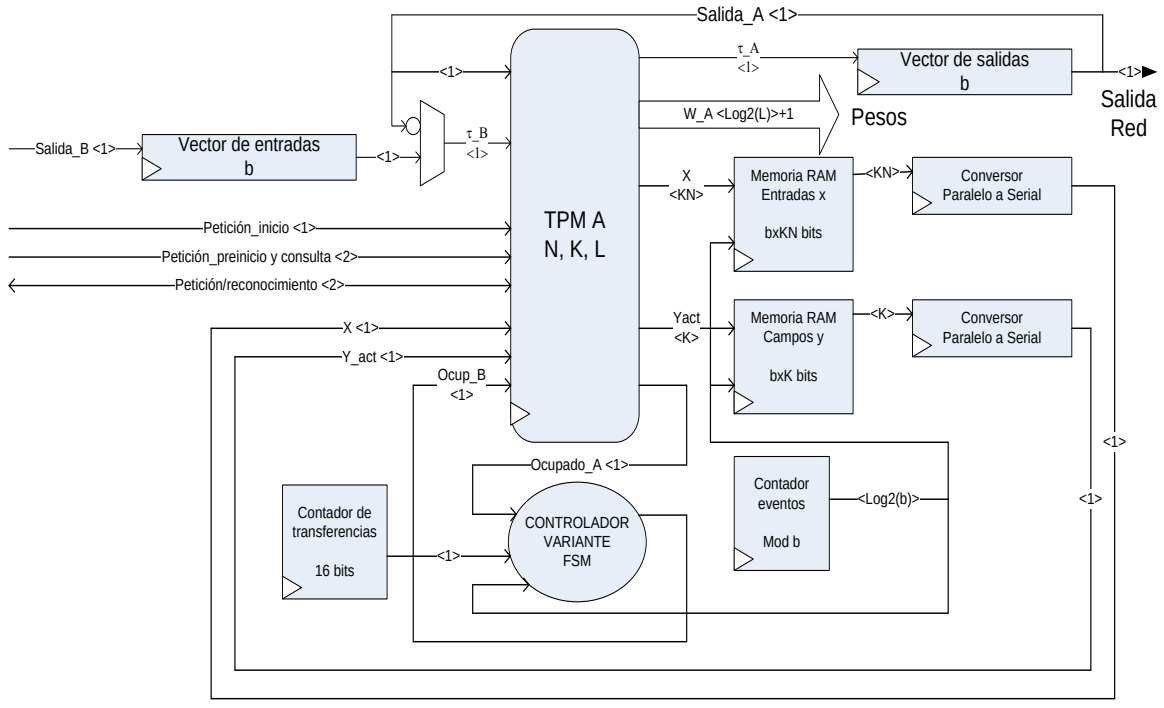
Tabla RTL		Palabras de Control			Salidas	
Estado	**Instrucción RTL**	**R. Ent**	**R. Sal**	**Activa**	**Sal**	**Ocupado**
Espera 1	*Dir ← 0* *LFSR←Semilla1* *Acum ← 0* *R. Ent ← 0* *R. Sal ← 0* *Act ← 0*	‘0’	‘0’	“00”	‘0’	‘0’
Precarga 2	*Dir ← Dir + 1* *RAM ← LFSR* *LFSR←LFSR>>1*	‘0’	‘0’	“00”	‘0’	‘1’
Listo 3	*Dir ← 0* *LFSR←Semilla2*	‘0’	‘0’	“00”	‘0’	‘0’
Computo 4	*Dir ← Dir + 1* *Acum ← RAM* *R.Ent ← LFSR* *LFSR←LFSR>1* *R. Sal ← Acum*	‘1’	‘0’ ‘1’ ‘0’	“00”	‘0’	‘1’
Análisis 5	*Dir ← Dir + 1* *Act ← R. Sal*	‘0’	‘1’	“01”	‘1’	‘0’ ‘1’
Act_WD 6	*Dir ← 0* *Acum ← 0*	‘0’	‘0’	“00”	‘0’	‘0’
Lectura 7	*Dir ← Dir* *Sum ← RAM*	‘0’	‘0’	“00”	‘0’	‘1’
Aprend 8	*Dir ← Dir* *Sum ← Entrad±1* *Act ← Ac >>1*	‘0’	‘0’	“10” “00	‘0’	‘1’
Escritura 9	*Dir ← Dir + 1* *RAM ← Sum* *R.Ent←R.Ent>>1*	‘1’	‘0’	“00”	‘0’	‘1’
Consulta 10	*Dir ← Dir + 1* *Salida ← RAM*	‘0’	‘0’	“00”	‘0’	‘1’

A partir de la Tabla 5 de palabras de control se construye la tabla de transición de registros que contiene la configuración de la máquina de estado. En la Tabla 6 se incluyen las instrucciones RTL donde se muestran las transiciones de los registros de cada fuente de *datapath*, las palabras de control de cada módulo y las señales de salida de la red, que son la señal de ocupado de la red y la habilitación de la salida de la red ya computada.

3.4. IMPLEMENTACIÓN DE LA VARIANTE DE PAQUETES

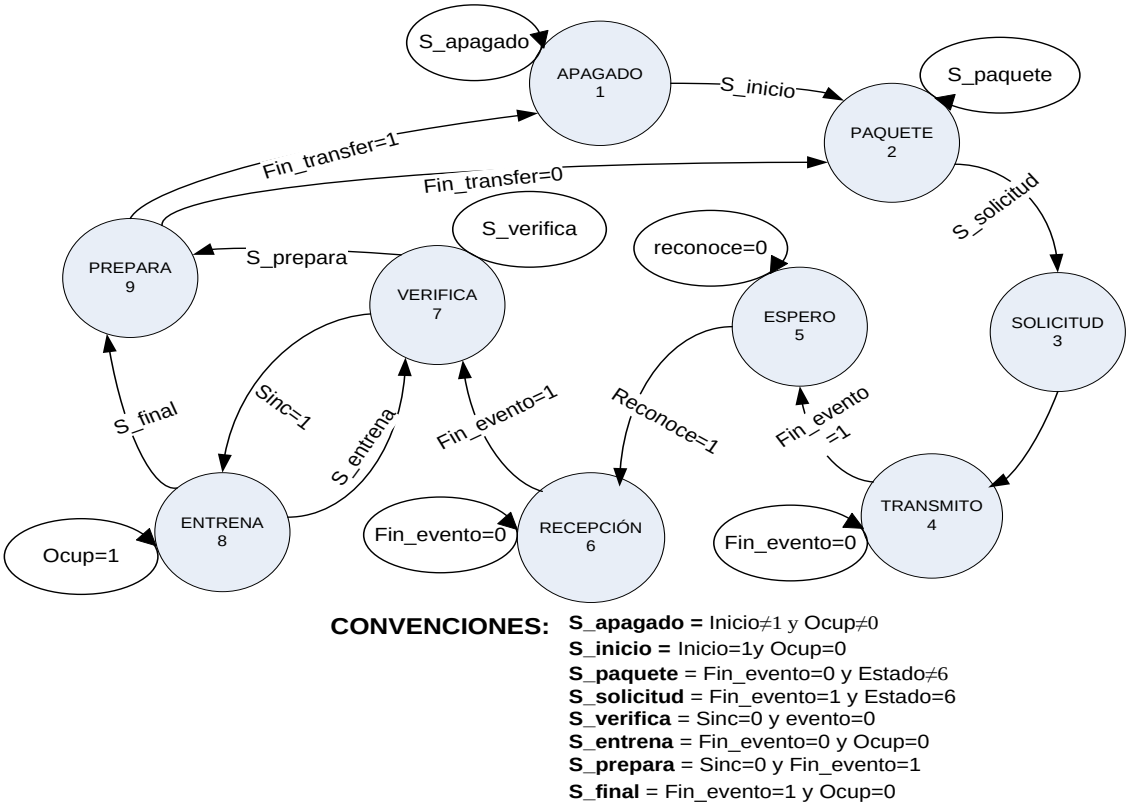
En un proceso de intercambio compuesto por dos ANN se transfieren las salidas para evaluar si se pasa a un paso de aprendizaje o se generan nuevas entradas. Como se documenta en el sección 3.1.1 se puede codificar dicha salida en una señal de un sólo bit que representa los dos valores posibles de la neurona de salida. En sistemas de comunicación basados en los modelos de capas, la carga del canal se supervisa, varía y optimiza según el tráfico del medio de transmisión, para enviar la mayor cantidad de información por transferencias hechas, ya que enviar de a un bit de salida por trama en la transmisión sería desaprovechar el MTU del sistema de comunicación.

Figura 24. *Datapath* y controlador para implementación de la variante de paquetes



Por lo anterior se ha documentado [6] [7] [8] [9] [10] la variante de paquetes de bits, que consiste en enviar varias salidas de la red en un paquete de tamaño b para luego realizar el proceso de aprendizaje. Esto permite disminuir la cantidad de intercambios bit a bit de salidas e incrementar la eficiencia del uso del empaquetamiento de los protocolos de comunicación.

Figura 25. Máquina de estados para la variante de paquete de bits



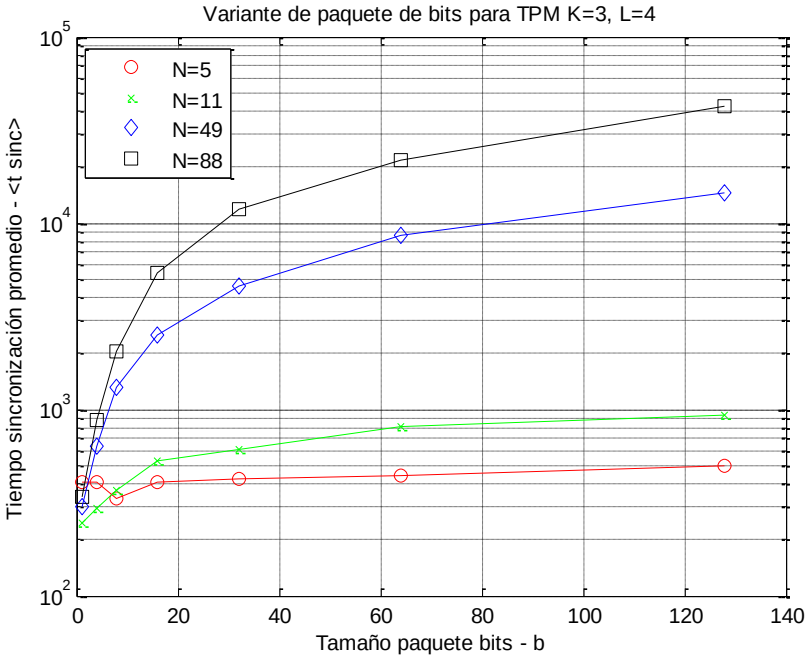
Se puede ver en la Figura 24 que a la entidad de intercambio entre bits se le agrega un nuevo recurso para realizar el control, agrupación y transmisión de los paquetes de salida de cada red. Entre las fuentes adicionadas están registros de desplazamiento para los paquetes recibidos y a enviar de tamaño b. Una memoria RAM para almacenar las entradas utilizadas para generar las salidas durante el

empaquetamiento y otra memoria con las habilitaciones de los campos intermedios. Registros convertidores de paralelo a serial para cada memoria RAM. Contadores para determinar el tamaño de paquetes y contar las transferencias de paquetes con el exterior. Para conformar el paquete se realimenta negativamente mediante un multiplexor la señal de salida de la red y en los casos de aprendizaje la salida actual de la red es la salida del vector de salidas donde se almacena el paquete a transmitir. Las transiciones del controlador de la variante se ilustran en la Figura 25.

Al analizar mediante el programa de simulación MATLAB® las características del proceso de sincronización mediante la transferencia de paquetes de bits de salida, se evidencia en la Figura 26 la dependencia de los parámetros de la red al cambio del tamaño del paquete y se muestra una dependencia fuerte de la cantidad de neuronas de entrada al tamaño del paquete. A partir de un parámetro $N > 49$ la eficiencia del sistema no aumenta, por lo cual la variante no seria un buen método de transmisión para dicha configuración de TPM.

En la Figura 27 se muestra que para un tamaño de paquete de $b < 30$ y al incrementar la cantidad de neuronas intermedias se incrementa exponencialmente el tiempo de sincronización promedio y se disminuye la cantidad de paquetes a intercambiar entre las redes para alcanzar la sincronización. Para paquetes con tamaño de $b > 30$ la curva muestra un comportamiento incremento lineal. Esto ocurre de igual manera para variaciones del parámetro L, lo cual esta documentado en la Figura 28 Tambien de la figuras se deduce que la cantidad de paquetes intercambiados disminuye al aumentar el tamaño del paquete para variaciones en los parámetro K y L.

Figura 26. Tiempo de sincronización promedio y cantidad de transferencias a variaciones del parámetro de neuronas de entrada ***N***



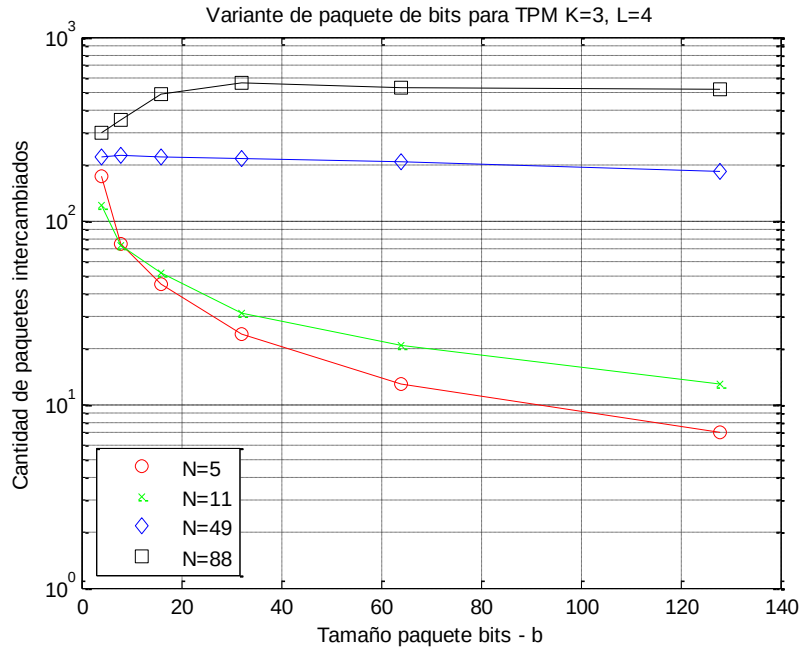
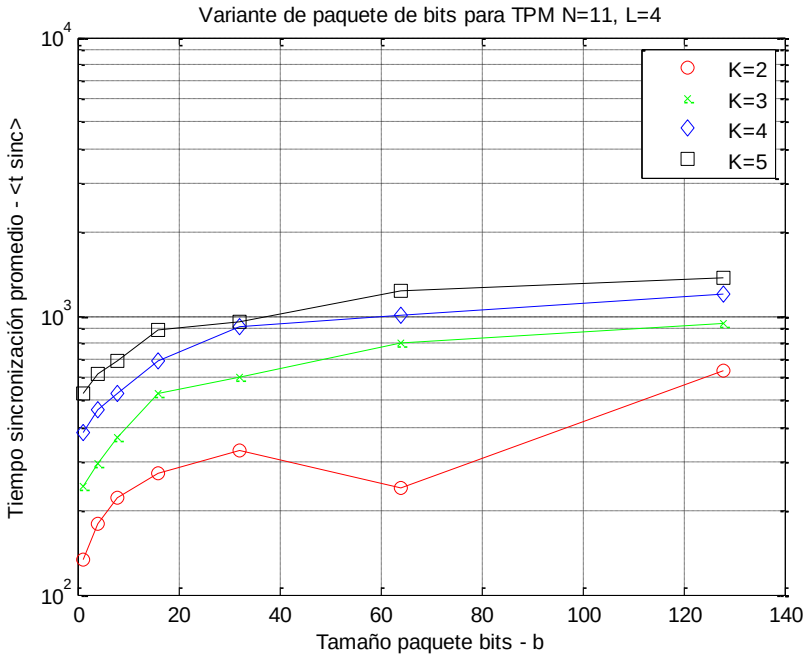


Figura 27. Tiempo de sincronización promedio y cantidad de transferencia de paquetes a variaciones del parámetro de neuronas intermedias



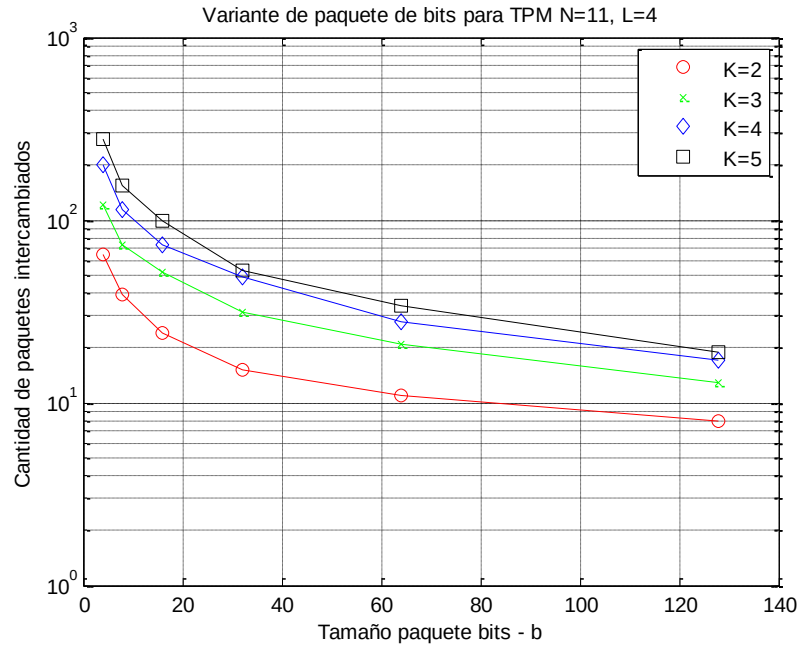
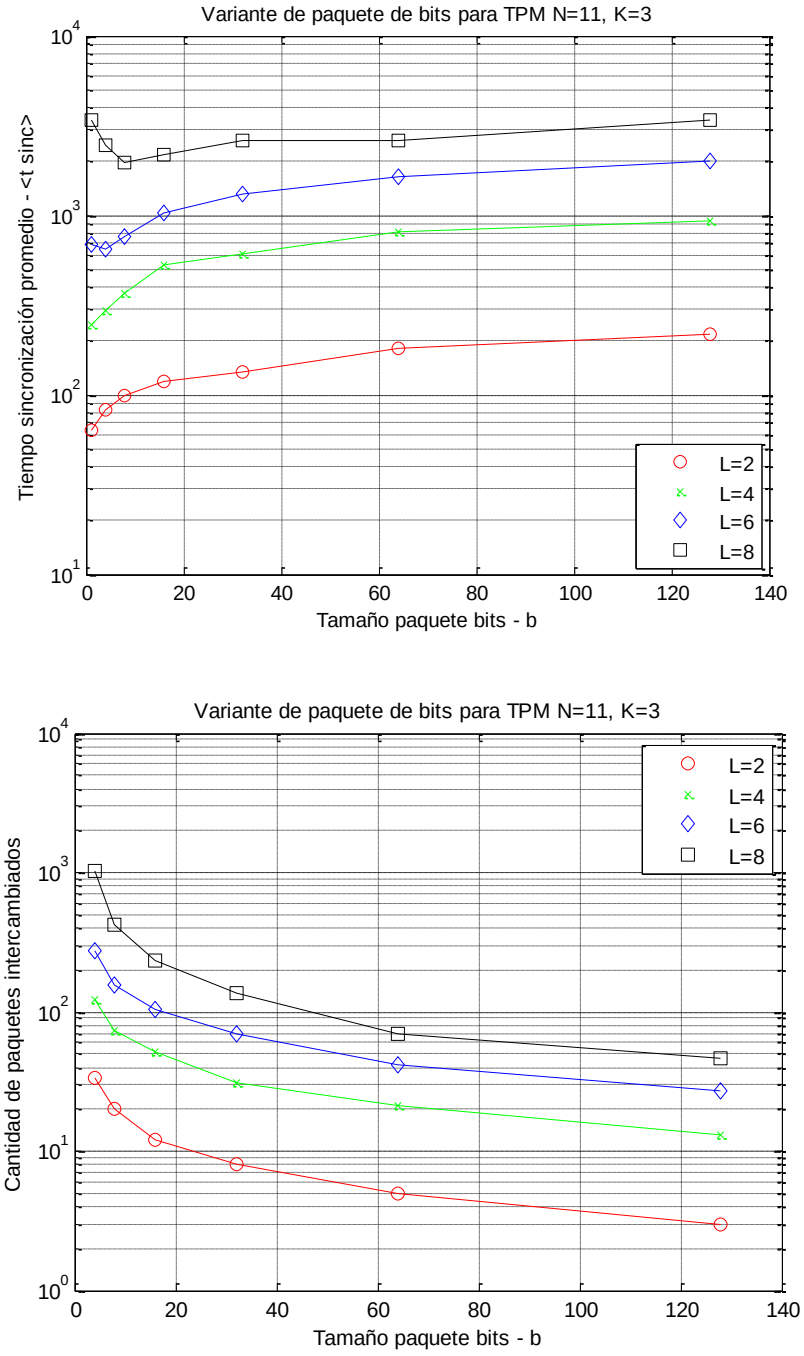


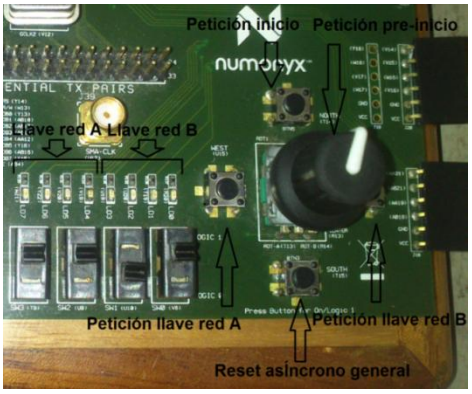
Figura 28. Tiempo de sincronización promedio y cantidad de transferencia de paquetes a variaciones del parámetro de profundidad de peso



4. ANÁLISIS Y RESULTADOS

Al final de la etapa de diseño se implementa un sistema de intercambio de llave simétrica con los procesadores descritos en el capítulo 3 y se verifican los resultados obtenidos del simulador Isim del programa ISE® versión 13.1 del fabricante Xilinx contrastados con el comportamiento de la descripción hecha en VHDL para la tarjeta *Spartan* 3AN y FPGA XC3S700AN. Mediante el simulador se puede verificar paso a paso del algoritmo de sincronización mutua en los cuatro niveles de simulación de la descripción diseñada y comprobar que las operaciones realizadas son las definidas por la FSM. En el archivo de construcción .UCF según [21] se definen cinco señales de entrada para el sistema de establecimiento correspondiente a las entradas de *reset*, petición de pre inicio, petición de inicio, petición de llave de la red A y petición de llave de la red B como se muestra en la Figura 29.

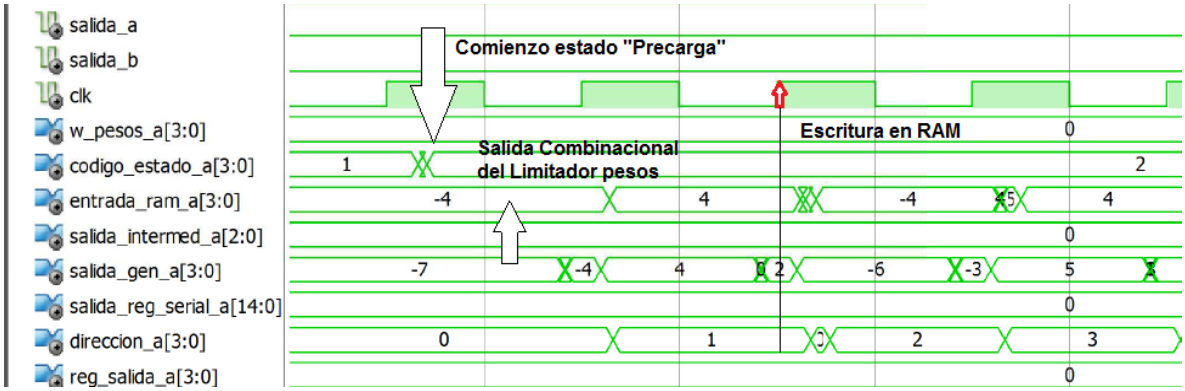
Figura 29. Descripción de las entradas y salidas en la tarjeta de desarrollo



Con cada *reset* asíncrono hecho por el usuario el circuito digital se ubica en el primer estado de la FSM llamado estado de espera, donde se prepara para

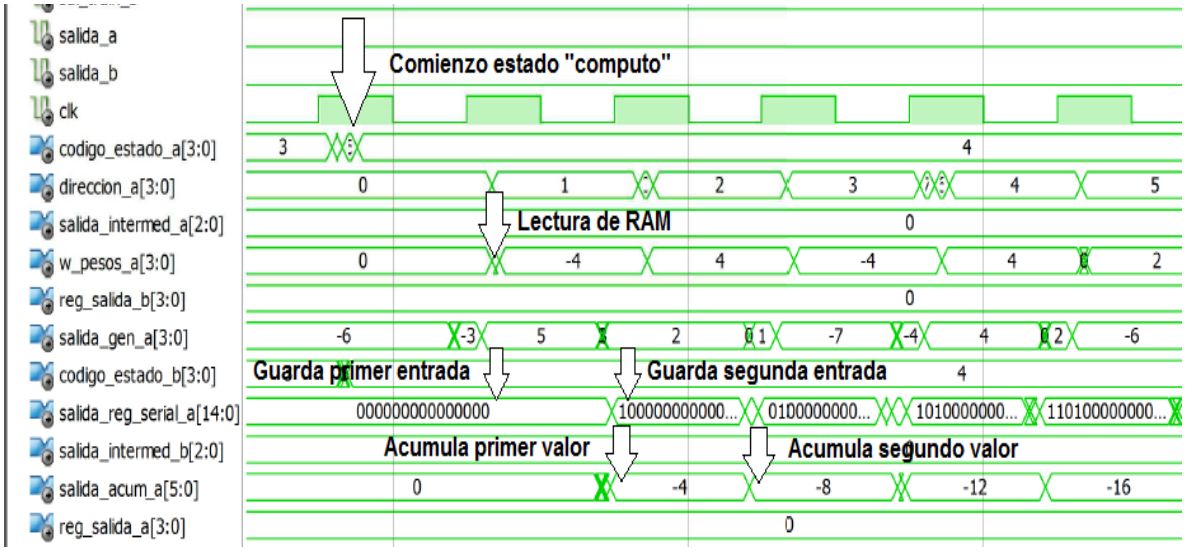
realizar el primer paso del algoritmo, el cual es inicializar la red con los valores de los pesos aleatorios mediante el estado precarga. Los generadores de PRBS se cargaron con dos semillas diferentes y al oprimir el botón de petición de pre inicio se comienza a desplazar el registro LFSR que genera un vector de bits pseudo aleatorio llevado a un módulo combinacional que realiza la reflexión de límites de acuerdo al parámetro L cuya salida se escribe en la memoria RAM con cada ciclo de reloj (código de estado 2 de la Figura 30).

Figura 30. Diagrama de tiempos de escritura en RAM en el estado de precarga



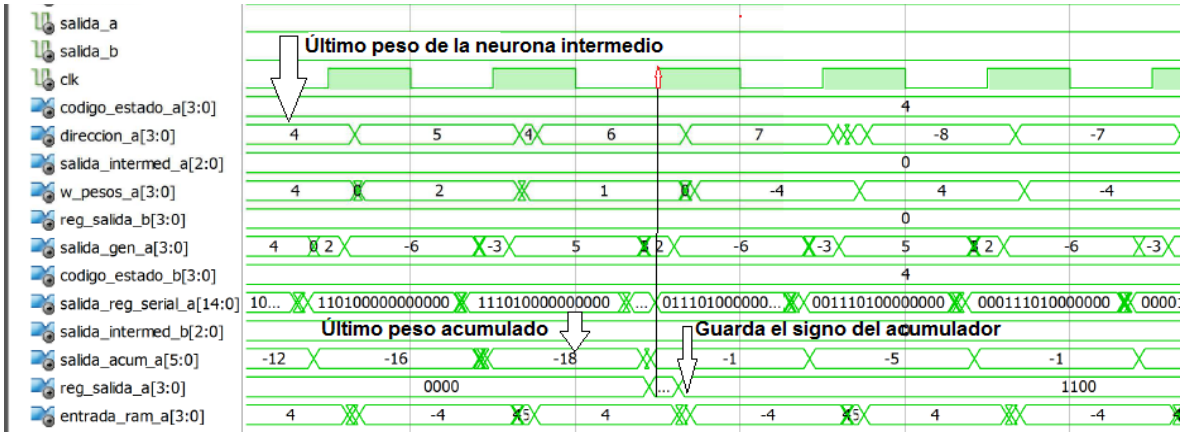
Al finalizar la escritura de la memoria RAM se pasa a un estado listo (código de estado 3) donde se carga una semilla igual en ambas redes para generar las entradas de entrenamiento y queda en espera de la señal de petición de inicio, que se da en los procesadores de la red para comenzar el proceso de sincronización mutua. Se da comienzo a este proceso en un estado computo (código de estado 4) correspondiente a la evaluación interna de la red neuronal, para ello se desplaza el registro LFSR y se guarda este bit en un registro serial. Se lee la memoria RAM y se extrae el valor del peso almacenado previamente, luego pasa a un bloque de inversión dependiendo del bit de entrada generado, para luego ser acumulado por el módulo correspondiente como se muestra en la Figura 31.

Figura 31. Diagrama de tiempos de acumulación de pesos en el estado de computo



El contador de direcciones tiene módulo KN y recorre la memoria RAM en los estados de precarga y computo; cada N direcciones corresponden a una neurona oculta de la red, por lo tanto al llegar a estos puntos se guarda el bit de signo del acumulador en un registro y se reinicia el conteo del acumulador. El bit de signo corresponde al valor del campo intermedio de la red y el registro se va desplazando a la vez que se va capturando el signo del acumulador en el registro serial paralelo (ver Figura 32), el cual calcula la salida de la red conforme se va almacenando la información por medio de la operación $\tau_{(t)} = y_{j-1} \oplus y_j$. Al finalizar el ciclo de computo se tiene el registro con la salida de la red seguido de los campos intermedios, a partir de este vector se genera un registro adicional correspondiente a los habilitadores del módulo de actualización donde $y_j == \tau_{(t)}$, resultado de la operación $y_j \odot \tau_{(t)}$.

Figura 32. Diagrama de tiempos de carga del signo del acumulador en el estado de computo

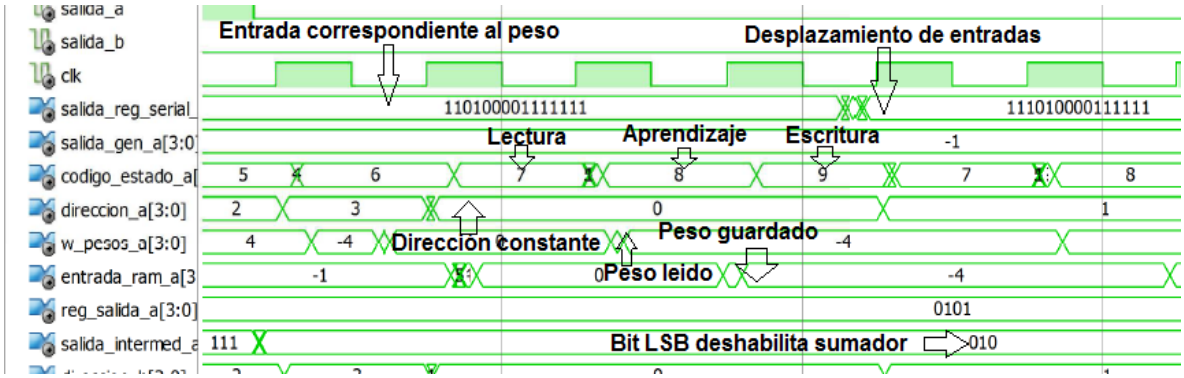


Un estado análisis (código de estado 5) espera al último valor acumulado y a que la red contraria salga del estado de ocupado en caso que por alguna razón tarde más en computar su valor interno. Al tener listas ya las salidas se pasa a un estado de activación del *watchdog* (código de estado 6) que compara las salidas y en caso de ser diferentes se vuelve al estado de computo donde se generan nuevas entradas, si son iguales se incrementa el contador de sincronizaciones y se pasa al estado de entrenamiento compuesto de tres estados: lectura, aprendizaje y escritura (códigos de estado 7, 8 y 9 respectivamente).

Durante estos tres estados visualizados en la Figura 33 se mantiene el valor del contador de direcciones y en el estado de lectura se lee la memoria RAM correspondiente al peso W_{ij} y pasa al siguiente estado de aprendizaje en el cual se genera parte de la regla de aprendizaje correspondiente a la operación de la salida de la red y la entrada X_{ij} de dicho peso, esta operación puede ser definida por el usuario para cambiar la regla de aprendizaje cambiando un parámetro de configuración. Además, se activa el sumador restador dependiendo del registro de activación valido para cada N datos leídos de la memoria; se desplaza el registro

de activación al finalizar la última neurona del campo intermedio. Para terminar el entrenamiento se pasa al estado de escritura de la RAM donde se almacena el peso actual que paso por el bloque sumador restador. Luego de pasar NK veces estos tres estados se compara si se llega al límite del contador de sincronizaciones, si es así se da por terminado el entrenamiento, si no es así se repite todo el proceso desde el estado de computo.

Figura 33. Entrenamiento de la red en los estados lectura, aprendizaje y escritura



Los estados de la implementación mostrados en los diagramas de tiempo pueden ser medidos en ciclos de reloj, los cuales dependen de los parámetros de la red. Es decir que en función de los parámetros configurados N, K y L, se puede determinar la demora de cada estado del proceso de sincronización, esto debido a la dependencia de cada fuente en VHDL a los parámetro de la red. El estado de pre inicio y el de cómputo tardan NK ciclos de reloj y el estado de análisis tarda 3 ciclos de reloj. Para la aplicación de la regla de aprendizaje se inicia en un estado de activación del *watchdog* que tarda un ciclo y se luego se repiten tres estados NK veces que duración cada uno un ciclo de reloj, por lo tanto el aprendizaje tiene una duración de $3NK + 1$ ciclos de reloj del sistema. Los demás, son estados de espera cuya duración depende de la activación del usuario o de la comunicación entre las redes.

4.1. INDICADORES DE TIEMPO ASOCIADOS AL SISTEMA

Se verifica experimentalmente que el comportamiento de la implementación corresponde con las simulaciones del Isim® realizando pruebas variando los parámetros de red N, K y L alrededor de los tiempos de sincronización promedios para cada estructura. Se realizan diez experimentos en el programa de simulación donde al comparar el valor de los pesos de la implementación en diferentes tiempos de sincronización se consigue llegar a la sincronización mutua mediante el aprendizaje *hebbiano*. Se toma un promedio de estas pruebas a variaciones de los parámetros de las redes y se muestran en la Figura 34 y la Figura 35. Como se aprecia en la Figura 34 punto de sincronización promedio oscila alrededor de los 350 pasos de sincronización, debido al punto de caída mostrado en la Figura 13, por lo cual para este aprendizaje es más probable sincronizar redes con número de neuronas de entrada alrededor de $N = 11$, después de este punto el crecimiento es proporcional.

Figura 34. Probabilidad de éxito de sincronización de dos redes TPM variando el parámetro ***N***

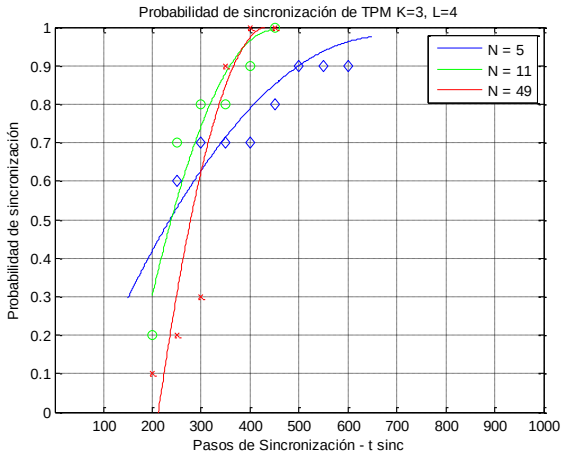
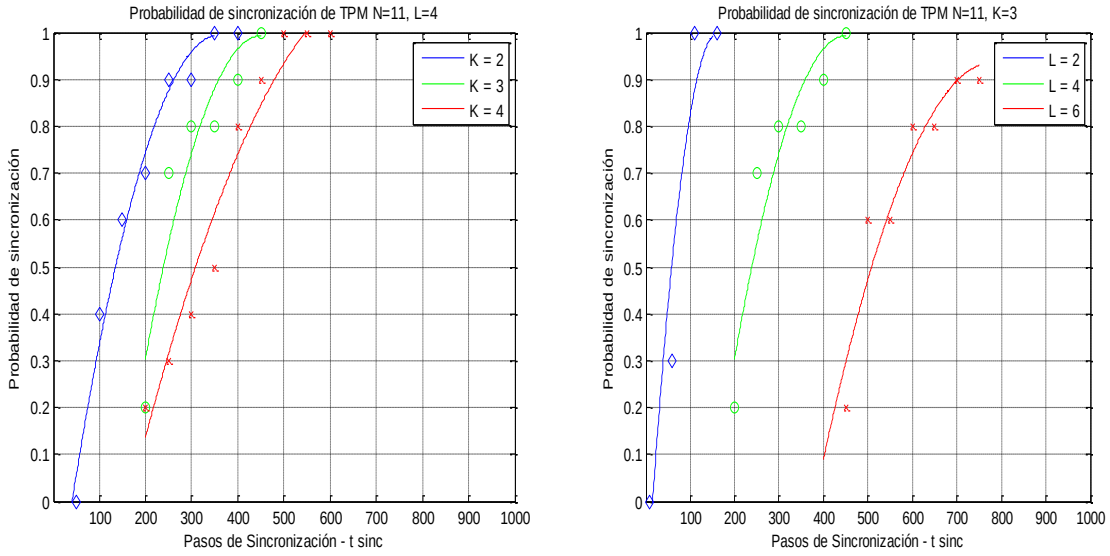


Figura 35. Probabilidad de éxito de sincronización de dos redes TPM variando el parámetro $\boldsymbol{K}$ y $\boldsymbol{L}$ de la red a diferentes tiempos de sincronización.



En la Figura 35 se ilustra el comportamiento de la implementación a variaciones de los parámetros de red K y L, donde se evidencia el crecimiento proporcional del punto de sincronización promedio al aumentar un parámetro en específico. Se concluye además que 200 pasos por encima del tiempo promedio de sincronización puede asumirse que es probable que las redes se encuentren sincronizadas. Sin embargo, estos puntos son sólo un criterio de parada para un sistema basado en sincronización mutua, ya que al no ser un proceso determinístico nunca puede asegurarse un 100% de probabilidad de alcanzar la sincronización. Por lo cual un usario debe verificar con un mensaje de prueba que las redes estan realmente sincronizadas, y si no lo estan repiten el proceso de establecimiento de llave.

Tabla 7. Ciclos de reloj promedio requeridos

Neuronas entrada N	***Neuronas Ocultas*** K	***Profundidad de pesos*** L	***Pasos de sincronización promedio***	***Ciclos de reloj promedios***
5	3	4	403	28.530
11	3	4	246	40.216
49	3	4	303	221.769
88	3	4	339	467.587
11	2	4	133	15.628
11	4	4	383	83.674
11	3	2	63	9.792
11	3	6	682	109.570

Se concluye del análisis del proceso de sincronzacion y de la Tabla 7 que para cada conjunto de parámetros de red K, N y L se debe seleccionar una constante diferente para el módulo del contador de sincronizaciones (*watchdog*), donde se deduce que es altamente probable que el proceso se haya realizado exitosamente. Estos tiempos de sincronización son etapas donde se llevo a cabo un movimiento de los estados de la red y para llevar a cabo este movimiento el procesador ha de pasar por los estados de computo y aprendizaje al menos una vez. Aunque desplazar los pesos pueda demorar alrededor $4NK + 5$ ciclos de reloj sin considerar retardos en la transmisión, el proceso no puede ser determinado con certeza ya que hay retrasos por el cómputo de la entradas que no son útiles para el proceso, por lo cual para llegar al punto recomendado de sincronización son requeridos unos ciclos de reloj promedio medidos en la implementación realizada.

El programa ISE® genera un reporte de la implementación descrita en VHDL con las consideraciones de los tiempos de retardo generados por las conexiones internas del circuito integrado, mostrando un análisis hecho para definir la frecuencia máxima donde el diseño es funcional, cumpliendo la ecuación (8), siendo t_{pcq} el tiempo de retardo a la salida de un FF, t_{pd} el mayor retardo de la

parte combinacional y t_{setup} el tiempo donde la entrada de un FF debe permanecer constante antes del flanco de activación del mismo para que no caiga en metaestabilidad[13]. Compilando la información generada de la descripción de la red TPM se obtiene una frecuencia máxima de trabajo del núcleo en el FPGA para cada red configurada por el usuario, como se muestra en la Tabla 8.

$$F_{máx} = \frac{1}{t_{pcq} + t_{pd} + t_{setup}} \tag{8}$$

Tabla 8. Frecuencia máxima de operación de la red TPM

Parámetros			**Frecuencia máxima de operación [MHz]**						
N	K	L	$b = 1$	$b = 4$	$b = 8$	$b = 16$	$b = 32$	$b = 64$	$b = 128$
5	3	4	127,372	117,600	115,295	111,727	109,175	124,839	124,008
11	3	4	127,291	116,048	114,927	119,019	114,953	116,167	115,975
49	3	4	116,662	116,052	113,328	111,359	108,141	103,959	107,784
88	3	4	116,009	115,420	115,420	115,420	114,492	114,360	109,016
11	2	4	129,166	118,565	116,222	112,597	113,608	121,071	123,442
11	4	4	127,486	126,534	125,203	118,189	111,269	120,173	115,227
11	3	2	130,736	115,525	114,927	120,207	116,303	116,167	115,975
11	3	6	122,100	116,048	114,927	119,603	116,303	114,820	115,975

Para la variante de paquetes las conexiones internas de la FPGA se extienden aun más en el circuito integrado, lo cual incrementa el tiempo de retardo entre unidades funcionales, por lo cual al analizar los reportes se presenta una caida en a frecuencia máxima de operación a incrementos del tamaño del paquete. Tambien influyen la sintetización que hace el programa de la descripción hecha en VHDL, ya que como se expone en la sección 4.2 y se muestra en el Anexo 2 el compilador posee diferentes formas para implementar dependiendo de la disponibilidad de los recursos y se observa en el reporte que el programa usa

[13] Metaestabilidad es el estado de un circuito digital secuencial en donde no es posible determinar el valor de la salida en un instante dado.

bloques de memoria RAM al tener un exceso memoria distribuida, esto conlleva finalmente a una conexion diferente del FPGA.

Tabla 9. Ciclos de reloj promedio para llegar a la cantidad de paquetes promedio

Parámetros			**Ciclos de reloj promedio**					
N	K	L	$b = 4$	$b = 8$	$b = 16$	$b = 32$	$b = 64$	$b = 128$
5	3	4	41.764	290.50	37.439	36.800	43.858	44.377
11	3	4	48.201	63.385	88.786	111.762	144.276	172.990
49	3	4	443.269	988.560	2.033.296	4.040.963	7.724.446	13.912.664
88	3	4	1.129.221	4.380.330	8.163.259	#	#	#
11	2	4	19.813	23.037	28.627	36.461	54.429	77.911
11	4	4	107.601	130.360	149.254	219.929	230.941	279.846
11	3	2	14.805	17.149	20.342	26.469	32.757	40.044
11	3	6	112.281	123.812	187.214	244.695	308.158	380.299

Se ilustra en la Tabla 9, la cantidad de ciclos de reloj requeridos para llegar al promedio de transferencias recomendadas promediadas en la Figura 26, Figura 27 y Figura 28 para 100 pruebas de cada configuración de red. En la Tabla 10 se expone en dicho punto cuantos pasos de sincronización se han llevado a cabo. En estas tablas se complementan con la información de la frecuencia máxima establecida por los reportes en la Tabla 8, así se puede calcular el tiempo real que tardaría estos procesadores en alcanzar los puntos recomendados para cada configuración. No se pudieron simular las redes de parámetros $N = 8, K = 3$ y $L = 4$ con tamaños de paquete $b \geq 32$ (definidos como # en la Tabla 9 y la Tabla 10) debido a que el archivo .WDB del simulador llega al número de datos máximos permitidos por el programa.

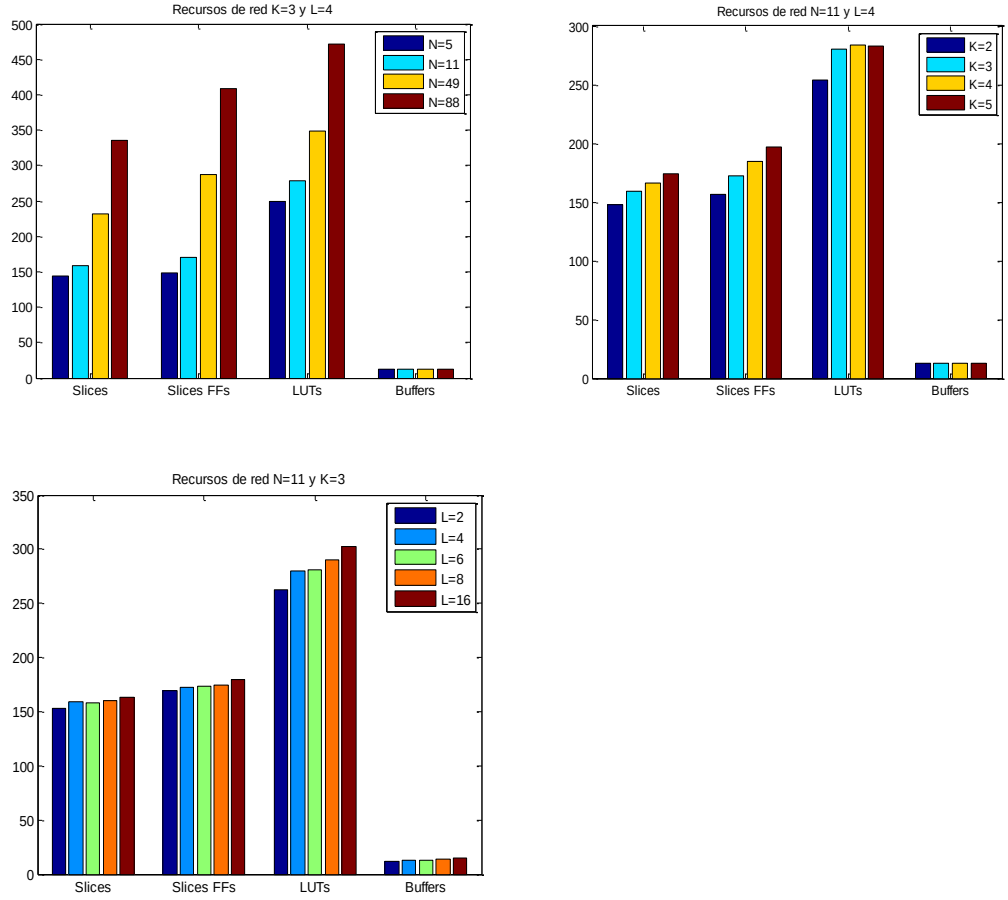
Tabla 10. Sincronizaciones promedios para llegar a la cantidad de paquetes promedio

Parámetros			Sincronizaciones promedio					
N	K	L	$b = 4$	$b = 8$	$b = 16$	$b = 32$	$b = 64$	$b = 128$
5	3	4	551	356	460	417	527	480
11	3	4	285	396	547	708	878	1.015
49	3	4	691	1.600	3.345	6.697	12.837	23.181
88	3	4	1.014	4.065	7.597	#	#	#
11	2	4	179	204	254	322	481	668
11	4	4	495	622	667	1.042	1.025	1.222
11	3	2	91	104	120	150	175	198
11	3	6	677	735	1.191	1.550	1.981	2.371

4.2. INDICADORES DE RECURSOS ASOCIADOS AL SISTEMA

Del reporte de síntesis del programa ISE® de la implementación descrita para el procesador TPM con diferentes parámetros de red podemos extraer la información acerca de los recursos utilizados por la FPGA. En el reporte hay descripciones a diferentes niveles de integración: reporte estándar, avanzado y de bajo nivel de síntesis; mostrando la configuración interna de las unidades utilizadas en cada descripción como se mostró en la sección 2.2. En la Figura 36 se muestra la variación en los bloques utilizados en la implementación de intercambio entre bits de salida ante cambios de parámetros de red.

Figura 36. Recursos usados de la FPGA para implementación TPM variable a N, K y L



De la Figura 36 se puede inferir la dependencia directa entre recursos de la implementación y variaciones del parámetro N, cuyo incremento tiende a ser exponencial en todos los recursos. Para el caso de K y L incremento de recursos no es tan drástico como en el caso de N, además el uso de *buffers* por cualquiera de los sistemas es constante para variaciones de N y K, ya que este recurso depende directamente de L.

En la familia 3AN del FPGA *Spartan*® están disponibles LUTs de hasta 4 entradas y la conexión interna que hace el sintetizador depende de las necesidades del diseño, utilizando LUTs de tipo local (LUT_L) cuando las conexiones son dentro de CBL, o LUT de tipo global (LUT_D) cuando se interconectan LUT de diferentes CBLs. Los LUTs en una FPGA pueden ser usados para implementar funciones lógicas, registros internos o memoria RAM distribuida, como se especifica en el reporte de síntesis en el apartado de resumen de utilización del dispositivo [22]. El dispositivo posee ademas recursos de tipo secuencial para implementar memorias y registros de diferentes tipos según la descripcion hecha por el usuario en el programa. Para ello configura registros denominados que son implementados a base de LC o *slices,* y también posee bloques de memoria RAM alrededor de los CBLs configurables por el sintetizador (para mayor descripción ver Anexo 2).

Cuando se evalúa el costo de una implementación basada en la variante de paquetes se tiene en cuenta la carga de los recursos adicionales mostrados en la Figura 37 y como se muestra en las gráficas de la Figura 38 hay un salto en el incremento de recursos entre $b = 1$ y $b = 4$. En el reporte final de sintetización se describe el uso de recursos de tipo combinacional en elementos básicos lógicos (BELs) como compuertas lógicas y exclusivas, multiplexores y LUTs; y de recurso secuencial como FFs, registros y memoria. En la Figura 4.9 se hace un paralelo para la variación de parámetros de la red de BELs y FFs usados, mostrando un incremento de alrededor de 250 BELs para variaciones de los tres parámetros de red. En el caso de los FFs el aumento depende de cada parámetro y del uso de los bloques de memoria RAM cuando se agotan los LUTs como memoria distribuida. Para cambios en el parámetro L se requiere aproximadamente 100 FFs de más entre una implementación de intercambio entre bit a bit y una implementación de variante de paquetes.

Figura 37. Uso de BELs y FFs a cambios de configuración de la red

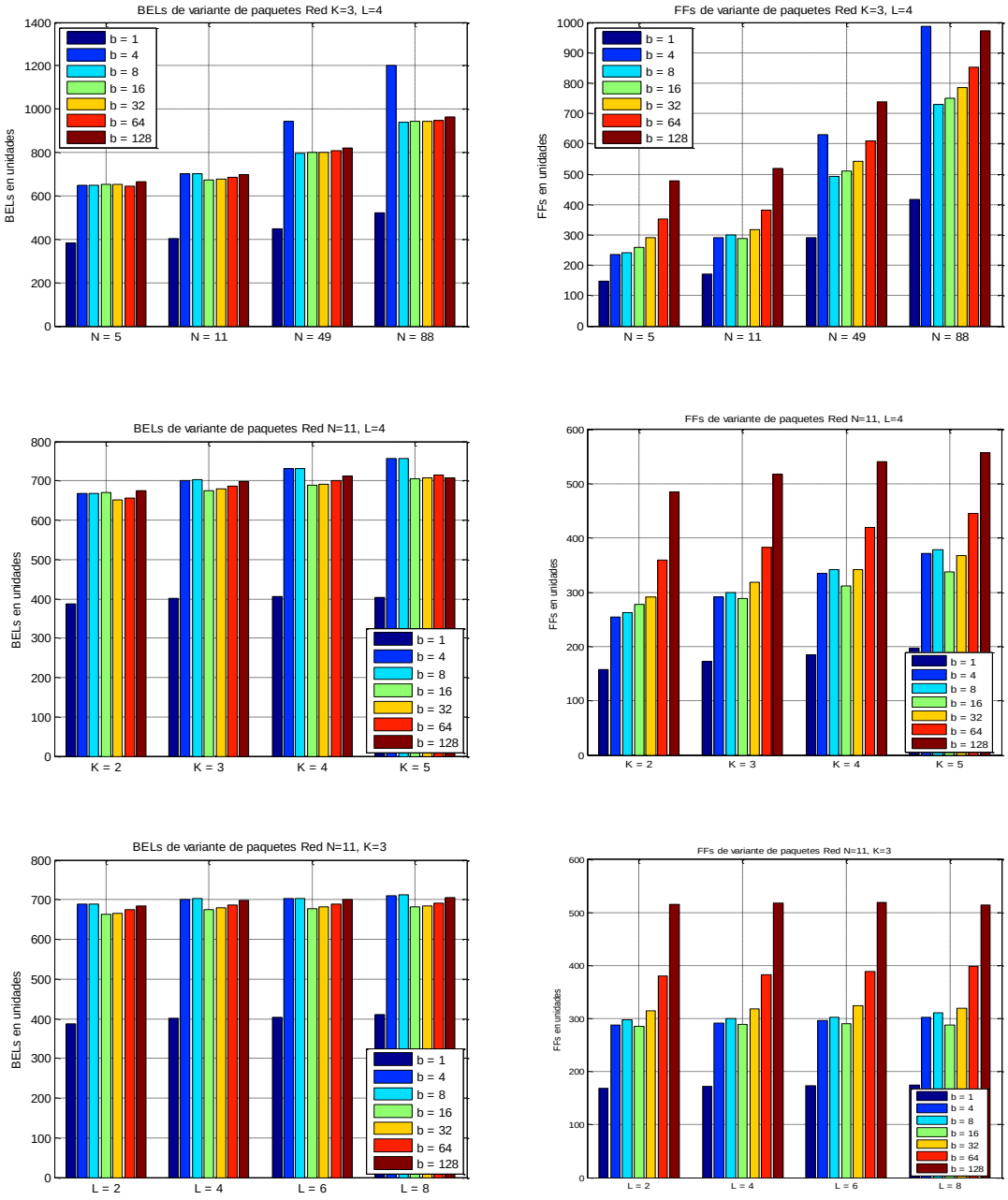


Figura 38. Uso de LUTs y *slices* a cambios de configuración de la red

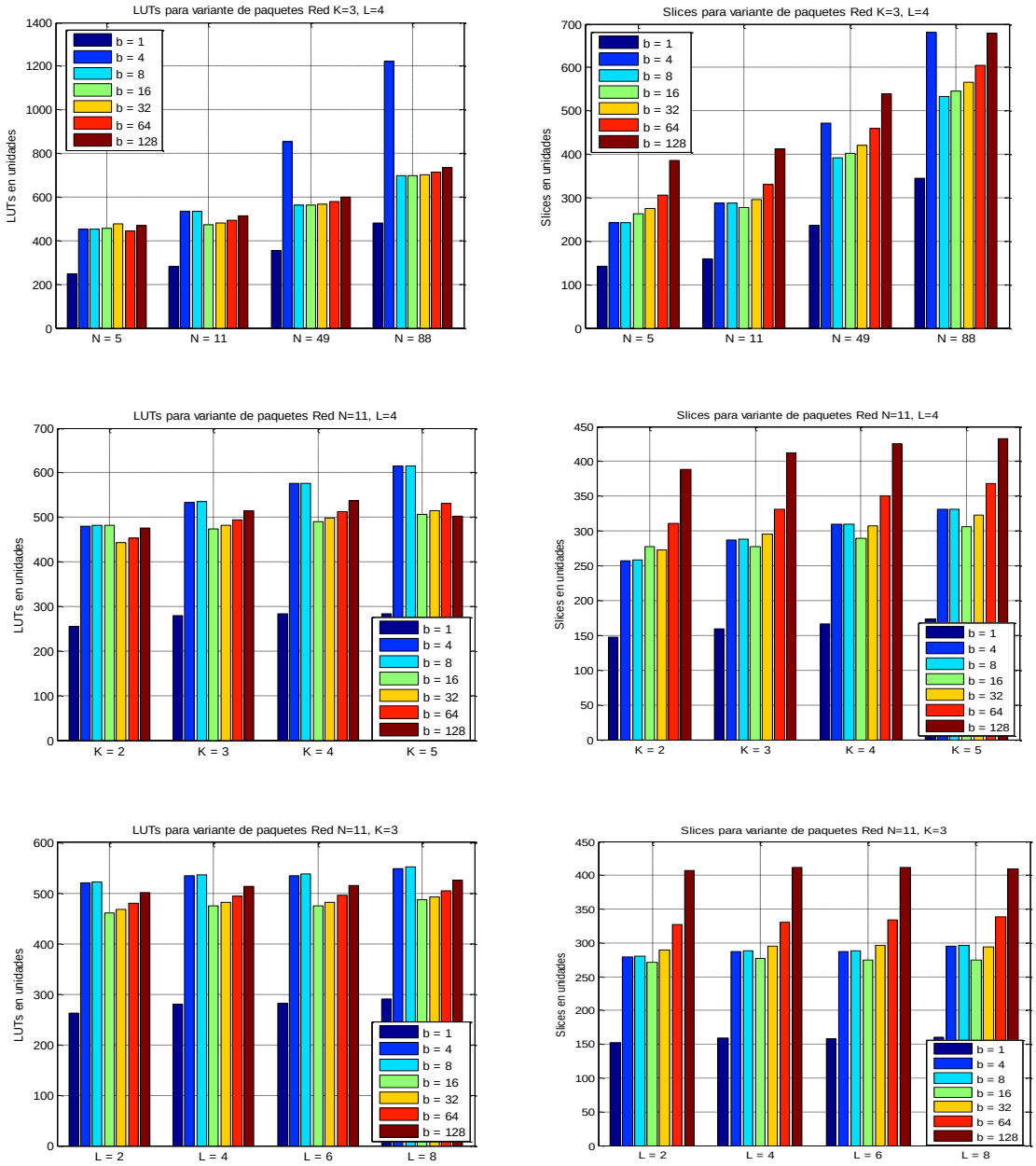
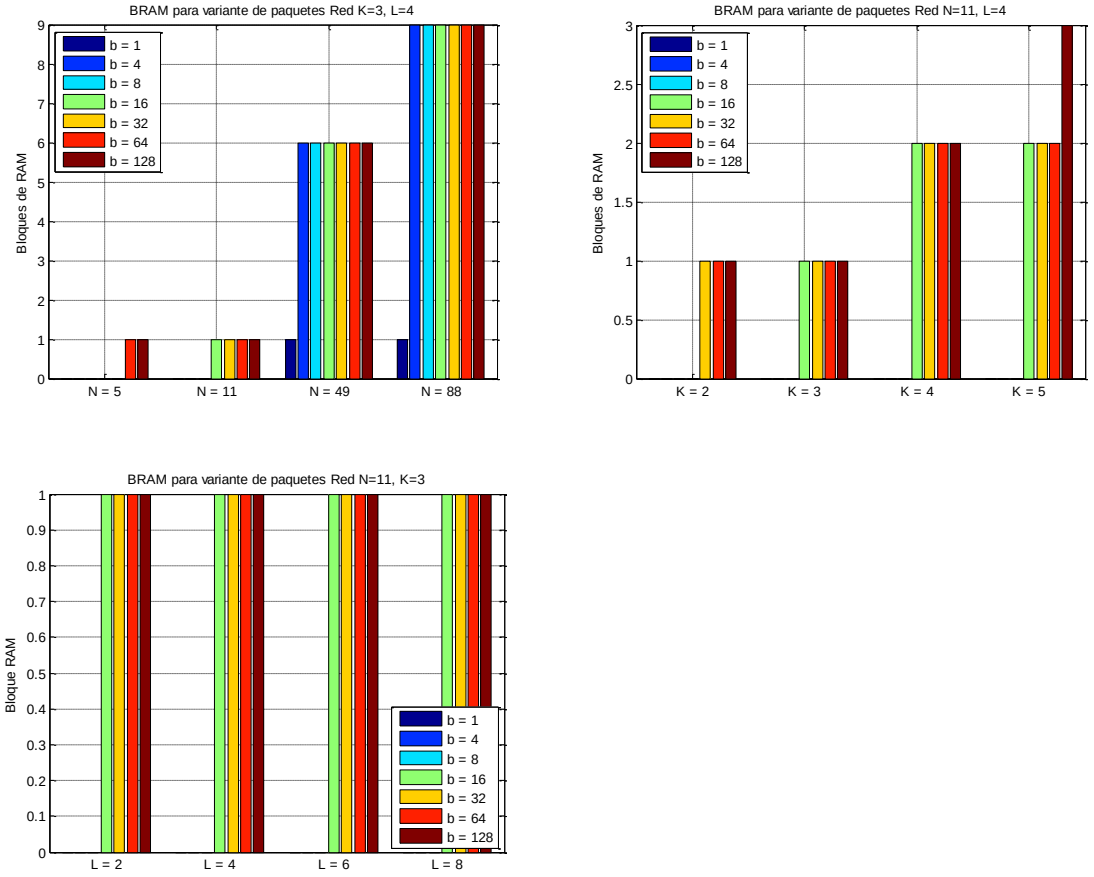


Figura 39. Uso de BRAM a diferentes configuraciones de red



En el reporte de utilización del dispositivo se resume el diseño en VHDL de acuerdo a los elementos de la FPGA descritos en la hoja de datos. En las gráficas de la Figura 39 se da un resumen de los recursos utilizados para la implementación de procesadores a diferentes parámetros de red, en términos de LUTs y *slices* totales utilizados. Recordando que para el circuito integrado referencia XC3S700AN posee 13.248 LUTs y 5.888 slices disponibles.

En las implementaciones de la Figura 39 donde la carga comienza a ser significativa para la FPGA se sintetizan con bloques de memoria BRAM alojada alrededor del circuito integrado, cuando esto ocurre se aumenta la frecuencia

máxima de operación y se disminuye el uso de otros recursos. La memoria BRAM usada es la referencia RAMB16BWE (Anexo 2), la cual es una memoria de SRAM doble puerto de 16kb configurable y de diseño específico, definido como macro recurso de la FPGA. Existen también puntos como en la Tabla 8 donde para parámetros de red $N = 11, K = 2$ y $L = 4$ y tamaño de paquete $b = 128$ se aprovecha al máximo los recursos internos de cada bloque de *slices* en el sistema y se ve un aumento en la frecuencia de operación.

5. CONCLUSIONES

- El diseño del circuito digital en VHDL fue hecho para un sistema de *datapath* síncrono y máquina de estados tipo Mealy con las fuentes descritas de forma tal que puedan ser sintetizadas por el usuario al cambiar los parámetros de red $N > 2$, $K \geq 2$ y $L \geq 2$ o las constantes de sincronización del *watchdog*. Los parámetros que determinan la estructura de la red y la llave establecida del sistema puede ser la asociada a los pesos sinápticos almacenados en la RAM del procesador luego del proceso de sincronización mutua, el cual se da por concluido cuando se alcanza el tiempo promedio de sincronización para cada conjunto de parámetros de red. Los aprendizajes hebbiano y de pasos aleatorios mostraron comportamientos estables y más rápidos que las redes entrenadas con aprendizaje anti-hebbiano, por lo cual se definió al primero como la mejor opción de entrenamiento para la implementación. El aprendizaje hebbiano muestra un punto mínimo para valores cercanos a $N = 11$ donde se obtiene el menor tiempo promedio de sincronización de 246 pasos, para las demás variaciones de parámetros de red el comportamiento es proporcional. El parámetro N es el que limita la transferencia de paquetes en la variante de bits, ya que para valores $N > 49$ la eficiencia del sistema disminuye y a pesar que se envíen más paquetes el tiempo de sincronización no se hace menor.

- Comparando los resultados al simular e implementar diferentes configuraciones de TPM con la literatura analizada acerca de los antecedentes de la neurocriptografía se encuentran similitudes en el comportamiento de la sincronización mutua para ANN y aunque el proceso

no es determinístico se encuentran tiempos de sincronización promedio que tienden a 400 pasos para variaciones de N como en los documentados en los artículos analizados, para cada configuración de red habrá un número de pasos promedios recomendados para finalizar el proceso de entrenamiento

- La distribución interna que hace el sintetizador no sigue una regla definida y sus conexiones se disponen depende de la ubicación de los recursos y de la necesidad del sintetizador de darle al usuario a una solución. Dependiendo de la cantidad de recursos que requiera un diseño, el programa puede sintetizar con bloques RAM para disminuir la carga asociada al uso de memoria distribuida de LUTs. Por lo anterior, es posible realizar comparaciones de recursos para la misma familia de FPGA, o entre las cuales se use la misma definición de sus elementos básicos..

- Se define un intervalo para la frecuencia máxima del reloj a que el sistema digital puede ser configurado siendo el límite mayor de 130 [MHz] para parámetros de red $N = 11, K = 3$ y $L = 2$ con tamaño de paquete $b = 1$ y el límite menor de cerca de 103 [MHz] para parámetros de red $N = 49, K = 3$ y $L = 4$ con tamaño de paquete $b = 64$ sin que el procesador caiga en estado de metaestabilidad, por lo cual el tamaño de la red no es proporcional a la velocidad del procesador. De la anterior declaración se estima que el reloj de un criptosistema basado en TPMs sobre una FPGA es de alrededor de 100 [MHz] siendo el tiempo promedio para el establecimiento de llave entre dispositivos con redes configuradas con los parámetros más grandes de 80 [ms]. La carga adicional aproximada al usar una variante de paquetes es de 250 BELs, 100 FFs y de 200 LUTs y 150 *slices*, la memoria en bloque se usa generalmente a partir de implementaciones para intercambios de paquetes de $b > 16$..

6. RECOMENDACIONES Y TRABAJO FUTURO

El anterior trabajo de investigación se muestra en forma general los resultados de las publicaciones más relevantes hechas en el campo de la neurocriptografía y busca comparar algunas de las características asociadas en el proceso de sincronización mutua de TPMs en la implementación sobre FPGAs. Además de dar a conocer los resultados de la síntesis sobre FPGA para que sean base para el desarrollo de ANN en futuras implementaciones en el campo de los sistemas digitales.

- Estos análisis abren la discusión acerca de la seguridad del uso de topologías de ANN como TPM [4] [3] y PPM [5] en la criptografía frente a diferentes tipos de ataques hechos por intrusos que obtienen la información compartida en el canal para replicar la llave secreta establecida. También se debe incluir en las implementaciones orientadas a aplicaciones el uso del protocolo de autenticación de los dispositivos como paso siguiente en el esquema del intercambio seguro luego del establecimiento de llave.

- Además, para que estas arquitecturas sean llevadas de forma exitosa a aplicaciones reales de comunicación ligera se debe integrar un generador de números aleatorios como fuente de carga inicial de las redes involucradas, ya que en el trabajo realizado se usó un generador de PRBS por LFSR. El grupo de investigación CPS está desarrollando trabajos de pregrado acerca de generadores aleatorios y su implementación en circuitos digitales.

- Luego de estos análisis y estudios el paso siguiente es incorporar estas tecnologías en sistemas de comunicación de capas como ETHERNET o sistemas inalámbricos de proximidad como los sistemas de RFID o NFC, donde la criptografía ligera aporta grandes ventajas al proceso de encriptación seguro.

Como complemento se hace una propuesta para versión alternativa del método de aprendizaje, basada en que una TPM es una forma generalizada de un perceptrón multicapas y el aprendizaje hebbiano y sus variantes son formas modificadas del entrenamiento recursivo con regla de aprendizaje delta.

Un perceptrón es un tipo de red neuronal que puede entrenarse con supervisión por entrenamientos recursivos [23] según el siguiente algoritmo:

1. Se fijan los pesos sinápticos iniciales con valores aleatorios

2. Se establecen los valores de las neuronas de entrada X_n

3. Se calcula la salida del perceptrón según su fórmula de activación

$$y_{(t)} = f_{act}\left(\sum_{i=1}^{N} X_i * W_i + \theta_{BIAS}\right)$$

4. Se actualizan los pesos de acuerdo a la regla de aprendizaje delta

$$W_{(t+1)} = W_{(t)} + \eta * X_{(t)} * e_{(t)}$$

Donde $e_{(t)=}\, y^*_{(t)} - y_{(t)}$ es el error de la salida dado por la diferencia entre el valor deseado $y^*_{(t)}$ y el obtenido $y_{(t)}$. Y η es la razón de aprendizaje que pertenece al intervalo $[0,1]$.

5. Se repite el procedimiento hasta que el error este dentro de un rango de tolerancia aceptable ξ.

$$e_{(t)} \leq \xi$$

Modificando los aprendizajes hebbiano, anti-hebbiano y de pasos aleatorios para actualizar todos los pesos en cada acuerdo de sincronización se puede disminuir el tiempo del proceso de sincronización.

1. Aprendizaje hebbiano modificado mediante fuerzas de atracción entre los pesos

$$W_{ij}^{A/B}(t) = g\left(W_{ij}^{A/B}(t-1) + X_{ij}^{A/B} * \tau^{A/B}\right)$$

2. Aprendizaje anti-h*ebbiano* modificado mediante fuerzas repulsivas entre los pesos

$$W_{ij}^{A/B}(t) = g\left(W_{ij}^{A/B}(t-1) - X_{ij}^{A/B} * \tau^{A/B}\right)$$

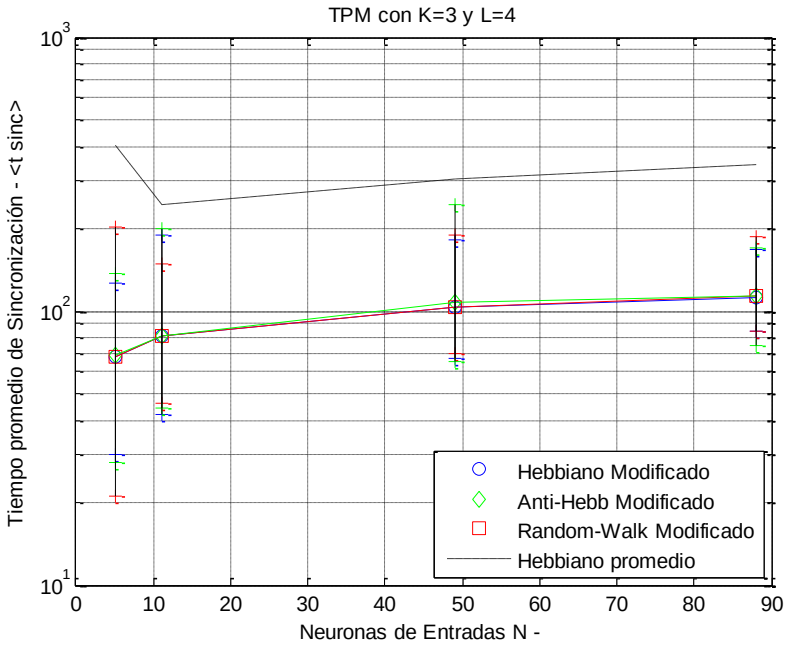
3. Aprendizaje de pasos aleatorios o *random-walk* modificado

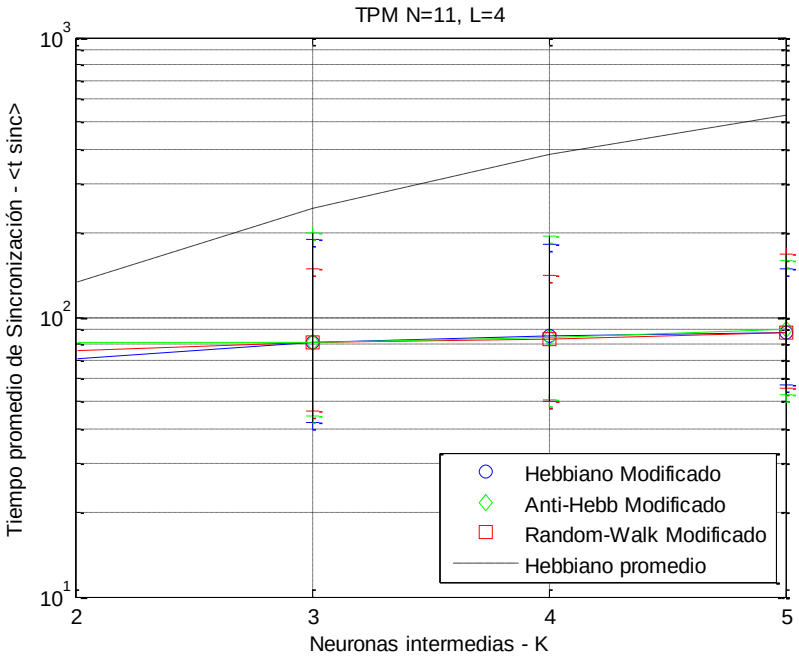
$$W_{ij}^{A/B}(t) = g\left(W_{ij}^{A/B}(t-1) + X_{ij}^{A/B}\right)$$

Donde la función $g(u)$ se define como:

$$g(u) = \begin{Bmatrix} sgn(u) * L, & para\ \ |u| > L \\ u \end{Bmatrix}$$

Figura 40. Tiempo de sincronización al aplicar la modificación a cambios de $\boldsymbol{N}$ $\boldsymbol{y}$ $\boldsymbol{K}$

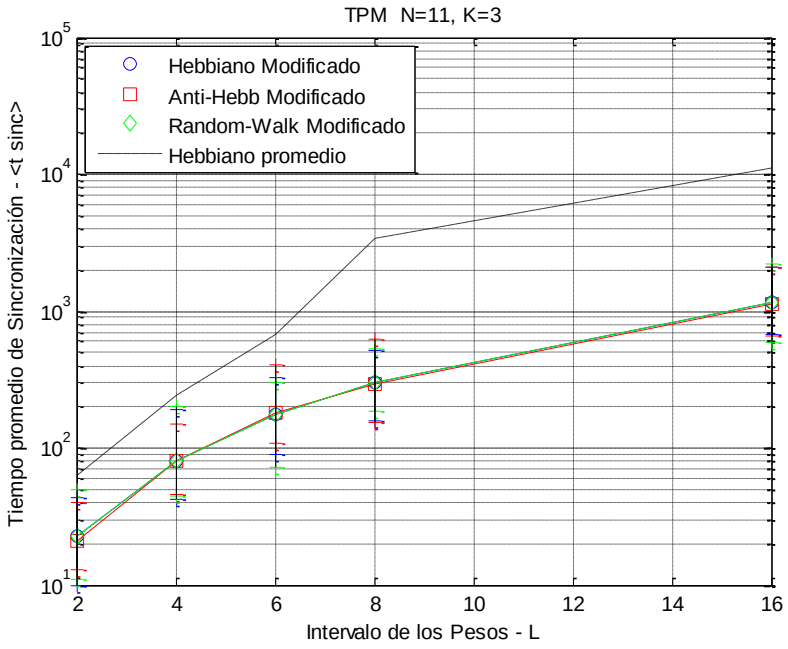




Utilizando el programa MATLAB® se realizan 100 pruebas para cada regla de aprendizaje hasta el momento en que se llega a la primera sincronización y se verifica el comportamiento de las redes a la modificación de la regla de

aprendizaje con los resultados mostrados en la Figura 40 y en la Figura 41 se puede ver que por medio de esta modificación se disminuye el tiempo de sincronización promedio de las TPM comparada con el valor promedio de la hebbiana ilustrada con líneas a trazos en las figuras.

Figura 41. Tiempo de sincronización al aplicar la modificación a cambios de L



De acuerdo a lo anterior, ignorando la seguridad que ofrece la regla y al actualizar todas las neuronas de entrada en vez de algunas pertenecientes a las neuronas intermedias cuyo valores son iguales a la salida de la red se disminuyen los pasos necesarios para llegar a la sincronización mutua, también se podría incrementar el número de neuronas del sistema o la profundidad de los pesos sinópticos del mismo para trabajar con arquitecturas más grandes o ubicar estas topologías de red en otras aplicaciones diferentes de la criptografía donde alcanzar la sincronización mutua en un tiempo menor sea una ventaja.

CITAS BIBLIOGRÁFICAS

[1] KARAMANIAN A., TENNETI S., DESSART F., "*PKI Uncovered certificate-based security solutions for next-generation networks*". Cisco Press, 2011.

[2] STALLINGS W., *Cryptology and Network Security: Principles and Practice*. Prentice Hall Editors, 1999.

[3] REYES O., "*Neuronal Synchronization and Light-weight Cryptography in Embedded System*." Shaker Verlag, 2012.

[4] KANTER I., KANTER E., KINZEL W., "*Secure Exchange of information by synchronitation of neuronal networks".* Europhys, 2002.

[5] REYES O., ZIMMERMANN K.-H., *"Key exchange protocol using Permutation Parity Machines*". INSTICC Press, 2010.

[6] VOLKMER M., SCHAUMBURG A., *Authenticated tree parity machine key exchange*. Europhysics Letters, 2004.

[7] VOLKMER M, WALLNER S., *Tree parity machine rekeying architectures*. IEEE Transactions on Computers, 2004.

[8] VOLKMER M, WALLNER S., *Tree parity machine rekeying architectures for embedded security*, 2005.

[9] VOLKMER M, WALLNER S., *A key establishment IP-Core for ubiquitous computing*. IEEE Transactions on Computers, 2005.

[10] VOLKMER M, WALLNER S., *Lightweight key exchange and stream cipher based solely on tree parity machine,* 2005.

[11] RUTTOR A., "*Neuronal synchronization and cryptografy"*. Disertación para el Doctorado en Ciencias de la Universidad Bayerischen Julius-Maximilians de la ciudad de Würzburg, Alemania, 2006.

[12] PONG C., "*FPGA prototyping by VHDL examples"*. Xilinx Spartan-3 version. John Wiley & sons, Inc., 2008.

[13] XILINX INC., "*Spartan-6 family overview*". Documento ds160 v2.0 (2011), disponible en la siguiente dirección web: http://www.xilinx.com/support/documentation/data_sheets/ds160.pdf

[14] XILINX INC., "*7 Series FPGAs configurable logic block"*. Documento ug474 v1.7 (2014), disponible en la siguiente dirección web: http://www.xilinx.com/support/documentation/user_guides/ug474_7Series_CLB.pdf

[15] XILINX INC., "*Spartan-3 generation FPGA user guide"*. Documento ug331 v1.8 (2011), disponible en la siguiente dirección web:

http://www.xilinx.com/support/documentation/user_guides/ug331.pdf

[16] XILINX INC., "*Spartan-6 FPGA configurable logic block"*. Documento ug384 v1.1 (2010), disponible en la siguiente dirección web: http://www.xilinx.com/support/documentation/user_guides/ug384.pdf

[17] FAJARDO C., RAMÓN J., Introducción al diseño digital avanzado. 2010.

[18] BOLUDA J., PARDO F., VHDL lenguaje para síntesis y modelado de circuitos. Editorial Ra-ma, 1999.

[19] HARRIS D., HARRIS S., *Digital design ad computer architecture*. Segunda edición, Morgan Kaufhann, 2012.

[20] XILINX INC., "*Linear feedback shift registers in Viryex devices*". Documento xapp210 v1.3 (2007), disponible en la siguiente dirección web:

http://www.xilinx.com/support/documentation/application_notes/xapp210.pdf

[21] XILINX INC., "*Spartan-3A/3AN FPGA starter kit board user guide*". Documento ug334 v1.1 (2008), disponible en la siguiente dirección web: http://www.gta.ufrj.br/ensino/EEL480/spartan3/ug334.pdf

[22] XILINX INC., "*Virtex-6 libraries guide for squematic Designs*" .Documento ug624 v14.7 (2013), disponible en la siguiente dirección web: http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_7/virtex6_scm.pdf

[23] HAYKIN S., *Neuronal Network: A comprehensive foundation*. Prentice Hall International Inc, segunda edición, 1999.

BIBLIOGRAFÍA

- BOLUDA J., PARDO F., VHDL lenguaje para síntesis y modelado de circuitos. Editorial Ra-ma, 1999.

- FAJARDO C., RAMÓN J., Introducción al diseño digital avanzado. 2010.

- HARRIS D., HARRIS S., *Digital design ad computer architecture*. Segunda edición, Morgan Kaufhann, 2012.

- HAYKIN S., *Neuronal Network: A comprehensive foundation*. Prentice Hall International Inc, segunda edición, 1999.

- KANTER I., KANTER E., KINZEL W., "*Secure Exchange of information by synchronitation of neuronal networks"*. Europhys, 2002.

- KARAMANIAN A., TENNETI S., DESSART F., "*PKI Uncovered certificate-based security solutions for next-generation networks*". Cisco Press, 2011.

- PONG C., "*FPGA prototyping by VHDL examples"*. Xilinx Spartan-3 version. John Wiley & sons, Inc., 2008.

- REYES O., "*Neuronal Synchronization and Light-weight Cryptography in Embedded System*." Shaker Verlag, 2012.

- REYES O., ZIMMERMANN K.-H., *"Key exchange protocol using Permutation Parity Machines*". INSTICC Press, 2010.

- RUTTOR A., "*Neuronal synchronization and cryptografy"*. Disertación para el Doctorado en Ciencias de la Universidad Bayerischen Julius-Maximilians de la ciudad de Würzburg, Alemania, 2006.

- STALLINGS W*., Cryptology and Network Security: Principles and Practice*. Prentice Hall Editors, 1999.

- VOLKMER M., SCHAUMBURG A., *Authenticated tree parity machine key exchange*. Europhysics Letters, 2004.

- VOLKMER M, WALLNER S., *A key establishment IP-Core for ubiquitous computing*. IEEE Transactions on Computers, 2005.

- VOLKMER M, WALLNER S., *Lightweight key exchange and stream cipher based solely on tree parity machine,* 2005.

- VOLKMER M, WALLNER S., *Tree parity machine rekeying architectures for embedded security*, 2005.

- VOLKMER M, WALLNER S., *Tree parity machine rekeying architectures*. IEEE Transactions on Computers, 2004.

- XILINX INC., "*7 Series FPGAs configurable logic block*". Documento ug474 v1.7 (2014), disponible en la siguiente dirección web: http://www.xilinx.com/support/documentation/user_guides/ug474_7Series_CLB.pdf

- XILINX INC., "*Virtex-6 libraries guide for squematic Designs*" .Documento ug624 v14.7 (2013), disponible en la siguiente dirección web: http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_7/virtex6_scm.pdf

- XILINX INC., "*Spartan-3 generation FPGA user guide*". Documento ug331 v1.8 (2011), disponible en la siguiente dirección web: http://www.xilinx.com/support/documentation/user_guides/ug331.pdf

- XILINX INC., "*Spartan-6 family overview*". Documento ds160 v2.0 (2011), disponible en la siguiente dirección web: http://www.xilinx.com/support/documentation/data_sheets/ds160.pdf

- XILINX INC., "*Spartan-6 FPGA configurable logic block*". Documento ug384 v1.1 (2010), disponible en la siguiente dirección web: http://www.xilinx.com/support/documentation/user_guides/ug384.pdf

- XILINX INC., "*Spartan-3A/3AN FPGA starter kit board user guide*". Documento ug334 v1.1 (2008), disponible en la siguiente dirección web: http://www.gta.ufrj.br/ensino/EEL480/spartan3/ug334.pdf

- XILINX INC., "*Linear feedback shift registers in Viryex devices*". Documento xapp210 v1.3 (2007), disponible en la siguiente dirección web:

  http://www.xilinx.com/support/documentation/application_notes/xapp210.pdf

ANEXOS

ANEXO A: Tabulación de sincronización para establecer los estados de los pesos en paralelo, resultados obtenidos con 100 pruebas

Aprendizaje	Parámetro			Pasos de Sincronización		
	N	K	L	Máximo	Promedio	Mínimo
HEBBIANO	5	3	4	2554	403	27
	11	3	4	905	246	53
	49	3	4	939	303	130
	88	3	4	756	339	341
	11	2	4	809	133	25
	11	4	4	1349	383	74
	11	5	4	1842	529	119
	11	3	2	247	63	16
	11	3	6	3201	682	174
	11	3	8	87211	3386	164
ANTI HEBBIANO	5	3	4	83200	62572	39786
	11	3	4	51937	19082	1852
	49	3	4	2248	1514	533
	88	3	4	1769	1062	422
	11	2	4	6684	3945	605
	11	4	4	179346	77818	25450
	11	5	4	165637	107800	46286
	11	3	2	223	125	84
	11	3	6	8768733	6870150	509156
RANDOM WALK	5	3	4	507	181	75
	11	3	4	610	287	105
	49	3	4	1079	402	173
	88	3	4	819	430	227
	11	2	4	464	184	58
	11	4	4	830	384	144
	11	5	4	1459	558	179
	11	3	2	144	65	27
	11	3	6	1653	676	305
	11	3	8	3134	1294	535
	11	3	16	11536	7103	4467

ANEXO B: Descripción de recursos internos de una FPGA

ITEM	Descripción	Función	Implementación
GND	Etiqueta a tierra	Conexión a nivel lógico bajo	'0'
INV	Inversor	Compuerta o bus de inversores	NOT
VCC	Etiqueta a Fuente	Conexión a nivel lógico alto	'1'
XORCY	Compuerta XOR para acarreo lógico	Compuerta XOR con salida general tipo "O"	XOR
LUT1	LUT de 1 bit de entrada	LUT de 1 entrada y salida general tipo "O"	LUT
LUTx	LUT de x bits de entrada	LUT de x entradas y una salida general tipo "O"	
LUTx_L	LUT de x bits de entrada	LUT de x entradas y una salida local tipo "LO"	
LUTx_D	LUT de x bits de entrada y dos salidas	LUT de x entradas y 2 salidas: tipo "O" y tipo "OL"	
MUXCY	Multiplexor 2 a 1 para acarreo lógico	Multiplexor 2 a 1 de un bit con salida general tipo "O"	LC
MUXF5	Multiplexor 2 a 1 para acarreo lógico	Multiplexor 2 a 1 de un bit con salida general tipo "O"	LUT
FD	Flip flop tipo D	FF con flanco ascendente de activación	D-FF
FD_1	Flip flop tipo D	FF con flanco descendente de activación	
FDC	FF tipo D con *clear* asíncrono	FD con *clear* asíncrono	
FDCE	FF tipo D con clear y habilitación (CE)	FD con clear asíncrono y habilitador de reloj	

ANEXO B (Continuación): Descripción de recursos internos de una FPGA

ITEM	Descripción	Función	Implementación
FDE	FF tipo D con habilitación (CE)	FD con habilitador de reloj	D-FF
FDPE	FF tipo D con habilitación y preestablecimiento	FD con habilitador de reloj y preestablecimiento asíncrono	
FDR	FF tipo D con reinicialización	FD con reinicialización síncrona	
FDRS	FF tipo D con re-inicialización y establecimiento	FD con re-inicialización y establecimiento síncrono	
FDS	FF tipo D con establecimiento	DF con establecimiento síncrono	
SRL16	Registro de desplazamiento	Registro de 16 bits con flanco ascendente de activación	LUT
SRL16E	Registro de desplazamiento con habilitación (CE)	Registro de 16 bits con habilitador de reloj	
SRLC16E	Registro de desplazamiento con habilitación y acarreo	Registro de 16 bits con habilitador de reloj y *acarreo*	
RAM16x1	Memoria RAM estática 16x1	RAM 16x1 de escritura asíncrona	SRAM
RAM16x1S	Memoria RAM estática 16x1	RAM 16x1 de escritura síncrona	
RAMB16BWE	Bloque RAM doble puerto de 16k	BRAM 16kb configurable	BRAM
IBUF	*Buffer* de entrada	*Buffer* o múltiples *buffers* de entrada	*Buffer*
OBUF	*Buffer* de salida	*Buffer* o múltiples *buffers* de salida	
BUFGP	*Buffer* de reloj	*Buffer* global de reloj	

ANEXO C: Tabulación para variante de paquetes de bits y regla de aprendizaje hebbiana, resultados obtenidos con 100 pruebas

Paquete bits b	Parámetros N	K	L	Sincronización Promedio	Transferencias Promedio	Bits salida
4	5	3	4	409	176	704
	11	3	4	294	121	484
	49	3	4	635	222	892
	88	3	4	873	298	1192
	11	2	4	180	65	264
	11	4	4	461	201	808
	11	5	4	622	280	1120
	11	3	2	83	34	136
	11	3	6	646	273	1092
	11	3	8	2439	1020	4080
8	5	3	4	334	74	592
	11	3	4	365	73	584
	49	3	4	1314	227	1824
	88	3	4	2039	352	2816
	11	2	4	224	39	320
	11	4	4	531	115	920
	11	5	4	694	153	1224
	11	3	2	100	20	168
	11	3	6	762	155	1248
	11	3	8	1964	426	3408
16	5	3	4	410	45	720
	11	3	4	525	52	848
	49	3	4	2511	224	3600
	88	3	4	5423	492	7888
	11	2	4	271	24	400
	11	4	4	686	73	1184
	11	5	4	892	99	1600
	11	3	2	118	12	208
	11	3	6	1034	105	1680
	11	3	8	2184	232	3728
32	5	3	4	425	24	768
	11	3	4	606	31	992
	49	3	4	4588	218	6976
	88	3	4	11764	564	18048
	11	2	4	328	15	478
	11	4	4	917	49	1590
	11	5	4	959	53	1696

ANEXO C (Continuación): Tabulación para variante de paquetes de bits y regla de aprendizaje hebbiana, resultados obtenidos con 100 pruebas

Paquete bits b	Parámetros N	K	L	Sincronización Promedio	Transferencias Promedio	Bits salida
32	11	3	2	135	8	256
	11	3	6	1324	69	2208
	11	3	8	2610	138	4416
64	5	3	4	442	13	832
	11	3	4	801	21	1344
	49	3	4	8511	207	13248
	88	3	4	21843	534	34176
	11	2	4	240	11	704
	11	4	4	1001	28	1792
	11	5	4	1231	34	2240
	11	3	2	181	5	320
	11	3	6	1634	42	2752
	11	3	8	2610	70	4544
128	5	3	4	497	7	1024
	11	3	4	938	13	1664
	49	3	4	14523	185	23680
	88	3	4	42047	523	67072
	11	2	4	640	8	1024
	11	4	4	1207	17	2176
	11	5	4	1380	19	2560
	11	3	2	218	3	384
	11	3	6	2020	27	3456
	11	3	8	3416	47	6016