UNIVERSIDAD AUTÓNOMA DE ZACATECAS

IMPLEMENTACIÓN DE REDES NEURONALES ARTIFICIALES EN FPGA PARA LA DETERMINACIÓN DEL ESPECTRO DE NEUTRONES FASE HARDWARE

JUAN MANUEL ÁLVAREZ MIRAMONTES

Tesis de Maestría

presentada a la Unidad Académica de Estudios Nucleares

de acuerdo a los requerimientos de la Universidad para obtener el título de

MAESTRO EN CIENCIAS NUCLEARES

Directores de tesis:

M. en C. Victor Martín Hernández Dávila y Dr. René Vega Carrillo

UNIDAD ACADÉMICA DE ESTUDIOS NUCLEARES

Zacatecas, Zac., 12 de abril de 2018

Dedicatoria

A mis *tanos* Juanito y Alejandro

"Hijos míos, cuánto tiempo desde el último abrazo que les dí. . . espero con el alma volver a sentir el roce de su piel y el dulce mirar de sus ojos, que hablan sin hablar y hacen sentir sin tocar, ¡los amo hijos míos!".

(María Suyapa Guadamuz)

A mis padres

Ustedes materializan el mejor ejemplo de amor, trabajo y perseverancia que un hijo pudo tener.

A Meche, Esmeralda, Lupita, Choche y Beto

Gracias hermanos por estar siempre ahí para mí.

A Cristi

Gracias amiga por apoyarme en momentos de tempestad y por alentarme a seguir adelante.

Agradezco a Dios por permitirme llegar hasta esta instancia y ponerme en ruta de una hermosa área del conocimiento, que no hace más que dar testimonio de la grandeza y majestuosidad de su obra.

Agradecimientos

Agradezco a mi asesor M. en C. Victor Martín Hernández Dávila por la propuesta de tema de tesis y por todo el apoyo y facilidades brindadas para que este trabajo se pudiese concretar.

Mi agradecimiento al Dr. René Vega Carrillo por sus consejos y sugerencias en la realización de esta tesis.

A todos los maestros que he tenido durante mi formación académica y que han sabido inculcar en mí, el gusto por la ciencia.

Al Consejo Nacional de Ciencia y Tecnología CONACyT por su apoyo en el proyecto SYNAPSIS (convenio SEP 2004-C01-46893) del cual forma parte este trabajo.

RESUMEN

Las Redes Neuronales Artificiales (RNAs), se han utilizado con gran éxito en la resolución de complejas tareas en las que los métodos convencionales alcanzan un nivel demasiado complicado como para ser utilizados. Recientemente las RNAs se han aplicado exitosamente en el campo de la física nuclear, específicamente en la reconstrucción de espectros de neutrones, una tarea realmente compleja que era resuelta con métodos iterativos u otras técnicas que ya se están viendo desplazadas por la aplicación de esta tecnología. Generalmente las RNAs son desarrolladas en alguna plataforma o software como por ejemplo MATLAB®, en donde el diseñador cuenta con varias herramientas para el desarrollo y puesta a punto de sus redes. Sin embargo, se ha observado que para el proceso de reconstrucción de espectros de neutrones por RNAs, el diseñador realiza un gran esfuerzo en la configuración y ajuste de la RNA y una vez que la red ha sido entrenada y probada en la resolución de este problema, se sigue dependiendo de MATLAB® y de una computadora para el único propósito de ejecutar la RNA; por lo tanto, se detecta un área de oportunidad para la implementación hardware de RNAs orientadas a la reconstrucción de espectros de neutrones. Si bien, dentro de la literatura no hay reportadas implementaciones de RNAs en la tarea de reconstrucción de espectros, si ha habido varios esfuerzos por llevar las RNAs a hardware, con la finalidad de acelerar su ejecución y de explotar su explícito paralelismo. Existe una clara tendencia de implementar RNAs en una tecnología hardware conocida como FPGA, que en los últimos años ha cobrado una notable popularidad, debido a que cuenta con la arquitectura y los recursos idóneos para su implementación. En este trabajo se ha diseñado una arquitectura de RNA implementada en un FPGA y que funciona a partir de una neurona, el procesamiento de la red se realiza de forma cíclica sin comprometer el tiempo de ejecución de la RNA y con la característica de que es configurable, es decir, el diseño permite aumentar o disminuir el número de neuronas de las capas de la RNA y el manejo de hasta 4 diferentes funciones de activación. Para su realización, se ha utilizado la tarjeta de desarrollo Basys2, una tarjeta de bajas prestaciones que incorpora un FPGA de la familia Spartan 3E de Xilinx. Así mismo, para el análisis del desempeño de nuestro diseño, se utilizaron tres

RNAs del Neural Network Toolbox de MATLAB®, ejecutándolas en sus versiones software y hardware, para posteriormente contrastar los resultados así obtenidos.

Palabras clave: Redes Neuronales Artificiales, RNA, Implementación, Fiel Programmable Gate Array, FPGA, reconstrucción, espectro, neutrones, Spartan3E.

ABSTRACT

The Artificial Neural Networks ANNs, have been applied with great success in the resolution of complex tasks in which the conventional methods reach a level of complexity that is too high to be used. Recently, ANNs have been used with great success in the nuclear physics field specifically in the reconstruction of neutron spectra, a rather cumbersome task that was once solved with iterative methods or other techniques that are already being displaced by the application of this technology. Generally, the ANNs are developed utilizing any platform or software such as MATLAB, in which the designer has a variety of tools for the design and calibration for the neural networks .Nevertheless, a pattern has been observed in the methodology utilized in the reconstruction of the neutron spectra for ANNs, the designer makes a great effort in the configuration and adjustment for the ANN, and once the network has been trained and tested in the resolution of this problem, the network still depends on MATLAB and a computer with the sole purpose to execute the ANN, therefore opening an area of opportunities for the hardware implementation of ANN oriented towards the reconstruction of neutron spectra. Despite the success of ANN unfolding of neutron spectra, a hardware device specific to this application, has not been developed. However, there have been multiple attempts to implement the ANNs using hardware, with the intention to accelerate their execution and exploit the explicit parallelism of the ANNs. There is a tendency to implement them using a technology known as FPGA, which in the last years it has gained popularity because it possess the resources and architecture suitable for the implementation of ANNs. In this work we have designed an architecture of ANN and implemented in FPGA that functions with a single neuron, the execution of the network is carried out in an iterative process without compromising the time of execution of the ANN and with a characteristic that makes it configurable, that is to say, the design allows the increase or decrease in the number of neurons in the layers of the ANN and it also allows up to four diferent activation functions. For its realization, the Basys2 FPGA development board was used, built around a Xilinx Spartan 3E FPGA. Likewise, for the performance analysis of our design, three ANNs from the Neural Network Toolbox of MATLAB were used, executing

them in their corresponding versions of software and hardware and making a comparison of the results thus obtained.

Keywords: Artificial Neural Network, ANN, Fiel Programmable Gate Array, FPGA, Spartan 3E, reconstruction, neutron, spectra.

Contenido General

Pag.

Lista de figuras

Figura Pag.

Figura Pag.

Nomenclatura

AG	Algoritmos Genéticos
$ASIC$	Application Specific Integrated Circuit
BP	Back Propagation error
CAD	Computer Aided Design
CI	Circuito Integrado
CLB	Configurable Logic Block
DSP	Digital Signal Processor
$FPGA$	Fiel Programmable Gate Array
FSM	Finite State Machine
IA	Inteligencia Artificial
LUT	Look-Up Table
MAC	Multiply Accumulate
MLP	Multi Layer Perceptron
RNA	Red Neuronal Artificial
$SEEB$	Sistema Espectrométrico de Esferas de Bonner

$VHDL$	Very High speed integrated circuit Hardware Description Language
μ_0	Constante de permeabilidad
α	Rayos Alfa
β	Rayos Beta
γ	Rayos Gamma

Capítulo 1

Introducción

Las Redes Neuronales Artificiales (RNAs) son un modelo computacional constituido por un gran número de elementos básicos llamados neuronas, altamente interconectados entre sí, que busca emular las habilidades del cerebro humano para aprender y tomar decisiones; por lo que las RNAs deben ser sometidas a un proceso de aprendizaje para posteriormente aplicarlas a un proceso de toma de decisiones. El desempeño de las RNAs dependerá de algunos factores clave tales como: la estructura de la red, el tipo de las funciones de activación de las neuronas y la calidad y cantidad de estímulos utilizados durante el proceso de aprendizaje [1], [2].

Así mismo, son bastante efectivas identificando y aprendiendo relaciones no lineales entre los datos de entrada y salida de un proceso, se han convertido en una herramienta sumamente socorrida en la resolución de problemas multidiciplinarios; uno de ellos y de interés particular en este trabajo, en el ámbito de la física nuclear, específicamente para la reconstrucción de espectros de neutrones [3],[4],[5],[6],[7].

Generalmente las RNAs son implementadas en software ya que tienen la gran ventaja de que el diseñador no necesita conocer el funcionamiento interno de los elementos que conforman la red y se concentra primordialmente en la aplicación de la RNA propiamente dicha. Sin embargo, esto conlleva también ciertas desventajas, como el hecho de que su ejecución sea más lenta que un diseño basado en hardware (lo cual es crítico en aplicaciones de tiempo real), o el hecho de tener que estar ligado a una computadora con sus inconvenientes implícitos. Bajo estas condiciones, una implementación hardware de una RNA resulta muy deseable ya que elimina parte del costo de la aplicación y proporciona un mejor nivel de adecuación para la

aplicación final [8], [9]. En este trabajo se propone el diseño hardware de una red neuronal artificial reconfigurable del tipo perceptrón multicapa, implementado en un dispositivo *Field Programmable Gate Array* (FPGA) de tal forma, que pueda servir como receptáculo de RNAs diseñadas en MATLAB® .

Las RNAs basadas en hardware pueden ser implementadas usando sistemas ya sea analógicos o digitales. El diseño digital es preferido ya que ofrece ventajas como mejor presición, mejor repetibilidad, menor sensibilidad al ruido, mayor flexibilidad y compatibilidad con otros tipos de tecnologías. Dentro de la implementación hardware de RNAs digitales, la literatura reporta tres principales grupos de diseño: basados en *Digital Signal Processors* (DSP), basados en *Application Specific Integrated Circuit* (ASIC) y basados en FPGA. La implementación basada en DSP es sequencial, y por tanto no preserva el paralelismo inherente en las RNAs. En los diseños basados en ASIC el desarrollador especifica todos los requerimientos y características necesarias para la aplicación específica, obteniendo así un sistema a la medida; sin embargo, esta solución resulta costosa en demasía como para ser considerada en la mayoría de las aplicaciones y no ofrece reconfigurabilidad para el usuario [10].

La tecnología FPGA resulta bastante conveniente para la implementación de RNAs, toda vez que preserva la arquitectura paralela de éstas, ofrece gran flexibilidad al ser un dispositivo reconfigurable y finalmente, ofrece una aceptable relación costo-desempeño. Características que las catapulta como la tecnología más socorrida en el diseño hardware de RNAs[11].

Si bien, los FPGA al ser dispositivos hardware reutilizables con la capacidad de albergar altos volúmenes tanto de lógica convinacional como lógica secuencial y de poseer una generosa cantidad de memoria; a la hora de diseñar RNAs, es necesario tener clara una estrategia de diseño ya que este tipo de arquitecturas, exige de una gran cantidad de recursos para su funcionamiento, sobre todo si se desea obtener el máximo grado de paralelismo que ofrece la naturaleza propia de las RNA. Aunado a lo anterior, hay que agregar que el diseño de las RNAs es sumamente cambiante, ya que dentro de la literatura se reportan diferentes topologías de RNAs y de tamaños que pueden ir desde unas cuantas neuronas, hasta cientos o inclusive miles de neuronas las que conforman la RNA. Por lo que, para la implementación de cada neurona, es imperante tener claros tres de los factores que desafían al diseñador de RNAs a

saber: representación de los datos, diseño de la MAC y diseño de la función de activación de las neuronas [12], [10], [8], [1].

1.1 Problema científico

Las redes neuronales artificiales se han aplicado con gran éxito en la reconstrucción de espectros de neutrones, no obstante, se observa que una vez que la RNA ha sido validada para tal actividad, su utilización se encuentra ligada a un software y por consecuencia a una computadora, lo cual limita el máximo potencial que podría extraerse de la RNA una vez que ya ha sido entrenada y probada con éxito para la tarea en cita. Por lo tanto, resultaría muy conveniente poder descargar el conocimiento de dicha RNA en un FPGA, que no sólo podría acelerar la ejecución de la RNA, sino que eliminaría la necesidad de contar con una computadora para su ejecución en la aplicación final, facilitando así la determinación de espectros de neutrones.

1.1.1 Objetivo General

Extraer el conocimiento de una red neuronal artificial del tipo perceptrón multicapa diseñada en MATLAB® y vaciarlo en un FPGA, de tal forma que la red neuronal pueda ejecutarse en un dispositivo hardware, eliminando así la necesidad de una computadora en la aplicación final.

1.1.2 Objetivos Particulares

1. Generar los códigos de programación necesarios para extraer y manipular la información de RNAs diseñadas en MATLAB® para su posterior implementación en un FPGA.

2. Diseñar una estrategia para el flujo y manejo de los datos bajo un esquema de representación en punto fijo de 16 bits.

3. Diseñar una neurona artificial

    3.1 Diseñar e implementar la función de activación sigmoide.

    3.2 Diseñar e implementar la unidad de aritmética *Multiply-Accumulate* MAc.

4. Diseñar una arquitectura de RNA configurable, que a partir de una única neurona implementada en un FPGA, permita emular a una RNA de MATLAB®.

5. Validar el prototipo mediante una comparación entre el desempeño de RNAs desarrolladas en MATLAB® y el desempeño de la RNA hardware.

Capítulo 2

Marco Teórico

2.1 Neutrones

2.1.1 Introducción

El conocimiento de la física del átomo y su núcleo comenzó con el descubrimiento de la radiactividad por H. Becquerel, quién la descubrió al intentar establecer una relación entre la radiación solar y la propiedad de velar placas fotográficas de sales de uranio; así, por serendipia, encontró que las sales de uranio seguían teniendo la capacidad de velar las placas aun a pesar de no recibir luz solar. Lo anterior, motivó a M. Curie en 1898, a investigar cual era la causa de la emisión de energía del uranio, llegando a identificar un nuevo elemento, el radio *Ra*, que emitía una radiación altamente penetrante, demostrando también que la tasa de emisión no se ve modificada por cambios físicos o químicos, es decir, cualquier cambio en la presión, volumen, temperatura o forma química, no cambian el porcentaje de emisión de la radiación [14]

E. Rutherford y F. Soddy en 1902 sugirieron que la radiactividad se debía a la desintegración espontánea de los átomos, y que los nuevos elementos formados, podrían tener propiedades totalmente diferentes con relación a las de los elementos iniciales. No obstante, que la desintegración del átomo es un proceso espontáneo, la radiactividad es un proceso prolongado, el cual se puede extender por períodos que van desde algunos segundos hasta millones de años [15].

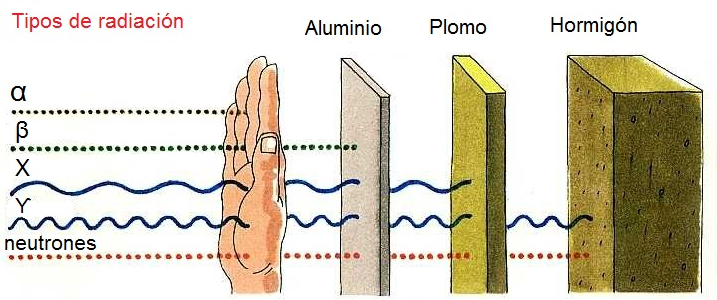

Figura 2.1 Tipos de Radiación y su poder de penetración en la materia

Después del descubrimiento de la radiactividad, muchos esfuerzos se dirigieron a estudiar las propiedades de esta radiación, especialmente en lo referente a el poder de penetración en diferentes materiales, la ionización específica en diferentes gases y el comportamiento sobre los efectos de un campo magnético y eléctrico. Como se aprecia en la figura 2.1, las radiaciones de sustancias radiactivas naturales, se clasificaron en componentes de acuerdo a su poder de penetración.

Al observar la figura 2.1, se aprecia un primer tipo de radiación con un poder de penetración sumamente débil, es detenido por una hoja de papel ordinario, pero causaba una ionización intensa en el aire, estos son los rayos α (rayos alfa o protones). Un segundo tipo de radiación tiene menor poder de ionización, pero mayor poder de penetración que los rayos α y puede pasar fácilmente a través de delgadas láminas de metal (de algunos milímetros de espesor), a esta radiación se le conoce como β (rayos beta, electrones o positrones); el tercer tipo causa a un menos ionización pero tiene la capacidad de penetrar placas de diversos materiales de hasta varios centímetros de espesor, a esta se le conoce como radiación γ (rayos gama o fotones).

En 1920 E. Rutherford sugirió que podía existir una partícula neutra, posiblemente un protón y un electrón estrechamente ligados, al que denominó neutrón. En 1930 los físicos alemanes W.Bothe y H. Becker produjeron una radiación fuertemente penetrante bombardeando núcleos de ${}_4Be^9$ con partículas α de 5.3 MeV procedentes de una fuente radiactiva, encontrando que muchos núcleos se transformaban en radiactivos y emitían lo que se pensaba eran rayos γ, ya que la radiación podía penetrar fácilmente varios centímetros de Pb (Plomo) y no

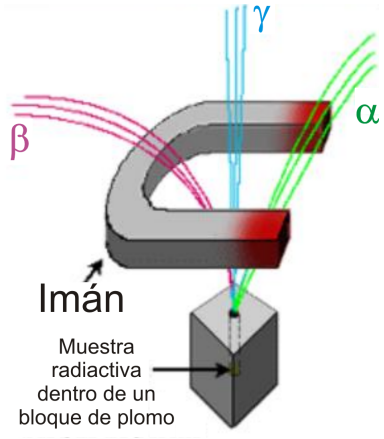

Figura 2.2 Comportamiento de partículas radiactivas en un campo magnético

era desviada por campos eléctricos y magnéticos como se muestra en la figura 2.2. Supusieron que se trataba de rayos altamente energéticos.

En 1932, James Chadwick, trabajando en el laboratorio Cavendish, sugirió que la radiación desconocida no constaba de rayos gama en absoluto, sino de una partícula sin carga, el neutrón (Chadwick recibió el premio Nobel en 1935 por este descubrimiento), aproximadamente del mismo tamaño que el protón; tal y como lo había sugerido Rutherford 12 años antes. Desde su descubrimiento, el neutrón ha jugado un rol significativo en el entendimiento de la estructura del núcleo. La ausencia de carga eléctrica le confiere un elevado poder de penetración y la posibilidad de interaccionar con la materia de manera distinta a la de las partículas cargadas y radiación electromagnética. Dadas las dimensiones ocupadas por los núcleos, estas partículas tienen una pequeña probabilidad de interaccionar (sección eficaz) y producen, excepto casos particulares, una escasa pérdida de energía, lo que conduce a que sea necesario un gran número de colisiones para que la energía del neutrón disminuya de manera significativa. Esta circunstancia hace que los neutrones juntamente con la radiación γ , sean mucho más penetrantes que las partículas cargadas y puedan de esta forma atravesar espesores importantes de materia, tal y como queda de manifiesto en la figura 2.1.

Debido a que los neutrones no son desviados por un campo eléctrico o magnético, y también porque no producen una ionización apreciable, se deben emplear métodos indirectos para detectarlos y medir su energía.

Tabla 2.1 clasificación de los neutrones de acuerdo a su energía

Neutrones	**Rangos de energía**
Térmicos	< 0.4 eV
Intermedios	0.4 eV - 10 keV
Rápidos	10 eV - 10 MeV
Relativistas	$>$ 10 MeV

2.1.2 Clasificación y propiedades de los neutrones

La variación rápida de la sección eficaz con la energía del neutrón, ha incitado a clasificar los neutrones en categorías según su energía cinética y tipo de interacción, si bien los límites que definen esta clasificación aun están sujetos a discusión, los neutrones se clasifican de acuerdo a su energía, ya que el tipo de reacción que sufre el neutrón depende directamente de ésta. La tabla 2.1 muestra una clasificación de los neutrones atendiendo a su energía.

Los neutrones térmicos, son aquellos que se encuentran en equilibrio térmico con los núcleos del medio donde se encuentran, se les puede aplicar las nociones de la teoría cinética de los gases y su distribución sigue la estadística de Maxwell Boltzman; los neutrones intermedios resultan de la colisión elástica de los neutrones rápidos en materiales de bajo número atómico, la distribución de estos neutrones es proporcional al inverso de su velocidad; los neutrones rápidos son aquellos que tienen una energía superior a unas decenas de keV, valor utilizado como límite por debajo del cual los instrumentos para la detección de estos neutrones resultan inadecuados. Finalmente, se consideran relativistas todos los neutrones cuya energía es superior a los 10 MeV [3].

2.1.3 Producción de neutrones

Para investigar un campo de neutrones desde un punto de vista espectrométrico y dosimétrico, el conocimiento de su origen y producción es una información a priori muy importante para la elección del tipo de instrumento de medida a utilizar y el análisis de los resultados de la medida.

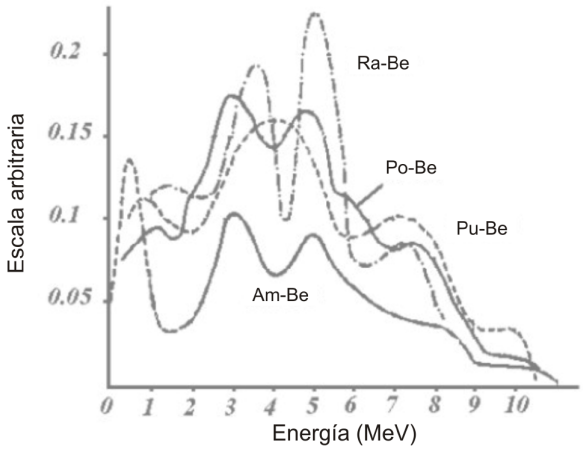

Figura 2.3 Espectro de neutrones del ^{241}AmBe, ^{239}PuBe, ^{226}RoBe, ^{210}RaBe, reacciones tipo (α, n)

Existen varias maneras de producir neutrones, entre las más importantes están: la fisión espontánea, como en el caso de la fuente de Ca-252 (Californio); las reacciones (α, n), como las fuentes isotópicas de ^{241}AmBe (Americio-Berilio) y ^{241}AmBo (Americio-Boro); las reacciones (γ, n), como las fuentes foto neutrónicas que utilizan como núcleos destino el Be^{9} o el ^{2}H (deuterio). Sin duda, la fuente más prolífica de neutrones es el reactor nuclear. La fisión de un núcleo de uranio o plutonio en un reactor nuclear va acompañada por la emisión de algunos neutrones. Estos neutrones de fisión tienen un amplio rango de energías que va de térmicos a relativistas.

Los neutrones producidos mediante reacciones nucleares como las ya mencionadas tienen un amplio espectro de energía tal y como se puede apreciar en la figura 2.3, esto debido a que las partículas alfa que inician la reacción tienen un amplio rango de energías. Normalmente se utiliza Radio, Polonio o Plutonio como emisor de partículas alfa para iniciar la reacción. Es posible producir también fuentes de neutrones monoenergéticas a partir de espectros neutrónicos amplios utilizando adecuadas combinaciones de filtros [14].

La distribución espectral de neutrones en el momento de su producción se modifica enormemente cuando atraviesan materiales diversos. La forma de los espectros encontrados con mayor frecuencia en el ámbito de la radio protección se encuentran en las publicaciones 180 y 318 de

la OIEA [17],[18].

2.1.4 Interacción de los neutrones con la materia

La interacción de los neutrones con la materia es un proceso bastante diferente al de las partículas cargadas y la radiación electromagnética. Dado que los neutrones no poseen carga eléctrica, no pueden ionizar directamente la materia que atraviesan y por lo tanto su interacción es únicamente con los núcleos de los átomos. Las interacciones más comunes del neutrón con la materia son colisiones inelásticas, captura y fisión. Un núcleo que colisiona inelásticamente con un neutrón queda en un nivel de energía más alto, entonces puede liberar esta energía emitiendo fotones, mediante la emisión de una partícula β o de ambas formas.

En la captura de neutrones, un núcleo afectado puede absorber el neutrón y expulsar energía en forma de rayos γ, rayos X, partículas β, partículas α o todas estas. Las partículas secundarias causan después ionización; en la fisión, un núcleo pesado puede absorber al neutrón y después de pasar por el estado del núcleo compuesto dividirse en dos núcleos más ligeros que casi siempre son radiactivos.

La materia orgánica esta compuesta esencialmente de elementos de bajo número másico como el *H, O, N y C*. Estos elementos son buenos moderadores de neutrones. Los neutrones térmicos e intermedios depositan esencialmente casi toda la energía en el tejido mediante colisiones inelásticas que los moderan y termalizan; y en este estado sufren la captura neutrónica los núcleos de H y N. La captura ${}^1H(n,\gamma)^2H$, produce un fotón por energía del orden de 2.2 MeV. Estos γ, interaccionan a su vez con el tejido por efecto fotoeléctrico, Compton o producción de pares, produciendo electrones que son los responsables de las ionizaciones producidas a lo largo de su recorrido. Estos rayos γ son la mayor contribución a la dosis en el cuerpo para neutrones de energía inferiores a 200 keV [14],[15].

La otra reacción de captura ${}^{14}N(n,p)^{14}C$, da lugar a la emisión de protones de 0.62 MeV. En este caso tanto los protones como los núcleos de retroceso al ser partículas cargadas, son los responsables de las ionizaciones producidas a lo largo de su recorrido. Esta reacción es importante en el cálculo de la dosis equivalente para energías inferiores a 100 keV.

Los neutrones rápidos ceden principalmente su energía al tejido mediante la dispersion elástica. En efecto, durante una colisión con el núcleo de H, el neutrón puede perder en promedio la mitad de su energía y la cede a los protones de H que al recular producen ionización, pudiendo los núcleos de H ser liberados. Mediante este proceso, un neutrón de 1 MeV, puede sufrir cerca de 20 colisiones para termalizarse, lo que equivaldría a un recorrido de unos 5 cm en el tejido.

El neutrón termalizado es, o bien capturado por uno de los procesos citados anteriormente, o en otros casos puede ser dispersado fuera del cuerpo humano dando lugar a los neutrones albedo.

Los neutrones rápidos también colisionan con los demás constituyentes del tejido, pero la pérdida de energía es relativamente pequeña comparada con el H, como lo prueba el hecho de que los neutrones con energías comprendidas entre 0.5 y 5 MeV pierden el 90% de su energía en colisiones con los núcleos de H [16].

Con el descubrimiento de la radiactividad, muchos esfuerzos se han dirigido a estudiar los efectos nocivos de la radiación ionizante y no ionizante, especialmente con relación a los neutrones, dadas sus propiedades intrínsecas. La detección de neutrones es un problema mucho más complejo que la detección de partículas cargadas y fotones, ya que los neutrones no transportan ninguna carga eléctrica y no causan ionización directa, aunque su poder de penetración ocasiona un mayor daño al personal que se ve expuesto a ellos.

2.1.5 Dosimetría de campos neutrónicos

2.1.5.1 Protección radiológica

La protección radiológica es el area de la ingeniería de la salud ambiental que trata de la protección del individuo y grupos de personas contra los efectos detrimentales de la radiación ionizante y no ionizante. La protección radiológica es responsable de los aspectos de seguridad en el diseño de procesos, equipos e instalaciones que utilizan fuentes de radiación, para que la exposición del personal sea minima y para que en todo tiempo, la radiación este dentro de los límites aceptables. Esta disciplina debe mantener al personal y al entorno bajo constante

supervision, para cerciorarse de que el uso y almacenamiento de las fuentes de radiación se hagan en condiciones seguras. Si se encuentra que las medidas de control no son efectivas o que han fallado, la seguridad radiológica debe ser capaz de evaluar el grado de riesgo y realizar las recomendaciones pertinentes tendientes a tomar acciones correctivas. Los aspectos científicos y de ingeniería en los que la seguridad radiológica interviene incluyen: las mediciones físicas de diferentes tipos de radiación y materiales radiactivos, el establecimiento de relaciones cuantitativas entre la exposición a la radiación y el daño biológico, el transporte de la radiactividad en el medio ambiente y el diseño de equipo radiológico, procesos y entornos seguros. Con base en lo anterior, las mediciones de la fluencia de dosis han llegado a ser el factor determinante que obligan a establecer procedimientos, practicas y entornos de trabajo que limiten la exposición a la radiación [14].

2.1.5.2 Espectrometría y dosimetría de neutrones

Con el descubrimiento de la radiactividad, muchos esfuerzos se han dirigido a estudiar las propiedades de la radiación. La detección de neutrones es un problema mucho más complejo que la detección de partículas cargadas y fotones, ya que los neutrones no transportan ninguna carga eléctrica y no causan ionización directa.

Desde el descubrimiento de los neutrones por Chadwick en 1932, se han desarrollado diversos métodos para medir la distribución de energía de los neutrones, también llamada espectro [28]. El objetivo de la espectrometría y dosimetría de neutrones en la protección radiológica, es proporcionar un método para valorar la magnitud de los efectos detrimentales en la salud debido a la exposición de radiación de neutrones. El logro satisfactorio de esta meta requiere alcanzar dos objetivos separados: el primero es identificar una cantidad que sea una medición razonablemente exacta del daño biológico ocasionado por la irradiación de neutrones, y el segundo es encontrar métodos, de preferencia simples, para la apropiada medición de esta cantidad en entornos de trabajo.

La espectrometría neutrónica ha contribuido de forma importante al desarrollo de la física nuclear y es un importante instrumento en otros campos tales como la tecnología nuclear,

fusion nuclear, radioterapia y radioprotección. Los métodos utilizados en espectrometría neutrónica se pueden clasificar atendiendo al principio utilizado para caracterizar o medir la energía del neutrón. De entre todos ellos destacan por su importancia y grado de utilización los basados en: la medida de la energía de los núcleos que reculan al interactuar con los neutrones, la medida de las energías de las partículas cargadas creadas en reacciones nucleares producidas por neutrones, la medida de la velocidad del neutrón, la medida de la radiactividad (una específica emisión gama o transición de fase) inducidas en ciertas reacciones producidas por los neutrones y denominadas de umbral, y finalmente métodos en los que la distribución energética de fluencia se obtiene por deconvolución de un conjunto de medidas realizadas con ciertos detectores y geometrías [19].

2.1.5.3 Evolución de la espectrometría neutrónica

Cuatro fases bien diferenciadas se ponen de manifiesto en la evolución de la espectrometría de neutrones. Las dos primeras corresponden a los periodos comprendidos entre 1932-1959 y 1960-1979 respectivamente[20],[21]. Muchos de los espectrómetros de neutrones utilizados actualmente se basan en métodos utilizados antes de 1960. Así, en los espectrómetros denominados de retroceso de esta época, se utilizaban cámaras de ionización y contadores proporcionales, emulsiones nucleares, centelladores orgánicos[22]. Un notable desarrollo se produce durante la segunda fase (1960-1979), con la introducción del método basado en el Sistema Espectrométrico de Esferas de Bonner (SEEB)[34] y los avances conseguidos en las técnicas de espectrometría neutrónica utilizando detectores de gas y de centelleo . En este período también supuso un importante desarrollo la aparición de los primeros códigos de deconvolución de espectros neutrónicos, obtenidos a partir de medidas realizadas con multicanales, así como la utilización por primera vez de los semiconductores en la espectrometría neutrónica y la introducción de los detectores de burbujas [30].

La tercera fase a partir del año 1979, se caracteriza por un gran avance tecnológico y el gran progreso que significó la incorporación de computadoras más potentes, que permitieron por

medio de métodos Monte Carlo, el cálculo de las funciones de respuesta y eficiencias de detección de los detectores utilizados, así como una mejora en los programas de deconvolución, para obtener espectros neutrónicos a partir de las lecturas realizadas con los espectrómetros [42]. Cabe mencionar que los códigos de deconvolución utilizados, y por tanto los métodos utilizados para su ejecución, están basados en métodos que contienen un determinado algoritmo de cálculo que el usuario debe conocer con precisión, esto representa una seria desventaja en el uso de estas herramientas. Con el avance de la ciencia moderna, en la cuarta y última fase desarrollándose en la época actual, se están utilizando técnicas alternativas basadas en la máxima entropía y en tecnologías de IA para realizar la espectrometría y dosimetría de neutrones. Entre las principales técnicas por medio de IA, se pueden mencionar los Algoritmos genéticos AG[31] y las Redes Neuronales Artificiales RNA, destacándose el uso de estas últimas [25], [26], [27], [28], [29], [5], [6].

Entre todas las técnicas disponibles para la espectrometría de neutrones, el Sistema Espectrométrico de Esferas de Bonner SEEB ha sido el más utilizado para los propósitos de la protección radiológica [32], debido a sus características, como son el amplio rango de energías, de térmicos a varios GeV, la gran variedad de sensores térmicos, pasivos o activos, que se pueden utilizar lo que permite adaptar la sensibilidad al espacio de trabajo específico, entre otras [33].

2.1.5.4 Espectrometría de neutrones por SEEB

En 1960 R.L. Bramblett, R.I. Ewing y T.W. Bonner describieron un espectrómetro de neutrones hecho con un detector de neutrones térmicos cubierto con esferas de polietileno, ver figura 2.4 [34].

Como se observa en la figura 2.5, en principio, a medida que los neutrones atraviesan las esferas de polietileno, se dispersan los neutrones rápidos y epitérmicos perdiendo energía hasta que alcanzan el balance térmico o salen del moderador. El detector responde a los neutrones térmicos que llegan al centro de la esfera moderadora [23].

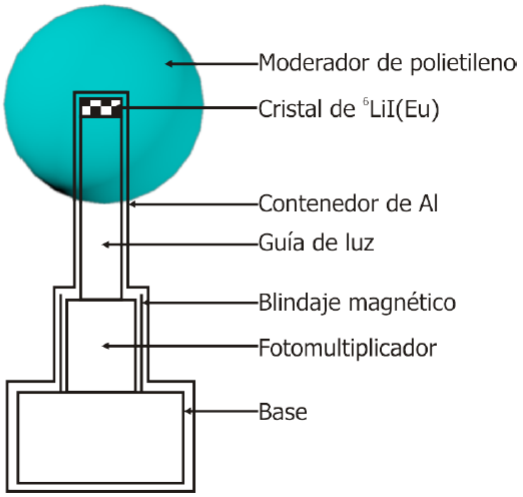

Figura 2.4 Elementos de un sistema espectrométrico de esferas de Bonner

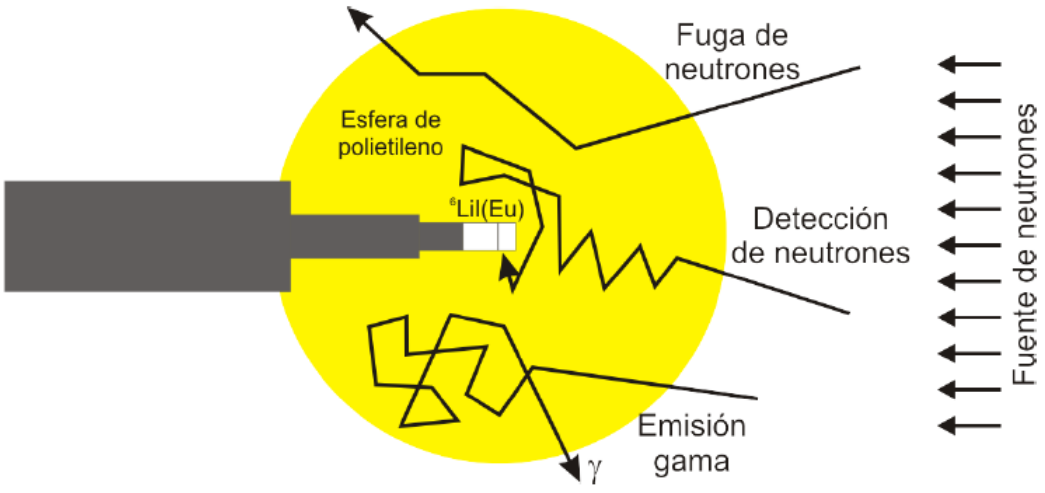

Figura 2.5 Detección de neutrones a partir del SEEB

Como se observa en la figura 2.6, la combinación detector-moderador tendrá una respuesta diferente a los neutrones en función de la energía. A partir de la tasas de conteo tomadas con las esferas, es posible reconstruir el espectro de neutrones, este proporciona información acerca de la distribución de energía de los neutrones incidentes. En el SEEB, cada detector se caracteriza por una función de respuesta. A medida que se incrementa el tamaño del moderador, el pico de la función de respuesta cambia hacia neutrones de energía mayor [5].

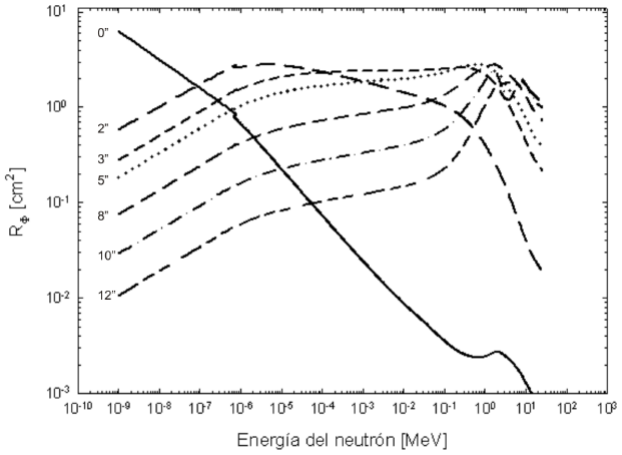

Figura 2.6 Matriz de respuesta del SEEB de la UAEN-UAZ

Cuando se utiliza el SEEB, se miden las tasas de conteo que los neutrones producen en cada uno de los detectores del sistema [36].Cuando neutrones de energía E_i alcanzan la esfera de diámetro D_j los neutrones pierden energía y alcanzan al detector produciendo una señal que se traduce en una tasa de conteo C_j [38]. El incrementar el diámetro de las esferas permite que se puedan detectar neutrones de energías mayores a las térmicas. Al conjunto de eficiencias en función de la energía de los neutrones se le llama "matriz de respuesta", la relación entre las tasas de conteo, en función del diámetro de la esfera $C(D)$, la matriz de respuesta $R(D,E)$ y el espectro de las muestras $\Phi_E(E)$, se muestran en la ecuación 2.1 [36].

$$C(D) = \int_{E\,\min}^{E\,\max} R(D,E)\Phi_E(E)dE \tag{2.1}$$

Esta es la ecuación integro-diferencial de Fredholm de primer tipo, que en su version discreta se indica en la ecuación 2.2

$$C_j = \sum_{i=1}^{m} R_{i,j}\Phi_i \tag{2.2}$$

donde C_j es la tasa de conteo del j-ésimo detector, $R_{i,j}$ es la matriz de respuesta del j-ésimo detector de neutrones en el i-ésimo intervalo de energía, Φ_i es la fluencia de neutrones en el i-ésimo intervalo de energía y m es el número de esferas del sistema espectrométrico.

Debido a que el número de detectores (esferas) es menor que el número de grupos de energía usados para describir el espectro, la ecuación 2.2 se convierte en un sistema de ecuaciones mal condicionado. En estos casos no existe una solución única, se tiene un número infinito de soluciones, el seleccionar aquella que tenga significado para el tipo de problema es parte del proceso de reconstrucción, por lo que se debe aplicar algún procedimiento para la deconvolución del espectro [38], [39],[40]. Existen varios métodos para resolver el problema de la reconstrucción de los espectros de neutrones y la mayoría de ellos, excepto aquellos que se basan en los métodos Monte Carlo, RNA [28],[29],[5], AG o una combinación de estas últimas, utilizan algoritmos iterativos.

2.1.6 Códigos y técnicas de deconvolución

Para poder obtener mediante un proceso de deconvolución los espectros neutrónicos a partir de las medidas realizadas con un SEEB convenientemente calibrado, es necesario conocer las respuestas de cada una de estas esferas en función de la energía del neutrón. La indicación de una esfera de Bonner se expresa por el producto de la convolución de su respuesta en función de la energía $R(E)$, por la tasa de fluencia energética de los neutrones en el punto de medida

$\Phi(E)$ como se describe en la ecuación 2.1. Una vez fijadas las condiciones de calibración del SEEB, así como la validación de las funciones de respuesta del mismo, la determinación del espectro neutrónico resulta de resolver el sistema de ecuaciones descrito en la ecuación 2.2.

La deconvolución del espectro neutrónico medido, consiste en establecer la tasa de distribución energética de fluencia $\Phi(E)$, la matriz de respuesta R(E) y el conjunto de medidas realizadas Mdr. Para $m = n$, el sistema de ecuaciones admite una solución exacta, si bien esta solución es raramente aceptable, las incertidumbres experimentales de los recuentos Mdr y los inevitables asociados a la matriz de respuesta suelen dificultar obtener un resultado satisfactorio. Si $m > n$, el sistema de ecuaciones es sobredeterminado y la solución se puede obtener por una técnica de regresión. No obstante y dado que el número de grupos de energías es pequeño,se consigue una pobre resolución energética. Si $m < n$, que es el caso más frecuente, el sistema es indeterminado, difícil de tratar y admite infinitas soluciones de las cuales es necesario encontrar aquella que represente adecuadamente el espectro medido.

2.1.6.1 El problema de la reconstrucción

La reconstrucción de espectros de neutrones utilizando datos obtenidos a partir de laminillas de activación, esferas de Bonner u otros detectores, generalmente implica resolver la ecuación 2.1 ó 2.2.

Aunque existen varios métodos para la solución formal de las ecuaciones integrales de primer orden, ninguno de esos métodos es generalmente aplicable cuando la función de respuesta del detector R_i, j no se conoce analíticamente. Este es el caso de todos los sistemas prácticos utilizados para la espectrometría de neutrones. En la práctica, R_i, j se calcula o se determina experimentalmente y es generalmente aproximado por una matriz de respuesta con valores discretos. La ecuación 2.2 se reemplaza, entonces por un conjunto de ecuaciones lineales, pero persiste el problema de resolver m ecuaciones con n incógnitas, en donde m es el número de detectores disponibles y n es el número de grupos de energía con que se define el espectro de neutrones. Lo anterior se podría expresar en notación matricial como se indica a

continuación:

$$\overline{Y} = A * \overline{\Phi} \tag{2.3}$$

donde $\overline{Y}$ y $\overline{\Phi}$ son vectores y A es una matriz de mxn. Generalmente n es mayor que m, de aquí que no exista una solución única, ya que se forma un sistema de ecuaciones mal condicionado que tiene un número infinito de soluciones, estas pueden tener amplias oscilaciones,algunas veces negativas, soluciones que tienen muy poco significado físico. Lo anterior nos conduce a introducir los términos "exacto", "aproximado" y "apropiado" para caracterizar las soluciones de la ecuación 2.3. Una solución exacta, si existe alguna, satisface exactamente la ecuación pero podría ser negativa u oscilatoria. Las soluciones aproximadas satisfacen la ecuación únicamente dentro de límites de error razonables. De las soluciones aproximadas, la selección de la solución más aceptable físicamente hablando produce una solución apropiada.

2.1.6.2 Métodos y códigos utilizados para la deconvolución

De acuerdo con el trabajo realizado por Matzke [38], la determinación de la fluencia de partículas a través de la reconstrucción de las lecturas de los detectores medidos ha sido investigada por muchos autores y se han desarrollado diversos métodos para resolver el problema de la detección de neutrones y la determinación del espectro. La parte más delicada de la espectrometría basada en el SEEB, es el proceso de reconstrucción. La deconvolución del espectro neutrónico medido consiste en establecer la tasa de distribución energética a partir de 2.1, para lo cual es necesario aplicar métodos de reconstrucción. Durante los últimos años se han logrado importantes progresos en el desarrollo de métodos y códigos en un intento por resolver los problemas de la reconstrucción del espectro de neutrones y el cálculo de dosis equivalentes.

En la dosimetría de reactores para la reconstrucción de algunos canales se utilizaron con éxito códigos basados en métodos de mínimos cuadrados lineales. Los códigos STAYSL, LEPRICON, LSL, DIFBAS, DIFMAZ, pertenecientes al paquete de programas HEPRO, y los códigos MSITER y MINCHI son ejemplos representativos de un gran número de códigos basados en métodos de ajuste de mínimos cuadrados, mismos que también se han aplicado con éxito para la reconstrucción de datos medidos a través del SEEB.

Un segundo grupo de códigos que excluyen valores negativos de fluencia son: SAND I y II, GRAVEL [41] (basado en algoritmo mejorado de SAND II), LSL-M2, LOHUI, BUNKI [42] RADAK. Estos códigos en principio, realizan un ajuste no lineal por medio de mínimos cuadrados, con la restricción de reconstruir únicamente fluencia de partículas no negativas. En este contexto, se tiene que mencionar el método propuesto por Schmittroth, quien utiliza una distribución logarítmica normal de las variables para evitar valores de fluencia negativos. Además, Schmittroth utiliza correlaciones en el espectro *a priori* como medio para asegurar un suavizado.

Durante los últimos años, se han realizado intentos por desarrollar nuevos códigos de deconvolución de espectros de neutrones, entre los cuales se pueden mencionar: MITOM [43] , FRUIT [44], BUMS [39]. E stos códigos están basado en algoritmos de reconstrucción iterativos. Sin embargo, se observa que el uso de estas herramientas ofrece ciertos inconvenientes, como es la necesidad de proporcionar información inicial a priori, como un "espectro inicial", mismo que deberá estar tan cerca como sea posible a la solución que se desea obtener y que en la mayoría de los casos es desconocido.

Los puntos críticos de los códigos mencionados son, en general: la complejidad en el uso de estos y en consecuencia, la necesidad de un usuario muy experto, así como el gran consumo de tiempo en el proceso de reconstrucción del espectro. Dentro de las complejidades mencionadas, el problema más serio que estos programas presentan, es la necesidad de que el usuario proporcione información real a priori, que como se ha mencionado, debe ser lo más cercano posible al espectro que se desea obtener, para poder llevar a cabo la reconstrucción del espectro de forma eficiente, lo que en la practica resulta ser complicado ya que en la mayoría de las ocasiones se desconoce el espectro que se desea medir. Las desventajas

de los códigos descritos han motivado a los investigadores a buscar nuevas propuestas de reconstrucción. Con la evolución de nuestra compleja sociedad tecnológica y la introducción de nuevas nociones y herramientas teóricas innovadoras en el campo de los sistemas inteligentes, el campo de la IA esta sufriendo una gran evolución. En las últimas décadas se han intentado procedimientos novedosos para el proceso de reconstrucción de espectros basados en AG [31], RNAs [25],[26],[27],[28],[29], [37] y técnicas híbridas de Redes Neuronales Artificiales evolucionadas por medio de Algoritmos Genéticos (RNAG) [45],[46]. La reconstrucción de espectros de neutrones y el cálculo de dosis equivalentes mediante RNA a partir de las tasas de conteo del SEEB, es una técnica alternativa que ha recibido gran atención durante los últimos años debido a los exitosos resultados que se han obtenido, mismos que resuelven muchos de los problemas antes mencionados cuando se aplican las técnicas clásicas.

2.2 Redes neuronales artificiales (RNAs)

2.2.1 Introducción

El interés de trabajar con RNAs, ha surgido en reconocimiento de que el cerebro humano trabaja de una forma totalmente distinta a la de una computadora. Se puede decir que el cerebro es un sistema de procesamiento de información altamente complejo, no lineal y de cómputo paralelo. Éste, tiene la capacidad de organizar sus elementos conocidos como neuronas a fin de desempeñar determinadas tareas (percepción, reconocimiento de patrones, control motor) de forma mucho más rápida que la mejor computadora digital creada hasta nuestros días. De una forma muy general, una RNA es una máquina que está diseñada para modelar la manera en que trabaja el cerebro para desempeñar determinada función de interés; la red usualmente es implementada utilizando componentes electrónicos, o en su defecto, simulada como software en una computadora digital. Para alcanzar un buen desempeño, las redes neuronales emplean una conexión masiva de células de procesamiento fundamentales, llamadas neuronas [47].

El primer paso hacia las RNA se dio en 1943, cuando McCulloch y Pitts describieron como podrían trabajar las neuronas y modelaron una red neuronal simple con circuitos eléctricos.

La investigación en las RNAs ha experimentado tres períodos de intensa actividad: el primero fue en 1940 debido al trabajo de McCulloch y Pitts, pioneros en este campo, el segundo ocurrió en 1960 con el trabajo de Rosemblatt con el teorema de convergencia del perceptrón, y el trabajo de Minsky y Papert mostrando las limitaciones de un perceptrón simple, el tercer período se dio en 1980. El principal desarrollo en este período incluye el método de energía de Hopfield en 1982 y el algoritmo de aprendizaje de propagación inversa para perceptrones multicapa, propuesto en primer lugar por Werbos, reinventado varias veces y popularizado por Rumelhart[48].

Las redes neuronales artificiales se han aplicado exitosamente en diversos campos tales como la industria, electrónica, medicina, automóvil, estadística, entretenimiento etc. Ello en virtud de su característica más importante, la capacidad para aprender a partir de datos de entrada [47].

2.2.2 Neurona biológica vs neurona artificial

El cerebro humano consiste de aproximadamente 100 billones de neuronas que están unidas por cerca de 100 trillones de conexiones, lo que forma el objeto más complejo conocido en el universo. El procesamiento en el cerebro es principalmente paralelo y distribuido: la información es almacenada en conexiones, principalmente en capas de mielina en el axon de la neurona y de aquí, distribuida sobre la red hacia un gran número de neuronas interconectadas en paralelo. El cerebro es adaptivo desde su nacimiento hasta su muerte y aprende de eventos que surgen en el mundo exterior. Análogamente, las RNA son modelos computacionales inspirados en los principios computacionales desempeñados por las redes neuronales biológicas del cerebro; están compuestas de elementos simples que operan en paralelo "neurona artificial". Como sucede en su contraparte biológica, la función de la red esta determinada por las conexiones entre los elementos, y una RNA se puede entrenar para que realice una función particular ajustando los valores de las conexiones (pesos) entre los elementos. Así pues, según [47],se puede decir que las RNAs se parecen al cerebro en dos cosas:

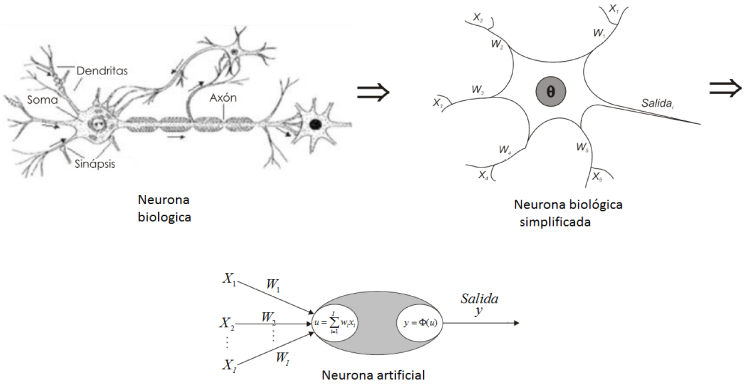

Figura 2.7 Modelo de neurona artificial a partir de su contraparte biológica

- 1.- El conocimiento de la red es adquirido de su entorno a través de un proceso de aprendizaje.

- 2.- La fuerza o peso de las conexiones interneuronales, conocidas como pesos sinápticos, son usadas para almacenar el conocimiento adquirido .

En la figura 2.7, se observa la estructura básica de una neurona biológica y las similitudes que guarda la neurona artificial respecto a ésta. De la figura 2.7 se observa lo siguiente:

-Las entradas Xi representan las señales que provienen de otras neuronas y que son capturadas por las dendritas.

-Los pesos Wi son la intensidad de la sinapsis que conecta dos neuronas; tanto Xi como Wi son valores reales.

-θ es la función umbral que la neurona debe sobrepasar para activarse; este proceso ocurre biológicamente en el cuerpo de la célula.

- Cada señal de entrada pasa a través de una ganancia o peso, llamado peso sináptico o fortaleza de la conexión cuya función es análoga a la de la función sináptica de la neurona biológica.

-Los pesos pueden ser positivos (excitatorios), o negativos (inhibitorios), el nodo sumatorio acumula todas las señales de entrada multiplicadas por los pesos y las pasa a la salida a través de una función de transferencia (también llamada función de activación o función de umbral).

La entrada neta (suma ponderada) a cada neurona puede escribirse mediante la ecuación 2.4:

$$u = \sum_{i=1}^{I} w_i x_i \tag{2.4}$$

donde w_i representa el i-ésimo peso sináptico y x_i la i-ésima entrada de la neurona. La salida y de la función de activación, se expresa en la ecuación 2.5

$$y = \Phi(u) \tag{2.5}$$

donde Φ denota la función de activación de la neurona.

2.2.3 Topología de Redes Neuronales Artificiales

Las RNAs están conformadas de elementos más simples llamados neuronas operando en paralelo; ahora bien, la manera en la cual estos elementos se estructuran dentro de la RNA, está íntimamente ligado con el algoritmo de aprendizaje usado para entrenar la RNA. Por consiguiente, podemos hablar de algoritmos o reglas de aprendizaje usados en el diseño de RNAs. Uno de los algoritmos más utilizados en el aprendizaje de las RNAs es el algoritmo de propagación inversa.

Como en su contraparte biológica, las conexiones entre estos elementos básicos determina en gran medida la funcionalidad de la red. Se puede entrenar una red neuronal artificial para que desempeñe cualquier función particular de interés, ajustando los valores de las conexiones (pesos) entre los elementos [47].

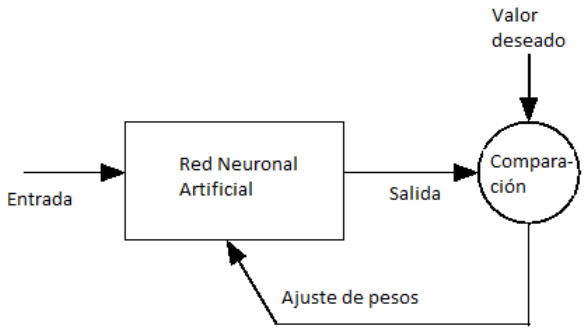

Figura 2.8 Esquema básico del entrenamiento de una RNA

Básicamente, las RNAs son entrenadas o ajustadas, de tal forma que una entrada en particular conduce a una salida deseada o esperada. Tal situación se ilustra en la figura 2.8, donde, basada en una comparación entre la salida misma de la red y el valor deseado, la red se ajusta a fin de empatar con el valor deseado.

Típicamente, una gran cantidad de pares entrada-salida son necesarios para entrenar una red. Tal y como se puede apreciar en la figura 2.9 a la neurona se le exhiben $p_1, p_2, ..., p_R$ entradas individuales, las cuales se multiplican con los pesos correspondientes $W_{1,1}, W_{1,2}, ..., W_{1,R}$ pertenecientes a la matriz de pesos W [49]. La neurona tiene una ganancia b (bias de la neurona), que llega al mismo sumador al que llegan las entradas multiplicadas por los pesos, para formular la salida n, expresándolo matemáticamente obtenemos la ecuación 2.6.

$$n = W_{1,1}p_1 + W_{1,2}p_2 + ... + W_{1,R}p_R + b \tag{2.6}$$

Para el caso en el que se tenga una neurona con demasiados parámetros, la notación anterior resulta inapropiada y tal vez sea más conveniente definir una notación abreviada tal y como se muestra en la figura 2.10 en donde el vector de entrada P se representa por la barra solida vertical de la izquierda. Las dimensiones de éste, se muestran en la parte inferior de la variable como *Rx1*, indicando que el vector de entrada es un vector columna de R elementos. Las

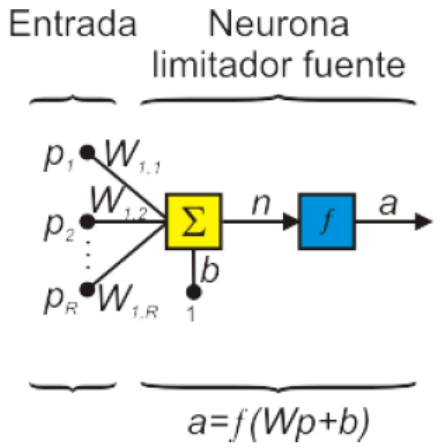

Figura 2.9 Modelo básico de una neurona con p_R entradas

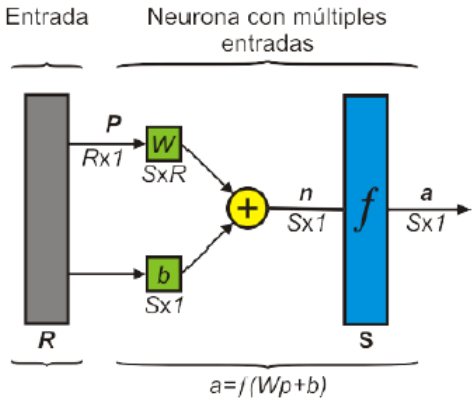

Figura 2.10 Notación comúnmente utilizada para representar los parámetros de una neurona en términos de matrices

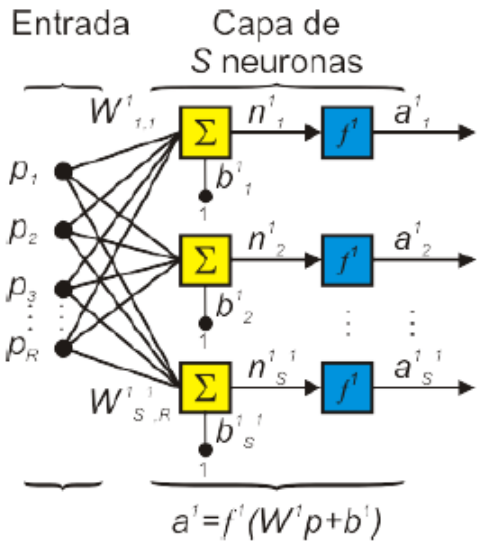

Figura 2.11 Capa de S neuronas

entradas van a la matriz de pesos W conformada por R filas. Una constante "1" entra a la neurona multiplicada por la ganancia escalar b. La salida de la red a, es en este caso un escalar resultado de evaluar n en la función de activación f.

Dentro de una RNA, los elementos de procesamiento se encuentran agrupados por capas. Una capa es una colección de neuronas; de acuerdo a la ubicación de la capa en la red neuronal, esta recibe diferentes nombres:

Capa de entrada: Recibe las señales de la entrada de la red, algunos autores no consideran el vector de entrada como una capa pues allí no se lleva a cabo ningún proceso.

Capas ocultas: Estas capas son aquellas que no tienen contacto con el medio exterior, sus elementos pueden tener diferentes conexiones y son estas las que determinan las diferentes topologías de la red.

Capa de salida: Recibe la información de la última capa oculta y transmite la respuesta al medio externo.

En la figura 2.11 se presenta una red de una sola capa con S neuronas, en la cual cada una de las R entradas se conecta a cada una de las neuronas, la matriz de pesos tiene ahora S filas.

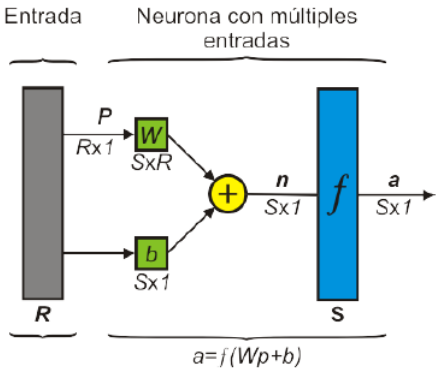

Figura 2.12 Capa de S neuronas en notación simplificada

La RNA anterior incluye la matriz de pesos, los sumadores, el vector de ganancias, la función de transferencia y el vector de salida de forma desglosada. Esta misma capa se observa en notación más simplificada en la figura 2.12.

En la figura 2.12 se han dispuesto los símbolos de las variables de tal manera que describan las características de cada una de ellas, por ejemplo, la entrada a la red es el vector P cuya longitud R aparece en su parte inferior, W es la matriz de pesos con dimensiones *SxR* expresadas debajo del símbolo que la representa dentro de la red, a y b son vectores de longitud S, que como se ha mencionado, representan el número de neuronas de la red.

Ahora, si se considera una red con varias capas (red multicapa), cada capa tendrá su propia matriz de pesos W, su propio vector de ganancias b, un vector de entradas netas n y un vector de salidas a.

La versión completa y la versión en notación abreviada de una red de tres capas, pueden ser visualizadas en las figuras 2.13 y 2.14 respectivamente.

Para esta red se tienen R entradas, S^1 neuronas en la primera capa, S^2 neuronas en la segunda capa, las cuales pueden ser diferentes; las salidas de las capas 1 y 2 son las entradas a las capas 2 y 3 respectivamente, así la capa 2 puede ser vista como una red de una capa con $R = S^1$ entradas, $S^1 = S^2$ neuronas y una matriz de pesos W^2 de dimensiones S^1xS^2.

En general, se pueden identificar tres clases fundamentales de arquitecturas de RNAs: redes de una capa con propagación hacia delante (Single-layer feedforward networks), redes multicapa de propagación hacia delante (Multilayer feedforward networks) y las redes neuronales

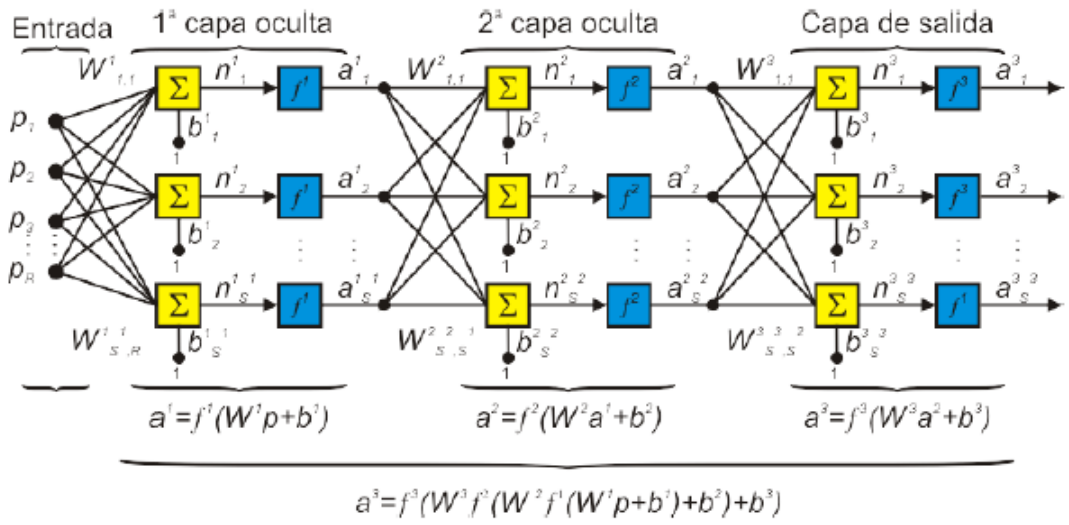

Figura 2.13 RNA de 3 capas con notación completa

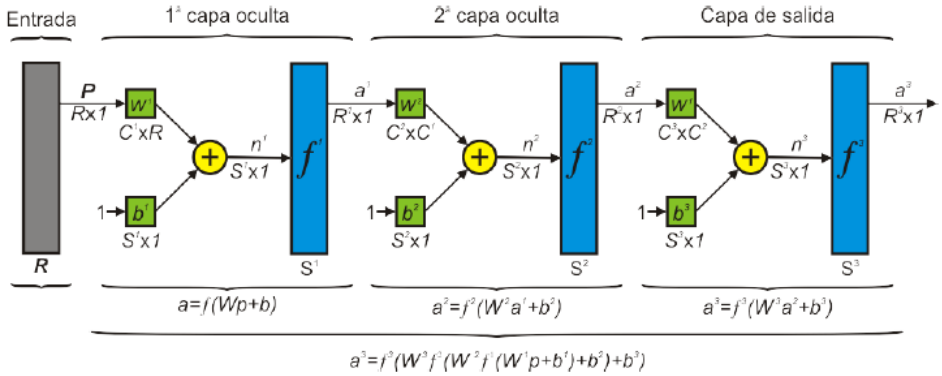

Figura 2.14 RNA de 3 capas con notación simplificada

artificiales recurrentes.

2.2.4 Red perceptrón multicapa (MLP)

El perceptrón multicapa MLP (Multi-Layer Perceptrón) por sus siglas en inglés, es una red neuronal del tipo propagación hacia delante (feedforward) que consiste de un nodo o capa de entrada, seguido de dos o más capas de perceptrones (nombre con el que se le conoce a la neurona en este tipo de redes) y con su última capa como capa de salida. De igual manera, las capas entre la capa de entrada y la capa de salida, son conocidas como capas ocultas. En estas RNAs la señal de entrada se propaga únicamente hacia delante de capa en capa, conformando así una generalización de la red perceptrón de una capa.

Este tipo de RNAs ha sido aplicado exitosamente para resolver difíciles y variados problemas, entrenándolas de manera supervisada con un algoritmo muy popular conocido como *error back-propagation algorithm* ó algoritmo de propagación inversa del error.

Básicamente, el algoritmo de aprendizaje *backpropagation*, consiste en dos pasos a través de las diferentes capas de la red. Un paso en dirección hacia delante y un paso hacia atrás. En el paso hacia delante un vector de entradas es aplicado a los nodos de la red, y se propaga a través de la red capa por capa. Finalmente, un conjunto de salidas se produce en la salida en respuesta a este estímulo. En cuanto al paso hacia atrás, los pesos sinápticos son todos ajustados en concordancia con una regla de corrección de error. Específicamente, la respuesta actual de la red es substraída de una respuesta deseada (objetivo), para producir una señal de error. Entonces, esta señal de error es propagada hacia atrás a través de la red en dirección contraria a la de las conexiones sinápticas; de aquí su nombre "propagación inversa del error".

Un MLP tiene tres características distintivas:

1.- El modelo de cada neurona en la red incluye una función de activación no lineal. Pero esta no linealidad debe de ser suave, tal que sea diferenciable en cualquier punto. Una función comúnmente utilizada que satisface estos requerimientos es la función sigmoidal, definida por

la función logística 2.7:

$$y_j = \frac{1}{1 + \exp(-v_j)} \tag{2.7}$$

donde v_j es el campo local inducido (la sumatoria de todas las entradas multiplicadas con su respectivo peso sináptico, más el bias) de la neurona j, y y_j es la salida de la neurona. La presencia de no linealidades es importante porque en cierta forma emula la fase refractaria de las neuronas reales.

2.- La red contiene dos o más capas de neuronas ocultas, que no forman parte ni de la entrada ni de la salida de la red. Estas neuronas ocultas posibilitan a la red de aprender tareas complejas al extraer las características más significativas de los patrones de entrada (vectores).

3.- La red exhibe un alto grado de conectividad determinado por la sinapsis de la red. Un cambio en la conectividad de la red requiere un cambio en la población de las conexiones sinápticas o de sus pesos.

Es a través de la combinación de estas características, junto con la habilidad de aprender a través de la experiencia de donde el perceptrón multicapa deriva todo su poder de procesamiento. No obstante, estas mismas características son responsables de nuestras deficiencias en cuanto al entendimiento sobre el comportamiento de estas redes; en cierto sentido, el proceso de aprendizaje es el que debe decidir que características del patrón de entrada debe de ser representado por las neuronas ocultas.

El desarrollo del algoritmo propagación inversa del error representa un punto de referencia, ya que proporciona un método computacional eficiente para el entrenamiento de RNAs.

2.3 Implementación hardware de RNAs

2.3.1 Introducción

El explícito paralelismo en las redes neuronales artificiales ha provocado gran interés en acelerar su ejecución usando hardware personalizado; es decir, hardware hecho a la medida de una RNA, basados en tecnologías tales como Application Specific Integrated Circuits (ASICs) y Field Programable Gate Arrays (FPGAs)[50].

Por mucho, la razón más citada para el desarrollo de tales sistemas, es que los microprocesadores de propósito general no explotan el paralelismo inherente en las redes neuronales y que arquitecturas altamente paralelas son requeridas para este fin. No obstante, esta premisa sólo puede justificarse únicamente si este alto desempeño deseado es alcanzable a un costo razonable.

Es evidente que en general las FPGAs no pueden compararse con procesadores ASIC en cuanto a desempeño; en este sentido, los FPGA siempre han estado rezagados en comparación con éstos. A pesar de esto, si uno considera las estructuras FPGA como una alternativa para implementar software, algo así como un tipo de procesador general, entonces, es posible que los FPGAs puedan entregar mejores relaciones costo-desempeño para determinadas aplicaciones. Y esto aunado a la capacidad de reconfiguración con la que cuentan, les permite extenderse a un amplio rango de aplicaciones; por ejemplo, diferentes tipos de redes neuronales [2]. Así, la principal ventaja de las FPGA es que pueden ofrecer una mejor relación costo-desempeño que cualquier ASIC o procesador de propósito general y con una mayor flexibilidad. Implementaciones dedicadas hardware de redes neuronales prometen proveer mayor velocidad y menor consumo de energía comparado con implementaciones software implementadas sobre microprocesadores. Las RNAs son populares entre la comunidad de "máquinas inteligentes" y han sido ampliamente aplicadas a problemas tales como clasificación, predicción y aproximación [50].

2.3.2 ASIC vs FPGA

Saltan a la luz dos puntos críticos que prevalecen sobre las RNAs basadas en ASICs, el primero es que si uno exige realizar una RNA con cierta flexibilidad, entonces, se acaba por realizar una estructura demasiado parecida a una FPGA. Esto es, un pequeño número de diferentes tipos de unidades funcionales que pueden ser configurados de diferentes maneras de acuerdo al tipo de red neuronal a implementar y que en general, termina por no contar con la misma flexibilidad que su contraparte FPGA, ya sea en su programabilidad o reconfigurabilidad. El segundo punto es que, el hardware por sí solo no constituye una estructura típica de cómputo, software también es requerido. Pero falta desarrollo de software para ASICs; esto debido a la limitada base de usuarios. Algo que no es tanto problema con las FPGAs ya que son más utilizados. Finalmente, una desventaja de los diseños basados en ASICs, es que la mayoría de las implementaciones tienden a concentrarse solo en el alto grado de paralelismo de las redes neuronales, generalmente ignorando las implicaciones de la ley de Amdahl, la cual cita:

> *"La velocidad se verá limitada por algún proceso serial o proceso paralelo de muy baja velocidad involucrado en el sistema".*

Además, incluso en la mayoría de redes neuronales donde se puede explotar el paralelismo, debe notarse que hay ciertas clases de paralelismo.

A pesar de todas las exigencias que se han hecho y que se seguirán haciendo en materia de neurocomputadoras personalizadas, a la fecha, no ha habido una neurocomputadora aplicada a problemas de redes neuronales que haya superado la mejor computadora convencional de sus tiempos. La promesa de los FPGAs es que, en esencia, ofrecen la posibilidad para realizar maquinas semipersonalizadas para redes neuronales; y con continuos desarrollos en su tecnología, son la mejor esperanza para superar los procesadores convencionales (relación costo desempeño)[10].

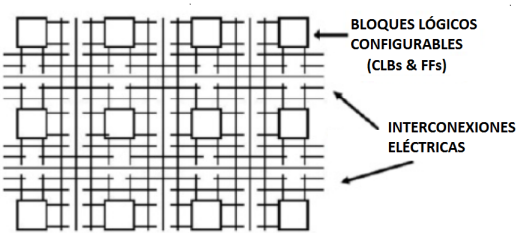

Figura 2.15 La estructura interna de un FPGA consiste de un arreglo bidimensional de bloques de lógica controlables CLBs

2.3.3 Arquitectura de FPGA

Los FPGAs se han convertido en una de las tecnologías más exitosas para el desarrollo de sistemas que requieren operación en tiempo real, además de que pueden cubrir un amplio rango de condiciones de operación. Los FPGA son arreglos bidimensionales de bloques lógicos y flip-flops, con interconexiones eléctricamente programables entre estos bloques. Dichas interconexiones consisten de interruptores programables, característica que diferencía a los FPGA del resto de los circuitos integrados ó CI (un CI habitual tiene interconexiones metálicas entre sus bloques lógicos). Los FPGA proporcionan a sus usuarios control sobre las conexiones de los bloques lógicos y la función que debe desempeñar cada bloque. Los bloques lógicos pueden ser configurados de manera que proporcionen funcionalidad tan simple como la de un transistor, o tan compleja, como la de un microprocesador. De igual forma, pueden ser usados tanto para funciones de lógica combinacional como de lógica secuencial.

El enrutamiento en los FPGAs consiste en "segmentos de cable" de longitud variable que pueden ser interconectados por medio de interruptores programables eléctricamente. La cantidad de bloques lógicos utilizados de un FPGA van en relación directa con la cantidad de segmentos de cable usados para el enrutamiento [51].

La figura 2.15 muestra en forma simplificada, la estructura interna de un FPGA. Los FPGA fueron introducidos como una alternativa a los CIs para implementar sistemas enteros sobre un circuito, y para proporcionar flexibilidad en cuanto a reprogramabilidad para el usuario. Otra enorme ventaja de esta tecnología es que con la ayuda de paquetes CAD (Computer Aided Design) las aplicaciones pueden ser implementadas en un corto tiempo (sin conexionado físico, sin diseño de máscaras, sin manufactura de CIs).

Este dispositivo puede ser programada utilizando un lenguaje llamado Hardware Description Lenguaje (HDL), y este contiene dos tipos de lenguajes, Very High Description Lenguaje (VHDL) y Verilog. Como su nombre indica, VHDL es un lenguaje descriptor de hardware para diseños digitales. Se originó de un programa del gobierno estadounidense en el desarrollo de circuitos integrados de muy alta velocidad. VHDL se parece a un lenguaje de programación cualesquiera (aunque tiene una estructura única), con modelado de sistemas tanto secuenciales como concurrentes. En el pasado, la gran mayoría de usuarios de esta tecnología eran diseñadores hardware con vasto conocimiento y experiencia en el diseño de circuitos ya fuese con Verilog o VHDL. Los lenguajes de descripción hardware fueron diseñados para facilitar la implementación de grandes diseños digitales mediante su representación en ecuaciones de lógica booleana, así como a través del uso de construcciones semánticas de alto nivel inspiradas en lenguajes de cómputo tradicionales.

2.3.4 Implementación de RNAs en FPGA

Paralelismo, modularidad y adaptación dinámica son tres características computacionales asociadas con RNAs. Las arquitecturas basadas en FPGAs resultan bastante apropiadas para implementar RNAs siempre y cuando se pueda explotar el paralelismo y la reconfigurabilidad para adaptar los pesos y topologías de una RNA [52].

La realización de RNAs con un gran número de neuronas es aun una tarea cambiante debido a que los algoritmos para RNA presentan una alta densidad de multiplicaciones y resulta relativamente costoso implementar esta operación en los FPGAs. Sin embargo, es posible explotar la reconfigurabilidad de esta tecnología mediante algunas estrategias, a fin de implementar estas RNAs en FPGAs de una forma barata y eficiente. Todas las implementaciones de RNAs intentan de una manera u otra, explotar la reconfigurabilidad hardware del FPGA y algunas de las estrategias a seguir para ello son las siguientes:

Prototipo y Simulación: Explota el hecho de que el hardware basado en FPGA puede ser rápidamente reconfigurado un número ilimitado de veces. Esta característica permite el

diseño de diferentes implementaciones de RNA y algoritmos de aprendizaje para su desarrollo o únicamente prueba de concepto.

Mejoramiento en densidad: Se refiere a métodos que incrementan la cantidad de funcionalidad efectiva por unidad de área de circuito utilizando lo que se conoce como *run-time reconfiguration*; por ejemplo, en[53] se implementó el algoritmo *back-propagation* divido en una secuencia de pasos adelante-atrás y la implementación de cada etapa es ejecutada sobre los mismos recursos del FPGA.

Adaptación en la topología: Se refiriere al hecho de que el dinamismo en la arquitectura de los FPGAs permite la implementación de RNAs con topologías modificables.

Como se mencionó en secciones atrás, la elaboración de RNAs involucra una fase de entrenamiento durante la cual la estructura de la red neuronal y los pesos sinápticos son iterativamente ajustados bajo el control de un algoritmo de entrenamiento, para identificar y aprender las características de los patrones de entrenamiento. La red así obtenida es entonces puesta a prueba con datos no vistos previamente por la red, midiendo su capacidad para generalizar [50]. El proceso de diseño para RNAs hardware típicamente involucra *offline learning*, lo que en traducción literal equivale a decir a que el aprendizaje de la RNA se realiza fuera del chip, utilizando simuladores software, para después de ello descargar la red entrenada hacia el chip FPGA. Y aunque existen en la literatura varias técnicas para la realización *on-chip learning* (realización de redes neuronales en su totalidad dentro de el FPGA), queda más allá de los objetivos de este trabajo. Por lo que el desarrollo de este documento gira en torno al primer método discutido.

Debido a la existencia de una amplia variedad de arquitecturas de redes neuronales y la falta de un profundo entendimiento de sus capacidades, la mayoría de las RNAs son implementaciones dedicadas de redes populares tales como, perceptrón multicapa, redes Hopfield o redes Kohonen. En [52] se puede encontrar una minuciosa revisión de implementaciones hardware en FPGA. Según [50], el universo de implementaciones hardware de RNAs, se puede clasificar tal y como se aprecia en la figura 2.16.

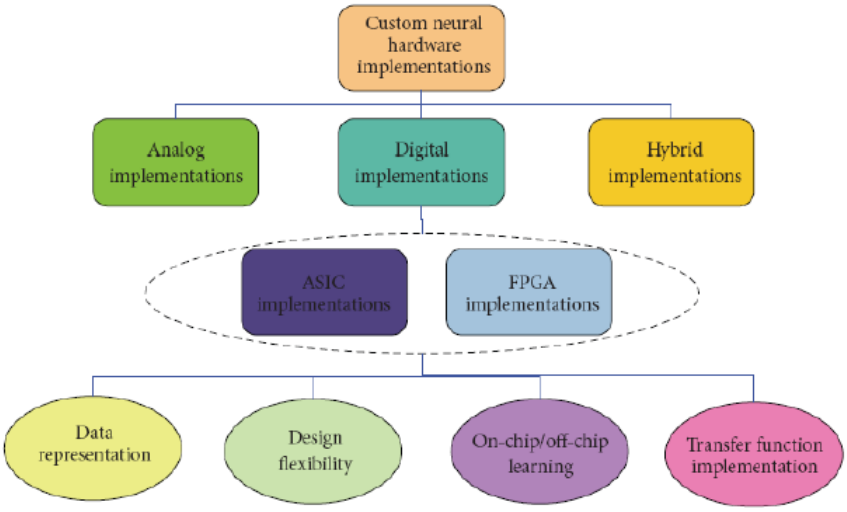

Figura 2.16 Fases en el diseño e implementación de una RNA

Las implementaciones de redes neuronales basadas en FPGAs ofrecen alta precisión, confiabilidad y programabilidad. Los pesos sinápticos y bias son típicamente almacenados en memorias digitales o registros dependiendo del compromiso que se quiera establecer en términos de velocidad, tamaño del circuito, tamaño de la red, flexibilidad del diseño, etc. Tal y como se desprende de la fase de diseño de la figura 2.16, son 4 las etapas más importantes a considerar en la implementación de redes neuronales artificiales y que se discuten a continuación.

2.3.4.1 Representación de los datos

La implementación digital de una red neuronal representa los valores reales tales como pesos, función de activación, entradas, salidas, bias, etcétera usando representaciones digitales de los datos tales como punto fijo, punto flotante o representaciones especializadas como por ejemplo codificación por tren de pulsos (pulse estream encoding). La elección particular de una representación es un compromiso entre velocidad, precisión de los datos y tamaño de la circuitería. Existen algunos trabajos de investigación que intentan demostrar que es posible entrenar RNAs con pesos enteros. El interés de utilizar representaciones enteras de los

datos proviene del hecho que las multiplicaciones (una operación muy importante en la implementación de RNAs) pueden ser implementadas más eficientemente que sus homólogas en punto flotante. Existen también algoritmos especiales de entrenamiento que utilizan números enteros que son potencias de dos como pesos. La ventaja estos enteros potencias de dos es que las multiplicaciones en una RNA pueden ser reducidas a series de corrimiento.

Pocos intentos se han realizado para implementar RNAs en hardware FPGA con pesos en punto flotante. ya que las unidades de aritmética de punto flotante son más lentas, grandes y complicadas de implementar que su contraparte en punto fijo. Para implementaciones en FPGA la elección predilecta ha sido la representación en punto fijo debido a las limitaciones en estos chips; sin embargo, avances en las densidades de las FPGAs pueden hacer las representaciones en punto flotante prácticas. Sin embargo, existe un compromiso entre precisión mínima, rango dinámico de los datos y el área requerida para la implementación de las unidades de aritmética. Una alta precisión contiene menos errores de cuantización pero requiere grandes unidades aritméticas; mientras que una menor precisión permite unidades aritméticas más pequeñas, más rápidas y más eficientes en potencia; pero pueden tener errores de cuantización muy grandes que pueden limitar la capacidad de la red neuronal para aprender y resolver un problema. Un análisis numérico debe de ser realizado para establecer un equilibrio entre estos compromisos y de esto modo determinar la mínima precisión requerida para determinada aplicación.

2.3.4.2 Flexibilidad del diseño

Una importante elección en la implementación de redes neuronales es el grado de adaptación estructural y flexibilidad de los parámetros sinápticos. Una implementación con una estructura de red y pesos fija únicamente puede ser usada en solo una aplicación, necesitándose así un rediseño total de la RNA para otra aplicación. Para implementaciones ASIC esto podría resultar bastante costoso; una ventaja de las implementaciones de RNAs en FPGAs es la capacidad *runtime reconfiguration* (ya citada anteriormente) para redirigir el mismo dispositivo FPGA hacia cualquier número de aplicaciones diferentes de RNAs, reduciendo sustancialmente los costos de la implementación. El runtime reconfiguration de los FPGAs puede ser utilizado

para el mejoramiento en densidad para lograr implementar redes de mayores tamaños multiplexando sus diferentes etapas en el tiempo. Esto incrementa la funcionalidad por unidad de área reconfigurable de la FPGA. A su vez, esta peculiaridad de la FPGA puede ser utilizada en la adaptación de la topología. RNAs con diferentes estructuras y parámetros pueden ser cargadas via runtime en el FPGA.

2.3.4.3 On-chip/off-chip Learning

Los algoritmos de entrenamiento para RNAs adaptan la estructura de la red y parámetros sinápticos de manera iterativa basados en una función de error entre la salida actual y la salida deseada. De aquí, un diseño de red en hardware debe tener la capacidad, de dinámicamente adaptar su estructura y parámetros internos. La mayoría de implementaciones reportadas en la literatura utilizan simulaciones software para entrenar la red, y la red así obtenida es descargada al hardware (offline). Un excelente ejemplo de *On-chip learning* puede verse en [53] quién realizó una implementación del algoritmo BP (backpropagation) diciéndolo temporalmente en tres secuencias: feedforward, error backpropagation, and synaptic weight update.

2.3.4.4 Función de activación

Las funciones de activación o funciones de transferencia usadas en RNAs son por lo regular no lineales con gran tendencia hacia funciones sigmoides, ejemplos de ello incluyen la tangente hiperbólica y la función logística sigmoidal. Para su representación digital, generalmente se utiliza aproximación lineal por segmentos ya que es implementada en hardware directamente como circuito o como una tabla *look-up*. La implementación directa de la función de activación como circuito requiere un rediseño de su hardware para cada aplicación que requiere función de activación diferente. En tal escenario, la LUT (Look-Up Table) como aproximación puede ser más flexible, ya que los valores de la función pueden ser precalculados fuera del FPGA y simplemente se guardan en la LUT. Sin embargo, las LUTs pueden ocupar un espacio significativamente mayor que su contraparte. Para redes neuronales, la implementación de

estas funciones es uno de los dos problemas de diseño aritmético más problemáticos y para su implementación hardware, precisión, desempeño y costo son factores realmente importantes. Los últimos dos implican que muchas de las mejores técnicas que han sido desarrolladas en análisis numérico (y que por lo regular son fácilmente implementadas en software) no son las más adecuadas para su implementación hardware.

Aproximaciones polinominales de grado superior pueden dar implementaciones con bajo nivel de error, pero debido al número de operaciones aritméticas involucradas (sumas y multiplicaciones) que deben de ser realizadas, no resultan muy aptas para su implementación en hardware. Algo parecido ocurre en los métodos que utilizan LUTs a menos que las tablas sean lo bastante pequeñas (tablas muy grandes resultaran muy lentas y con altos costos de implementación).

Dadas las tendencias en tecnología, resulta aparente que en lo presente, la mejor técnica para la evaluación hardware de una función es una combinación de polinomios de grado inferior y pequeñas LUTs. Este es el caso tanto para tecnologías ASIC como FPGA, y especialmente esta última, ya que en el estado del arte de estos dispositivos se puede contar con sustanciales cantidades de memoria dispersa en el dispositivo, así como con unidades aritméticas (multiplicadores y sumadores). La combinación de polinomios de grado inferior (principalmente los de grado uno) y LUTs no es nueva, los principales cambios han sido hechos en como seleccionar los mejores puntos de interpolación y en como hacer para que las tablas se mantengan pequeñas. Este tipo de implementación tiene dos ventajas, la primera es que exactamente el mismo hardware puede ser utilizado para realizar diferentes funciones, dado que únicamente los coeficientes de los polinomios necesitan ser cambiados (contenidos en las LUTs). La segunda es que se relacionan bastante bien con los dispositivos FPGA, que actualmente ya cuentan con sumadores, multiplicadores y memoria como parte de su estructura interna. En base a las consideraciones anteriores, el método de interpolación lineal para la aproximación de la función sigmoidal, resulta clave en el desarrollo de este trabajo.

Capítulo 3

Materiales y métodos

3.1 Tarjeta electrónica de desarrollo *Basys2*

Como ya se mencionó anteriormente, existe una importante razón para que el desarrollo hardware se realice en FPGAs y es porque este tipo de tecnología, ofrece el camino más directo para su implementación al ofrecer cantidades importantes de memoria, procesamiento paralelo y reconfigurabilidad del sistema.

La realización de este trabajo se desarrolló al amparo de la tarjeta de desarrollo Basys2. Ésta tarjeta es una plataforma para el diseño y la implementación de circuitos electrónicos, basado en un FPGA manufacturado por Xilinx, de la familia **Spartan 3E**. De tal forma que proporciona todos los elementos necesarios para la elaboración de circuitos que pueden ir de lo simple a lo muy complejo.

La Basys2 cuenta con una amplia colección de periféricos de entrada-salida, además de todos los elementos necesarios para soportar un FPGA, por lo que incontables diseños pueden llevarse a buen término sin la necesidad de añadir componentes.

En la figura 3.1 se ilustran los periféricos de entrada salida, interconectados de un FPGA Xilinx Spartan 3E, de la misma figura podemos destacar lo siguiente:

- Un FPGA Sartan3E de Xilinx de 100,000 compuertas en encapsulado CP132
- Puerto USB2 como interface para el flujo de los datos manejado por un chip Atmel
- Memoria Flash ROM para almacenar las configuraciones del FPGA

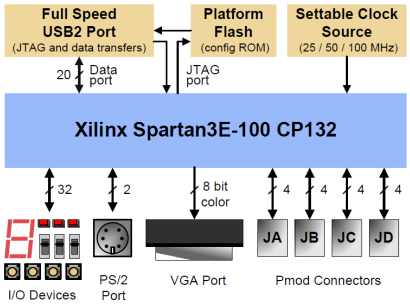

Figura 3.1 Diagrama de Bloques de la tarjeta de desarrollo Basys2

- 8 LEDs, 4 visualizadores de 7 segmentos, 4 botones, 8 interruptores de deslizamiento
- Puertos de 8 bits PS2 y VGA
- Reloj configurable (25/50/100MHz), más un socket para un segundo reloj.
- 4 receptáculos de 6 pines para conectores de expansión (conversores A/D, PWM, etc)
- Protección cortacircuitos en todos los componentes de E/S.

Claro está que se puede hacer todo un compendio de las características y bondades del FPGA Spartan 3E, sin embargo, nos abocamos a describir las características más importantes de la arquitectura de este dispositivo; para un estudio en detalle de este dispositivo se puede consultar [54].

La arquitectura de la familia Spartan 3E consiste de cinco elementos programables principales:

- **Bloques Lógicos Configurables (CLBs)** Contienen *Look-Up Tables* (LUTs) flexibles que implementan lógica y/o elementos de memoria.
- **Bloques de Entrada/Salida(IOBs)** Controlan del flujo de datos entre los pines de E/S y la lógica interna del dispositivo.
- **Bloques RAM** Provee el almacenamiento de datos en bloques de doble puerto de 18k bit.

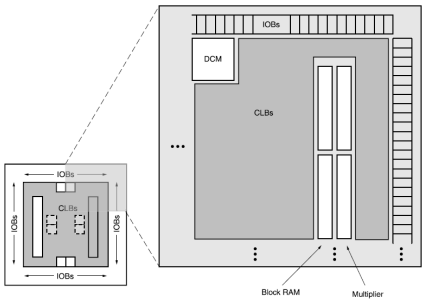

Figura 3.2 Arquitectura de la familia Spartan 3E de Xilinx

- **Bloques multiplicadores** Aceptan dos operandos de 18 bits como entrada y calcula su producto.
- **Bloques Manejadores de Reloj (DCM)** Proporcionan una solución totalmente digital de auto calibración, retraso, distribución, división y corrimiento de fase de las señales de reloj.

Estos elementos están organizados dentro del chip, tal y como se puede apreciar en la figura 3.2. Así mismo, este dispositivo está dotado de una basta red de interconexiones para entrelazar estos elementos básicos y trasmitir señales entre ellos (Cada elemento funcional tiene asociado una matriz de interruptores que permite multiples conexiones para el enrutado).

Esta tarjeta de desarrollo fue diseñada específicamente para funcionar con el ambiente de desarrollo ISE de Xilinx. El cual es una plataforma que permite desde la generación de código en VHDL, generación de diagramas eléctricos, maquinas de estados, pasando por la simulación y hasta su implementación en Chip.

3.2 Arquitectura de la Red Neuronal Artificial

Considerando que la mira de este trabajo es la descarga de una RNA previamente entrenada en MATLAB para la reconstrucción del espectro de neutrones; era necesario establecer una arquitectura de RNA lo más modular posible, con la finalidad de poder empatar la RNA hardware a su similar RNA software. Si bien, la mayoría de la literatura busca explotar el paralelismo inherente de las RNA para obtener tiempos menores de procesamiento, hay que pagar

un alto precio por el uso de esta filosofía, ya que los recursos pueden mermar rápidamente con el incremento del número de neuronas en la red. Además, bajo este esquema, difícilmente se puede obtener un diseño, que con mínimas modificaciones nos permita incrementar el número de neuronas en cada capa o inclusive el número de capas de procesamiento de la RNA.

Para la resolución de nuestro problema científico se optó por una arquitectura de RNA **cíclica**, que si bien se pierde en el ámbito de procesamiento paralelo, se gana mucho en la administración de los recursos y tan es así, que se pueden vaciar RNA de tamaños considerables en un FPGA Spartan 3E, que dicho sea de paso, es de las familias más modestas manufacturas por Xilinx, viéndose enormemente favorecido desde el punto de vista costo-beneficio.

En adición a lo anterior, en un proceso cíclico es necesario llevar un control estricto sobre el flujo de los datos (lo cual resulta antagónico en demasía si comparamos con el procesamiento paralelo) y es este estricto control, lo que nos permitirá en determinado momento indicarle a nuestra RNA si deseamos agregar más neuronas, cambiar de una función de activación a otra, o inclusive agregar más capas a la RNA.

Dicho lo anterior, en la figura 3.3 se ilustra la arquitectura propuesta para el desarrollo de este trabajo, de donde podemos observar que son tres los bloques de memoria ROM (extrema izquierda), los que recibirán las entradas, pesos y bias respectivamente; y que precisamente es en éstos dos últimos, donde reside el conocimiento de la RNA previamente entrenada. De estos puntos fluye la información hacia el bloque etiquetado MAC donde se calcula la suma de productos, y que a su vez se conecta con un bloque sumador, cuya función será la de añadir el *umbral* ó bias de la neurona. Conectado a éste último tenemos una variedad de hasta cuatro funciones de activación, de entre las cuales podemos elegir gracias a un multiplexor al que están acopladas.

Hasta esta punto hemos conformado la arquitectura esencial de una neurona artificial, y es justamente esta neurona la que habrá de llevar el cálculo de todas y cada una de las neuronas que conformarán la RNA, desde luego mediante un tratamiento cíclico de los datos. Función que vienen a complementar los bloques restantes de la figura 3.3, a saber, memoria RAM, multiplexores y unidad de control (FSM).

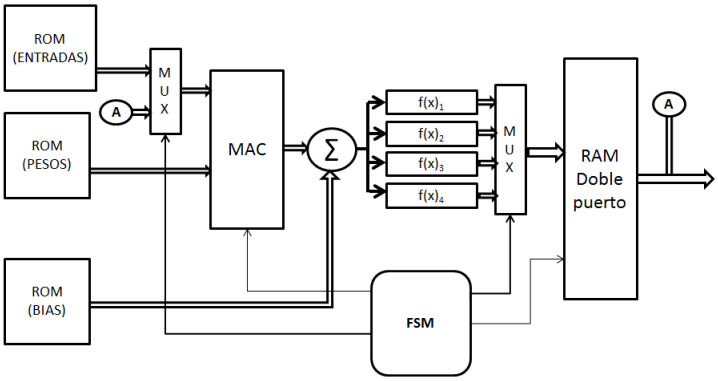

Figura 3.3 Arquitectura propuesta para el diseño de la RNA

En cuanto a la representación de los datos, todo el sistema fue concebido para trabajar bajo un esquema de 16 bits en punto fijo. De donde los 4 bits más significativos se usan para representar la parte entera y los doce restantes para representar la parte fraccionaria. Se hace notar que el resultado de la multiplicación en MAC debería de expresarse a 32 bits, no obstante, para mantener homogéneo el flujo de los datos, el resultado es truncado a 16 bits, cuidando se hagan los corrimientos necesarios para no alterar el resultado, no obstante, esto limita la precisión de la RNA a costa de simplicidad en el diseño.

En lo que se refiere a la RNA entrenada en MATLAB y que posteriormente se descargará hacia la RNA de la figura 3.3, es importante señalar que nuestro diseño únicamente admite RNAs del tipo perceptrón multicapa de dos capas, pero con la opción de añadir y prescindir de neuronas en ambas capas, así como la opción de cambiar de función de activación entre capas.

El número de unidades de procesamiento (neuronas), que se puedan configurar en el sistema dependerá de la cantidad de conexiones sinápticas entre neuronas, pero bajo la arquitectura propuesta, bien podrían procesarse hasta mil neuronas repartidas entre la capa uno y dos. Viéndose únicamente limitado nuestro diseño, por la capacidad de los bloques de memoria ROM *Entradas, Pesos y Bias*. Y los tiempos de procesamiento, que se podrían convertir en un factor a considerar en redes tan grandes.

La configuración o parametrización (como se le prefiera llamar) de nuestra RNA hardware es posible gracias al bloque de control etiquetado *FSM* (Finite State Machine), en donde el usuario puede realizar cambios tales como: número de neuronas en la capa uno, número de

neuronas en la capa dos, elegir de entre cuatro opciones de funciones de activación diferentes; o inclusive, la posibilidad de incrementar el número de capas de la red (para lo cual ya habría que realizar cambios más significativos dentro del código).

3.3 Generalidades del entorno de desarrollo

Para la resolución del problema científico planteado, partimos del punto en que ya existe una RNA entrenada y evaluada en MATLAB para la tarea de la reconstrucción del espectro de neutrones. Enseguida es necesario extraer el conocimiento de esta RNA y pre-procesar esta información de tal manera, que pueda ser "interpretada" por la RNA hardware. Ante tal reto, hubo que elaborar algunos códigos de programación en MATLAB (M-files), códigos de programación en VHDL, hacer uso de software para el diseño de circuitos en FPGA y finalmente delimitar los alcances en el desarrollo de este trabajo.

En la figura 3.4, se muestra un diagrama de flujo de las vicisitudes para la implementación de la RNA, partiendo como ya se dijo líneas atrás, de una RNA software (Diseñada, entrenada y evaluada en MATLAB), hasta su descarga en una RNA hardware(RNA embebida dentro del FPGA).

3.3.1 VHDL

VHDL es un lenguaje descriptor de hardware, es decir, mediante código computacional es posible describir el comportamiento de de un circuito o un sistema electrónico, a partir del cual, el circuito o sistema mismo puede ser alcanzado (implementado) . VHDL surgió como una iniciativa del Departamento de Defensa de los Estados Unidos en los 80s.

VHDL fue el primer software descriptivo de hardware en ser estandarizado por el IEEE, a través del estándar IEEE 1076. VHDL fue diseñado para la síntesis de circuitos así como para su simulación. La principal y fundamental ventaja de usar VHDL es que es un estándar, es decir, es totalmente independiente de cualquier desarrollador de tecnología y por tanto es un lenguaje portable y reusable. Las dos principales aplicaciones inmediatas de VHDL son en campo de dispositivos lógicos programables CPLDs y en el campo de ASICs. Una vez

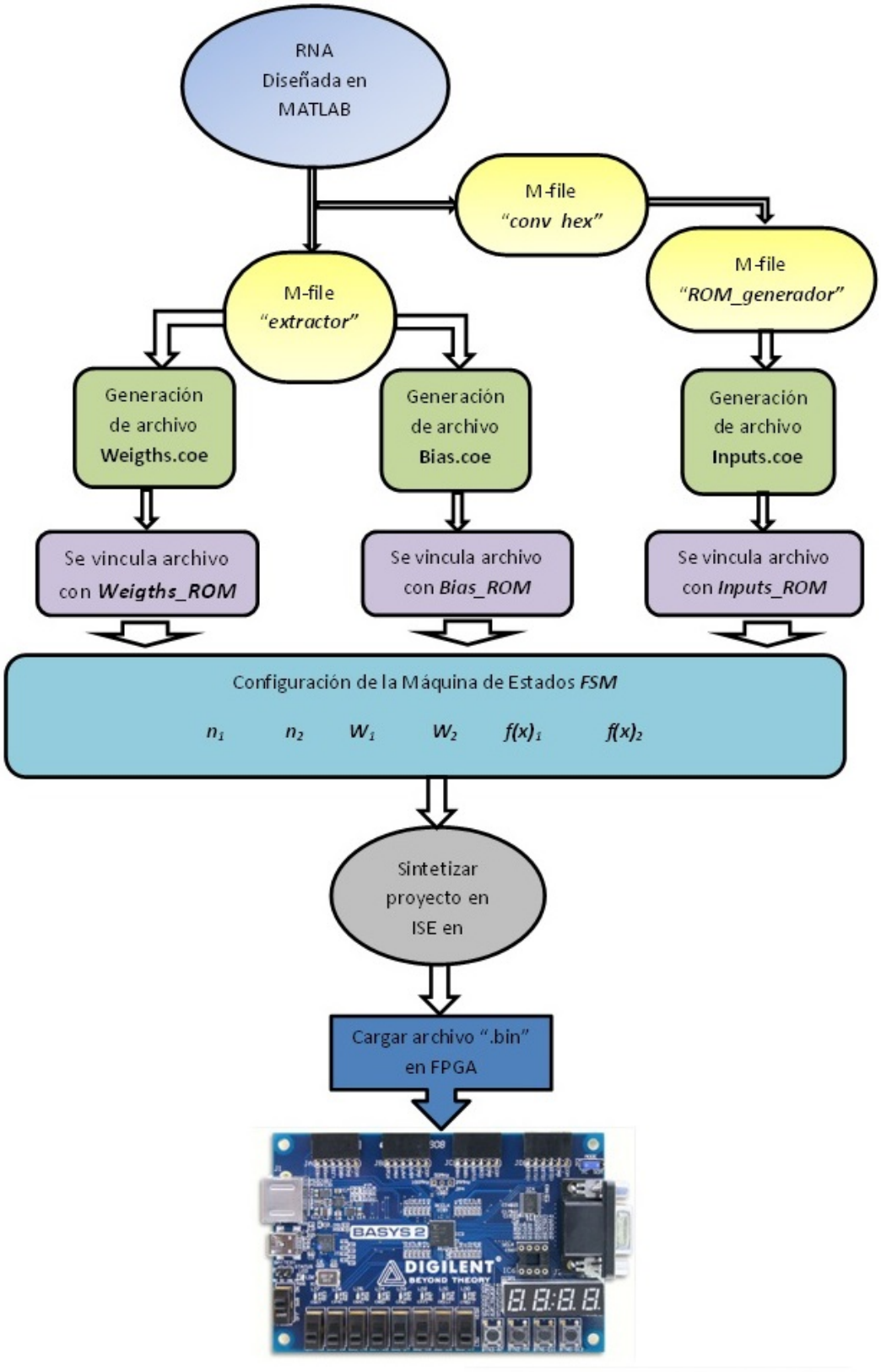

Figura 3.4 Proceso para la descarga de la RNA software en el FPGA

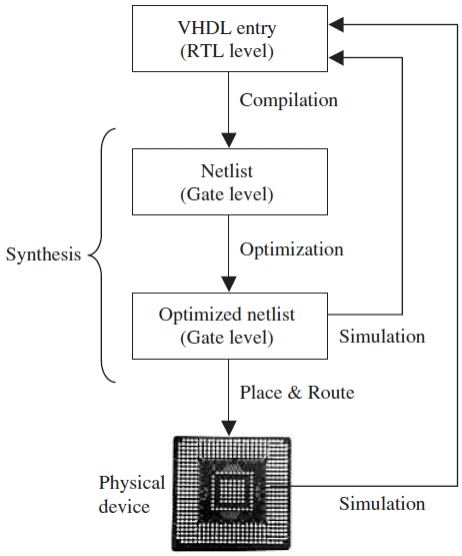

Figura 3.5 Fases en el diseño VHDL

que el código VHDL ha sido escrito, este puede ser usado para implementar el circuito en un dispositivo programable (de altera, Xilinx, Atmel, etc.) o puede ser enviado a una fabrica de circuitos integrados para generar un ASIC chip. Actualmente, muchos complejos circuitos integrados comerciales emplean estas fases de diseño durante su desarrollo.

Un aspecto importante de VHDL es que, contrario a un programa de computadora que es secuencial, VHDL es inherentemente concurrente (paralelo); por tal motivo, VHDL es considerado un código más que un programa.

Como se mencionó anteriormente, una de las mayores ventajas de VHDL es que permite la síntesis de un circuito en un dispositivo programable (PLD o FPGA) o en un ASIC. Los pasos a seguir para este fin se resumen en la figura 3.5

El diseño se inicia escribiendo el código VHDL, el cual se guarda con la extension .vhd y con el mismo nombre de la entidad (bloque principal del código). Como primer paso en el proceso de síntesis se compila, es decir, se realiza una conversion del lenguaje de alto nivel VHDL que describe el circuito deseado a nivel de registros de transferencia (RTL) hacia el netlist, nivel de compuertas (Gate level). El segundo paso es la optimización, la cual se realiza

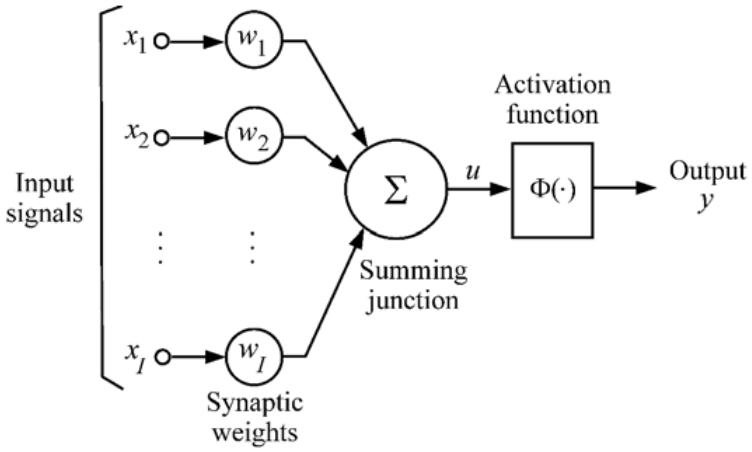

Figura 3.6 Modelo de una neurona artificial

en el netlist ya sea para velocidad o area; hasta este punto, el diseño puede ser simulado. Finalmente un software place-and-route generara el enrutado físico para el PLD/FPGA, o generara las máscaras para un ASIC.

Existen varias herramientas disponibles para la síntesis, implementación y simulación de circuitos conocidas como EDA Electronic Design Automation por sus siglas en inglés. Algunas de ellas ofrecidas de forma gratuita en kits de desarrollo tales y como la Quartus II de altera o el Spartan 3E de Xilinx.

3.4 Neurona Artificial Digital

La neurona artificial es el elemento básico de una red neuronal artificial; a su vez, los elementos básicos que conforman una neurona artificial son (1) un nodo de señales de entrada con índices $1, 2, ...I$, que reciben las correspondientes señales de entrada o vector de entradas, por decir x = $(x_1, x_2, ...x_I)^T$; (2) Un conjunto de conexiones sinápticas cuya fuerza o intensidad están representadas por un conjunto de pesos, denotados aquí por w = $(w_1, w_2, ...w_I)^T$; y (3) una función de activación Φ, que relaciona la entrada sináptica total con la salida (activación) de la neurona. Estos elementos principales en comento se ilustran en la figura 3.6.

La entrada sináptica total, u de la neurona esta dado por el producto interno del vector de entrada y el vector de pesos de la siguiente manera:

$$u = \sum_{i=1}^{I} w_i x_i \tag{3.1}$$

donde se asume que el umbral de activación (bias) esta incorporado en el vector de pesos. La salida de la función de activación, y , esta dado por

$$y = \Phi(u) \tag{3.2}$$

donde Φ denota la función de activación de la neurona. Invariablemente, el cómputo de los productos internos es una de las más importantes operaciones aritméticas a realizarse para la implementación hardware de una red neuronal. Esto no significa multiplicaciones y adiciones individuales, sino una multiplicidad de ellas, en otras palabras, una secuencia de multiplicaciones y adiciones conocidas como MAc (multiply-accumulate) por sus siglas en inglés.

La entrada total sináptica es transformada en la salida por medio de una función de activación no lineal.

3.5 Implementación de la Función de Activación

Existe una gran variedad de funciones de activación, y la elección de una u otra dependerá en gran medida del resultado deseado, e inclusive, muchas de las veces, de un procedimiento de prueba y error. Las funciones de activación más comúnmente utilizadas en RNAs, por lo regular son las no lineales, con gran tendencia hacia funciones sigmoides. Ejemplos de éstas incluyen la tangente hiperbólica y la función logística sigmoidal. Para nuestros fines habremos

de centrarnos en la implementación de la función de activación sigmoide unipolar, ya que es la función que se ha usado con éxito en la reconstrucción de espectros de neutrones.

Para las RNA, el diseño de estas funciones es uno de los dos problemas más importantes de diseño. Existen muchas técnicas para su implementación, tales como, aproximaciones polinomiales, algoritmos CORDIC, manejo de tablas, etc. Y la que mejor se adecúe a un problema en particular, dependerá de los recursos disponibles, la presición, el desempeño y el precio que se este dispuesto a pagar.

La implementación directa de la función de activación como un circuito requiere un rediseño hardware para cada vez que cambie la aplicación. Ante tal escenario, la LUT (Look-Up Table) como aproximación puede ser mucho más flexible, ya que los valores de la función pueden ser precalculados y simplemente almacenados en estas memorias LUT.

Para redes neuronales, la implementación de estas funciones es uno de los problemas de diseño aritmético más problemático y para su implementación hardware precisión, desempeño y costo son importantes. Los últimos dos implican que muchas de las mejores técnicas que han sido desarrolladas en análisis numérico (y que generalmente son fácilmente implementadas en software), no son las más apropiadas para implementar en hardware.

Aproximaciones polinomiales de grado superior pueden brindar un ajuste con bajo nivel de error, pero debido al número de operaciones matemáticas involucradas (sumas y multiplicaciones) que deben realizarse, no resultan muy aptas para su implementación en hardware. Algo parecido ocurre en los métodos que utilizan LUTs a menos que las tablas sean lo bastante pequeñas, ya que tablas muy grandes resultaran muy lentas y con altos costos para su implementación.

De lo anteriormente dicho, resulta aparente que la mejor técnica para la evaluación hardware de una función es una combinación de polinomios de grado inferior y pequeñas LUTs. Este es el caso tanto para tecnologías ASIC como para FPGA; especialmente esta última, ya que esta tecnología incorpora sustanciales cantidades de memoria dispersa dentro del mismo dispositivo. Así como también unidades aritméticas de suma y multiplicación.

La combinación de polinomios de grado inferior, especialmente los de grado uno y LUTs no es nueva, los principales cambios han sido hechos en como seleccionar los mejores puntos

de interpolación, y en como hacer para que las tablas se mantengan pequeñas. Este tipo de implementación guarda dos ventajas, la primera es que exactamente el mismo hardware puede ser utilizado para evaluar diferentes funciones ya que únicamente, los coeficientes precalculados de los polinomios y almacenados en LUTs son cambiados. La segunda es que, se relacionan bastante bien con los FPGA por contener dentro de su estructura interna, memoria, sumadores y multiplicadores.

Así pues, con base en lo anteriormente dicho, para la implementación de la función de activación sigmoide, proponemos una aproximación de primer orden como sigue:

dejemos $I = [L, U]$ sea un intervalo real con $L < U$, donde L representa el límite inferior y U el límite superior de dicho intervalo. Y digamos que $f : I \to \Re$, sea la función a aproximar, donde $\Re$, denota el conjunto de los números reales.

Supongamos ahora que $\widehat{f} : I \to \Re$, es una función lineal, esto es:

$$\widehat{f}(x) \; = \; c1 + \; c2x \tag{3.3}$$

para algunas constantes $c1$ y $c2$ que aproximan a f. Pudiendo además, definir el error relativo de tal aproximación de la siguiente manera:

$$\varepsilon(x) = \frac{f(x) - \widehat{f}(x)}{f(x)}, \quad x \in \; I \tag{3.4}$$

ahora, para encontrar unos valores razonablemente buenos para $c1$ y $c2$ de tal forma que 3.4 permanezca pequeño, se impone la siguiente condición:

$$f(L) = \widehat{f}(L), \qquad f(U) = \widehat{f}(U) \tag{3.5}$$

aplicando la condición anterior para la función sigmoide obtenemos el siguiente sistema de ecuaciones:

$$\begin{cases} c_1 + c_2 L \ = \ \frac{1}{1+e^{-U}} \\ c_1 + c_2 U = \ \frac{1}{1+e^{-L}} \end{cases} \tag{3.6}$$

resolviendo el sistema de ecuaciones 3.6 y dejando por simplicidad:

$$\theta = Ue^L - Le^U - Le^{(L+U)} + Ue^{(L+U)}$$

tenemos que la solución para c_1 y c_2 se puede expresar de la siguiente manera:

$$c_1 = \frac{\theta}{\theta + U - L + Ue^U - Le^L} \tag{3.7}$$

$$c_2 = \frac{e^U - e^L}{\theta + U - L + Ue^U - Le^L} \tag{3.8}$$

y la función de aproximación $\widehat{f}(x) = c_1 + c_2 x$ toma la forma:

$$\hat{f}(x) = \frac{\theta + x(e^U - e^L)}{\theta + U - L + Ue^U + Le^L}, \quad x \in I \tag{3.9}$$

y de igual forma, sustituyendo para el error relativo obtenemos:

$$\varepsilon(x) = \frac{-Le^L - L + Ue^U + U - e^{-x}\theta - xe^U}{\theta + U - L + Ue^U + Le^L} + \frac{-xe^{U-x} + xe^L + xe^{(L-x)}}{\theta + U - L + Ue^U + Le^L}, \quad x \in I \tag{3.10}$$

Resuelto el sistema de ecuaciones 3.6, tanto para $c1$ como para $c2$, así como también, para el error relativo de nuestra aproximación; se elaboró un código en MATLAB para evaluar 3.7 y 3.8, para posteriormente sustituirlas en 3.9 y así llegar finalmente a obtener una función sigmoide mediante aproximaciones de polinomios de primer grado.

Está claro, que cuantos más segmentos se utilicen para la función a aproximar cuanto mejor será la aproximación de la función. Por ejemplo, se ilustran dos situaciones, la primera de ellas con una aproximación de 2^4 segmentos, y la segunda de ellas con 2^5 segmentos.

Caso 1

Función a aproximar:

$$f(x) = \frac{1}{1 + e^{-x}}\ , \qquad \begin{cases} -8 \leq x < 8 \\ k = 4 \end{cases}$$

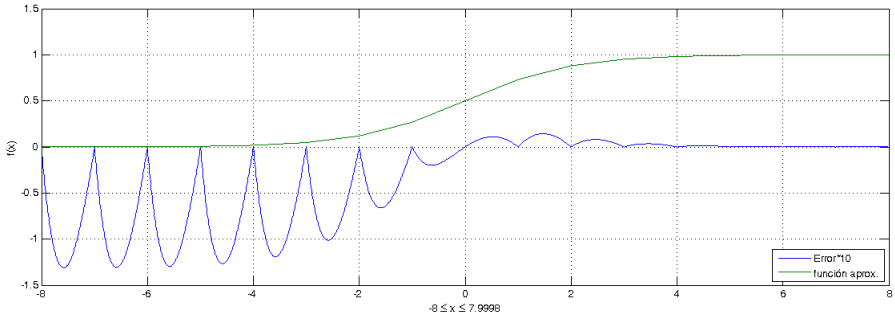

Figura 3.7 Aproximación de la función sigmoide con 2^4 segmentos

Se corre nuestro código de MATLAB® *aprox_sigmoide* para las condiciones descritas y se obtienen los resultados que se ilustran en la figura 3.7, donde 2^k son los intervalos en los que se divide la función a aproximar.

Para una mejor apreciación del error dentro de la gráfica, se ha aplicado una ganancia de 10 al error relativo.

Caso 2 tenemos:

Función a aproximar:

$$f(x) = \frac{1}{1+e^{-x}}\ , \qquad \begin{cases} -8 \leq x < 8 \\ k = 5 \end{cases}$$

Nuevamente se corre nuestro programa, pero ahora para $k = 5$, esperando obtener una mejora en la aproximación de nuestra función. El resultado se ilustran en la figura 3.8, donde 2^k son los subintervalos en los que se divide la función a aproximar.

De la figura 3.8 está claro que el error relativo disminuyó de forma importante en comparación con el caso 1. Evidentemente incrementar el valor de k vendrá en beneficio de nuestra función pero en detrimento de los recursos necesarios para implementarla en el FPGA.

La implementación hardware de nuestra función de activación se realizó bajo el esquema 3.9 :

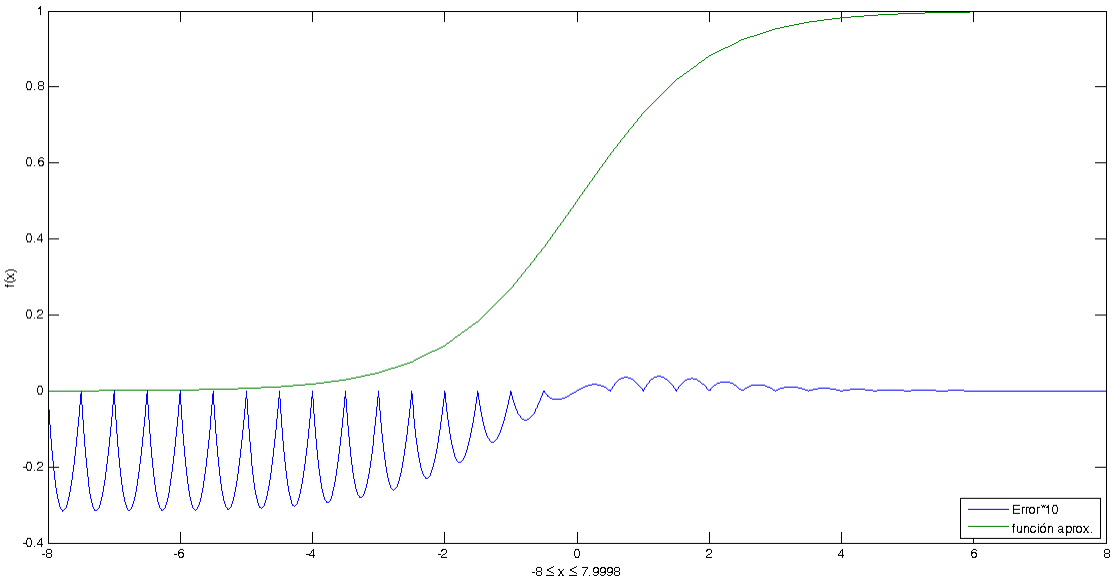

Figura 3.8 Aproximación de la función sigmoide con $2^{k=5}$ segmentos

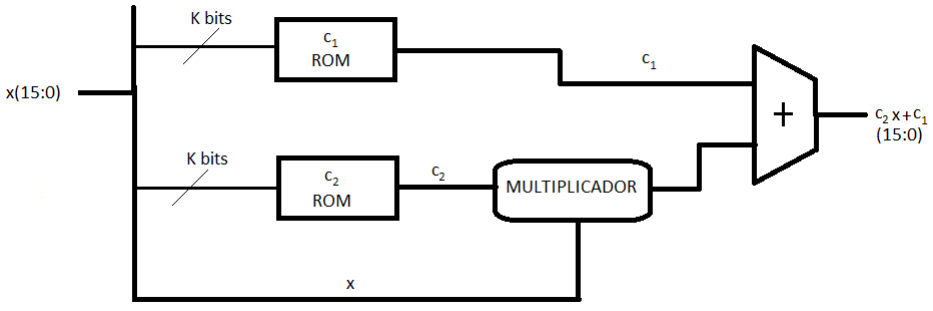

Figura 3.9 Esquemático de la organización hardware para función de activación sigmoide

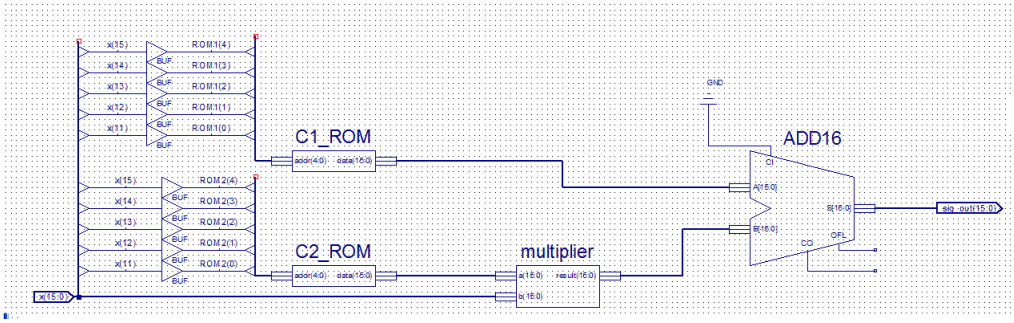

Figura 3.10 Diagrama eléctrico de la función de activación en el FPGA

Del diagrama anterior, se puede inferir que tanto c_1 como c_2 fueron almacenadas en memorias ROM, por ser uno de los métodos más sencillos de implementar, así como uno de los que menos recursos del FPGA consumen; tal y como se discutió en la sección anterior.

Finalmente, en la figura 3.10 se aprecia una captura de pantalla del diagrama del circuito realizado en la plataforma de desarrollo ISE.

3.6 Diseño de MAC

El diseño de la MAC es otro de los bloques fundamentales de una neurona y no menos importante que la función de activación, ya que es en este punto donde se lleva acabo el cómputo de la entrada sináptica total, que dicho en otras palabras, es una serie de multiplicaciones y adiciones que una vez concretadas, su producto neto es aplicado a la función de activación.

Hay ciertas variables a considerar para el diseño de la MAC, la primera de ellas es que tal y como se indicó el capítulo anterior; impera la necesidad de establecer una representación de los datos en *punto fijo*, más específicamente, en un formato de representación a 16 bits, donde 4 bits se utilizan para la parte entera y los 12 bits restantes para la representación de la parte fraccionaria. De igual forma, se deben tener presentes los corrimientos generados en la multiplicación, para realizar los ajustes necesarios en el resultado y seguir manteniendo un formato $Q_{4.12}$ es decir, 4 bits para la parte entera y el resto para la parte fraccionaria.

Sumado a lo anterior y no menos importante, requerimos de un *registro* igualmente de 16 bits para la acumulación progresiva en cada una de las iteraciones ya que hay que estar monitoreando esta operación aritmética para asegurarse que no se produzca *sobrecarga (overload)*.

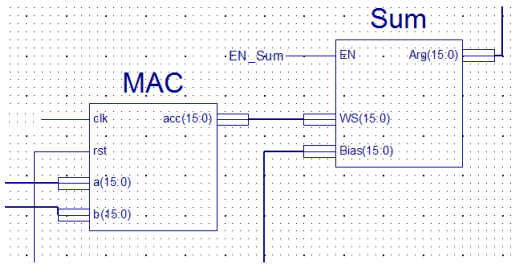

Figura 3.11 Simbolo generado a partir de su código VHDL para la MAc(izquierda) y una unidad sumadora (derecha)

La sobrecarga se produce cuando al realizar la suma de dos operandos, llámese A y B, ambos del mismo signo (ya sea positivo o negativo), se produce un resultado con signo contrario; por lo que si fuese el caso, debemos asegurarnos que el registro almacene la cifra máxima del formato de representación y evitar así, el desbordamiento en comento.

Considerado todo lo anterior, se elaboró el código en VHDL y a partir de dicho código, se generó el circuito que se puede apreciar en la figura 3.11. Una vez corroborada su completa funcionalidad por medio se simulaciones en ISE, se integró al *módulo principal* del diseño que dicho sea de paso, se elaboró bajo el perfil de diagrama electrónico.

En la figura 3.11 podemos obervar que la MAc conecta por su derecha con un bloque sumador, cuya función es la de realizar la adición de la suma neta del cómputo *entradas-pesos* con el *umbral* ó *bias* de la neurona. Por lo que a ambos bloques se les pudiese considerar una misma unidad, pero para fines prácticos se optó por colocarlos de forma independiente como un criterio más de diseño.

3.7 Gestor de Señales de Reloj DCM

La familia de FPGA Spartan 3E provee de bloques *Digital Clock Manager* o DCM por sus siglas en inglés, que permiten un control total sobre la señal de reloj, en términos de frecuencia de la señal, cambio de fase y control de la forma de la señal de reloj (*skew*) [54]. Para el funcionamiento de nuestra RNA (y de cualquier circuito síncrono) se requiere un estricto control sobre la señal de reloj del sistema, para que la propagación de las señales se realice lo más limpia posible (libre de ruido), en el momento preciso; para así, estar en condiciones de

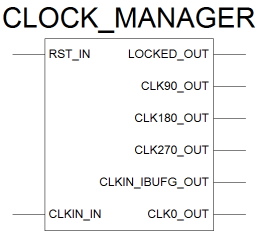

Figura 3.12 Bloque DCM que gestiona las señales de reloj dentro de la RNA

obtener un resultado libre de errores. Por esta razón se incluyó un bloque DCM cuyo objetivo será, el control de las señales de reloj de nuestra RNA.

En la figura 3.12, se puede apreciar el bloque inferido a partir de la utilería *IP CORE GENERATOR* de la plataforma de desarrollo. La descripción de los pines de entrada/salida del bloque es la siguiente:

- PINES DE ENTRADA
  - RST_IN: Entrada de restablecimiento del DCM
  - CLK_IN: Es la entrada de señal de reloj principal del sistema, sobre la cual se lleva el procesamiento. Para nuestro diseño, este pin va conectado al cuarzo de la tarjeta electrónica.
- PINES DE SALIDA
  - LOCKED_OUT: Señal de control que avisa un correcto ämarrede las señales de reloj. Es activo en 1́, y no es hasta que se activa esta bandera, que podemos hacer uso de la señal de reloj con plena certidumbre de que el DCM ha realizado los ajustes necesarios en la señal de reloj.
  - CLK90_OUT: Señal de reloj con 90^o de desfase respecto de CLK_IN
  - CLK180_OUT: Señal de reloj con 180^o de desfase respecto de CLK_IN
  - CLK270_OUT: Señal de reloj con 270^o de desfase respecto de CLK_IN
  - CLK_IBUFG_OUT: Comparte las mismas características que la señal de entrada CLK_IN, reforzada por un buffer

- CLK0_OUT: Es un bypass de la señal de entrada CLK_IN

3.8 Máquina de estados

La etapa de control de la red neuronal artificial es llevada a cabo por medio de una *máquina de estados* ó FSM por sus siglas en inglés, la cual funge como el principal centro de gestión detrás de todo el diseño, entre las principales de sus tareas están: el control sobre el direccionamiento de todas las memorias de la red (se hace notar que la dirección la generan contadores digitales insertados en la red pero es la máquina de estados quién dice cuándo contar y cuando restablecer), el control de la señal de *reset* de los bloques que así lo requieren, el multiplexado de señales, el control de lectura y escritura de la memoria de doble puerto (en ésta se almacena la salida de cada neurona) y finalmente el procesamiento por neurona y por capa en la RNA.

Cabe señalar que para el diseño de esta etapa se buscó obtener la mayor *portabilidad* posible, ya que como se mencionó secciones atrás, existe una cantidad inmensa de tipos y configuraciones de redes neuronales; por lo que para alcanzar cierta adaptabilidad en nuestro diseño de RNA, era necesario tomar esta característica como una de nuestras prioridades. Aunado a lo anterior, resulta imposible diseñar una RNA hardware que reciba una universalidad de RNA software, razón por la cual inherentemente hubo que limitar los criterios de diseño.

Entrando en materia, el circuito derivado del código VHDL escrito en ISE se muestra en la figura 3.13, de donde tenemos la siguiente descripción para cada uno de sus pines entrada-salida

- PINES DE ENTRADA
  - clk_lock: Cuando se pone en activo "1" indica que el bloque manejador de señales de reloj DCM ha sintonizado sus señales de reloj. Es el punto de partida para la FSM
  - clk: Es la entrada de señal de reloj
  - mrst: Restablece toda la RNA a su punto de arranque.

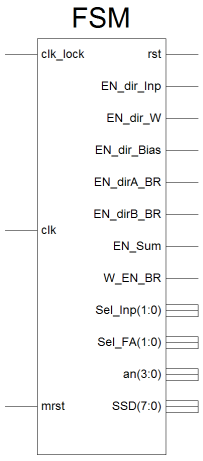

Figura 3.13 Circuito o símbolo generado a partir del código VHDL de la máquina de estados

- PINES DE SALIDA
  - rst: Señal de control utilizada para restablecer los contadores a "0"
  - EN_dir_Inp: Habilita o inhibe direccionamiento de la memoria ROM *Inputs_ROM*
  - EN_dir_W: Habilita o inhibe direccionamiento de la memoria ROM *Weigths_ROM*
  - EN_dir_Bias: Habilita o inhibe direccionamiento de la memoria ROM *Bias_ROM*
  - EN_Sum: Habilita o inhibe la adición del *bias* con la salida del bloque *MAC*
  - W_EN_BR: Bit de control de escritura de la memoria de doble puerto, activo en "1"
  - Sel_Inp: Señal de control del flujo de las señales de entrada de la RNA
    - Sel_Inp="00": Indica que se debe tomar como *vector de entradas* los valores almacenados en Inputs_ROM
    - Sel_Inp="01": Indica que se debe tomar como *vector de entradas* los valores almacenados en la Block_RAM
  - Sel_FA: Señal de selección para una de 4 posibles funciones de activación

Para la ejecución de la RNA de dos capas se codificaron los estados $state0$ $state1$ $state2$ y $state3$

- **state0** En este estado se configura la RNA a su punto de arranque, es decir, contadores a cero, punteros a cero, registros a cero y la RNA entra en un estado de *congelamiento* en espera de la señal de sincronía de las señales de reloj *clk_lock*

- **state1** En este estado se realiza el cálculo de la capa 1 de la RNA, neurona por neurona y los resultados se almacenan en la memoria de doble puerto *Block_RAM*.

- **state2** En este estado es donde se realiza el cómputo de la capa 2 o capa de salida, al igual que en $state1$ la salida de cada una de las neuronas se almacenan en la memoria de doble puerto *Block_RAM*.

- **state3** En este estado la RNA ha finalizado el cómputo y el FPGA entra en modo de visualización.

Como ya anteriormente se explicó, se buscó dar la mayor *portabilidad* al diseño, por lo cual, la máquina de estados escrita en VHDL se dotó de *Generics*, que es un tipo especial de variables que el usuario puede manipular sin la necesidad de tener que hacer cambios importantes dentro del código.

Lo anterior es cierto en la medida que el usuario respete la arquitectura de la red, a saber: Red perceptrón multicapa de propagación hacia adelante, con una capa de entrada y una capa de salida. Bajo este orden de ideas se enlistan los parámetros establecidos para ser modificados por el usuario final, dentro del código de la FSM:

- $n1$: Variable tipo *generic* que indica la cantidad de neuronas en la capa uno

- $n2$: Variable tipo *generic* que indica la cantidad de neuronas en la capa dos o capa de salida

- $W1$: Variable tipo *generic* que indica la cantidad de conexiones sinápticas en la capa uno de la RNA

- $W2$: Variable tipo *generic* que indica la cantidad de conexiones sinápticas en la capa dos de la RNA

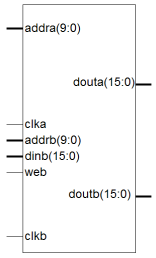

Figura 3.14 Memoria de doble puerto de la RNA

Si bien, es posible realizar modificaciones mínimas en la máquina de estados para ampliar las capacidades de la RNA (por ejemplo agregar un estado más para realizar el cálculo de una capa más), se decidió restringir el desarrollo de este trabajo a los criterios de diseño discutidos anteriormente.

3.9 Memoria de doble puerto Block RAM

Los FPGA cuentan con cantidades importantes de memoria *distribuida*, esto es, las *LUTs* que dicho sea de paso, son el elemento fundamental de los FPGA, se pueden agrupar con la finalidad de potenciar su capacidad de almacenamiento. Adicionalmente, pueden contar con bloques de memoria *dedicados*, es decir, bloques cuya única función es la de almacenar datos.

La familia Spartan-3E de Xilinx incorpora desde 4 y hasta 36 bloques dedicados de memoria, organizados en bloques de 18k bit *doble puerto* totalmente**configurables**. Para el caso que nos atañe, el FPGA de nuestra tarjeta de desarrollo cuenta con 12 bloques de este tipo.

Éstos bloques cuentan con una estructura de doble puerto, ambos idénticos, llámense puerto A y puerto B, que permiten acceso totalmente independiente y por ende, es posible realizar operaciones de lectura y escritura simultáneamente sobre cualquiera de los dos [54].

Éstas características se aprovecharon en el diseño de nuestra RNA, permitiéndonos que con sólo un bloque de memoria alcancemos operaciones de lectura-escritura simultáneos, es decir, mientras en el puerto A se están leyendo los datos del cálculo de la capa uno de la RNA, en el puerto B se están escribiendo los resultados el cómputo de la capa dos, optimizando así el desempeño y funcionalidad de la RNA.

Para las características de nuestra RNA, se infirió el bloque de memoria de la figura 3.14

La descripción de pines del bloque 3.14 es la siguiente:

- **PINES DE ENTRADA**
  - *addra(9:0)*: Direccionamiento de puerto A con profundidad de 2^{10} palabras de 16 bit.
  - *CLK_a*: Entrada de reloj del puerto A
  - *addra(9:0)*: Direccionamiento de puerto B con profundidad de $2^{1}0$ palabras de 16 bit
  - *Wr_EN_BR*: Bit de control de escritura de puerto B
  - *CLK_b*: Entrada de reloj del puerto B
- **PINES DE SALIDA**
  - *outa(15:0)*: Salida de puerto A a 16 bits
  - *outb*: Salida de puerto B a 16 bits

3.10 Propuesta de diseño de RNA para FPGA

Como solución al problema científico planteado en este trabajo, se ha diseñado un sistema capaz de extraer el conocimiento de una RNA previamente entrenada y probada en MATLAB®, para posteriormente descargar dicho conocimiento en un dispositivo electrónico conocido como FPGA, quién será el encargado de la ejecución de la RNA hardware . El sistema a su vez, cuenta con la capacidad de modificar el número de neuronas de cualquiera de sus dos capas y puede también, admitir una variedad de hasta cuatro funciones de activación diferentes; parámetros ajustables por el diseñador, lo que permite empatar la red del FPGA, a una variedad de RNAs diseñadas en MATLAB. Esto último es cierto, en la medida que no se rebasen los criterios y límites establecidos en nuestro diseño o los límites mismos de la tarjeta de desarrollo.

De tal forma que, en la figura 3.15 se muestra el diagrama electrónico del diseño final de nuestra RNA desarrollado en ISE Xilinx versión 9.1, e implementado en un FPGA de la familia

Spartan 3E. Si comparamos este diagrama, con nuestro diagrama de bloques 3.3 de la sección anterior, notaremos la adición de algunas compuertas OR usadas para complementar la acción de *reset* y de 5 contadores etiquetados *CB16CE*, cuya única función es la de direccionar las memorias de nuestro diseño.

Ahora bien, para poner a prueba nuestro diseño, era necesario contar con RNAs entrenadas y puestas a prueba en MATLAB® para de esta manera, estar en condiciones de ajustar los parámetros de nuestra RNA hardware *esqueleto*, con los parámetros de la RNA *software* que deseamos implementar.

Para cumplir dicho requerimiento de la forma más directa y simple posible, se optó por utilizar el *Neural Network toolbox* de MATLAB®, que es una utilería de este programa, diseñada única y exclusivamente para el desarrollo de RNAs; por lo que tenemos a la mano, una gran variedad de redes ya probadas, y de las cuales seleccionamos algunas de ellas, para implementarlas en nuestro diseño y así tener un marco de comparación. Cabe resaltar que ésta utilería de MATLAB utiliza algunas funciones de preprocesamiento de los datos, que se encuentran integradas a la red misma, de las cuales destaca la función *mapminmax*, función que se encarga de normalizar tanto el vector de entradas como el vector de salidas, a un rango [-1 1], lo cual es un factor importante a considerar, al momento de vaciar o extraer el conocimiento de estas RNAs al FPGA; o dicho en otras palabras, habrá que **desnormalizar** los resultados obtenidos por nuestro diseño, a fin de que estos tengan un significado físico y puedan ser objeto de comparación.

No menos importante es mencionar que este trabajo, ha sido desarrollado a nivel hardware, por lo que manejar cifras binarias, formatos de representación, simulaciones, uso de recursos, etc., es el lenguaje más apropiado para medir el *performance* de nuestra RNA. Por lo que, para una mejor comprensión de los resultados obtenidos, se buscó expresar el resultado final bajo un sistema de representación decimal (puesto que es el que habitualmente manejamos), no implicando que dichos resultados sean visibles de esta manera, en cualquiera de las etapas de nuestro sistema.

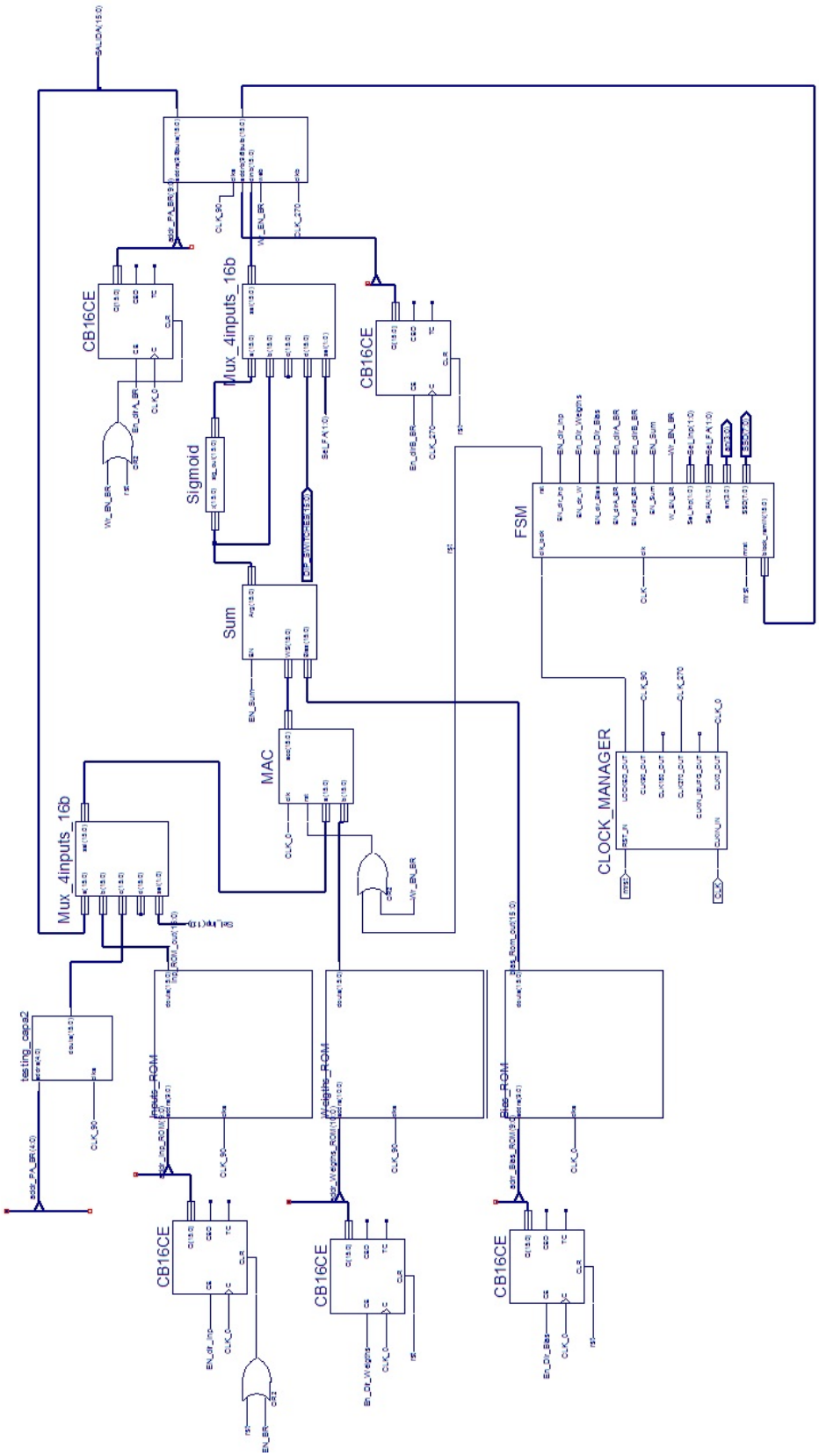

Figura 3.15 Diseño final de la Red Neuronal Artificial

Claro está, que solo nos resta poner a prueba nuestro diseño, para lo cual se analizarán tres casos diferentes; pero previo a ello, entiéndase que para cada uno de los casos estudiados, se realizaron los siguientes pasos (remítase a 3.4) para la consecución de cada uno de ellos:

- Cargar la RNA que se desea implementar, en el área de trabajo de MATLAB®
- Extraer el conocimiento de la RNA mediante el M-file "extractor", de donde se obtienen los archivos *Weigths_Rom y Bias_ROM*, mismos que se vinculan con las memorias ROM de la RNA.
- Generar el estímulo de entradas que requiere la RNA para posteriormente almacenarlo en *Inputs_ROM* (se usa cualesquiera de los datos de entrada proporcionados por mismo MATLAB)
- Cargar los archivos *Weigths_Rom, Bias_ROM, Inputs_ROM* con su cada cual, dentro de nuestra RNA.
- Ajustar la máquina de estados de la RNA hardware, de acuerdo con las características de la RNA de MATLAB que deseamos implementar.
- Sintetizar nuestra RNA dentro de ISE para que el compilador de este software genere un archivo ".bin".
- Vaciar el archivo *".bin"* a nuestra tarjeta de Desarrollo Basys2

Capítulo 4

Resultados y Discusión

4.1 Resultados

4.1.1 Caso 1

Dentro de las aplicaciones del *toolbox* de RNAs de MATLAB®, encontramos una RNA diseñada para el reconocimiento de caracteres, la cual puede ser cargada fácilmente en el área de trabajo de MATLAB® ingresando el comando **appcr1**. Esta RNA fue diseñada y entrenada para reconocer las 26 letras del alfabeto (alfabeto inglés); teóricamente, habría un sistema que digitaliza letras del alfabeto dentro de algún proceso, produciendo una matriz booleana de salida de 5x7 por cada caracter leído. Por ejemplo, la letra "A" se representa gráficamente como en la figura 4.1

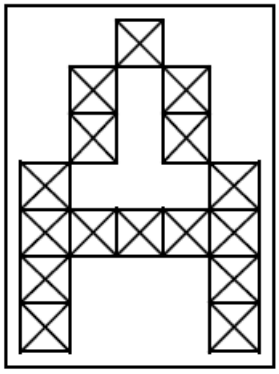

Figura 4.1 Letra "A" en su representación gráfica de una matriz de 5x7

No obstante, el sistema no es perfecto y puede presentar lecturas con ruido tal como en la figura 4.2

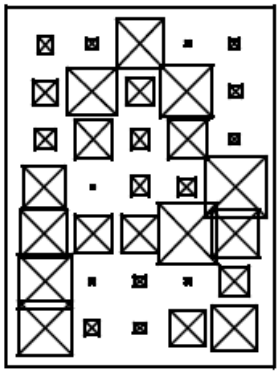

Figura 4.2 Letra "A" en su representación gráfica con ruido

Esta red es entrenada para obtener una clasificación perfecta de entradas sin ruido, y para obtener una presición razonablemente buena en el caso de entradas con ruido.

Si urgamos dentro del código de esta aplicación, fácilmente encontramos que la topología, así como demás parámetros de configuración para esta RNA son como sigue: Red perceptrón multicapa, una capa oculta, 25 neuronas en capa oculta con función de activación sigmoide y 26 neuronas en la capa de salida con función de activación lineal. Idealmente esta RNA debe producir un "1" en la neurona de salida que representa la posición de la letra dentro del alfabeto, por ejemplo, para la letra "A" se debe producir un "1" en la neurona de salida número "1" y "-1" en las 25 neuronas restantes. Pues bien, ejecutando esta RNA en MATLAB® y posteriormente hacemos lo propio adaptando los parámetros de nuestra RNA hardware, en concordancia con los de la red propiamente dicha y una vez hemos vaciado el conocimiento de dicha red a nuestra tarjeta, estamos finalmente en condiciones de poner a prueba nuestra RNA hardware.

Debido a la naturaleza propia del problema, el formato más apto para el manejo de los datos de entrada-salida, es el hexadecimal, por lo que la visualización y sobre todo la comparación entre la RNA de MATLAB® y la RNA del FPGA no resulta tan directa. Es por ello que se optó por mostrar los resultados en la forma de la tabla 4.3. En donde para una mayor claridad, se despliegan los resultados obtenidos para ambas redes en formatos, tanto decimal como hexadecimal.

LETRA	$MATLAB_{10}$	$MATLAB_{16}$	ISE_{16}	ISE_{10}	Error Relativo
A	0.9802	07d7	07c6	0.9717	0.87%
B	-1.0091	f7ed	f7e2	-1.0146	0.55%
C	-1.0132	f7e5	f7d8	-1.0195	0.62%
D	-1.0005	f7ff	f7f4	-1.0059	0.54%
E	-1.0083	f7ef	f7e2	-1.0146	0.62%
F	-1.0073	f7f1	f7e5	-1.0132	0.59%
G	-0.9984	f803	f7f7	-1.0044	0.60%
H	-1.0006	f7ff	f7f4	-1.0059	0.53%
I	-0.9836	f822	f819	-0.9878	0.43%
J	-1.0062	f7f3	f7e6	-1.0127	0.65%
K	-0.9869	f81b	f815	-0.9897	0.28%
L	-0.9839	f821	f81b	-0.9868	0.29%
M	-0.9981	f804	f7f9	-1.0034	0.53%
N	-1.0063	f7f3	f7e8	-1.0117	0.54%
O	-0.9969	f806	f7fa	-1.0029	0.60%
P	-0.9883	f818	f810	-0.9922	0.39%
Q	-1.0011	f7fe	f7f2	-1.0068	0.57%
R	-1.0003	f7ff	f7f5	-1.0054	0.51%
S	-0.9891	f816	f80f	-0.9927	0.36%
T	-0.9991	f802	f7f8	-1.0039	0.48%
U	-0.9952	f80a	f7ff	-1.0005	0.53%
V	-1.0011	f7fe	f7f2	-1.0068	0.57%
W	-0.9925	f80f	f805	-0.9976	0.51%
X	-1.0002	f800	f7f5	-1.0054	0.52%
Y	-1.0041	f7f8	f7eb	-1.0103	0.62%
Z	-1.0006	f7ff	f7f5	-1.0054	0.48%

Figura 4.3 Comparativa de los resultados obtenidos para la RNA ejecutada en MATLAB y para la RNA en el FPGA

En dicha tabla se aprecia también una columna que cuantifica el error de la red, dicho error es expresado en porcentaje y deja ver claramente el error que presenta la RNA, respecto de la RNA de MATLAB® (tomada desde luego como valor verdadero).

Para esta primera valoración, cuyo valor de entrada para ambas redes es la letra "A", representada por un vector de entrada de 5x7 con ruido de media 0 y desviación estándar de 0.2, se puede dilucidar que la RNA implementada en el FPGA no solo reconoce el estímulo que se le requiere identifique, sino que se ajusta bastante bien a su similar en MATLAB®.

Continuando con las pruebas, al igual que en el ejemplo anterior, se le presenta ahora a ambas RNAs la letra "G", y los resultados se ilustran en la tabla 4.4

Una vez más, vemos que aunque nuestra RNA es capaz de generalizar, e indicarnos que el estímulo presentado corresponde con la letra "G", no obstante, podemos apreciar de la columna del error que esta vez el ajuste no fue tan bueno como para el caso anterior y existe un incremento del error en cada una de las neuronas de salida, respecto de la red de MATLAB®.

Bajo este mismo orden de ideas, se realiza una tercera y última prueba, esta vez se estimulan las RNAs con la letra "R", escogida una vez más de forma totalmente arbitraria. Y nuevamente, los resultados obtenidos de ambas RNAs se despliegan en la tabla 4.5

4.1.2 Caso 2

Para el segundo caso, una vez más indagamos dentro de las aplicaciones del *toolbox* de RNAs de MATLAB®, y encontramos una RNA diseñada para la clasificación de vinos de tres cavas diferentes, de acuerdo a sus características químicas. Esta RNA cuenta con un total de 178 muestras de vino obtenidas de 3 bodegas diferentes; a dichas muestras se les realizó un estudio químico para extraer los siguientes atributos:

- 1. Alcohol
- 2. Ácido Málico los archivos
- 3. Ceniza
- 4. Alcalinidad de la ceniza

LETRA	$MATLAB_{10}$	$MATLAB_{16}$	ISE_{16}	ISE_{10}	Error Relativo
A	-0.9962	f808	f617	-1.2388	24.35%
B	-1.0027	f7fb	f60a	-1.2451	24.17%
C	-1.0154	f7e0	f705	-1.1226	10.56%
D	-0.9944	f80b	fb3c	-0.5957	40.09%
E	-1.0155	f7e0	f967	-0.8247	18.79%
F	-1.0122	f7e7	f6df	-1.1411	12.73%
G	0.9933	07f2	0674	0.8066	18.80%
H	-0.9956	f809	f6fb	-1.1274	13.24%
I	-0.995	f80a	f7d0	-1.0234	2.85%
J	-1.0034	f7f9	f596	-1.3018	29.74%
K	-0.9983	f803	f9c5	-0.7788	21.99%
L	-0.9843	f820	f79a	-1.0498	6.65%
M	-0.9994	f801	f7ad	-1.0405	4.11%
N	-1.0031	f7fa	f59a	-1.2998	29.58%
O	-0.9996	f801	f956	-0.833	16.67%
P	-0.9961	f808	fb98	-0.5508	44.70%
Q	-1.0045	f7f7	f867	-0.9497	5.46%
R	-0.9828	f823	f452	-1.46	48.56%
S	-0.9832	f822	f711	-1.1167	13.58%
T	-1.0072	f7f1	fa40	-0.7188	28.63%
U	-1.0083	f7ef	fa2a	-0.7295	27.65%
V	-1	f800	f7ce	-1.0244	2.44%
W	-0.9933	f80e	f7d8	-1.0195	2.64%
X	-1.0084	f7ef	f9dc	-0.7676	23.88%
Y	-0.9949	f80a	f7e3	-1.0142	1.94%
Z	-0.9998	f800	f842	-0.9678	3.20%

Figura 4.4 Resultado de aplicar la letra "G" con ruido a las RNAs tanto en su versión en MATLAB, como su versión en FPGA

LETRA	$MATLAB_{10}$	$MATLAB_{16}$	ISE_{16}	ISE_{10}	Error Relativo
A	-0.9949	f80a	f804	-0.998	0.31%
B	-0.9994	f801	f7e9	-1.0112	1.18%
C	-0.9901	f814	f823	-0.9829	0.73%
D	-1.0109	f7ea	f7fe	-1.001	0.98%
E	-0.993	f80e	f7de	-1.0166	2.38%
F	-1.0083	f7ef	f806	-0.9971	1.11%
G	-1.0129	f7e6	f7c2	-1.0303	1.72%
H	-0.9845	f820	f7f7	-1.0044	2.02%
I	-1.004	f7f8	f80a	-0.9951	0.89%
J	-1.0159	f7df	f7db	-1.0181	0.22%
K	-0.9933	f80e	f80e	-0.9932	0.01%
L	-1.0329	f7bd	f7ce	-1.0244	0.82%
M	-0.9985	f803	f7ef	-1.0083	0.98%
N	-0.9861	f81c	f81f	-0.9849	0.12%
O	-1.008	f7f0	f7fb	-1.0024	0.56%
P	-0.9893	f816	f84a	-0.9639	2.57%
Q	-0.9918	f811	f7de	-1.0166	2.50%
R	0.9735	07ca	0789	0.9419	3.25%
S	-0.9856	f81e	f802	-0.999	1.36%
T	-0.9792	f82b	f80f	-0.9927	1.38%
U	-1.0237	f7cf	f7de	-1.0166	0.69%
V	-0.9993	f801	f7fd	-1.0015	0.22%
W	-0.9881	f818	f815	-0.9897	0.16%
X	-0.9898	f815	f803	-0.9985	0.88%
Y	-1.0161	f7df	f7dd	-1.0171	0.10%
Z	-0.9815	f826	f80d	-0.9937	1.24%

Figura 4.5 Resultado de aplicar la letra "R" con ruido a las RNAs tanto en su versión en MATLAB, como su versión en FPGA

- 5. Magnesio
- 6. Fenoles
- 7. Flavanoides
- 8. Fenoles no Flavanoides
- 9. Proantocianinas
- 10. Intensidad del Color
- 11. Matiz
- 12. OD280/OD315 de vinos diluidos
- 13. Prolina

De esta manera, son las propiedades químicas de la muestra y la bodega a la cual pertenece, los parámetros de entrada-salida utilizados para el entrenamiento de la RNA.

Al igual que en el caso anterior, se ejecuta esta RNA tanto en su versión de MATLAB®, como en nuestro diseño hardware para algunas de las muestras que proporciona la aplicación misma y se obtienen los resultados que se ilustran a partir de la tabla 4.6 y hasta la tabla 4.10.

Muestra de Vino					
Cava	**$MATLAB_{10}$**	**$MATLAB_{16}$**	**ISE_{16}**	**ISE_{10}**	**Error Relativo**
1	0.9998	3ffd	21fe	0.5311	46.88%
2	0.0127	00d0	00d6	0.0131	3.15%
3	0	0000	0006	0.0004	0.04%

Figura 4.6 Muestra 1 aplicada tanto a la RNA de MATLAB® como a la RNA hardware

Muestra de Vino: 38					
Cava	**$MATLAB_{10}$**	**$MATLAB_{16}$**	**ISE_{16}**	**ISE_{10}**	**Error Relativo**
1	0.9921	3f7f	21fe	0.5311	46.47%
2	0.0412	02a2	02b3	0.0422	2.43%
3	0000	0000	0006	0.0004	0.04%

Figura 4.7 Muestra 38 aplicada tanto a la RNA de MATLAB® como a la RNA hardware

Muestra de Vino: 65					
Cava	**$MATLAB_{10}$**	**$MATLAB_{16}$**	**ISE_{16}**	**ISE_{10}**	**Error Relativo**
1	0	0000	0006	0.0004	0.04%
2	0.9784	3e9f	3e92	0.9777	0.07%
3	0.0000	0000	0006	0.0004	0.04%

Figura 4.8 Muestra 65 aplicada tanto a la RNA de MATLAB® como a la RNA hardware

Muestra de Vino: 93					
Cava	**$MATLAB_{10}$**	**$MATLAB_{16}$**	**ISE_{16}**	**ISE_{10}**	**Error Relativo**
1	0.0000	0000	0006	0.0004	0.04%
2	0.8984	3980	3951	0.8956	0.31%
3	0.0000	0000	0006	0.0004	0.04%

Figura 4.9 Muestra 93 aplicada tanto a la RNA de MATLAB® como a la RNA hardware

Muestra de Vino: 140					
Cava	**$MATLAB_{10}$**	**$MATLAB_{16}$**	**ISE_{16}**	**ISE_{10}**	**Error Relativo**
1	0	0000	0006	0.0004	0.04%
2	0.1276	082a	085c	0.1306	2.35%
3	0.9847	3f05	3a87	0.9145	7.13%

Figura 4.10 Muestra 140 aplicada tanto a la RNA de MATLAB® como a la RNA hardware

4.1.3 Caso 3

Como tercer y último caso se utilizó la RNA *crab classification* de MATLAB®, red diseñada para determinar el sexo de los cangrejos de acuerdo a seis dimensiones físicas de esta especie. Esta aplicación cuenta con 200 muestras, para cada una de las cuales se determinó si el cangrejo es una hembra o un macho.

Siguiendo con la misma filosofía de los casos anteriores, se contrasta el diseño de RNA en FPGA, contra la misma RNA pero ejecutada en MATLAB®. Se realiza una comparativa de ambas redes para las muestras 1, 10, 20 y 30 que proporciona la misma aplicación y que se ilustran a continuación.

Muestra x=1					
Sexo	**$MATLAB_{10}$**	**$MATLAB_{16}$**	**ISE_{16}**	**ISE_{10}**	**Error Relativo**
HEMBRA	0	0000	0000	0	0.00%
MACHO	1	4000	3ffb	0.9997	0.03%

Figura 4.11 Muestra 1 aplicada tanto a la RNA de MATLAB® como a la RNA del FPGA

Muestra x=10					
Sexo	**$MATLAB_{10}$**	**$MATLAB_{16}$**	**ISE_{16}**	**ISE_{10}**	**Error Relativo**
HEMBRA	0.9999	3ffe	3fdd	0.9979	0.20%
MACHO	0	0000	000d	0.0008	0.08%

Figura 4.12 Muestra 10 aplicada tanto a la RNA de MATLAB® como a la RNA del FPGA

Muestra x=20					
Sexo	**$MATLAB_{10}$**	**$MATLAB_{16}$**	**ISE_{16}**	**ISE_{10}**	**Error Relativo**
HEMBRA	0.0026	002a	0005	0.0003	88.46%
MACHO	0.9438	3c67	3fc4	0.9963	5.56%

Figura 4.13 Muestra 20 aplicada tanto a la RNA de MATLAB® como a la RNA del FPGA

Muestra x=30					
Sexo	**$MATLAB_{10}$**	**$MATLAB_{16}$**	**ISE_{16}**	**ISE_{10}**	**Error Relativo**
HEMBRA	1.0000	4000	3ffe	0.9999	0.01%
MACHO	0	0000	0000	0	0.00%

Figura 4.14 Muestra 30 aplicada tanto a la RNA de MATLAB® como a la RNA del FPGA

4.2 Discusión

Se ha evaluado el desempeño de la RNA implementada en un FPGA Spartan 3E contrastándolo con una RNA equivalente en MATLAB, de tal forma que para cada uno de los tres casos, hubo que ejecutar en primera instancia la RNA en MATLAB y obtener una salida para un determinado estímulo o entrada. Esta salida en representación decimal, afecta a la segunda columna de las tablas presentadas, etiquetada como *MATLAB10*. Posteriormente, esta misma columna es convertida a su equivalente en representación hexadecimal, obteniendo así la tercera columna *MATLAB16*. La razón de convertir las salidas de la RNA a su valor en hexadecimal, es con el objeto de tener el mismo formato de representación que se obtiene de la ejecución de la RNA en el FPGA. Posteriormente, se extraen los pesos sinápticos y bias de cada una de las neuronas de la RNA de MATLAB y se vacían en la RNA del FPGA; para ello, se optimizó el proceso mediante un *M-file* o programa en MATLAB llamado *extractor*. Acto seguido se ejecuta la RNA del FPGA para la misma entrada que su contraparte y se obtienen los valores de la cuarta columna *ISE16* en formato hexadecimal. Si bien, la representación

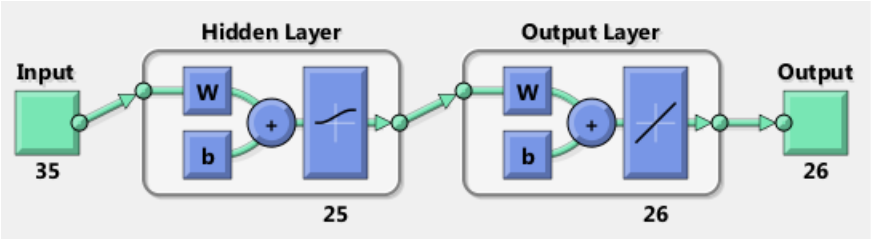

Figura 4.15 Topología de la RNA para reconocimiento de caracteres de caso 1

en hexadecimal es el formato por excelencia para trabajar a nivel máquina, no resulta sencillo identificar la diferencia entre ambas RNAs por lo que se desarrolló un segundo M-file *conv_dec* que convierte los valores registrados en la cuarta columna *ISE16* a su valor en decimal, y así tener cifras del formato al que estamos habituados manejar. Esta conversión se pone de manifiesto en la quinta columna *ISE10*. Por último se presenta una sexta columna con el error relativo de la RNA del FPGA, para lo cual se consideraron las salidas de la RNA de MATLAB como ideales.

Para el caso 1, nuestra RNA se adaptó y configuró fácilmente por medio de las herramientas ya discutidas con anterioridad, de tal modo de empatarse con la RNA de MATLAB que pretendemos emular y que guarda la forma de la figura 4.15.

De los resultados obtenidos en 4.3, 4.4 y 4.5 podemos decir que la RNA es capaz de generalizar y decirnos que las entradas de la red fueron las letras A, G y R respectivamente (recordar que idealmente la red debe presentar el valor de un "1" en la neurona de salida que corresponde con la posición de la letra dentro del alfabeto), mostrando un muy buen ajuste para las letras A y R. En cuanto a la letra G, para este estímulo tal y como se aprecia en 4.4, hubo un notable incremento del error para algunas de las salidas de la red . Inicialmente se creyó que este error podía deberse a un problema de sincronización en la extracción de los datos de las memorias ROM ya fuese de la entrada, pesos o bias; por que si bien era de esperar un error por truncamiento (al disponer un formato de representación fija con 4 bits para la parte entera y 12 bits para la parte fraccionaria) o un error inevitable debido a la aproximación de la función de activación, lo observado para este caracter en particular difiere de los buenos resultados observados para los estímulos "A" y "R". Por lo que se desarrolló en MATLAB® el código *depuring*

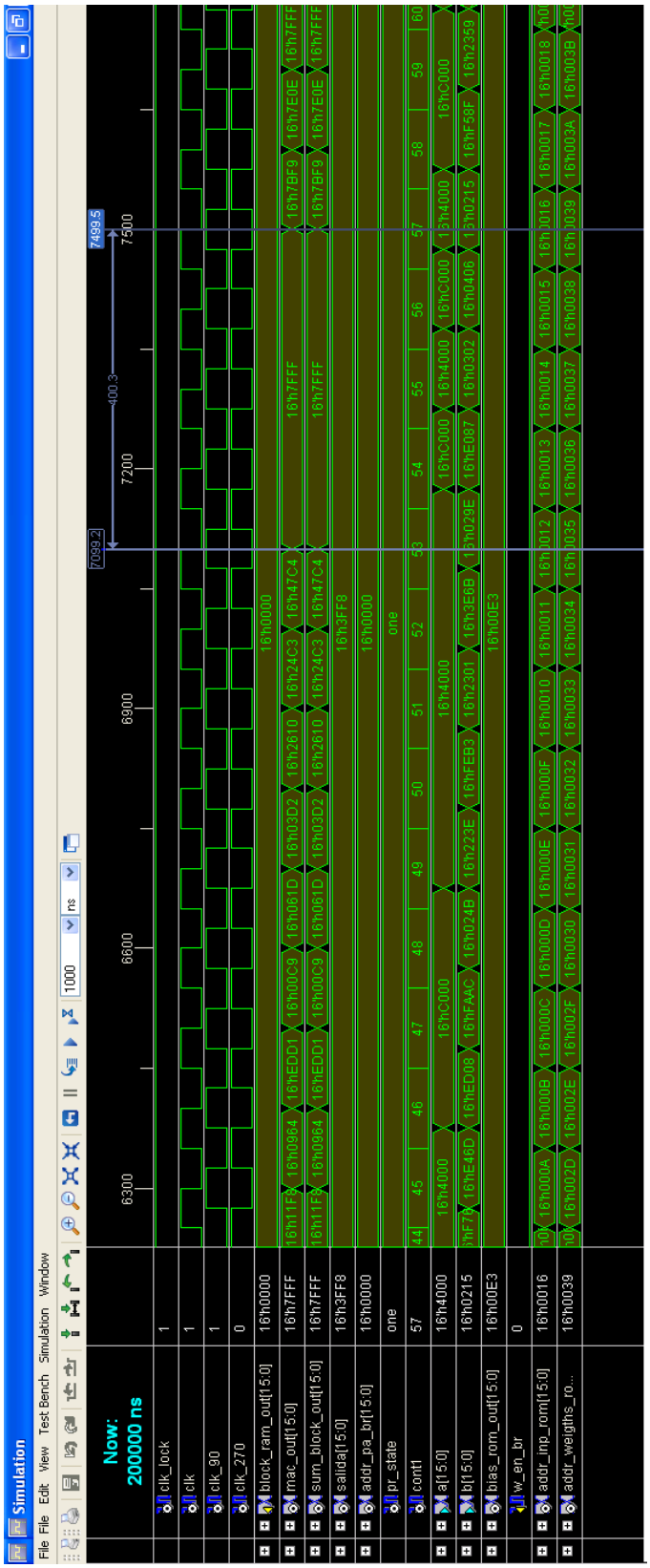

Figura 4.16 Simulación de la RNA en la que se aprecia la saturación de la unidad MAC

Device Utilization Summary			
Logic Utilization	**Used**	**Available**	**Utilization**
Total Number Slice Registers	137	4,896	2%
Number used as Flip Flops	109		
Number used as Latches	28		
Number of 4 input LUTs	412	4,896	8%
Logic Distribution			
Number of occupied Slices	270	2,448	11%
Number of Slices containing only related logic	270	270	100%
Number of Slices containing unrelated logic	0	270	0%
Total Number of 4 input LUTs	492	4,896	10%
Number used as logic	412		
Number used as a route-thru	80		
Number of bonded IOBs	24	92	26%
IOB Latches	4		
Number of Block RAMs	11	12	91%
Number of GCLKs	5	24	20%
Number of DCMs	1	4	25%
Number of MULT18X18SIOs	2	12	16%
Number of RPM macros	42		
Total equivalent gate count for design	732,597		
Additional JTAG gate count for IOBs	1,152		

Figura 4.17 Uso de recursos del FPGA para la implementación de la RNA del caso 1

con la finalidad de comparar paso a paso la ejecución de la RNA en MATLAB® y la ejecución de la RNA en el simulador de ISE (esto es, para cada pulso de la señal de reloj CLK).

Tras un exhaustivo análisis de la RNA del FPGA a nivel máquina, se encontró una limitante no prevista en la etapa de diseño de este trabajo, específicamente dentro del circuito *MAC*. Es en esta sección de la RNA, en donde se desarrolla la suma de productos que posteriormente es aplicada a la función de activación. Pues bien, tal y como se aprecia en la figura 4.16, que es una captura de pantalla de la simulación para este caso en específico, en la zona marcada por dos líneas verticales, se aprecia una **saturación** del circuito (valor hexadecimal *7fff*), o dicho en otras palabras, a la MAC se le esta pidiendo almacenar un valor más allá de los límites que se impusieron en el diseño; si bien estas saturaciones se producen esporádicamente en la ejecución de la red, si terminan por influir en el resultado final. Razón por la cual los resultados obtenidos pueden diferir de manera importante de los resultados esperados.

Por otro lado, en cuanto al consumo de recursos del FPGA para la implementación de esta RNA, el software de desarrollo ISE nos arroja una tabla como la que se ilustra en la figura 4.17, una vez que se compila e implementa el código en el FPGA. En esta tabla se exhibe la cantidad de recursos utilizados por nuestro diseño.

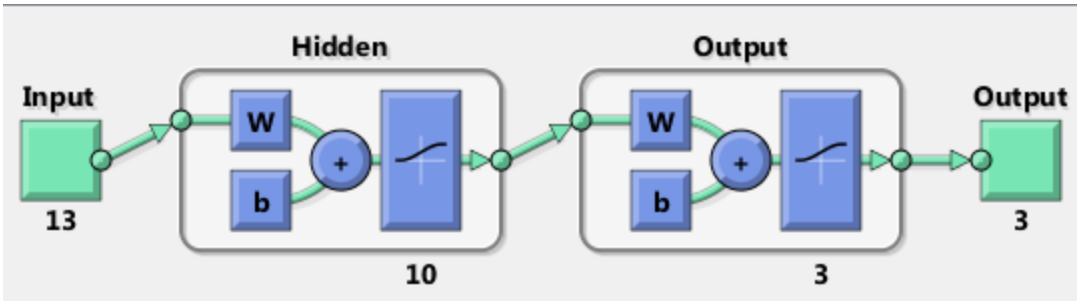

Figura 4.18 Topología de la RNA para la clasificación de vinos

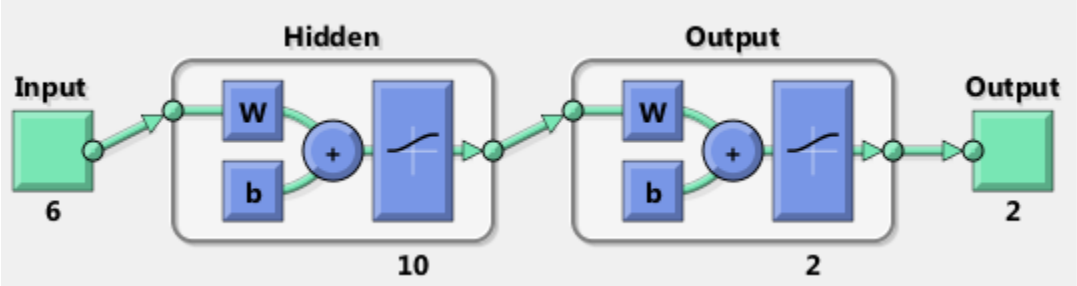

Figura 4.19 Topología de la RNA para la determinación de sexo en cangrejos

Para el caso 2, una vez más se configuró nuestro diseño de acuerdo con 4.18, obteniendo una buena generalización de nuestra red, tal y como se puede dilucidar de 4.6, 4.7, 4.8, 4.9 y 4.10; no obstante, se condujo un análisis tal y como el planteado líneas atrás, encontrándose también algunas saturaciones en MAC, que provocan algunas desviaciones en el resultado final.

Al igual que el caso anterior, en cuanto al consumo de recursos del FPGA para esta RNA, en la figura 4.20 se exhiben los recursos absorbidos esta vez por la RNA del caso 2. Salta a la vista que los recursos necesarios para implementar esta RNA fue menor que el caso 1, resultado que no es de sorprender ya que se trata de una RNA con menos neuronas.

Para el caso 3, una vez más se hicieron los ajustes correspondientes para ejecutar una RNA como la que se ilustra en la figura 4.19. En cuanto al desempeño de la RNA se refiere, ésta fue capaz de determinar el sexo de los cangrejos de acuerdo con lo esperado; exhibiendo una muy buena aproximación en comparación con la red de MATLAB.

Finalmente, en 4.21 se muestra el resumen de los recursos requeridos del FPGA para la implementación de esta RNA.

Device Utilization Summary			
Logic Utilization	**Used**	**Available**	**Utilization**
Total Number Slice Registers	135	4,896	2%
Number used as Flip Flops	109		
Number used as Latches	26		
Number of 4 input LUTs	405	4,896	8%
Logic Distribution			
Number of occupied Slices	265	2,448	10%
Number of Slices containing only related logic	265	265	100%
Number of Slices containing unrelated logic	0	265	0%
Total Number of 4 input LUTs	481	4,896	9%
Number used as logic	405		
Number used as a route-thru	76		
Number of bonded IOBs	24	92	26%
IOB Latches	4		
Number of Block RAMs	11	12	91%
Number of GCLKs	4	24	16%
Number of DCMs	1	4	25%
Number of MULT18X18SIOs	2	12	16%
Number of RPM macros	42		
Total equivalent gate count for design	732,524		
Additional JTAG gate count for IOBs	1,152		

Figura 4.20 Uso de recursos del FPGA para la implementación de la RNA del caso 2

Device Utilization Summary			
Logic Utilization	**Used**	**Available**	**Utilization**
Total Number Slice Registers	132	4,896	2%
Number used as Flip Flops	109		
Number used as Latches	23		
Number of 4 input LUTs	392	4,896	8%
Logic Distribution			
Number of occupied Slices	258	2,448	10%
Number of Slices containing only related logic	258	258	100%
Number of Slices containing unrelated logic	0	258	0%
Total Number of 4 input LUTs	468	4,896	9%
Number used as logic	392		
Number used as a route-thru	76		
Number of bonded IOBs	24	92	26%
IOB Latches	4		
Number of Block RAMs	11	12	91%
Number of GCLKs	4	24	16%
Number of DCMs	1	4	25%
Number of MULT18X18SIOs	2	12	16%
Number of RPM macros	42		
Total equivalent gate count for design	732,389		
Additional JTAG gate count for IOBs	1,152		

Figura 4.21 Uso de recursos del FPGA para la implementación de la RNA del caso 3

De los tres casos discutidos, el primero de ellos es el que sin lugar a dudas requiere mayor procesamiento por contener un número significativo de neuronas, además de plantear un hipotético procesamiento en tiempo real, lo cual podría representar todo un reto para la arquitectura de nuestro diseño. Saber si es posible o no alcanzar tal velocidad de procesamiento dependería de varios factores ajenos a la red que ya van más allá de los alcances de este trabajo, no obstante, por poner algunas cifras sobre la mesa, del caso referido tenemos que son requeridos 1525 ciclos de reloj para el procesamiento de todas y cada una de las neuronas de la red (capa1 = 35*25 + capa2 = 25*26) más unos cuantos ciclos de reloj que son requeridos para la sincronización de señales. Por lo tanto, para nuestra tarjeta de desarrollo con un reloj a 50 MHz, son requeridos poco más de $30us$ para el procesamiento de la RNA, lo cual nos hace pensar que cuidando ciertos detalles en el diseño, se podrían realizar tareas a alta velocidad con esta RNA hardware.

Conclusiones

1. Se elaboró el M-file *extractor* cuya función es la de extraer los pesos sinápticos y bias de la RNA de MATLAB® (que es donde reside el conocimiento de la red) y de generar los archivos *WEIGTHS.COE* y *BIAS.COE*, archivos que son cargados en las respectivas memorias ROM del FPGA.

2. Se diseñó una arquitectura hardware de RNA que trabaja en un formato de representación de punto fijo y con una longitud de 16 bits, de donde los 4 bits más significativos se utilizan para representar la parta entera y los 12 bits restantes para el manejo de la parte fraccionaria. Para facilitar la conversión de los datos del formato decimal a su versión en punto fijo, se creó el M-file *conv_hex*.

3. Se implementó una red neuronal artificial dentro de un FPGA con la particularidad de que funciona con una sola neurona y con un procesamiento cíclico, es decir, el cómputo de toda la red neuronal se realiza dentro de una unidad fundamental (neurona hardware), donde la información de toda la red fluye de forma controlada; la salida de esta neurona hardware es alimentada a una memoria RAM de doble puerto, en donde se almacena para su posterior procesamiento en capas subsecuentes. Además, en cierta medida se puede decir que el diseño es adaptable y reconfigurable ya que puede fungir como molde receptor de una variedad de RNAs creadas en MATLAB.

4. Se implementó la función de activación sigmoide para nuestro prototipo; se optó por esta función debido a que es una de las más exitosas en la resolución de problemas no lineales, no obstante, el diseño de nuestra RNA es capaz de manejar hasta 4 diferentes funciones de activación. Para la aproximación de la función de activación sigmoide se utilizaron polinomios de grado uno. Para este propósito se creó el M-file *aprox_sigmoide*.

5. El ahorro de recursos en hardware vino a costa de perder en exactitud, ya que hubo que truncar los resultados para que se pudiesen representar en únicamente 16 bits. No obstante, tal y como se hizo evidente en la fase de pruebas de este trabajo, nuestro diseño fue capaz de generalizar para las RNAs de los tres casos estudiados, acertando a las pruebas que se les impusieron y mostrando un error relativamente pequeño respecto de su contraparte en MATLAB. Además, gracias a la generación de los M-file citados anteriormente, son necesarios unos cuantos pasos para extraer la información contenida en la RNA de MATLAB y vaciarla dentro de la red del FPGA.

6. De las RNAs analizadas, la red del caso 1 es la más grande de las tres, conteniendo 25 neuronas en la capa oculta y 26 neuronas en la capa de salida, y no esta ni cerca de agotar la lógica disponible en el FPGA, lo cual habla de la capacidad de este diseño para albergar RNAs de mayores proporciones.

7. Era de esperar una diferencia entre los resultados de una RNA ejecutada en MATLAB y la RNA ejecutada en un FPGA, debido a dos principales factores: el error por truncamiento y el error implícito en la aproximación de la función de activación. Recordar además que el flujo de datos se realiza de forma secuencial, por lo que existe una propagación del error entre capa y capa.

8. Para los casos discutidos nuestro diseño demostró tanto adaptabilidad como reconfigurabilidad, siendo su punto débil la exactitud, ya que con los criterios de diseño establecidos, el FPGA reproduce la RNA con cierto grado de error. No obstante, mediante los programas desarrollados en MATLAB®, son necesarios unos cuantos pasos para llevar la red neuronal del software al hardware.

9. Es posible mejorar el desempeño de nuestro diseño de forma relativamente directa, esto es, incrementar la cantidad de bits tanto para la representación de la parte entera como la parte fraccionaria de los datos (para disminuir el error por truncamiento), e incrementar el número de segmentos utilizados para la aproximación de la función de activación.

10. Se ha alcanzado el objetivo principal de este trabajo pues de tres casos analizados de RNAs diseñadas en MATLAB®, las tres fueron depositadas con éxito dentro del FPGA, donde se ejecutaron y mostraron tener resultados muy similares a su versión original en software. Por lo que, en este punto podemos concluir que una vez que el investigador ha diseñado una RNA para la tarea de la reconstrucción de espectros (o alguna otra), esta RNA puede ser vaciada en un FPGA para su uso en campo, prescindiendo así de la necesidad de una computadora y dotando a la RNA de gran portabilidad.

Apéndice A: Código Matlab

En el disco compacto adjunto se encontrará la siguiente relación de códigos de MATLAB® elaborados para la realización de este trabajo; de igual manera se agrega el proyecto ISE que implementa la RNA en el FPGA.

1. **aprox_sigmoide**
2. **conv_bin**
3. **conv_dec**
4. **conv_hex**
5. **depuring**
6. **extractor**
7. **hexTOdec**
8. **ROM_generador**
9. **Proyecto ISE**

Referencias

[1] Sahin, I., Koyuncu, I. Design and implementation of neural networks neurons with radbas, logsig and tansig activation functions on FPGA. Elektronika ir Elektrotechnika, 120(4), 51-54. 2012.

[2] Omondi, A.R. and Rajapakse, J.C. (Editors). FPGA implementation on Neural networks. Dordrecht, The Netherlands. Springer

[3] Ortiz-Rodriguez, J. M., Martinez-Blanco, M. R., Ornelas-Vargas, G., Guerrero-Ozuna, H. A., Mendoza-Hasso, J. H., Solis-Sanchez, L. O., & Castañeda-Miranda, R. A study using the robust design of artificial neural networks methodology in neutron spectrometry. In Industrial Technology (ICIT), 2016 IEEE International Conference. Pp. 1600-1606, 2016.

[4] Ortiz Rodriguez J.M., et al. A neutron spectrum unfolding code based on generalized regression artificial neural networks. Proccedings of the ISSSD 2015. Mexico, city. September 26-30, 2015.

[5] Vega-Carrillo H.R., Hernandez-Davila V.M., Manzanares-Acuña E., et al. Artificial neural networks technology for neutron spectrometry and dosimetry. Radiation Protection Dosimetry 126(1-4): 408-412, 2007.

[6] Vega-Carrillo H.R., Hernandez-Davila V.M., Manzanares-Acuña E., Mercado-Sanchez G.A., Iñiguez de la Torre M.P., Barquero R., Palacios F., Méndez-Villafañe R., Arteaga-Arteaga T. and Ortiz-Rodríguez J. M. Neutron spectrometry using artificial neural networks. Radiation Measurements, 41:425-431, 2006.

[7] Vega-Carrillo H.R., Hernandez-Davila V.M., Manzanares-Acuña E., Mercado-Sanchez G.A., Ortiz-Rodríguez J.M., Iñiguez de la Torre M.P., Barquero R., and Arteaga-Arteaga T. Neutron spectrometry with artificial neural networks. XVI Congreso Anual de la SNM y XXIII Reunion Anual de la SMSR, 10-13/Jul/2005.

[8] Esraa Zeki M. and Haitham Kareem A. Hardware Implementation of Artificial Neural Network Using Field Programmable Gate Array. International Journal of Computer Theory and Engineering, 5(5), 780-783, 2013.

[9] Arroyo León M.A., Ruiz Castro A. and Leal Ascencio R.R. An Artificial Neural Network on a Field Programmable Gate Array as a virtual sensor. ITESO, Departamento de Electrónica, Sistemas e Informática, Tlaquepaque, Jalisco, 45090, MÉXICO.

[10] Muthuramalingam A., Himavathi, S. and Srinivasan, E. Neural Network Implementation Using FPGA: Issues and Application. World Academy of Science, Engineering and Technology. International Journal of Electrical, Computer, Energetic, Electronic and Communication Engineering , 2(12), 2008.

[11] Santos-Diego, Nunes-Henrique, Morgado-Fernando. Dual processor neural network implementation in FPGA. Madeira Interactive Technologies Institute and Centro de Competências de Ciências Exactas e da Engenharia, Universidade da Madeira. Project PEst-OE/EEI/LA0009/2011.

[12] Ortega-Zamorano F., Jerez José M., Urda-Muñoz D., Luque-Baena R. Efficient implementation of the Backpropagation algorithm in FPGAs and microcontrollers. IEEE Transactions on Neural Networks and Learning Systems, Volume: 27, Issue: 9, Sept. 2016. The final version of record is available at $http$: $//dx.doi.org/10.1109/TNNLS.2015.2460991$

[13] Ortiz Rodriguez J.M. Diseño robusto de redes neuronales artificiales aplicadas a la espectrometría de neutrones. Tesis de maestría, Universidad Autónoma de Zacatecas, 2005.

[14] Cember H. Introduction to Health Physics. McGraw-Hill, 3a edition, 1996.

[15] Knoll G.F. Radiation Detection and Measurement. John Wiley$\&$ Sons, 3a edition, 1999.

[16] Mackovicka L. Contribution à la dosimètrie neutron-gamma: Etude d´un ensemble radiateur-dètecteur type CR 39. PhD thesis, Université de Limoges, 1987.

[17] IAEA. Compendium of neutron spectra in criticality accident dosimetry. Technical Report 180, 1979.

[18] IAEA. Compendium of neutron spectra and detector responses for radiation protection purposes. Technical Report 318, 1990.

[19] Brooks F. D. and Klein H. Neutron spectrometry historical review and present status.Nuclear Instruments and Methods in Physics Research A, 476:1-11, 2002.

[20] Marion J.B. and Fowler J.F. Fast neutron physics, volume 1 & 2. Interscience, New York,1960.

[21] Bromley D.A. Detectors in nuclear science. Nuclear instruments and methods, 162, 1979.

[22] Ferguson A.T.G. Fast neutron physics, volume 1, page 179. Interscience, New York, 1960

[23] Dearnaley G. Progress in fast neutron physics, page 173. University of Chicago, 1963.

[24] Miller S.C. AFITBUNKI: a modified iterative code to unfold neutron spectra from Bonner sphere detector data. Master‘s thesis, 1993.

[25] Fehrenbacher G., Schutz R., Hahn K., Sprunck M.E., Biersack J.P. and Wahl W. Proposal of a new method for neutron dosimetry based on spectral information obtained by application of artificial neural networks. Radiation Protection Dosimetry, 83(4):293-301,1999.

[26] Cordes E., Fehrenbacher G., Schutz R., Sprunck M., Hahn K., Hofmann R., Biersack J.P. and Wahl W. An approach to unfold the response of a multi-element system using an artificial neural network. IEEE Transactions on Nuclear Science, 45(3):1464-1469, 1998.

[27] Braga C.C., Dias M.S. Application of neural networks for unfilding neutron spectra measured by means of Bonner spheres. Nuclear Instruments and Methods in Physics Research Section A, 476(1-2):252-255, 2002.

[28] Kardan M.R., Setayeshi S., Koohi-Fayegh R. and Ghiassi-Nejad M. Neutron spectra unfolding in Bonner spheres spectrometry using neural networks. Radiation Protection Dosimetry, 104(1):27-30, 2003.

[29] Kardan M. R., Koohi-Fayegh R., Setayeshi S. and Ghiassi-Nejad M. Fast neutron spectra determination by threshold activation detectors using neural networks. Radiation Measurements, 38:185-191, 2004.

[30] Apfel R.E. The superheated drop detector. Nuclear Instruments and Methods, 162:603-608, 1979.

[31] Mukherjee B. ANDI-03: a genetic algorithm tool for the analysis of activation detector data to unfold high-energy neutron spectra. Radiation Protection Dosimetry, 110(1-4):249-254, 2004.

[32] Thomas D.J. Neutron spectrometry for radiation protection. Radiation Protection Dosimetry,110(1-4):141-149, 2004.

[33] Alevra A.V., Cosack M., Hunt J.B., Thomas D.J. and Schraube H. Experimental determination of the response of four Bonner sphere sets to monoenergetic neutrons (II). Radiation Protection Dosimetry, 40(2):91-102, 1992.

[34] Bramblett R.L., Ewing R.I. and Bonner T.W. A new type of neutron spectrometer. Nuclear Instruments and Methods, 9:1-12, 1960.

[35] Vega-Carrillo H.R., Hernandez-Davila V.M., Manzanares-Acuña E., Mercado Sanchez G.A., Gallego E., Lorente A., Perales-Muñoz W.A. and Robles-Rodríguez J.A. Artificial neural networks in neutron dosimetry. Radiation Protection Dosimetry, 118(3):251-259,2006.

[36] Vega-Carrillo H.R., Wehring B.W., Veinot K.G. and Hertel N.E. Response matrix for a multisphere spectrometer using a ^{6}LiF thermoluminescence dosimeter. Radiation Protection Dosimetry, 81(2):133-139, 1999.

[37] Vega-Carrillo H.R., Hernandez-Davila V.M., Manzanares-Acuña E., Gallego E., Lorente A. and Iñiguez M.P. Artificial neural networks technology for neutron spectrometry and dosimetry. Radiation Protection Dosimetry, 126(1-4):408-412, 2007.

[38] Matzke M. Unfolding procedures. Radiation Protection Dosimetry, 107(1-3):155-174,2003.

[39] Sweezy J., Hertel N. and Veinot K. BUMS-Bonner sphere unfolding made simple: an html based multisphere neutron spectrometer unfolding package. Nuclear Instruments and Methods in Physics Research A. 476(1-2):263-269, 2002.

[40] Bedogni R. Neutron spectrometry and dosimetry for radiation protection around a high energy electron/positron collider. Tesis doctoral, Universidad Autónoma de Barcelona. 2006.

[41] Lacoste V., Reginatto M., Asselineau B. and Muller H. Bonner sphere neutron spectrometry at nuclear workplaces in the framework of the ENVIDOS project. Radiation Protection Dosimetry, 125(1-4):304-308, 2007.

[42] Miller S.C. AFITBUNKI: a modified iterative code to unfold neutron spectra from Bonner sphere detector data. Master theses, United States Air Force University, 1993.

[43] Tomás M., Fernández F., Bakali M. and Muller H. MITOM: a new unfolding code based on a spectra model method applied to neutron spectrometry. Radiation Protection Dosimetry,110(1-4):545-548, 2004.

[44] Bedogni R., Domingo C., Esposito A. and Fernández F. FRUIT: An operational tool for multisphere neutron spectrometry in workplaces. Nuclear Instruments and Methods in Physics Research A,580(3):1301-1309, 2007.

[45] Ortiz-Rodríguez J.M., Martínez-Blanco M.R., Gallego E., and Vega-Carrillo H.R. Artificial neural networks modeling evolved genetically, a new approach applied in neutron

[46] Ortiz-Rodríguez J.M., Martínez-Blanco M.R., Gallego E., and Vega-Carrillo H.R. Neutron spectrometry and dosimetry based on a new approach called genetic artificial neural networks. 12th International Congress of the International Radiation Protection Association (IRPA´2008), 19-24/Oct/2008.

[47] Haykin S., Neural Networks: A comprehensive Foundation, Second Edition, Prentice Hall (1999).

[48] Gupta, M.M., Jin, L. and Homma N. Static and dynamic neural networks: from fundamentals to advanced theory. 2003.

[49] Martínez Blanco, M.R. NSDann: Una herramienta de cómputo para la espectrometría y dosimetría de neutrones basada en RNA. Tesis de maestría, Universidad Autónoma de Zacatecas. 2009.

[50] Saumil, G.M. and Gregory, D.P. 2010. Evolvable Block-Based Neural Network Design for Applications in Dynamic Environments. Department of Electrical and Computer Engineering, George Washington University, 20101 Academic Way, Ashburn, VA 20147-2604, USA

[51] Sulaiman N., Zeyad Assi Obaid, M. H. Marhaban and M. N. Hamidon. Design and Implementation of FPGA-Based Systems - A Review. Australian Journal of Basic and Applied Sciences, 3(4): 3575-3596, 2009 ISSN 1991-8178c 2009, INSInet Publication

[52] Zhu, J.H. and Sutton, P. 2003. FPGA implementations of neural networks - A survey of a decade of progress. In: Cheung, P.Y.K., Constantinides, G.A. and De Souza, J.T. (eds.) Field-Programmable Logic and Applications, Proceedings.

[53] Eldredge, J.G. and B.L. Hutchings. Density enhancement of a neural network using FPGAs and run-time reconfiguration. in Proceedings of IEEE Workshop on Field- Programmable Custom Computing Machines, 1994. pp 180-188.

[54] Spartan-3E FPGA Family Data Sheet, DS312 (v3.8) August 26, 2009 Data Sheet