UNIVERSIDAD TECNOLÓGICADE LA MIXTECA

DIVISIÓN DE ESTUDIOS DE POSGRADO

“Implementación de redes neuronales sobre lógica reconfigurable”

PARA OBTENER EL GRADO DE:

Maestro en Electrónica, Opción: Sistemas Inteligentes Aplicados

PRESENTA:

Ing. Magdiel Pascual García Juárez

DIRECTOR:

Dr. Enrique Guzmán Ramírez

Huajuapan de León, Oaxaca; enero de 2015.

Tesis presentada en enero de 2015 ante los sinodales:

Dr. Antonio Orantes Molina

Dr. Felipe de Jesús Trujillo Romero

Dr. Ricardo Pérez Águila

Dr. Rosebet Miranda Luna

Director de tesis:

Dr. Enrique Guzmán Ramírez

Resumen

El presente trabajo de tesis versa sobre el diseño e implementación de un sistema enfocado al modelado e implementación de una Red Neuronal Artificial (RNA) tipo perceptrón multicapa (MLP, *Multi-layer Perceptron*) sobre un FPGA *(Field Programmable Gate Array)*. Además, el sistema permite monitorear el desempeño de la red cuando se adapta a una aplicación específica.

El sistema, denominado un **Sistema para el modelado de RNA-MLP sobre lógica reconfigurable**, tiene como elemento central de procesamiento a un FPGA Spartan-3E 500 manufacturado por la compañía Xilinx.

El sistema cuenta con una interfaz gráfica de usuario, la cual implementa el algoritmo de entrenamiento "*backpropagation*" y envía los pesos sinápticos, resultantes de este procedimiento, a la RNA modelada en el FPGA para su adaptación a una aplicación específica. Además, esta interfaz permite al usuario enviar patrones de prueba a dicha red.

El complemento de este sistema es un conjunto de módulos que describen los diferentes elementos, mediante el lenguaje VHDL (*Very High Speed Hardware Description Language*), que componen a un sistema neuronal, y que pueden ser instanciados para crear la estructura de una RNA. Se realizó el modelado del cálculo del producto punto utilizando una arquitectura convencional y una arquitectura basada en Aritmética Distribuida. Respecto al cálculo de la función de activación, ésta fue modelada usando las técnicas Ley A (*Efficient implementation*

of piecewise linear activation function for digital VLSI neuronal networks), Alippi (*Simple Approximation of Sigmoidal Functions: Realistic Design of Digital Neural Networks Capable of Learning*), PLAN (*Piecewise linear approximation applied to nonlinear function of a neural network*) y CRI (*Centred Recursive Interpolation*).

Finalmente, se muestra el desempeño del sistema al modelar RNA para resolver los problemas de la XOR y de la planta Iris, exponiendo un análisis de resultados tomando en cuenta los parámetros de

- Porcentaje de reconocimiento.
- Tiempos de operación.
- Recursos consumidos.

Dedicatoria

Con especial afecto y cariño a:

Mi esposa

Lorena

Mi hija

Carolina

Agradecimientos

A mi director de tesis Dr. Enrique Guzmán Ramírez por su amistad, el conocimiento y tiempo invertido para la realización de la presente.

A mis sinodales Dr. Antonio Orantes Molina, Dr. Felipe de Jesús Trujillo Romero, Dr.Ricardo Pérez Águila, Dr Rosebet Miranda Luna cuyas observaciones y sugerencias contribuyeron al mejoramiento de la presente tesis.

A mi esposa e hija Lorena y Carolina por sus palabras de aliento y constante apoyo durante todo el periodo que duró el presente trabajo de tesis.

A mi madre y hermanos que siempre me han apoyado.

Índice General

Resumen……v
Dedicatoria……..vii
Agradecimientos……..ix
Índice General……xi
Índice de Figuras……xv
Índice de Tablas……xix
Capítulo 1 Introducción 1
1.1 Planteamiento del problema.....3
1.2 Justificación 3
1.3 Limitantes de la propuesta 4
1.4 Hipótesis 4
1.5 Objetivos 5
1.6 Metas.....5
1.7 Descripción de la propuesta 6
1.8 Estructura de la tesis 7

1.9 Publicaciones generadas....8
Capítulo 2 Marco teórico 9
2.1 Redes Neuronales Artificiales....9
2.1.1 Fundamento biológico....13
2.1.2 Modelo de una Neurona Artificial....15
2.1.3 Definición de Redes Neuronales Artificiales....19
2.1.4 Arquitectura de las Redes Neuronales Artificiales....20
2.1.5 Algoritmo de entrenamiento o aprendizaje....23
Capítulo 3 Métodos y tecnología utilizada 27
3.1 Perceptrón multicapa y Backpropagation....27
3.2 RNA's sobre lógica reconfigurable....37
3.3 Arreglo de Compuertas Programables de Campo....39
3.4 Estado del arte....42
Capítulo 4 Diseño del sistema 47
4.1 Sistema para el modelado de RNA-MLP sobre lógica reconfigurable....47
4.2 Interfaz de usuario....49
4.3 Sistema embebido RNA-MLP....53
4.3.1 Controlador DLP-USB....54
4.3.2 RNA lógica reconfigurable....55
4.3.2.1 Driver DLP-USB....55
4.3.2.2 Unidad de control general....57
4.3.2.3 RNA modular....58
4.3.2.3.1 Definición de la estructura de la neurona artificial....59
Definición del módulo Pesos sinápticos....61
Definición del módulo Regla de propagación....62
Arquitectura convencional....64
Arquitectura basada en Aritmética distribuida....67
Definición del módulo Función de activación....70
Ley A....72
Alippi....74
Aproximación por Interpolación Recursiva Centrada....77
PLAN....78
4.3.2.3.2 Estructura de la RNA....81

Ejecución de una capa con Arquitectura Convencional....85
Ejecución de una capa con Arquitectura basada en Aritmética Distribuida....86

Capítulo 5 Pruebas y resultados 89

5.1 Problema XOR....90
5.1.1 Definición del problema XOR....90
5.1.2 Implementación en hardware de la RNA para el problema XOR....90
5.1.2.1 Implementación RNA de la XOR en el sistema embebido RNA-MLP91
5.1.2.2 Adecuación de la Interfaz de usuario.... 94
5.1.2.3 Resultados para la RNA del problema XOR.... 97
5.2 Problema de la planta Iris....101
5.2.1 Definición del problema de la planta Iris....101
5.2.2 Implementación de la RNA para el problema de la Iris Planta....102
5.2.2.1 Implementación RNA de la planta Iris en el sistema embebido RNA-MLP……...103
5.2.2.2 Adecuación de la interfaz de usuario para la planta Iris 106
5.2.2.3 Resultados para la RNA del problema de la planta iris.... 108

Capítulo 6 Conclusiones y trabajos futuros 113

6.1 Conclusiones....113
6.2 Trabajos futuros 115

Bibliografía……117

Índice de Figuras

Figura 1.1 Diagrama a bloques general del sistema propuesto..... 6

Figura 2.1 Estructura de una neurona biológica [55]..... 14

Figura 2.2 Modelo de Neurona Artificial [53]..... 16

Figura 2.3 Modelo de Neurona Artificial Estándar [53]..... 18

Figura 2.4 RNA monocapa. 22

Figura 2.5 RNA multicapa..... 22

Figura 2.6 RNA recurrentes..... 23

Figura 2.7 Proceso de aprendizaje supervisado [48]. 24

Figura 2.8 Clasificación de las RNA's por el tipo de aprendizaje y la arquitectura [53]..... 25

Figura 3. 1. Arquitectura general de una RNA-MLP. 29

Figura 3. 2 Arquitectura del FPGA..... 41

Figura 3.3 Diagrama a bloques de la tarjeta Nexys-2..... 43

Figura 4. 1 Diagrama a bloques del sistema propuesto. 49

Figura 4. 2 Sub-sistema Interfaz de usuario (a) Especificación de parámetros de la RNA, (b) Resultados del clasificador, (c) Pruebas del sistema, (d) Especificación de parámetros del backpropagation. 50

Figura 4. 3 Estructura de las tramas del protocolo de comunicaciones. 52

Figura 4. 4 Sistema embebido RNA-MLP. 54

Figura 4.5 Símbolo del módulo Driver DLP-USB. 56

Figura 4. 6 Símbolo del módulo Unidad de control general. 57

Figura 4.7 Símbolo del módulo Neurona. 59

Figura 4.8 Estructura interna de la Neurona Artificial. 61

Figura 4.9 Símbolo del módulo Pesos sinápticos. 62

Figura 4.10 Símbolo del módulo Regla de propagación. 63

Figura 4.11 Esquema MAC. 64

Figura 4.12 Definición de la arquitectura convencional. 65

Figura 4.13 Símbolo del módulo Multiplicador. 65

Figura 4.14 Símbolo para el módulo Suma. 67

Figura 4.15 Arquitectura basada en aritmética distribuida. 69

Figura 4.16 Definición de la arquitectura basada en Aritmética Distribuida. 70

Figura 4. 17 Símbolo del módulo Función de activación. 71

Figura 4.18 Selector del bit 1 más significativo. 74

Figura 4.19 Arquitectura completa de la Ley A modificada. 74

Figura 4.20 Función sigmoidea, breakpoints Pn,Qn. 75

Figura 4.21 Arquitectura de la función de aproximación Alippi. 76

Figura 4. 22 Diagrama para la implementación en hardware de CRI 79

Figura 4.23 Obtención del rango del registro de entrada x. 79

Figura 4.24 Propuesta del hardware para la DTX. 80

Figura 4.25 Propuesta en hardware de PLAN. 81

Figura 4.26 Conexión de una capa en una RNA. 81

Figura 4.27 Símbolo del módulo ControlWr_w. 82

Figura 4.28 Símbolo del módulo Registro de capa. 83

Figura 4. 29 Símbolo del módulo Control de capa 84

Figura 4.30 Símbolo del módulo preRegistro 85

Figura 5.1 Definición de la RNA para el problema de la compuerta XOR. 91

Figura 5.2 Diagrama de conexiones de la RNA para la compuerta XOR. 92

Figura 5.3 Conexiones de la RNA con Aritmética distribuida. 93

Figura 5. 4 Especificación de parámetros para el diseño de la RNA. 95

Figura 5.5 Especificación de los parámetros para el entrenamiento de la RNA. 96

Figura 5.6 Pestaña para establecer fase de prueba de la RNA implementada en el FPGA. 97

Figura 5.7 Resultados del clasificador de la RNA de la compuerta XOR. 98

Figura 5.8 Ventana para introducir los descriptores de un patrón. 98

Figura 5.9 Clasificador para diversos valores de entrada. 98

Figura 5.10 Definición de la RNA para el problema de la planta Iris. 103

Figura 5.11 Diagrama de conexiones de la RNA con arquitectura convencional para la planta Iris. 104

Figura 5.12 Conexiones para la RNA con aritmética distribuida para la planta Iris. 105

Figura 5.13 Especificación de parámetros para el diseño de la RNA de la panta Iris. 106

Figura 5.14 Especificación de los parámetros para el entrenamiento de la RNA de la planta iris. 107

Figura 5.15 Pestaña para establecer fase de prueba de la RNA implementada en el FPGA. . 108

Figura 5.16 Pestaña de pruebas para la RNA con Aritmética distribuida. 109

Figura 5.17 Mensaje de salida para la prueba de un patrón. 109

Índice de Tablas

Tabla 2.1 Funciones de activación habituales [53]. 17

Tabla 4. 1. Señales del módulo Driver DLP-USB. 56

Tabla 4.2 Señales del módulo Unidad de control general. 57

Tabla 4.3 Descripción de las señales del módulo Neurona. 59

Tabla 4.4 Descripción de las señales del módulo Pesos sinápticos. 62

Tabla 4.5 Descripción del módulo Regla de propagación. 63

Tabla 4.6 Descripción de las señales del módulo Multiplicador. 66

Tabla 4.7 Descripción de las señales del módulo Suma. 66

Tabla 4.8 Memoria RAM con aritmética distribuida. 69

Tabla 4.9 Descripción de las señales del módulo Función de activación. 71

Tabla 4. 10 Valores de la función propuesta. 73

Tabla 4.11a Tabla de verdad de las entradas de la Ley A modificada 73

Tabla 4.11b. Tabla de verdad de las salidas de la Ley A modificada. 73

Tabla 4.12 Tabla de valores de q y Δ. 78

Tabla 4.13 Representación de la aproximación PLAN. 78

Tabla 4.14 Descripción del módulo ControlWr_w. 82

Tabla 4.15 Descripción de las señales del módulo Registro de capa. 83

Tabla 4.16 Descripción de las señales del módulo Control de capa. 84

Tabla 4.17 Descripción de las señales del módulo preRegistro. 85

Tabla 5.1 Rasgos y clasificación del problema XOR. 90

Tabla 5. 2 Conjunto de entrenamiento para el problema XOR. 95

Tabla 5. 3 Valores iniciales de pesos sinápticos y umbrales para el problema XOR. 96

Tabla 5. 4 Resultado de los pesos sinápticos y umbral después del entrenamiento. 96

Tabla 5. 5 Porcentaje de reconocimiento para la compuerta XOR con Arq. Convencional. 99

Tabla 5. 6 Porcentaje de reconocimiento para la compuerta XOR con Arq. b/Aritmética Dist. 99

Tabla 5.7 Recursos utilizados por la implementación con Arq. Convencional de la XOR. ... 100

Tabla 5.8 Recursos utilizados por la implementación con Arq. b/Aritmética Dist.de la XOR 100

Tabla 5. 9 Recursos utilizados por las funciones de activación. 100

Tabla 5.10 Reporte de tiempos de operación de la función de activación. 101

Tabla 5.11 Tiempos de operación de la RNA para la XOR. 101

Tabla 5.12 Descriptores de los patrones del problema de la Iris Planta. 102

Tabla 5.13 Definición de las salidas para la RNA para el problema de la planta iris. 102

Tabla 5.14 Valores de los pesos sinápticos y umbra de la RNA para la planta iris. 107

Tabla 5.15 Porcentaje de reconocimiento la planta Iris con arquitectura convencional. 109

Tabla 5.16 Porcentaje de reconocimiento la planta Iris con arquitectura b/Aritmética distribuida. 110

Tabla 5.17 Resultados de la RNA para el problema de la planta Iris 110

Tabla 5.18 Recursos utilizados por la implementación con Arq. Convencional de la planta Iris. 111

Tabla 5.19 Recursos utilizados por la implementación con Arq. b/Aritmética Dist. de la planta Iris. 111

Tabla 5.20 Tiempos de operación de la RNA para la planta Iris 112

Capítulo 1 Introducción

La capacidad del cerebro humano para realizar en forma eficiente procesos tan complejos como razonar, almacenar y procesar información, generar conocimiento de la experiencia, percibir su entorno, responder a situaciones nuevas, etc., ha inspirado a muchos científicos a intentar o procurar modelar el comportamiento del cerebro humano.

En este sentido, una de las ramas más destacadas del campo científico de la Inteligencia Artificial es la que corresponde a las Redes Neuronales Artificiales, entendiendo como tales aquellas estructuras en las que existen unidades básicas de procesamiento de información de cuyas interacciones locales depende el comportamiento del sistema [36]. El nombre dado a esta rama de las ciencias se deriva de dos hechos principalmente, primero, la unidad básica de procesamiento está inspirada en la célula fundamental del sistema nervioso humano, la neurona; y segundo, la interconexión de estas unidades intentan emular le red de interconexiones que presentan las neuronas en el cerebro humano.

Debido a que una RNA representa una eficiente herramienta de análisis de información, tiene un amplio rango de aplicaciones, por ejemplo, en análisis y predicción financiera,

reconocimiento de patrones, análisis y procesamiento de señales, imágenes y video, diagnóstico médico, etc. En general, las RNAs resultan bastante apropiadas para aplicaciones en las que no se dispone a priori de un modelo identificable que pueda ser programado, pero se dispone de un conjunto básico de ejemplos de entrada. También han sido aplicadas con un éxito más que razonable a aquellos problemas de los cuales no existen modelos matemáticos precisos o algoritmos con complejidad razonable.

De acuerdo a respetables investigadores en el área de las RNAs, como Hecht-Nielsen, Caudill, Butler, Feldman, Ballard, Rumelhart, Hinton, Williams y Kohonen, una RNA es un sistema de procesamiento de información que consta de un gran número de unidades simples de procesamiento, altamente interconectadas y jerarquizadas en capas o niveles, capaz de adaptarse a diversas aplicaciones para responder dinámicamente a estímulos externos [33], [15], [21], [72], [46]. A esta definición, se le puede agregar que una RNA presenta un alto grado de concurrencia en los diferentes elementos que la integran. En este sentido, de acuerdo con Hecht-Nielsen [35] y Freman-Skapura [22] "*una RNA es una estructura de procesamiento de información, cuyos elementos se encuentran distribuidos, trabajan en forma paralela y se interconectan entre sí en forma de grafo dirigido*".

Por otro lado, la lógica reconfigurable, que tiene como principal representante al FPGA, consiste de un arreglo matricial de bloques lógicos reconfigurables (CLB, *Configurable Logic Block*), utilizados para implementar funciones lógicas combinacionales o secuenciales simples, comunicados a través de recursos de interconexión programables [10]. La popularidad de un FPGA para implementar algoritmos de diversas áreas se ha incrementado significativamente en los últimos años, sobre todo en aquellos que presentan un alto grado de paralelismo, por ejemplo, operaciones aritméticas de punto flotante, procesamiento de imágenes [9], [54], [27], [31] encriptación de información [74], compresión de información [26], extracción de características [13], algoritmos para clasificación [39], reconocimiento de rostros [20], control digital [83], [1], algoritmos genéticos [19], y por supuesto RNAs [69], [37], [64], [42].

El presente trabajo de tesis versa sobre la implementación de RNA sobre lógica reconfigurable, por tal motivo, fue necesario realizar un estado del arte referente al vínculo existente entre estas dos áreas, el cual ha sido vertido en el sub-tema 3.4.

1.1 Planteamiento del problema

Usualmente suele considerarse a las RNAs como modelos naturales de cómputo paralelo; esto es debido a los diferentes tipos de paralelismo que una RNA exhibe. De acuerdo a Omondi *et al*. [64], una RNA posee los siguientes tipos de paralelismo.

- Paralelismo en la etapa de entrenamiento.
- Paralelismo a nivel de capa.
- Paralelismo a nivel de nodo (externo).
- Paralelismo a nivel de nodo (interno).
- Paralelismo a nivel de bit.

Considerando al paralelismo como una importante característica de una RNA, y que al aprovecharlo su rendimiento puede mejorar, resulta de gran importancia la realización de estudios minuciosos e implementación de herramientas que tengan por objetivo ayudar a resolver problemas relacionados con el aprovechamiento de dicha característica, como

1. Buscar métodos existentes que permitan explotar los diferentes niveles de paralelismo presente en una RNA.
2. Determinar el hardware o arquitectura adecuada para el mapeo de estos métodos.
3. Adaptar y evaluar su desempeño para determinar cuál de estos métodos representa la opción más apropiada para una aplicación específica.
4. Detectar y proponer nuevos paradigmas que permitan sacar provecho a los diferentes niveles de paralelismo que una RNA posee.
5. Generar un ambiente propicio para su evaluación.

1.2 Justificación

Resulta frecuente que las RNAs sean implementadas en procesadores, ya sea en microcontroladores, procesadores digitales de señales (DSP) o computadoras personales (PC), basadas en microprocesadores. Aunque existe un amplio rango de aplicaciones en las que este tipo de arquitectura es adecuada, debido a que la ejecución de la implementación de un algoritmo en ellas se lleva a cabo en forma secuencial, el paralelismo implícito en una RNA

no puede ser aprovechado. Por su parte, un FPGA tiene la capacidad de evaluar varios procesos en forma concurrente.

Por otro lado, para la implementación de RNA's en aplicaciones donde se requiera explotar el paralelismo inherente en una RNA suelen emplearse diversos dispositivos de cálculo paralelo como

- Computadoras paralelas de propósito general
- Neuro-computadoras
- Circuitos integrados de aplicación específica (asics) analógicos y digitales
- FPGA's

Girau realizó una evaluación de las tecnologías mencionadas tomando en cuenta la velocidad de procesamiento, área de semiconductor utilizada, costo de un producto final, el tiempo de diseño y la confiabilidad de la implementación final, en su estudio muestra que un FPGA representa la opción más equilibrada para la implementación de RNA`s [28]. Una explicación más detallada al respecto se presenta en la sección 3.2.

Los argumentos vertidos justifican en demasía el por qué utilizar un FPGA como elemento de procesamiento en la implementación de RNA.

1.3 Limitantes de la propuesta

Considerando que el punto medular del trabajo propuesto es el estudio de implementar arquitecturas de RNA sobre lógica reconfigurable, es necesario mencionar los alcances del mismo.

1. El presente trabajo sólo se plantea para el modelo del MLP
2. El algoritmo de entrenamiento queda excluido del estudio, es decir, el entrenamiento de la RNA se llevara a cabo en una computadora personal
3. Las arquitecturas utilizadas en el estudio serán probados en aplicaciones sobre funciones booleanas y reconocimiento de patrones

1.4 Hipótesis

Un análisis profundo relacionado con técnicas utilizadas en la implementación de RNAs sobre lógica reconfigurable puede ayudar a obtener información objetiva que permita contar

con criterios que pueden ser utilizados para determinar cuál técnica resulta más adecuada para cierta arquitectura de RNA y/o su utilización en una aplicación específica, encontrar nuevos métodos que permitan optimizar su implementación y/o desempeño y brindar un ambiente de diseño propicio para experimentar con nuevas técnicas aplicadas a este tema.

1.5 Objetivos

El objetivo principal del presente trabajo de tesis es **realizar un estudio que permita evaluar el desempeño de las principales técnicas utilizadas en cada uno de los elementos que componen a una RNA en su implementación sobre lógica reconfigurable.**

Para cumplir con el objetivo planteado, los siguientes objetivos secundarios son necesarios:

1. Implementar una neurona artificial sobre lógica reconfigurable usando una técnica modular.
2. Implementar una RNA sobre lógica reconfigurable usando la neurona artificial modular diseñada.

1.6 Metas

Las metas planteadas por objetivo son:

1.1. Definir la estructura de la RNA y de la neurona artificial que será utilizada en el presente estudio.

1.2. Identificar cada elemento de una neurona artificial cuya implementación sea viable en un FPGA.

1.3. Realizar un estudio de las técnicas utilizadas en cada uno de estos elementos e implementarlas en un FPGA.

2.1 Realizar un estudio de las técnicas utilizadas en el modelado de RNAs en un FPGA e implementarlas.

Finalmente las metas para conseguir el objetivo principal son:

1. Realizar una comparativa entre las diferentes técnicas implementadas, teniendo como principales parámetros la velocidad de procesamiento y la cantidad de recursos utilizados.
2. Realizar una comparativa para medir la eficiencia de las técnicas empleadas cuando son implementadas en aplicaciones específicas.

1.7 Descripción de la propuesta

El resultado principal que este trabajo de investigación ha generado es un sistema que se ha denominado **Sistema para el modelado de RNA-MLP sobre lógica reconfigurable**. Dicho sistema es una herramienta que permite al usuario llevar a cabo la implementación de un MLP sobre un FPGA y monitorear su desempeño cuando se adapta a una aplicación específica. El sistema cuenta con diversos módulos que describen los diferentes elementos que componen a un sistema neuronal, y que pueden ser instanciados para crear la estructura de una RNA. Estos módulos fueron diseñados como elementos de tamaño genérico, lo que permite fácilmente adaptar la RNA a una aplicación específica. Es decir, con la herramienta propuesta es fácil construir una RNA de un determinado número de entradas, de capas ocultas, de neuronas en las capas ocultas y de salidas.

El modelado de estos módulos se realizó mediante un HDL utilizando niveles de abstracción RTL y comportamental. La herramienta EDA-CAD utilizada en la síntesis e implementación del sistema diseñado es el software ISE Foundations versión 12.1 de la compañía Xilinx Inc. y, aunque los módulos fueron descritos favoreciendo la portabilidad del sistema propuesto, la herramienta EDA-hardware en la que fue implementado y probado es la tarjeta de evaluación Nexys 2 de la compañía Digilent Inc., que cuenta con un FPGA Spartan 3E.

El diagrama de bloques del sistema propuesto se muestra en la Figura 1.1.

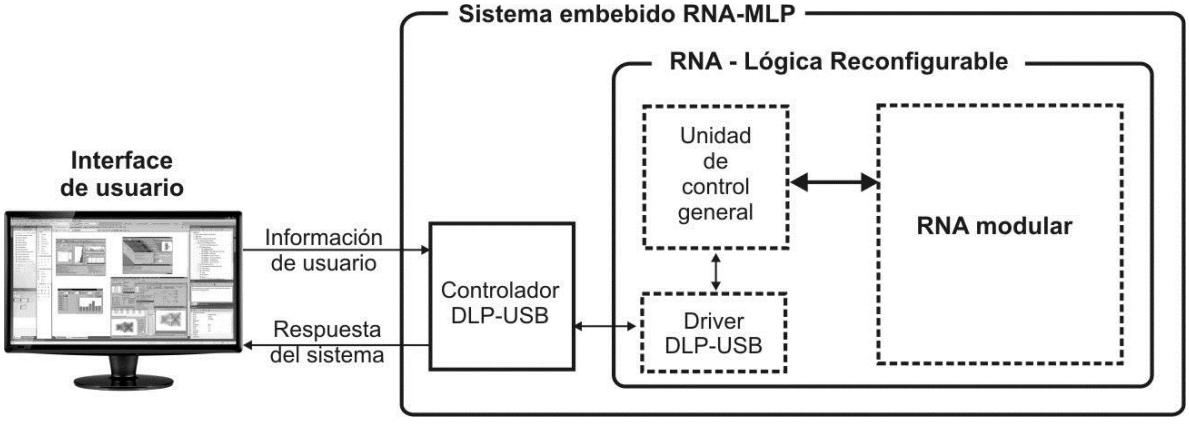

Figura 1.1 Diagrama a bloques general del sistema propuesto.

Con la finalidad de facilitar el uso de la herramienta propuesta, el diseño del sistema fue desarrollado buscando una estructura modular. De esta forma, el sistema quedo conformado por los siguientes sub-sistemas:

- Un programa de software, desarrollado con el lenguaje C++ Builder y ejecutado en una computadora personal (PC), que permite al usuario interactuar con el hardware del sistema. A partir de este momento, este sub-sistema será denominado **Interfaz de usuario**.
- Un sistema embebido cuyo elemento central de procesamiento es un FPGA y que representa al hardware del sistema. Este sub-sistema será denominado **Sistema embebido RNA-MLP** y ésta constituido por los siguientes componentes:
  - **Controlador DLP-USB**. Se trata de un dispositivo DLP-USB245M [URL1] que tiene por función permitir la transferencia de información entre la **Interfaz de usuario** y el **RNA-Lógica Reconfigurable** vía el puerto USB.
  - **RNA lógica reconfigurable**. Un FPGA que representa al elemento central de procesamiento del sistema propuesto. Está integrado por los siguientes módulos:
    - **Driver DLP-USB**. Este módulo incluye el modelado del protocolo que permite establecer la comunicación con el **Controlador DLP-USB**.
    - **Unidad de control general**. Este módulo interpreta la información proveniente de la **Interfaz de usuario**, mediante la cual se administrara el funcionamiento de la RNA construida dentro del FPGA
    - **RNA modular**. Se trata de un conjunto de módulos que describen la estructura o el comportamiento de cada uno de los elementos que son utilizados para construir la estructura de una RNA.

1.8 Estructura de la tesis

La estructura de este documento de tesis está dividida en 6 capítulos, detallados a continuación.

Capítulo 1. Introducción. Capítulo presente, donde se explica en forma sucinta el tema a desarrollar.

Capítulo 2. Marco teórico. Este capítulo contiene una breve descripción de los conceptos fundamentales necesarios para la comprensión del trabajo desarrollado

Capítulo 3. Métodos y tecnología utilizados. Este capítulo versa sobre el modelo neuronal MLP, el algoritmo de aprendizaje *backpropagation*, y la lógica reconfigurable, todos ellos conforman los métodos y tecnologías utilizadas en este trabajo de tesis.

Capítulo 4. Descripción de la propuesta. Este capítulo describe en detalle el diseño del sistema propuesto. Muestra, inicialmente, como se diseñó una estructura modular de la neurona usada en el sistema. Posteriormente se muestra como, usando el modelo de neurona diseñado, se puede construir una RNA para una aplicación específica..

Capítulo 5. Resultados y discusión. En este capítulo, al incluir aplicaciones de la propuesta generada por este trabajo de tesis, se demuestra la funcionalidad de la herramienta desarrollada.

Capítulo 6. Conclusiones y trabajo futuro. Este capítulo presenta las conclusiones obtenidas de este trabajo de tesis y las perspectivas o trabajos futuros que se plantean para la continuación de esta investigación.

Al final de la tesis, se presenta la bibliografía utilizada como base para el desarrollo la presente investigación.

1.9 Publicaciones generadas

1. E. Guzman-Ramírez, M.P. García, J.L. Barahona & Oleksiy Pogrebnyak (2014). "A generic size neural network based on FPGA". *2014 International Conference on Multimedia, Communication and Computing Application (MCCA2014) - CRC Press*. (Accepted).
2. E. Guzman-Ramírez, M.P. García, J.L. Barahona & Oleksiy Pogrebnyak (2014). "Efficient modular hardware architecture of a neural network". *International Journal of Engineering and industries*, ISSN: 2093-5765 (Accepted).

Capítulo 2 Marco teórico

Este capítulo contiene una breve descripción de los conceptos fundamentales necesarios para la comprensión del trabajo desarrollado. Se describe algunos hechos históricos fundamentales que permitieron evolucionar y adaptar las RNA's a las diversas áreas de la ciencia. Posteriormente se expone los fundamentos generales teóricos de las RNA, empezando con el análisis del modelo de una neurona artificial, siguiendo con la definición y arquitectura de una RNA y finalizando con los diversos tipos de algoritmos de aprendizaje o entrenamiento comúnmente empleados durante el proceso de adaptación de la RNA a una aplicación específica.

2.1 Redes Neuronales Artificiales

Con la finalidad de establecer la evolución de los modelos neuronales hasta la aparición de los que este trabajo de tesis utiliza, el Perceptrón Multicapa y el algoritmo de aprendizaje de retropropagación del error o propagación del error hacia atrás *(EBP, Error Backpropagation)*en esta sección se hace un recuento de hechos históricos referentes a las

RNA que de alguna manera influyeron en el planteamiento y desarrollo de los modelos neuronales es estudio.

El cerebro humano siempre ha sido un “sistema” de gran interés para investigadores de diversas áreas, por lo que, han intentado emular su comportamiento buscando explotar su poder de procesamiento al implementarlas en diversas aplicaciones. En este sentido, el primer modelo artificial de una neurona fue propuesto por Warren McCulloch y Walter Pitts, a través de la primera teoría matemática de un modelo neuronal presentada en el año de 1943 [55]. A partir de este suceso surgieron nuevos y diversos trabajos que dieron paso al área de estudio denominada Redes Neuronales Artificiales.

En el año de 1949 Donal Hebb desarrolló una regla de aprendizaje conocida como “Aprendizaje Hebbiano”, el cual muestra como las neuronas utilizan el refuerzo para fortalecer las conexiones de entrada más importantes [32].

Posteriormente, en 1954, Gabor presenta el "Aprendizaje filtrado", el cual usa el gradiente descendente para obtener el peso sináptico óptimo que minimiza el error cuadrático medio entre las señales de salida actuales y las generadas anteriormente [56]. El mismo año, Marvin Minsky desarrolló una máquina de aprendizaje en donde las conexiones podían ser adaptadas automáticamente [57].

En 1956 Taylor introduce la memoria asociativa utilizando la regla de Hebb [75]. Beurle analiza la activación y propagación de la actividad del cerebro a gran escala [52]. Von Neuman muestra como introducir redundancia y tolerancia a fallas en una red neuronal y muestra como la activación síncrona de muchas neuronas pueden ser usadas para representar cada bit de información [56]. Uttley demostró que las RNA’s con conexiones modificables pueden aprender a clasificar patrones con pesos sinápticos que representan probabilidad condicional. Desarrollo un separador lineal en el que los pesos son ajustados utilizando la entropía de Shannon [76], [9], [10].

Un avance importante se dio cuando Frank Rosenblatt introduce al perceptron en el año 1958 [71]. El perceptrón representa un método de aprendizaje para un modelo neuronal formado por una red de unidades binarias de decisión y que actúa como una función que mapea un conjunto de patrones en un conjunto de clases. Esencialmente, Cada una de las neuronas está basada en el modelo McCulloch-Pitts.

Una continuación natural al modelo de Rosenblatt, es la red neuronal denominada *ADALINE* (***ADA****ptive* ***LI****near* ***NE****uron* o ***ADA****ptive* ***LIN****ear* ***E****lement*) desarrollada por Bernard Widrow y Marcian Edward Hoff [80]. La arquitectura de este modelo es muy similar a la del perceptrón. La diferencia entre estos modelos radica en el algoritmo de aprendizaje, ya que la red ADALINE, y su versión múltiple (MADALINE), utilizan la regla delta, regla Widrow-Hoff o regla del mínimo error cuadrado medio (algoritmo LMS, *Least Mean Square*), el cual supone que la actualización de los pesos sinápticos de la red es proporcional al error que la neurona comete, dicho error es determinado comparando la diferencia entre el valor deseado y la salida lineal obtenida.

Un fuerte estímulo al desarrollo de la auto-organización de los modelos neuronales artificiales se da en el año 1961, cuando Hubel y Wiesel conducen un importante estudio biológico de las propiedades de las neuronas en el córtex visual de los gatos, sentando las bases para el desarrollo de modelos artificiales que simulaban esta propiedad [43].

Avances importantes que siguieron a los mencionados son el desarrollo de un teorema de convergencia del perceptrón de Rosenblat propuesto por Novikoff en 1963 [56], [63] y el desarrollo de RNAs, por parte de Utteley en 1966, donde la información entre los patrones de la neurona estaba representada por la intensidad sináptica [77].

En 1969 Marvin Minsky y Seymour Papert publicaron el libro "*Perceptrons: An Introduction to Computational Geometry*" donde mostraron las limitantes del perceptrón simple [58]. Este trabajo demostró que el perceptrón solo es capaz de resolver problemas que son linealmente separables y que fallaba en problemas relativamente simples, como el problema XOR, que no cumplen esta característica, lo que generó una drástica reducción en la investigación de las RNA's.

A pesar de este suceso, siguieron surgiendo trabajos sobre modelos neuronales, por ejemplo, Kaoru Nakano describió un sistema asociativo, denominado "*Associatron*", junto con algunas sugerencias de cómo utilizar la información más eficientemente en una RNA [61]; Shun-Ichi Amari publicó un trabajo teórico sobre la auto organización en redes con elementos de umbral (*Self-Organizing Nets of Threshold Elements*) [4]; James Anderson realizó un modelo de memoria asociativa lineal siguiendo el planteamiento de Hebb que llamo "*Interactive Memory"*[6]; Teuvo Kohonen presentó sus "*Correlation Matrix Memories*" [44]. Cabe hacer

mención que los trabajos de Anderson y Kohonen, y en cierta medida el de Nakano, dieron lugar al modelo que actualmente se conoce con el nombre genérico de asociador lineal (*Linear Associator*).

Leon Cooper y Charles Elbaum, en 1973, son los primeros en explotar la patente y desarrollo comercial de RNA's (*Nestor Associates*), con su modelo RCE (*Reduced Coulomb Energy*) que tenía su propio sistema de aprendizaje el NLS (*Nestor Learning System*).

En 1974 Paul Werbos desarrolló los principios básicos del backpropagation, mientras desarrollaba su tesis doctoral, al implementar un sistema que estimaba un modelo dinámico para predecir comunicaciones sociales y nacionalismo [78], [79].

A principios de la década de los 80´s, Kunihiko Fukushima presenta la RNA denominada *Neocognitron* [23]. Inspirada por el modelo de Hubel-Wiesel [43], el *Neocognitron* estaba formado por múltiples tipos de celdas conectadas en cascada donde cada una cumplía funciones específicas. El *Neocognitron* inicialmente fue utilizado para el reconocimiento visual de patrones, sus creadores demostraron que podía ser entrenado para reconocer números arábigos escritos por humanos aun cuando estos presentaran deformaciones en su forma [24].

De acuerdo a James A. Anderson y Edward Rosenfeld, editores del compendio *Neurocomputing*, la era moderna de las RNA y las memorias asociativas inicia cuando John Joseph Hopfield presenta su red neuronal, conocida como red de Hopfield, ya que ésta sirvió para recuperar el interés en la investigación sobre modelos neuronales [7]. La red de Hopfield es una red de adaptación probabilística, recurrente y que funcionalmente entraría en la categoría de las memorias autoasociativas. Son arquitecturas de una capa con interconexiones totales y recurrentes; funciones de activación booleana de umbral y regla de aprendizaje no supervisado. La principal aportación de Hopfield consistió en realizar un análisis de las RNA recurrentes, donde mostró la estabilidad de este tipo de redes [82], [41].

El mismo año en que Hopfield presentó su modelo neuronal, otro avance de importancia en esta área fue dado cuando Teuvo Kohonen crea los Mapas Auto-Organizados (SOM, *Self-Organizing Map*) [45]. Los SOM son considerados una RNA de aprendizaje competitivo que han sido utilizados con mucho éxito en la creación de nuevos esquemas para la cuantificación vectorial (VQ) [47].

Una de las primeras RNA inspirada en la estadística, la máquina de Boltzmann, fue propuesta por Geoff Hinton y Terence Sejnowski en el año 1983 [38]. Este modelo neuronal puede ser visto como una extensión de la red Hopfield; es una red neuronal recurrente estocástica que representa la información a partir de una distribución de probabilidad, es decir, modela la distribución de probabilidad subyacente en un conjunto de datos dado. La máquina de Boltzmann se ha utilizado en aplicaciones de segmentación y restauración de imágenes y optimización combinacional.

Durante 1984 y 1885, Robert Hecht Nielsen, Todd Gutschow y Robert Kuczewski diseñaron el TRW MARK III, uno de los primeros neuro-computadores digitales.

En 1986 Rumelhart, Hinton y Wilson formalizaron un método para que una RNA tipo MLP aprendiera la asociación que existe entre un conjunto de patrones de entrada y las clases correspondientes, este método es conocido como algoritmo *backpropagation* [72]. Con el tiempo, *backpropagation* se ha convertido en uno de los modelos neuronales más utilizados demostrando ser una eficiente herramienta en aplicaciones de reconocimiento de patrones, modelado dinámico, análisis de sensibilidad, y el control de los sistemas en el tiempo, entre otros. La sección 3.1 presenta un análisis de este modelo neuronal.

2.1.1 Fundamento biológico

El cerebro humano, formado por aproximadamente 100 mil millones de neuronas comunicadas entre sí por unos 100 billones de conexiones, es el objeto más complejo conocido en el universo.

El estudio de las RNA's se remonta al trabajo de Santiago Ramón y Cajal, considerado el padre de las neurociencias, quien a finales del siglo XIX, descubrió el sistema nervioso y determinó que está compuesto por una red de células individuales, denominadas neuronas, y que éstas están ampliamente interconectadas entre sí.

Una neurona recibe señales eléctricas de sus neuronas vecinas, procesa las señales y genera señales hacia otras neuronas vecinas. El procesamiento de la neurona es paralelo y distribuido. Los elementos principales que forman una neurona son las dendritas (10 micras de longitud), el soma (80 micras de longitud), el axón (100 micras hasta 1m de longitud) y la sinapsis (ver Figura 2.1). Las dendritas actúan como receptor de señales provenientes de neuronas vecinas,

estas señales son procesadas por el soma y el resultado es enviado a través del axón. En el extremo del axón existe una unidad o proceso llamada sinapsis cuya función es controlar el flujo de las señales [48].

La sinapsis permite la transferencia de información entre neuronas. Existen dos tipos de sinapsis, eléctrica y química. El tipo de sinapsis más común es la química, en donde no existe contacto físico entre las neuronas, estas permanecen separadas por un pequeño vacío de unas 0.2 micras (denominada hendidura sináptica) y la información fluye sólo en un sentido por medio de sustancias químicas denominadas neurotransmisoras. La sinapsis eléctrica por su parte, hace uso de descargas eléctricas producidas en el cuerpo celular, y que se propagan por el axón. La sinapsis eléctrica ofrece una vía de baja resistencia entre neuronas, generando un retraso mínimo en la transmisión sináptica debido a que no existe un mediador químico. Las sinapsis pueden ser excitadoras o inhibidoras [53].

Finalmente, las neuronas se agrupan para formar redes neuronales. La entrada a una RNA es proporcionada por receptores sensoriales, éstos proporcionan estímulos del interior del cuerpo y del mundo externo a través de órganos de los sentidos. En respuesta, la red neuronal procesa esta información y genera señales para cumplir cierto comportamiento, en función de la información percibida, la cual se ve reflejada en los órganos actuadores, por ejemplo extremidades, [83].

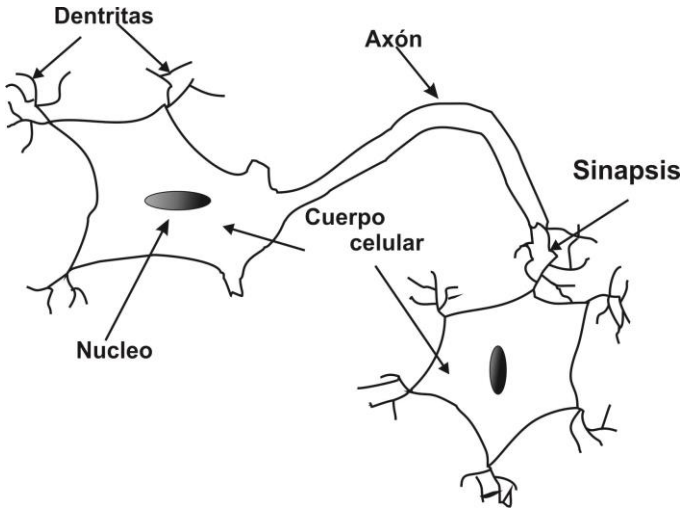

Figura 2.1 Estructura de una neurona biológica [55].

2.1.2 Modelo de una Neurona Artificial

Una neurona artificial posee características inspiradas en el conocimiento o supuestos que se tienen acerca del funcionamiento de la neurona biológica, tanto en aspectos morfológicos como fisiológicos, y se basa en modelos matemáticos de su comportamiento.

Se denomina procesador elemental o neurona artificial a un dispositivo simple de cálculo que, basados en modelos matemáticos del comportamiento de una neurona biológica y a partir de un vector de entrada procedente del exterior o de otras neuronas, proporciona una respuesta determinada [53]. Los elementos que constituyen una neurona artificial se muestran en la Figura 2.2.

Para ser más explícitos, considerando a la $j-$ésima neurona, ésta cuenta con:

- Un vector de entrada $\mathbf{x}$ formado por n elementos, etiquetados como $x_i(t)$, $i=1,2,...,n$. Este vector forma parte del conjunto de vectores representativo del sistema al que se adaptara la neurona o RNA.
- Pesos sinápticos, w_{ji} que representan la intensidad de interacción entre cada neurona presináptica i y la neurona postsináptica j.
- Regla de propagación, $\sigma(w_{ji}, x_i(t))$, define el valor del potencial postsináptico de la neurona i en función de sus pesos y entradas.
- Función de activación, $f_j(a_j(t-1), h_j(t))$ que proporciona el estado de activación actual $a_j(t)$ de la neurona j, en función de su estado anterior $a_j(t-1)$ y de su potencial postsináptico actual.
- Función de salida $F_j(a_j(t))$, que proporciona la salida actual $y_j(t)$ de la neurona j en función de su estado actual.

Las **variables de entrada y salida** pueden ser binarias (digitales) o continuas (analógicas), dependiendo del modelo y aplicación. Al conjunto de p elementos, formados por una entrada y su correspondiente salida, que definen el comportamiento del sistema al cual será adaptada la RNA suele denominársele conjunto fundamental de asociaciones, que está definido por

$$\{(\mathbf{x}^1,\mathbf{y}^1),(\mathbf{x}^2,\mathbf{y}^2),...,(\mathbf{x}^p,\mathbf{y}^p)\}=\{(\mathbf{x}^\mu,\mathbf{y}^\mu)\mid \mu=1,2,...,p\} \tag{2.1}$$

donde $\mathbf{x}^\mu=[x_i^\mu]_n$ y $\mathbf{y}^\mu=[y_j^\mu]_m$. Para el caso de una neurona y_j^μ.

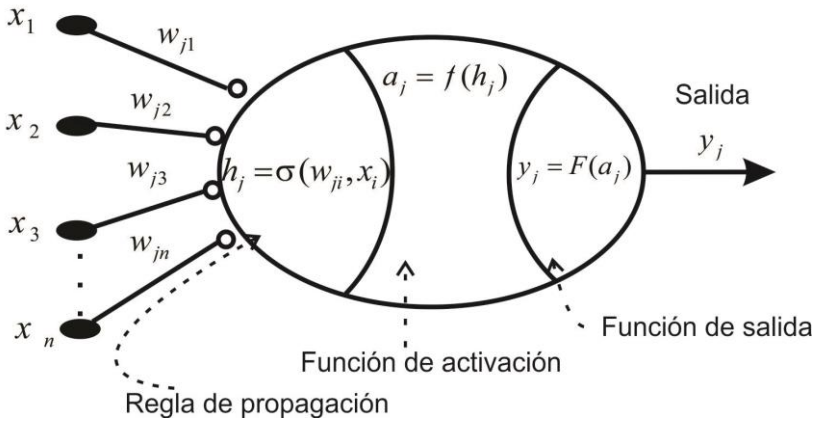

Figura 2.2 Modelo de Neurona Artificial [53].

El **peso sináptico** w_{ji} define la intensidad de interacción entre la neurona presináptica i y la postsináptica j. Dada una entrada positiva, si el peso es positivo tenderá a excitar a la neurona postsináptica, si el peso es negativo tenderá a inhibirla.

La **regla de propagación** permite obtener, a partir de las entradas y los pesos sinápticos, el valor del potencial postsináptico h_j de la neurona

$$h_j(t) = \sigma_j\left(w_{ji}, x_i(t)\right) \tag{2.2}$$

La función más habitual es de tipo lineal, y se basa en la sumatoria de las entradas ponderadas por los pesos sinápticos

$$h_j(t) = \sum_i w_{ji} x_i \tag{2.3}$$

que formalmente también pueden interpretarse como el producto escalar de los vectores de entrada y pesos sinápticos

$$h_j(t) = \sum_i w_{ji} x_i = \mathbf{w}_j^T \cdot \mathbf{x} \tag{2.4}$$

Otra regla de propagación para los modelos de RNA basados en el cálculo de distancias entre vectores como los mapas de Kohonen, es la distancia euclídea, que representa la distancia entre los vectores de entrada y los pesos.

$$h_j{}^2(t) = \sum_i \left(x_i - w_{ji}\right)^2 \tag{2.5}$$

La **función de activación o de transferencia** define el estado actual de la neurona $a_j(t)$ a partir del potencial postsináptico $h_j(t)$ y del propio estado anterior $a_j(t-1)$

$$a_j(t) = f_j\left(a_j(t-1), h_j(t)\right) \tag{2.6}$$

Sin embargo, en muchos modelos de RNA se considera que el estado actual de la neurona no depende de su estado anterior, sino únicamente del actual

$$a_j(t) = f_j\left(h_j(t)\right) \tag{2.7}$$

Algunas de las funciones de activación, $y = f(x)$, comúnmente empleadas en las RNA se muestra en la Tabla 2.1.

Tabla 2.1 Funciones de activación habituales [53].

	Función	Rango
Identidad	$y = x$	[-α,+ α]
Escalón	$y = signo(x)$ $y = H(x)$	{-1,+1} {0,+1}
Lineal a tramos	$y = \begin{cases} -1, & si \quad x < -l \\ x, & si \quad l \le x \le -1 \\ 1, & si \quad x > +l \end{cases}$	[-1,+1]
Sigmoidea	$y = \frac{1}{1+e^{-x}}$ $y = tgh(x)$	[0,+1] [-1,+1]
Gaussiana	$y = A \cdot e^{-Bx^2}$	[0,+1]
Sinusoidal	$y = A \cdot sen(wx + \varphi)$	[-1,+1]

La **función de salida** proporciona la salida global de la neurona $y_j(t)$ en función de su estado actual $a_j(t)$. Muy frecuentemente la función de salida es simplemente la identidad $F(t)=x$, de modo que el estado de activación de la neurona se considera como la propia salida

$$y_j(t)=F_j\left(a_j(t)\right)=a_j(t) \tag{2.8}$$

De este modo, la operación de la j-ésima neurona puede expresarse como

$$y_j=F_j\Big(\ f_j\left[a_j(t-1),\sigma_j\left(w_{ji},x_i(t)\right)\right]\ \Big) \tag{2.9}$$

El modelo expuesto de neurona artificial resulta ser muy general, en la práctica se suele utilizar uno más estándar, considerando que la regla de propagación consiste en la sumatoria de las entradas ponderadas y que la función de salida es la identidad, la neurona estándar queda definida como [53]:

- Conjunto de entradas $x_i(t)$ y pesos sinápticos w_{ji}.
- Regla de propagación $h_j(t)=\sigma\left(w_{ji},x_i(t)\right)$; $h_j(t)=\sum w_{ji}x_i$ es la más común**.**
- Una función de activación $y_j(t)=f_j\left(h_j(t)\right)$, que representa simultáneamente la salida de la neurona y su función de activación.

En la Figura 2.3 se muestra el modelo de la neurona artificial estándar.

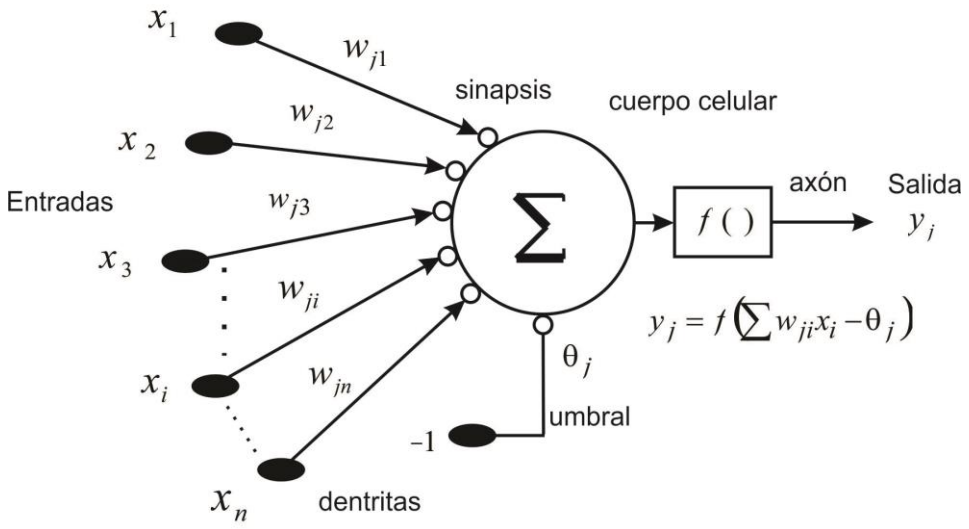

Figura 2.3 Modelo de Neurona Artificial Estándar [53].

Con frecuencia se añade al conjunto de pesos de la neurona un parámetro adicional θ_j, denominado umbra, el cual se resta del potencial postsináptico, por lo que el argumento de la función de activación queda

$$\sum w_{ji} x_i - \theta_j \tag{2.10}$$

En algunos casos este parámetro representa el umbral de disparo de la neurona, es decir, el nivel mínimo que debe alcanzar el potencial postsináptico para que la neurona se dispare o active.

En conclusión, el modelo de esta neurona queda definido como

$$y_j(t) = f_j\left(\sum_i w_{ji} x_i - \theta_j\right) \tag{2.11}$$

Si los índices i y j comienzan en 0, podemos definir $w_{j0} = \theta_j$ y $x_0 = -1$, con lo que el potencia postsináptico se obtiene realizando la suma desde $i = 0$

$$y_j(t) = f_j\left(\sum_{i=0}^{n} w_{ji} x_i\right) \tag{2.12}$$

2.1.3 Definición de Redes Neuronales Artificiales

En este apartado se ha hecho un compendio de argumentos donde se especifica el concepto de RNA y/o se define su estructura.

De acuerdo a respetables investigadores en el área de las RNAs, como Hecht-Nielsen, Caudill, Butler, Feldman, Ballard, Rumelhart, Hinton, Williams y Kohonen, una RNA es un sistema de procesamiento de información que consta de un gran número de unidades simples de procesamiento, altamente interconectadas y jerarquizadas en capas o niveles, capaz de adaptarse a diversas aplicaciones para responder dinámicamente a estímulos externos [33], [15], [21], [72], [46].

De acuerdo con Stergiou y Siganos una RNA es un paradigma utilizado como un sistema de procesamiento de información inspirado en el sistema nervioso biológico, que consiste en un número de elementos de procesamiento interconectados para resolver un problema específico [40].

Por su parte, Krös, y Van Der Smagt argumentan que una RNA consiste en un grupo de neuronas artificiales, elementos de procesamiento, que se comunican entre sí mediante el envío de señales a través de un gran número de conexiones ponderadas [49].

Yegnanarayana define a una RNA como un sistema que consta de unidades de procesamiento (o neuronas artificiales) interconectadas de manera predeterminada para realizar una tarea deseada [82].

La definición presentada por Martín, B.B. y Sanz, M.A es la siguiente: un sistema neuronal artificial es aquel que tiene como elemento esencial de su estructura básica a una neurona artificial y que se organiza en capas; varias capas constituirán una red neuronal; y, por último, una red neuronal (o un conjunto de ellas), junto con las interfaces de entrada y salida, más los módulos convencionales adicionales necesarios, constituirán el sistema global de proceso. Este sistema neuronal se adapta a aplicaciones específicas mediante el proceso de aprendizaje [53].

Finalmente, desde el punto de vista de Rumelhart, McClelland y Hinton, miembros del PDP (*Parallel Distributed Processing Research Group*) de la Universidad de San Diego, un sistema neuronal o conexionista, está compuesto por los siguientes elementos [72]:

- Un conjunto de procesadores elementales o neuronas artificiales.
- Arquitectura de la RNA. Estructura o patrón de conexiones entre las neuronas artificiales.
- Una dinámica de cómputo, que expresa el valor que toman las neuronas artificiales, y que se basa en las funciones de activación (o de transferencia) de la neurona, las cuales especifican como se transforman las señales de entrada de la unidad de proceso en la señal de salida.
- Una regla o dinámica de aprendizaje. Procedimiento para determinar el valor de los pesos sinápticos
- El entorno donde opera, definido por el conjunto de entrenamiento.

2.1.4 Arquitectura de las Redes Neuronales Artificiales

Se denomina arquitectura de una RNA a la topología, estructura o patrón de conexiones que guardan las neuronas que la integran. Es decir, la arquitectura define la organización y forma

en que están conectadas las neuronas que forman a la RNA y depende de los requerimientos de la aplicación para la que es diseñada.

Las neuronas se suelen agrupar en unidades estructurales denominadas capas, el conjunto de una o más capas constituyen la RNA [53]. En este sentido, es posible diferenciar tres tipos de capas en los modelos de RNA:

- Entrada. Una capa de entrada, denominada sensorial, está compuesta por neuronas que reciben datos o señales procedentes del entorno.
- Salida. Una capa de salida es aquélla que contiene las neuronas que proporcionan la respuesta de la red neuronal.
- Oculta. Una capa oculta es aquella que no tiene una conexión directa con el entorno, es decir, no se conecta directamente ni a elementos sensores ni a efectores.

Además, la arquitectura de una red neuronal principalmente está definida por el número de capas ocultas, numero de neuronas en las capas ocultas, número de neuronas en la salida y la función de activación de las neuronas [50]; donde, algunas generalidades de estos elementos son:

- Capas ocultas: Las capas ocultas son requeridas principalmente para resolver problemas no-lineales.
- Número de nodos en la capa oculta: No existe una regla para seleccionar el número de neuronas en la capa oculta.
- Número de nodos de salida: Las redes neuronales con múltiples salidas
- Función de activación: La función de activación es matemáticamente formulada para determinar la respuesta de una neurona y frecuentemente es deseable que las neuronas de la capa oculta tengan funciones de activación no-lineales y diferenciables.

En la RNA, las neuronas se interconectan entre sí por medio de conexiones sinápticas, que definen el comportamiento de la red. Las conexiones entre las neuronas de las capas de la RNA pueden ser excitatorias o inhibitorias; un peso sináptico negativo define una conexión inhibitoria, mientras que uno positivo determina una conexión excitatoria.

En relación a la estructura de sus capas, las RNAs se clasifican en redes monocapa y redes multicapa.

Las redes monocapa son las más sencilla ya que tiene una capa de neuronas que proyectan las entradas a una capa de neuronas de salida donde se realizan diferentes cálculos, en la Figura 2.4 se muestran ejemplos de esta categoría.

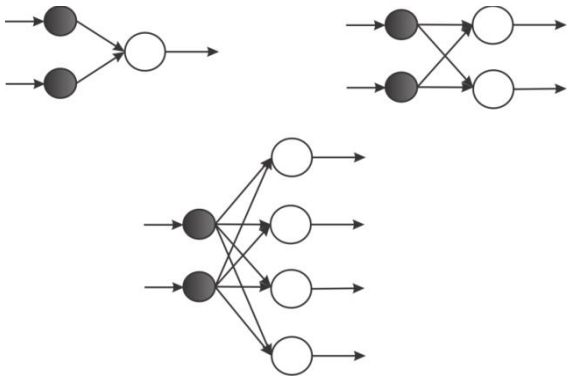

Figura 2.4 RNA monocapa.

Las redes multicapa son aquellas donde las neuronas se organizan en varias capas, es una generalización de la anterior existiendo un conjunto de capas intermedias entre la entrada y la salida (capas ocultas), en la Figura 2.5 se muestran ejemplos de estas redes.

Por otro lado, considerando el flujo de datos, las RNAs se pueden clasificar en redes no recurrentes (*feedforward*) y en redes recurrentes (*feedback*). En las redes no recurrentes, la información circula en un solo sentido, desde las neuronas de entrada hacia las neuronas de salida; ejemplos de este tipo de RNA se muestran en las Figura 2.4 y Figura 2.5. En las redes recurrentes o realimentadas la información puede ir entre las capas en cualquier sentido, incluido el de salida-entrada; la Figura 2.6 muestra ejemplos de RNA recurrentes.

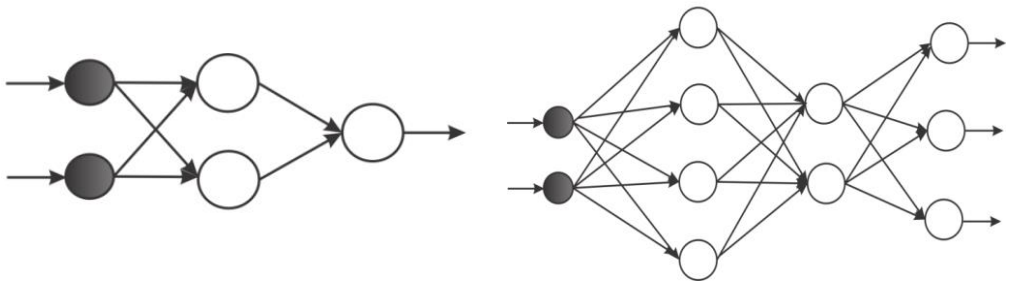

Figura 2.5 RNA multicapa.

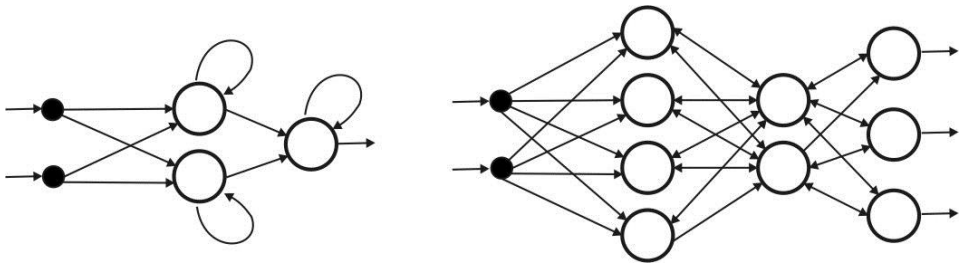

Figura 2.6 RNA recurrentes.

2.1.5 Algoritmo de entrenamiento o aprendizaje

Como se ha mencionado, el campo de aplicación de las RNA es basto, esto se debe especialmente a una de sus principales características, su capacidad para aprender interactuando con su entorno o con alguna fuente de información. Esta función es cumplida por su mecanismo de aprendizaje o algoritmo de entrenamiento, el cual es un proceso adaptativo que modifica los pesos sinápticos de la red buscando mejorar su comportamiento y/o adaptarlo a una aplicación específica [48].

De manera general, una RNA modifica su peso sináptico w_{ji} correspondiente a la conexión entre la neurona presináptica i y la neurona postsináptica j mediante la regla de aprendizaje de la forma.

$$w_{ji}(k+1) = w_{ji}(k) + \Delta w_{ji}(k) \tag{2.13}$$

El proceso de aprendizaje se considera terminado cuando los pesos sinápticos permanecen estables

$$\frac{\partial w_{ji}}{\partial t} = 0 \tag{2.14}$$

Los cambios producidos por el proceso de aprendizaje son, la creación de conexiones entre neuronas, la modificación de conexiones entre neuronas, la destrucción de conexiones entre neuronas y la destrucción de neuronas.

Algunos paradigmas referentes al aprendizaje de RNA son:

- Aprendizaje supervisado
- Aprendizaje no supervisado
- Aprendizaje por refuerzo

El proceso de aprendizaje supervisado requiere un administrador que presente el conjunto de entrenamiento que caracteriza la aplicación para determinar las respuestas que deben ser generadas por la RNA a partir de las entradas. En la Figura 2.7 se muestra un diagrama a bloques de un proceso de aprendizaje supervisado.

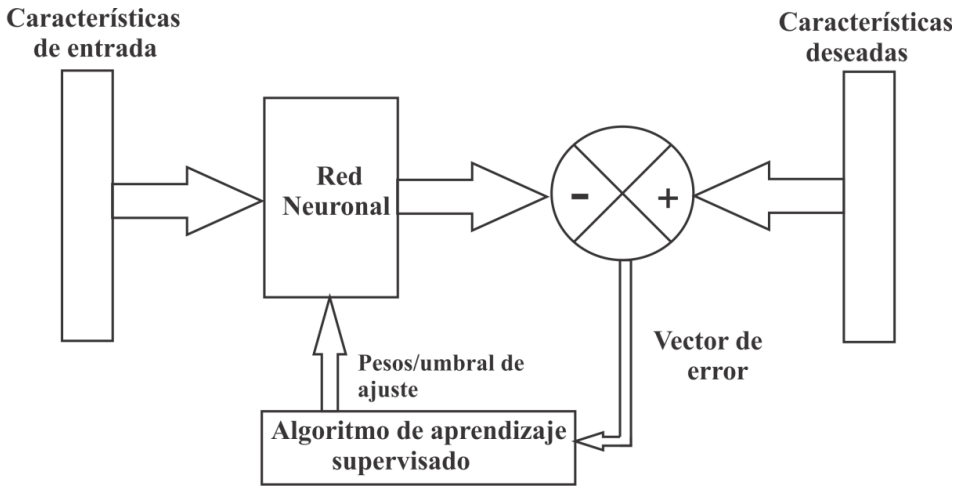

Figura 2.7 Proceso de aprendizaje supervisado [48].

El proceso de aprendizaje no supervisado es requerido en muchos problemas de reconocimiento, cuando el patrón destino es desconocido, intenta generar un conjunto único de pesos para una clase particular de patrones. Por lo tanto, asigna una clase de objetos a una clase de pesos. La RNA sólo dispone de las entradas, así, el algoritmo aprende a categorizar las entradas (*clustering*). El algoritmo está constituido por un número de reglas que proporcionan a la red facilidad de aprender el comportamiento más adecuado, atendiendo ciertos criterios. Las redes neuronales *feedforward* más populares con un entrenamiento sin supervisión son los SOM's o "mapas de Kohonen" [47].

El aprendizaje por refuerzo puede ser considerado como una forma intermedia de los dos tipos anteriores de aprendizaje. Aquí, la máquina de aprendizaje hace alguna acción sobre el medio y crear una respuesta de retroalimentación de éste. Generalmente, se continúa el ajuste de parámetros hasta que se produce un estado de equilibrio, después de lo cual no habrá más cambios en sus parámetros.

Es posible hacer una clasificación de las RNA según sea el modelo de la neurona que se utilice, la arquitectura o topología de conexión y el algoritmo de aprendizaje. En la Figura 2.8 se muestra una clasificación de las RNA [53].

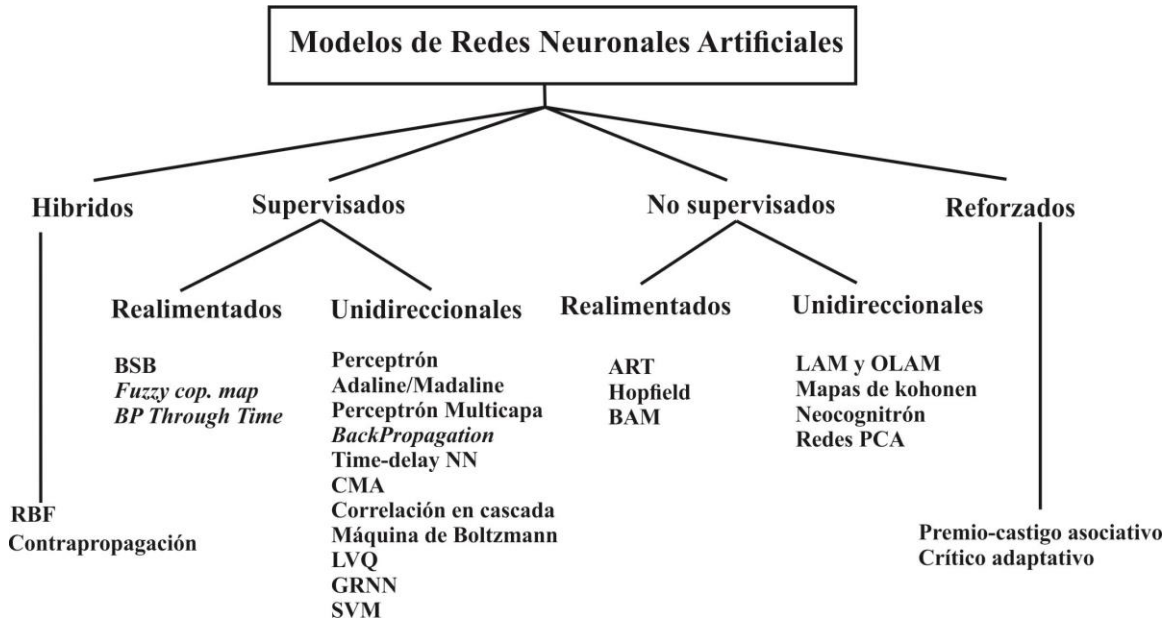

Figura 2.8 Clasificación de las RNA's por el tipo de aprendizaje y la arquitectura [53].

Como comentario final del capítulo, cabe hacer mención que, aunque las RNA’s están inspiradas en la neurociencia no pretenden ser buenos modelos neuronales biológicos, lo que se busca sobre todo es conseguir o emular la capacidad computacional o de cálculo de una red neuronal biológica.

Capítulo 3 Métodos y tecnología utilizada

Este capítulo versa sobre el modelo neuronal MLP, el algoritmo de aprendizaje *backpropagation*, y la lógica reconfigurable, todos ellos conforman los métodos y tecnologías utilizadas en este trabajo de tesis. El capítulo también incluye la justificación de la implementación de una RNA sobre lógica reconfigurable. El capítulo finaliza con un estado del arte de trabajos relacionados con la implementación de RNA's sobre lógica reconfigurable.

3.1 Perceptrón multicapa y Backpropagation

Si se añaden capas intermedias (ocultas) a un perceptrón simple, se obtiene un MLP. El MLP, es un sistema complejo capaz de modelar las relaciones existentes entre las variables de entrada y salida de un sistema, permite predecir la salida en base a los objetos de entrada, por lo que resulta ser muy popular y utilizado para diversas tareas. La red neuronal MLP es no recurrente, donde elementos no-lineales (neuronas) son organizados en capas sucesivas y la información fluye de la capa de entrada a la capa de salida a través de las capas ocultas. Kumar, G.J. menciona como características principales de un MLP las siguientes [50]:

- El número de entradas es variado.
- Generalmente una capa de entrada no es considerada una capa de la RNA desde que su función se limita a canalizar los valores del vector de entrada a todas y cada una de las neuronas de la primera capa oculta.
- Cuenta con una o más capas ocultas las cuales a su vez cuentan con un número variado de neuronas.
- Las neuronas de las capas ocultas generalmente usan funciones de activación no-lineales (típicamente sigmoideas).
- La capa de salida tiene un número variable de neuronas que pueden incluir funciones de activación lineales (para problemas de regresión) o funciones de activación no-lineales (para tareas de clasificación o reconocimiento).

Considerando lo anterior, se puede definir la arquitectura de un RNA-MLP. La Figura 3.1 muestra un MLP que tiene r capas ocultas, donde:

$\mathbf{x}^{\mu} = \left[x_i^{\mu}\right]_n$ es el $\mu-$ésimo vector de entrada, perteneciente al conjunto fundamental de asociaciones definido en la expresión 2.1

$\left(\mathbf{y}^{l}\right)^{\mu} = \left[\left(y_{j^l}^{l}\right)^{\mu}\right]_{m^l}$ representa la salida de la $l-$ésima capa oculta cuando el $\mu-$ésimo vector de entrada es presentado a la red; donde $l = [1,2,...,r]$ y m^l es variable para cada capa e indica el número de neuronas de la capa

$\mathbf{z}^{\mu} = \left[z_k^{\mu}\right]_q$ representa la salida generada por el MLP cuando el $\mu-$ésimo vector de entrada es presentado a la red; q indica el número de neuronas de la capa de salida

$\mathbf{t}^{\mu} = \left[t_k^{\mu}\right]_q$ representa la salida que la red debe generar cuando el $\mu-$ésimo vector de entrada es presentado

$\mathbf{W}^1 = \left[w_{j^1 i}^1\right]_{m^1 \times n}$ son los pesos sinápticos de la primer capa oculta y $\boldsymbol{\theta}^1 = \left[\theta_{j^1}^1\right]_{m^1}$ sus umbrales

$\mathbf{W}^l = \left[w_{j^l j^{l-1}}^l\right]_{m^l \times m^{l-1}}$ son los pesos sinápticos de las capas ocultas restantes y $\boldsymbol{\theta}^l = \left[\theta_{j^l}^l\right]_{m^l}$ sus umbrales; para $l = [2,...,r]$

$\mathbf{W}^{r+1} = \left[w_{kj^r}^{r+1} \right]_{q \times m^r}$ son los pesos sinápticos de la capa de salida y $\boldsymbol{\theta}^{r+1} = \left[\theta_k^{r+1} \right]_q$ sus umbrales

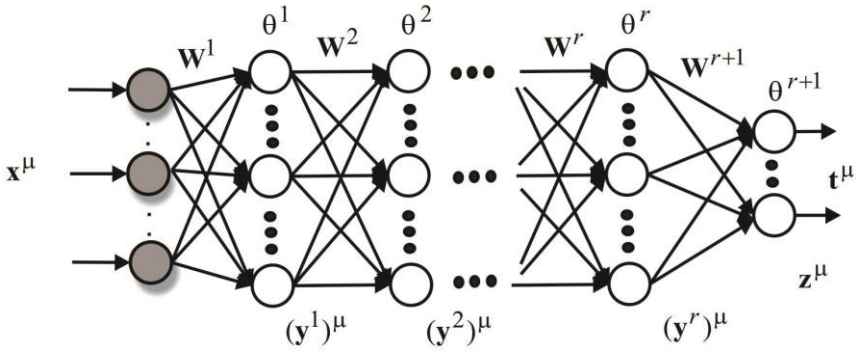

Figura 3. 1. Arquitectura general de una RNA-MLP.

Con la finalidad de evitar confusiones y clarificar la interpretación de los índices involucrados en las definiciones anteriores, se debe considerar lo siguiente a lo largo del trabajo: $w_{j^l j^{l-1}}^{l}$ se interpreta como el j^{l-1}–ésimo peso sináptico de la j^l–ésima neurona de la capa oculta, o de salida, l; y $\theta_{j^l}^{l}$ como el umbral de j^l–ésima neurona de la capa oculta, o de salida, l.

Se puede ahora definir la operación del MLP con r capas ocultas. Entonces, la salida de la primera capa oculta es expresada matemáticamente de la siguiente manera:

$$y_{j^1}^{1} = f\left(\sum_i w_{j^1 i}^{1} \cdot x_i - \theta_{j^1}^{1} \right) \tag{3.1}$$

Para la segunda capa oculta

$$\begin{aligned} y_{j^2}^{2} &= f\left(\sum_{j^1} w_{j^2 j^1}^{2} \left(f\left(\sum_i w_{j^1 i}^{1} \cdot x_i - \theta_{j^1}^{1} \right) \right) - \theta_{j^2}^{2} \right) \\ y_{j^2}^{2} &= f\left(\sum_{j^1} w_{j^2 j^1}^{2} \cdot y_{j^1}^{1} - \theta_{j^2}^{2} \right) \end{aligned} \tag{3.2}$$

Para la r–ésima capa oculta

$$y_{j^r}^{r} = f\left(\sum_{j^{r-1}} w_{j^r j^{r-1}}^{r} \cdot y_{j^{r-1}}^{r-1} - \theta_{j^r}^{r} \right) \tag{3.3}$$

Finalmente, la operación de la red queda definida por:

$$z_k = g\left(\sum_{j^r} w_{kj^r}^{r+1} \cdot y_{j^r}^{r} - \theta_k^{r+1} \right) \tag{3.4}$$

Es necesario resaltar el hecho de que no basta con agregar capas ocultas a una RNA para que su desempeño sea mejorado, algo que repercute más en este aspecto es el incluir funciones de activación no-lineales y diferenciables substituyendo a funcione de activación de umbral, típicas en un perceptron.

De acuerdo a lo mencionado, en un MLP las funciones de activación de las capas ocultas, $f(\cdot)$, son no-lineales, típicamente sigmoideas; por ejemplo, la función logística (ecuación 3.5), y la función tangente hiperbólica (ecuación 3.6).

$$f(x) = \frac{1}{1+e^{-x}} \tag{3.5}$$

$$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{3.6}$$

Mientras que las funciones de activación de las capas de salida, $g(\cdot)$, pueden ser lineales o no-lineales, dependiendo de la aplicación.

Por otro lado, al considerar la fase de entrenamiento o aprendizaje del MLP, si se utiliza el algoritmo del perceptrón simple para llevar a cabo este proceso, este algoritmo únicamente permitirá entrenar a las neuronas en forma individual.

El algoritmo denominado *Backpropagation* resuelve este problema, el cual es denominado de asignación de crédito y se basa en la contribución del error en la salida de la red neuronal de cada uno de los nodos ocultos.

Este método de aprendizaje ha sido propuesto en diferentes formas por varios investigadores, de acuerdo a R. Hecht-Nielsen [34] y a Gori-Tesi [30], entre otros, el origen del algoritmo EBP se relaciona con el campo de la teoría de control óptimo cuando en 1969 Bryson y Ho desarrollaron un algoritmo, muy similar al del EBP, para un control no-lineal adaptativo [14]. Posteriormente en 1971, en forma independiente, fue re-descubierto por Werbos cuando desarrolló un algoritmo de entrenamiento de retropropagación, el cual publicó por primera vez en su tesis doctoral [78]. El trabajo de Werbos no fue apreciado hasta que en 1982 Parker re-

descubrió la técnica [67] y en 1985 escribió un reporte referente a este trabajo [68] cuando laboraba en el MIT.

Finalmente, en el año 1986 este algoritmo fue formalizado por el grupo PDP, integrado por Rumelhart, Hinton y Williams, como un método de aprendizaje global para una RNA tipo MLP [72].

Otro trabajo de importancia relacionado con el algoritmo EBP fue el propuesto por Le Cun en 1988 [51], el cual está basado en el trabajo de Bryson y Ho.

El algoritmo EBP es la solución al problema de entrenar los nodos de las capas ocultas de un MLP, la idea básica de este algoritmo plantea que la actualización de los pesos sinápticos de las neuronas de una capa depende del error cometido por ésta y por el error cometido por todas las capas que le siguen. Es decir,

- El error de la capa de salida (salida de la red) influye en la actualización de los pesos sinápticos de todas las capas ocultas y de los de ella misma.
- El error cometido por la última capa oculta influye en la actualización de los pesos sinápticos de todas las capas ocultas que le preceden y de los de ella misma.
- El error cometido por la penúltima capa oculta influye en la actualización de los pesos sinápticos de todas las capas ocultas que le preceden y de los de ella misma.
- Y así sucesivamente.

Se procede al análisis del EBP, para lo cual se considera un MLP de una capa oculta, cuya arquitectura se presenta en la Figura 3.2.

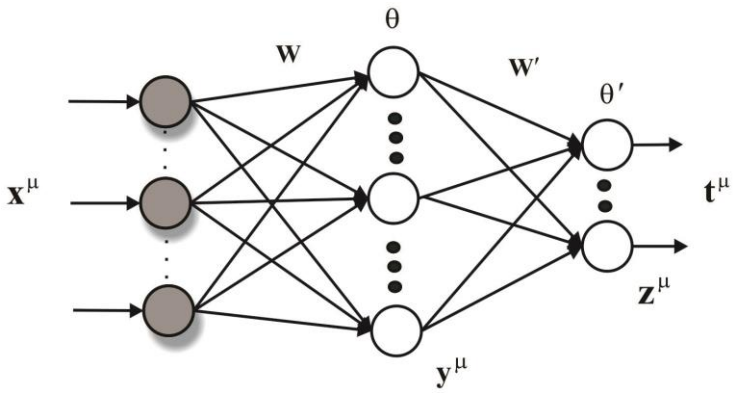

Figura 3.2. Arquitectura del MLP de tres capas.

Para facilitar el análisis, y tomando en cuenta los parámetros de la Figura 3.1, para el caso del MLP de la Figura 3.2 se hacen las siguientes consideraciones:

- Como existe sólo una capa oculta, entonces $l=1$ y $\left(\mathbf{y}^{l}\right)^{\mu}=\left[\left(y_{j^{l}}^{l}\right)^{\mu}\right]_{m^{l}}$ es simplemente $\mathbf{y}^{\mu}=\left[y_{j}^{\mu}\right]_{m}$.
- Los pesos sinápticos y umbrales de la capa oculta, $\mathbf{W}^{1}=\left[w_{j^{1}i}^{1}\right]_{m^{1}\times n}$ y $\boldsymbol{\theta}^{1}=\left[\theta_{j^{1}}^{1}\right]_{m^{1}}$, son representados como $\mathbf{W}=\left[w_{ji}\right]_{m\times n}$ y $\boldsymbol{\theta}=\left[\theta_{j}\right]_{m}$.
- Los pesos sinápticos y umbrales de la capa de salida, $\mathbf{W}^{2}=\left[w_{kj^{1}}^{2}\right]_{q\times m^{1}}$ y $\boldsymbol{\theta}^{2}=\left[\theta_{k}^{2}\right]_{q}$, son representados como $\mathbf{W}'=\left[w_{kj}'\right]_{q\times m}$ y $\boldsymbol{\theta}'=\left[\theta_{k}'\right]_{q}$.
- Los demás elementos de la red, $\mathbf{x}^{\mu}=\left[x_{i}^{\mu}\right]_{n}$, $\mathbf{z}^{\mu}=\left[z_{k}^{\mu}\right]_{q}$, y $\mathbf{t}^{\mu}=\left[t_{k}^{\mu}\right]_{q}$, son representados de la misma manera.

Para el inicio del análisis y con base en la Ecuación 3.4, se determina la operación global de la red para el patrón de entrada $\mathbf{x}^{\mu},(\mu=1,...,p)$,

$$\begin{aligned} z_{k}^{\mu} &= g\left(\sum_{j} w_{kj}'\cdot y_{j}^{\mu}-\theta_{k}'\right) \\ &= g\left(\sum_{j} w_{kj}' f\left(\sum_{i} w_{ji}\cdot x_{i}^{\mu}-\theta_{j}\right)-\theta_{k}'\right) \\ &= g\left(\mathbf{w}_{k}' f\left(\mathbf{w}_{j}\cdot\mathbf{x}^{\mu}-\theta_{j}\right)-\theta_{k}'\right) \end{aligned} \tag{3.7}$$

Ahora, se considera al error cuadrático medio como función de coste

$$\begin{aligned} E(w_{ji},\theta_{j},w_{kj}',\theta_{k}') &= \frac{1}{2}\sum_{\mu}\sum_{k}\left(t_{k}^{\mu}-z_{k}^{\mu}\right)^{2} \\ &= \frac{1}{2}\sum_{\mu}\sum_{k}\left(t_{k}^{\mu}-g\left(\sum_{j} w_{kj}'\cdot y_{j}^{\mu}-\theta_{k}'\right)\right)^{2} \end{aligned} \tag{3.8}$$

La minimización de esta función se lleva a cabo mediante el descenso por el gradiente.

Para esta caso, habrá un gradiente respecto de los pesos de la capa de salida w'_{kj} y otro respecto de la oculta w_{ji}

$$\Delta w'_{kj} = -\varepsilon \frac{\partial E}{\partial w'_{kj}}; \qquad \Delta w_{ji} = -\varepsilon \frac{\partial E}{\partial w_{ji}} \tag{3.9}$$

Aplicando la regla de la cadena con respecto a w_{ij} y w'_{kj}

$$\frac{\partial E}{\partial w_{ji}} = \frac{\partial E}{\partial u_j}\frac{\partial u_j}{\partial w_{ji}} \quad \text{y} \quad \frac{\partial E}{\partial w'_{kj}} = \frac{\partial E}{\partial u'_k}\frac{\partial u'_k}{\partial w'_{kj}}$$

donde

$$u_j = \sum_i w_{ji} x_i^{\mu} - \theta_j \quad \text{y} \quad u'_k = \sum_j w'_{kj} y_j^{\mu} - \theta'_k \tag{3.10}$$

Ahora tomando en cuenta que

$$\frac{\partial u_j}{\partial w_{ji}} = x_i, \qquad \frac{\partial u'_k}{\partial w'_{kj}} = y_j$$

y sabiendo que la señal del error de la capa oculta y de la de salida están definidas por

$$\delta y_j = -\frac{\partial E}{\partial u_j}, \qquad \delta z_k = -\frac{\partial E}{\partial u'_k}$$

Entonces, el ajuste para la capa oculta y la capa de salida respectivamente está dado por

$$\Delta w_{ji} = \varepsilon \cdot \delta y_j \cdot x_i \tag{3.11}$$

$$\Delta w'_{kj} = \varepsilon \cdot \delta z_k \cdot y_j \tag{3.12}$$

Se procede a determinar las señales de error δz_k y δy_j. Se inicia con la capa oculta, al aplicar la regla de la cadena, la señal de error queda definida como

$$\delta z_k = \frac{\partial E}{\partial u'_k} = \frac{\partial E}{\partial z_k}\frac{\partial z_k}{\partial w'_{kj}} = (t_k - z_k) g'(u'_k) \tag{3.13}$$

por lo tanto el ajuste de los pesos sinápticos de la capa de salida quedan definidos como

$$\begin{aligned} w'_{kj} &= w'_{kj} + \Delta w'_{kj} \\ &= w'_{kj} + \varepsilon \cdot \delta z_k \cdot y_j \\ &= w'_{kj} + \varepsilon (t_k - z_k) g'(u'_k) y_j \end{aligned} \tag{3.14}$$

Y en notación vectorial queda definida como

$$\mathbf{W}' = \mathbf{W}' + \eta \cdot \boldsymbol{\delta}_z \cdot \mathbf{y}^T \tag{3.15}$$

donde $\mathbf{W}' = \left[w'_{kj} \right]_{q \times m}$, $\mathbf{y} = \left[y_j \right]_m$, $\boldsymbol{\delta}_\mathbf{z} = \left[\delta z_k \right]_q$

El siguiente paso es calcular el término señal de error de la capa oculta $\delta y_j = -\frac{\partial E}{\partial u_j}$, este paso es uno de los más importantes de la regla delta generalizada; se procede a aplicar la regla de la cadena sobre esta señal,

$$\delta y_j = -\frac{\partial E}{\partial y_j} \frac{\partial y_j}{\partial u_j} \tag{3.16}$$

Se observa que el segundo término corresponde a la derivada de las salidas, es decir de la función de activación, de las neuronas de la capa oculta, $f(\cdot)$

$$\frac{\partial y_j}{\partial u_j} = f'_j(u_j) \tag{3.17}$$

Por otro lado, aplicando el segundo término, $\frac{\partial E}{\partial y_j}$, sobre la función de coste de la Ecuación 3.8, y considerando la Ecuación 3.10

$$\begin{aligned} \frac{\partial E}{\partial y_j} &= \frac{\partial}{\partial y_j} \left(\frac{1}{2} \sum_{\mu=1}^{p} \sum_{k=1}^{q} \left(t_k^\mu - g\left(\sum_{j=1}^{m} w'_{kj} \cdot y_j^\mu - \theta'_k \right) \right)^2 \right) \\ &= \frac{\partial}{\partial y_j} \left(\frac{1}{2} \sum_{k=1}^{q} (t_k - g(u'_k))^2 \right) \\ &= -\sum_{k=1}^{q} (t_k - z_k) - \frac{\partial}{\partial y_j} (g(u'_k)) \\ &= -\sum_{k=1}^{q} \underbrace{(t_k - z_k) g'(u'_k)}_{\delta z_k} \frac{\partial u'_k}{\partial y_j} \end{aligned} \tag{3.18}$$

Ahora como $u'_k = w'_{k1}y_1 + w'_{k2}y_2 + ... + w'_{kj}y_j + ... + w'_{kq}y_q$ entonces $\frac{\partial u'_k}{\partial y_j} = w'_{kj}$, por lo que

$$\frac{\partial E}{\partial y_j} = -\sum_{k=1}^{q} \delta z_k \cdot w'_{kj} \tag{3.19}$$

Sustituyendo las ecuaciones 3.19 y 3.17 en 3.16 obtenemos

$$\delta y_j = f'_j(u_j)\sum_{k=1}^{q} \delta z_k \cdot w'_{kj} \tag{3.20}$$

Finalmente sustituyendo esta ecuación en la ecuación 3.11

$$\Delta w_{ji} = \varepsilon \cdot x_i \cdot f'_j(u_j)\sum_{k=1}^{q} \delta z_k \cdot w'_{kj} \tag{3.21}$$

La ecuación 3.21 es la más importante de la generalización de la regla delta, en ella está implícito el aprendizaje (adaptación, cambio, entrenamiento, optimización) de los pesos sinápticos de la capa oculta. Por consiguiente, en cada iteración, el peso sináptico w_{ji} será ajustado mediante

$$\begin{aligned} w_{ji} &= w_{ji} + \Delta w_{ji} \\ &= w_{ji} + \varepsilon \cdot x_i \cdot f'_j(u_j)\sum_{k=1}^{q} \delta z_k \cdot w'_{kj} \\ &= w_{ji} + \varepsilon \cdot x_i \cdot \delta y_j \end{aligned} \tag{3.22}$$

En notación vectorial, la ley de aprendizaje de descenso por el gradiente para la neurona de la capa oculta es

$$\mathbf{W} = \mathbf{W} + \eta \cdot \boldsymbol{\delta}_{\mathrm{y}} \cdot \mathbf{x}^T \tag{3.23}$$

donde $\mathbf{W} = [w_{ji}]_{m\times n}$, $\mathbf{x} = [x_i]_n$, $\boldsymbol{\delta}_{\mathbf{y}} = [\delta y_j]_m$

Resumiendo, las expresiones que permiten calcular la actualización de los pesos sinápticos para la capa de salida están definidas por las ecuaciones 3.14, 3.13 y 3.10

$$\begin{aligned} w'_{kj} &= w'_{kj} + \Delta w'_{kj} = w'_{kj} + \varepsilon \cdot \delta z_k \cdot y_j \\ \delta z_k &= (t_k - z_k)g'(u'_k) \\ u'_k &= \sum_j w'_{kj} y_j - \theta'_k \end{aligned}$$

Y para la capa oculta por las ecuaciones 3.22, 3.20 y 3.10

$$w_{ji} = w_{ji} + \Delta w_{ji} = w_{ji} + \varepsilon \cdot \delta y_j \cdot x_i$$

$$\delta y_j = f'_j(u_j) \sum_{k=1}^{q} \delta z_k \cdot w'_{kj}$$

$$u_j = \sum_i w_{ji} x_i - \theta_j$$

Se observa que en estas expresiones está implícito el concepto de propagación hacia atrás de los errores, esto debido a

1. Se calcula la expresión δz_k, denominada señal de error, que es proporcional al error de la salida actual de la red.
   - Con este error, se calcula la actualización $\Delta w'_{kj}$ de los pesos de la capa de salida.
2. Se propagan hacia atrás los errores δz_k a través de las sinapsis, generando así las señales de error δy_j, correspondientes a las sinapsis de la capa oculta.
   - Con estas se calcula la actualización Δw_{ji} de las sinapsis ocultas.

La actualización de los umbrales, se realiza considerándolos un caso particular de peso sináptico cuya entrada es igual a -1.

Siguiendo el mismo desarrollo, el algoritmo puede ser extendido fácilmente a una arquitectura de r capas ocultas, como la de la Figura 3.1. De esta forma, las expresiones obtenidas para la arquitectura mencionada son las siguientes

$$\Delta w^{r+1}_{kj^r} = \varepsilon \cdot \delta z_k \cdot y^r_{j^r}, \text{ con } \delta z_k = (t_k - z_k) g'(u^{r+1}_k) \text{ y } u^{r+1}_k = \sum_{j^r} w^{r+1}_{kj^r} y^r_{j^r} - \theta^{r+1}_k$$

$$\Delta w^r_{j^r j^{r-1}} = \varepsilon \cdot \delta y^r_{j^r} \cdot y^{r-1}_{j^{r-1}}, \text{ con } \delta y^r_{j^r} = f'^r_{j^r}\left(u^r_{j^r}\right) \sum_k \delta z_k \cdot w^{r+1}_{kj^r} \text{ y } u^r_{j^r} = \sum_{j^{r-1}} w^r_{j^r j^{r-1}} y^{r-1}_{j^{r-1}} - \theta^r_{j^r}$$

$$\Delta w^2_{j^2 j^1} = \varepsilon \cdot \delta y^2_{j^2} \cdot y^1_{j^1}, \text{ con } \delta y^2_{j^2} = f'^2_{j^2}\left(u^2_{j^2}\right) \sum_{j^3} \delta y^3_{j^3} \cdot w^3_{j^3 j^2} \text{ y } u^2_{j^2} = \sum_{j^1} w^2_{j^2 j^1} y^1_{j^1} - \theta^2_{j^2} \tag{3.24}$$

$$\Delta w^1_{j^1 i} = \varepsilon \cdot \delta y^1_{j^1} \cdot x_i, \text{ con } \delta y^1_{j^1} = f'^1_{j^1}\left(u^1_{j^1}\right) \sum_{j^2} \delta y^2_{j^2} \cdot w^2_{j^2 j^1} \text{ y } u^1_{j^1} = \sum_i w^1_{j^1 i} \cdot x_i - \theta^1_{j^1}$$

Se debe de considerar que en el aprendizaje en serie, el orden en la presentación de los patrones debe ser aleatorio, puesto que, si siempre se sigue el mismo orden el entrenamiento

estaría viciado en favor del último patrón del conjunto de entrenamiento, cuya actualización, por ser la última siempre predominaría sobre las anteriores. Esta aleatoriedad presenta importante ventaja, puesto que en ocasiones permite escapar de mínimos locales alcanzándose mínimos del error más profundos [12], [81].

3.2 RNA's sobre lógica reconfigurable

A la definición de RNA vertida en la sección 2.1.3, se le puede agregar que una RNA presenta un alto grado de concurrencia en los diferentes elementos que la integran. En este sentido, de acuerdo con Hecht-Nielsen [35] y Freman-Skapura [22] "*una RNA es una estructura de procesamiento de información, cuyos elementos se encuentran distribuidos, trabajan en forma paralela y se interconectan entre sí en forma de grafo dirigido*".

Por su parte, Omondi *et al*. [64] mencionan que usualmente suele considerarse a las RNAs como modelos naturales de cómputo paralelo; esto es debido a los diferentes tipos de paralelismo que una RNA exhibe. En este sentido, argumentan que una RNA posee los siguientes tipos de paralelismo

- Paralelismo en la etapa de entrenamiento. Para explotar el paralelismo a nivel de datos que un algoritmo de entrenamiento posee, varias sesiones de éste pueden ser ejecutadas concurrentemente. El paralelismo a este nivel es medio y puede ser implementado en FPGA's.
- Paralelismo a nivel de capa. En una RNA multicapa, las diferentes capas que la integran pueden ser procesadas en paralelo. El paralelismo a este nivel depende del número de capas que integren a la RNA y típicamente es bajo, esté paralelismo puede ser explotado mediante una arquitectura pipeline.
- Paralelismo a nivel de nodo (externo). Se refiere a cuántos nodos o neuronas artificiales pueden estar operando concurrentemente. Este nivel repercute en el paralelismo de los niveles anteriores, por este motivo frecuentemente es considerado el nivel de paralelismo más importante presentado por una RNA, el cual se ve limitado por el número de neuronas que constituyen a la RNA, y puede ser explotado por un FPGA, ya que este cuenta con un número considerables de celdas lógicas, las cuales operan de forma paralela.

- Paralelismo a nivel de nodo (interno). La operación de una neurona artificial involucra operaciones aritméticas de suma y multiplicación, el paralelismo se hace presente cuando algunas de estas operaciones pueden trabajar simultáneamente.
- Paralelismo a nivel de bit. El paralelismo a este nivel ocurre cuando el ancho de palabra con la que opera una unidad de procesamiento es incrementado. Resulta evidente que, al igual que en casi todos los algoritmos, por no decir todos, en una RNA este nivel de paralelismo está presente y depende del diseño de cada una de las unidades funcionales que la integran.

Sin embargo, resulta frecuente que RNAs sean implementadas en procesadores, ya sea en microcontroladores, procesadores digitales de señales (DSP) o computadoras personales (PC), basadas en microprocesadores. Aunque existe un amplio rango de aplicaciones en las que este tipo de arquitectura es adecuada, debido a que la ejecución de los algoritmos implementados en ellas se llevan a cabo en forma secuencial, el paralelismo implícito en una RNA no puede ser aprovechado.

Por otro lado, existen dispositivos paralelos que han sido utilizados en la implementación de RNAs, estos son computadoras paralelas de propósito general, neuro-computadoras, circuitos integrados de aplicación específica (ASICs, *Application-Specific Integrated Circuit*) analógicos, ASICs digitales y FPGAs, los cuales permiten explotar el paralelismo inherente de las RNA's y cuentan con una gran capacidad de cálculo.

Implementaciones de RNA sobre computadoras paralelas de propósito general presentan problemas de conectividad, lo que resulta en costosos intercambios de información. Además, estas arquitecturas son costosas y no pueden ser utilizadas en aplicaciones de sistemas embebidos. Su principal aplicación se centra en algoritmos de entrenamiento.

Las neuro-computadoras son sistemas paralelos dedicados al cómputo neuronal. Sus principales desventajas son su elevado costo y que rápidamente se vuelven obsoletos.

Por su parte, los ASICs analógicos ofrecen implementaciones altamente densas de RNAs, de alta velocidad y de bajo consumo. Sin embargo, requieren un alto conocimiento de la tecnología analógica, resultan en implementaciones poco robustas, presentan problemas de precisión y almacenamiento de datos, son soluciones costosas y poco flexibles, y los algoritmos de aprendizaje son difíciles de implementar en esta tecnología.

En comparación con los ASICs analógicos, los ASICs digitales proporcionan RNAs de mayor precisión, más robustas, y pueden manejar eficientemente cualquier cálculo neural estándar. Sin embargo, al igual que su contraparte, requieren que el diseñador tenga conocimiento específico de la tecnología para lograr diseños funcionales, y es una opción muy costosa en volúmenes bajos de producción. Por lo general, esta tecnología es utilizada para implementar partes específicas de redes neuronales, a fin de ser incluidas en neuro- computadoras.

La última tecnología utilizada frecuentemente en la implementación de RNAs son los FPGAs, estos dispositivos tienen como principal característica el ser reprogramables. Además, las herramientas EDA-CAD (*Electronic Design Automation – Computer Aided Design*) para FPGAs existentes reducen el tiempo de diseño y lo hacen un proceso más sencillo, además de que permiten manejar un modelado del sistema utilizando un enfoque de software a través de lenguajes de descripción de hardware (HDL, *Hardware Description languaje*).

De acuerdo con Girau [28], de las tecnologías mencionadas, los FPGA representa la opción más equilibrada para la implementación de RNAs. En su evaluación Girau consideró como parámetros la velocidad de procesamiento, el área de semiconductor utilizada, el costo de un producto final, el tiempo de diseño y la confiabilidad de la implementación.

A las características de un FPGA mencionadas y al argumento vertido por Girau, se puede agregar que la arquitectura de un FPGA resulta ideal para explotar el paralelismo presente en una RNA, el modelado para un FPGA específico puede ser portable a otro dispositivo de una compañía diferente e incluso a futuros FPGAs, un FPGA posee la capacidad para implementar sistemas en un solo circuito integrado (SoC, *System on Chip*), puede ser usado para aplicaciones de sistemas embebidos y un diseño en un FPGA puede ser llevado a un ASIC lo que mejorara su relación velocidad-área-consumo.

3.3 Arreglo de Compuertas Programables de Campo

Los arreglo de compuertas programable de campo (FPGA) fueron introducidos por la compañía Xilinx en el año de 1985, es un dispositivo lógico cuya función puede ser modificada utilizando solamente una parte de los componentes que lo integran y/o cambiando las interconexiones entre ellos.

La arquitectura de un FPGA consiste en un arreglo bidimensional de bloques lógicos configurables (CLB), recursos de interconexión, bloques de entrada/salida (IOB, *Input Output Block*) y bloques de aplicación específica embebidos (ver Figura 3. 2) [16].

La versatilidad que tiene un FPGA para implementar cualquier función secuencial o combinacional está relacionada con las capacidades de los CLBs, elementos que representan la unidad lógica más básica en un FPGA, y la gran flexibilidad que tienen para interconectarse entre sí.

Los componentes internos de un CLB varían entre los diferentes fabricantes, por lo general contienen un circuito lógico llamado tabla de búsqueda (LUTs, *look-up table*). Una LUT consiste en una SRAM de dimensión $2^m \times 1$, la cual puede ser usada para representa una tabla de verdad de cualquier función lógica de m entradas.

Las líneas de entrada conectadas a la SRAM corresponden a las entradas de la tabla de verdad y la salida de la SRAM provee el valor de la función lógica [10]. Los CLBs pueden ser conectados gracias a la estructura de enrutamiento configurable.

En estos dispositivos, los CLB's pueden ser conectados eficientemente con CLB contiguos en la misma columna o renglón. Además, las conexiones configurables permiten conectar CLBs con IOBs, lo que habilita una conexión con el exterior. Considerando la complejidad de los CLBs, se puede distinguir dos tipos de FPGA, aquellos que tienen CLBs de complejidad baja que son denominados de granulidad fina, y los FPGAs de granulidad gruesa, constituidos por CLBs de complejidad alta [10].

La función de los IOBs es la de comunicar la lógica interna del FPGA con el exterior, es decir, controlan la entrada y salida de los datos. Entre los elementos que integran a los IOBs están flip-flop, latchs, bufer de tercer estado, etc. Las características que los IOBs otorgan a la lógica implementada en un FPGA son, bidireccionalidad, alta impedancia, polaridad de la salida programable, resistencias *pull-up* y *pull-down*, etc.

Los recursos de interconexión son elementos que también son configurables y que tienen por función comunicar a los CLBs entre sí y a estos con los IOBs.

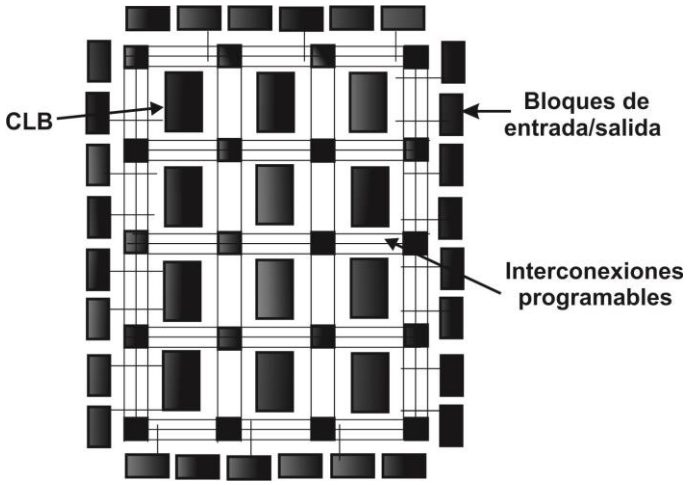

Figura 3. 2 Arquitectura del FPGA.

Estos recursos están constituidos principalmente por líneas horizontales y verticales que recorren los espacios existentes entre los CLBs. Elementos adicionales que forman parte de los recursos de interconexión son los puntos de interconexión programables y las matrices de interconexión.

Finalmente, los bloques de aplicación específica embebidos son aquellos elementos de arquitectura fija que forman parte de la arquitectura del FPGA. Elementos que comúnmente son integrados a un FPGA son aquellos que son empleados en una gran variedad de aplicaciones y/o consumen bastos recursos (CLBs) en su implementación. Elementos embebidos comunes en un FPGA son los bloquesRAM (BRAM), multiplicadores y administradores de señales de reloj (DCM, *Digital Clock Manager*).

En la actualidad, diversas compañías como Xilinx, Altera, Lattice Semiconductor, Actel, QuickLogic, Atmel, Achronix Semiconductor, fabrican dispositivos FPGA. Los campos de aplicación de los FPGA son bastos, tales como: uso militar, automotriz, medicina, video y audio. El fabricante Xilinx ofrece una amplia gama de dispositivos FPGA integrados en familias, como Spartan, Artix, Kintex, Virtex. Dentro de la familia Spartan se tienen los siguientes dispositivos:

- Automotive-grade XA Spartan-6 FPGA
- Automotive-grade XA Spartan-3A FPGA
- Automotive-grade XA Spartan-3A DSP FPGA
- Automotive-grade XA Spartan-3E FPGA

- Spartan-3 FPGA
- Spartan-3A FPGA
- Spartan-3AN FPGA
- Spartan-3E FPGA

El presente trabajo hace uso de la tarjeta de desarrollo Nexys-2 de la compañía Digilent Inc., la cual cuenta con un FPGA Spartan 3E, y que puede ser empleada en un sin número de aplicaciones basadas en FPGA. Esta tarjeta ofrece recursos que permite que los diseños puedan crecer fácilmente. Dichos recursos son representados por cuatro conectores de expansión de 12 pines, denominados PMOD, y son utilizados para integrar al sistema periféricos como, controladores de motor, ADC y DAC, circuitos de audio, y una serie de interfaces de sensores y actuadores. Todas las señales de la tarjeta Nexys2 cuentan con protección contra descargas electrostáticas (ESD, *electrostatic discharge*) y contra cortocircuitos, lo que garantiza una larga vida útil en cualquier entorno. Adicionalmente, la tarjeta Nexys-2 (ver Figura 3.3) cuenta con los siguientes recursos:

- Puerto USB para la configuración del FPGA y trasferencia de datos
- 16 MB de PSDRAM Micron y 16MB de ROM Intel StrataFlash
- Plataforma Flash Xilinx para las configuracions del FPGA
- Osilador de 50MHz
- Conectores de Entrada/Salida de proposito general
- Puerto de datos VGA, RS-232 y PS/2
- 8 LEds, 4 displays de 7 segmentso, 4 bottones y 8 switchs

3.4 Estado del arte

El presente trabajo de tesis versa sobre la implementación de RNAs sobre lógica reconfigurable. Por tal motivo, a continuación se hace mención de algunos trabajos donde interactúan estas dos áreas científicas.

Eldredge y Hutchings propusieron la implementación del algoritmo *backpropagation* en un FPGA, denominada RNA reconfigurable en tiempo de ejecución (RRANN, *Run-Time Reconfiguration Artificial Neural Networks*), cuya arquitectura es altamente escalable y hace uso eficientes de los recursos ofrecidos por el FPGA [18].

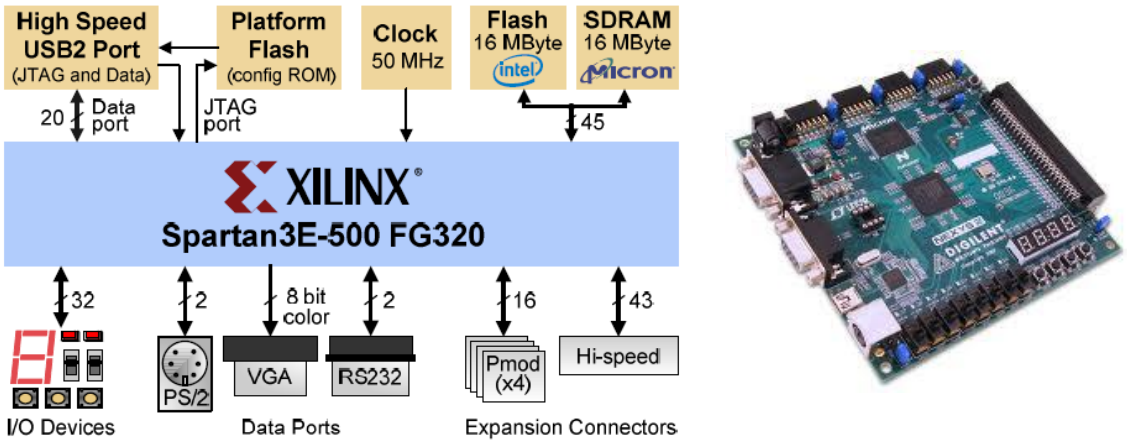

Figura 3.3 Diagrama a bloques de la tarjeta Nexys-2.

La principal característica de RRANN es que mejora el desempeño de la RNA explotando el paralelismo natural existente en el algoritmo *backpropagation*. La estructura de RRANN está constituida por un controlador global y varios procesadores neuronales. El controlador global es responsable de administrar la operación de los procesadores neuronales. Cada procesador neuronal contiene seis neuronas y subrutinas de hardware implementados como máquinas de estado, y es responsable de ejecutar los cálculos requeridos por el algoritmo *backpropagation*.

La implementación de un arreglo sistólico usado en la implementación de un MLP en un FPGA Virtex XCV400 se presenta en [25]. La implementación incluye el modelado del algoritmo *backpropagation* en su versión *on-line* a través de una estructura pipeline. Adicionalmente, los autores emplean un paralelismo denominado "*forward-backward parallelism*", donde las fases *forward* y *backward* para diferentes patrones de entrenamiento pueden ser desarrolladas concurrentemente.

Un novedoso modelo de un bloque basado en RNA's (BBNN, *Block-based Neural Netwrks*) se presenta en [59]. La arquitectura de BBNN consiste en un arreglo de dos dimensiones del bloque con cuatro variables de entrada/salida y conexiones de los pesos. Cada bloque puede tener una de cuatro diferentes configuraciones internas dependiendo de los requerimientos de la estructura. El modelo BBNN incluye la restricción de que el arreglo bidimensional y los pesos deben de ser enteros, esto con la finalidad de hacer más fácil la implementación en un FPGA.

De acuerdo a Oniga y Buchman, la implementación de dispositivos con la capacidad de aprendizaje y arquitecturas adaptativas es posible gracias al uso de RNA's [65]. En su trabajo, describen un nuevo método que permite un rápido diseño, entrenamiento e implementación de una RNA sobre un FPGA. Se emplean bloques específicos creados por el autor, en la herramienta de *System Generator* de *Simulink* para el diseño de la RNA. El *system Generator* permite también generar el código de Lenguaje de descripción de Hardware (HDL) del sistema representado en *simulink*. El VHDL generado puede ser posteriormente sintetizado para la implementación en un dispositivo de la familia *Xilinx*.

Por su parte Sahin *et al*., argumentan que el uso de un FPGA para la implementación de RNA provee de flexibilidad al sistema, por lo que plantean la implementación de RNA utilizando un FPGA Spartan IIE de la compañía Xilinx [73]. En este trabajo se diseñó una librería en VHDL enfocada al manejo de operaciones aritméticas de punto flotante con una precisión de 32 bits, que soporta el estándar IEEE-754. La RNA resultante es modular, compacta, eficiente; además, el número de neuronas, el número de capas y el número de entradas son fácilmente adaptables a diversas aplicaciones.

Ortigosa *et al*. presentaron la implementación en hardware de diversos modelados de un MLP mediante diferentes niveles de abstracción, principalmente nivel de transferencia de registros (RTL, *Register-Transfer Level*) y nivel algorítmico a través de los lenguajes de descripción de hardware VHDL y Handel-C [66]. En este trabajo, los autores estudiaron la eficiencia y la viabilidad de una implementación de RNAs en FPGAs para aplicaciones que involucren sistemas embebidos. Como caso particular presentan un sistema de reconocimiento de voz en el cual definen una arquitectura paralela de un MLP de 24 neuronas ocultas.

El desarrollo y la implementación de la arquitectura del *backpropagation*-MLP descrita mediante el lenguaje VHDL se describe en [29]. El reto es encontrar una arquitectura que minimiza los costes de hardware al tiempo que maximiza el rendimiento, la precisión, y la parametrización. El trabajo descrito en el presente documento es una plataforma que ofrece un alto grado de parametrización manteniendo al mismo tiempo un rendimiento comparable a otras implementaciones basadas en hardware MLP.

Un algoritmo para una implementación compacta de RNAs en hardware fue presentada por Dinu *et al*. en [17]. En su propuesta, los autores describen la operación de una neurona

artificial, con función de activación escalón, en términos de lógica booleana. Con este fin, el modelo matemático de la neurona es digitalizado y expresado mediante una estructura lógica de compuertas y optimizada eliminando la redundancia lógica existente. El modelado de la neurona se lleva a cabo usando VHDL, este modelado es portable lo que permite su fácil reutilización en diversas tecnologías FPGA.

Recientemente, Nedjah *et al*. presentaron la arquitectura en hardware de una RNA con una estructura MLP que explota el paralelismo inherente en las RNA [62]. La arquitectura propuesta se caracteriza por permitir cambiar, en tiempo de operación, el número de entradas a la red, el número de capas que la integran y el número de neuronas por capa; esta característica hace posible que una gran cantidad de aplicaciones relacionadas con RNAs pueda ser implementada usando la arquitectura propuesta. Con la finalidad de reducir el tiempo de procesamiento, la arquitectura planteada representa a los números reales usando fracciones de enteros, de esta forma las operaciones aritméticas son realizadas con valores enteros y modeladas con circuitos combinacionales. Además, se utiliza una función de activación sigmoidea, la cual es implementada a través de una aproximación polinomial que solo requiere sumas de productos.

En [11], Bhargav y Nataraj presentaron la implementación de la arquitectura *backpropagation* en un FPGA descrito en VHDL. La arquitectura de propósito general busca servir como una herramienta para realizar prototipos de funciones, pruebas de RNNA y como un sistema para la mejor comprensión de las RNA's implementadas en hardware. Para ello se ofrece un alto grado de parametrización, mientras se mantiene el diseño de red generalizado con un rendimiento comparable a otras implementaciones del MLP basadas en hardware.

Varias implementaciones en hardware de un MLP y una versión modificada llamada eXtended Multilayer Perceptron (XMLP) implementadas en un FPGA fueron presentadas por Ranjan *et al*. [70]. La versión modificada del MLP cuenta con 2 capas dimensionales y las interconexiones configurables. Las conexiones de las capas pueden restringirse acorde a la definición de los patrones. Con esto se produce un sistema que es rápido y con capacidades similares de clasificación. XMLP cuenta con funciones de activación configurables. El diseño se realizó utilizando 2 diferentes niveles de abstracción: nivel de registros de transferencia (*Register transfer level*, VHDL) y lenguajes de alto nivel (Handel-C).

Finalmente, La descripción de la red neuronal MLP en un FPGA es descrita en [2], además de la implementación de las funciones de activación tanh-sigmoid y log-sigmoid. Los resultados de este trabajo permitieron comprobar que la función tanh-sigmoid es una mejor opción que la función de activación log-sigmoid para la implementación en hardware propuesta.

Capítulo 4 Diseño del sistema

El presente capítulo describe en detalle el diseño del sistema propuesto. Muestra, inicialmente, como se diseñó una estructura modular de la neurona usada en el sistema; esta estructura tiene por finalidad brindar al usuario la facilidad de incluir y probar diversas descripciones de cada uno de los elementos que conforma a una neurona artificial. Posteriormente se muestra como, usando el modelo de neurona diseñado, se puede construir una RNA para una aplicación específica; para conseguir este fin, fue necesario modelar unidades de control que administren la operación de las capas que componen a la red.

4.1 Sistema para el modelado de RNA-MLP sobre lógica reconfigurable

El **Sistema para el modelado de RNA-MLP sobre lógica reconfigurable** desarrollado en este trabajo de tesis, es una herramienta que permite al usuario llevar a cabo la implementación de un MLP sobre un FPGA y monitorear su desempeño cuando se adapta a una aplicación específica. El sistema cuenta con diversos módulos que describen los diferentes elementos que componen a un sistema neuronal, y que pueden ser instanciados para crear la estructura de una RNA. Estos módulos fueron diseñados como elementos de tamaño genérico,

lo que permite fácilmente adaptar la RNA a una aplicación específica. Es decir, con la herramienta propuesta es fácil construir una RNA de un determinado número de entradas, de capas ocultas, de neuronas en las capas ocultas y de salidas.

El modelado de estos módulos se realizó mediante un HDL utilizando niveles de abstracción RTL y comportamental. La herramienta EDA-CAD utilizada en la síntesis e implementación del sistema diseñado es el software ISE Foundations ver. 12.1 de la compañía Xilinx Inc. y, aunque los módulos fueron descritos favoreciendo la portabilidad del sistema propuesto, la herramienta EDA-hardware en la que fue implementado y probado es la tarjeta de evaluación Nexys 2 de la compañía Digilent Inc., que cuenta con un FPGA Spartan 3E.

El diagrama de bloques del sistema propuesto se muestra en la Figura 4. 1. Es fácil distinguir una estructura modular y jerárquica que integra a los siguientes sub-sistemas:

- Un programa de software, desarrollado con el lenguaje C++ Builder y ejecutado en una computadora personal (PC), que permite al usuario interactuar con el hardware del sistema. A partir de este momento, este sub-sistema será denominado **Interfaz de usuario**.
- Un sistema embebido cuyo elemento central de procesamiento es un FPGA y que representa al hardware del sistema. Este sub-sistema será denominado **Sistema embebido RNA-MLP** y ésta constituido por los siguientes componentes:
  - **Controlador DLP-USB**. Se trata de un dispositivo DLP-USB245M que tiene por función permitir la transferencia de información entre la **Interfaz de usuario** y el **Sistema embebido RNA-MLP** vía el puerto USB.
  - **RNA lógica reconfigurable**. Un FPGA que representa al elemento central de procesamiento del sistema propuesto. Está integrado por los siguientes módulos:
    - **Driver DLP-USB**. Este módulo incluye el modelado del protocolo que permite establecer la comunicación con el **Controlador DLP-USB**.
    - **Unidad de control general**. Este módulo interpreta la información proveniente de la **Interfaz de usuario**, mediante la cual se administrara el funcionamiento de la RNA construida dentro del FPGA

- **RNA modular**. Se trata de un conjunto de módulos que describen la estructura o el comportamiento de cada uno de los elementos que son utilizados para construir la estructura de una RNA.

Los siguientes apartados describen detalladamente cada uno de los elementos mencionados.

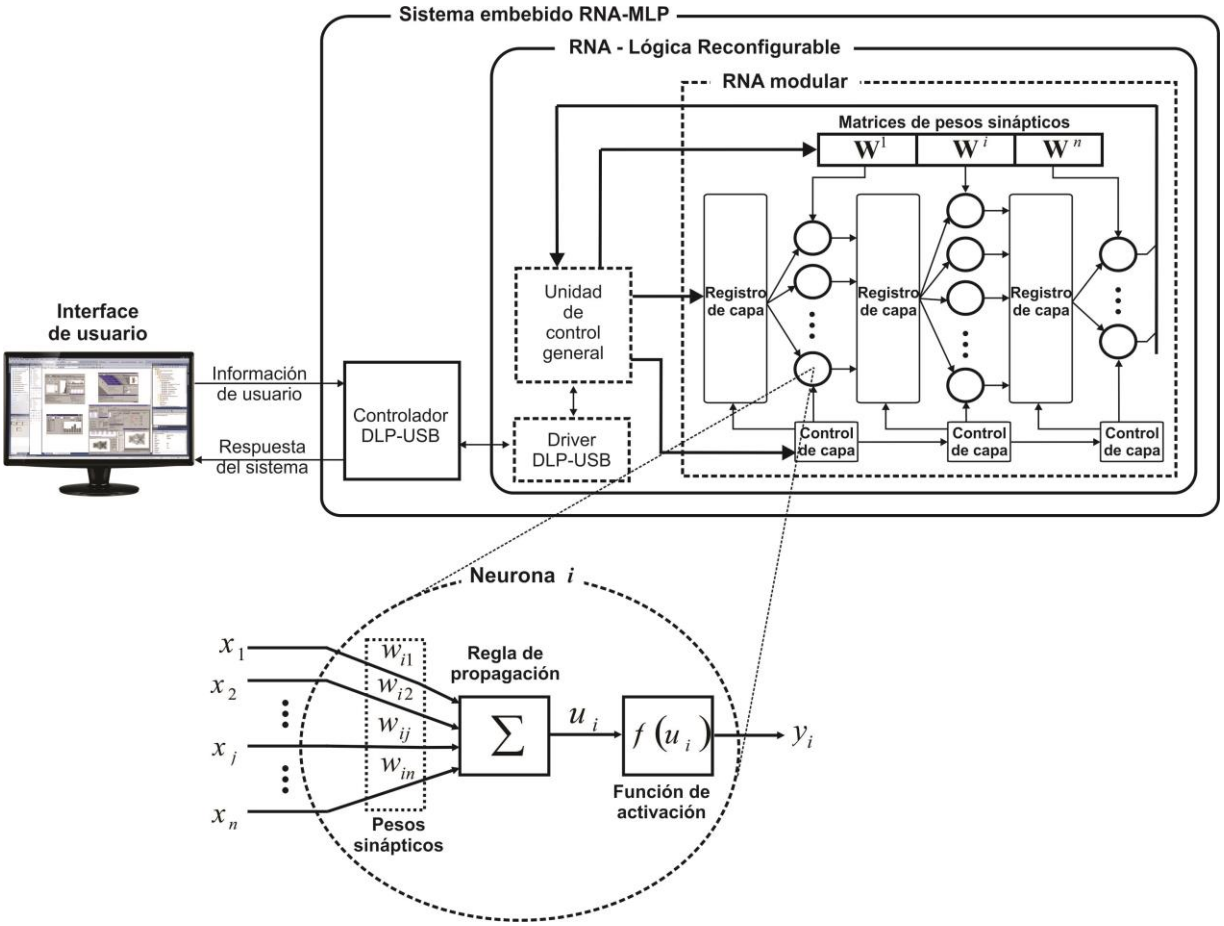

Figura 4. 1 Diagrama a bloques del sistema propuesto.

4.2 Interfaz de usuario

El contar con una interfaz gráfica de usuario (GUI, *Graphical User Interface*) que permita interactuar con aplicaciones de hardware basadas en sistemas embebidos es una innegable necesidad. La GUI del sistema propuesto, denominada **Interfaz de usuario** (ver Figura 4. 2), es una aplicación software que es ejecutada en una PC y que con la finalidad de contar con soporte y propiciar su evolución, fue desarrollada con base en C++. La comunicación entre la **Interfaz de usuario** y **Sistema embebido RNA-MLP** se lleva a cabo por medio del puerto USB.

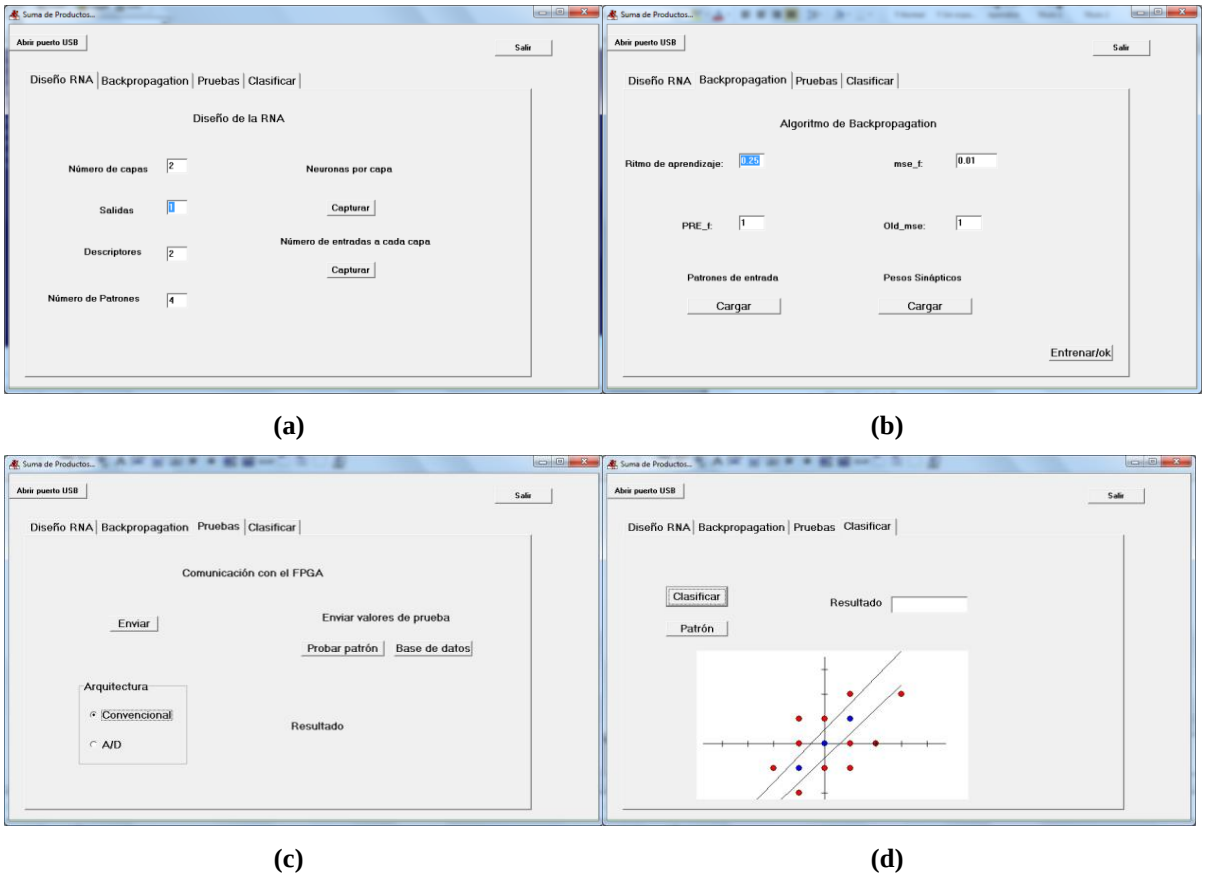

Figura 4. 2 Sub-sistema Interfaz de usuario (a) Especificación de parámetros de la RNA, **(b)** Resultados del clasificador, **(c)** Pruebas del sistema, **(d)** Especificación de parámetros del backpropagation**.**

Las funciones de la **Interfaz de usuario** son las siguientes:

- Entrenamiento de la RNA y envío de los pesos sinápticos. Debido a que en este trabajo de investigación no se considera un aprendizaje durante la operación de la RNA (aprendizaje *on-line*), este proceso es llevado a cabo por la **Interfaz de usuario** para posteriormente conformar la estructura de la red mediante el envío y actualización de los pesos sinápticos.
- Introducir patrones de prueba a la RNA. Una vez que los pesos sinápticos de la RNA estructurada en el FPGA han sido actualizados, la **Interfaz de usuario** permite al usuario enviar patrones de entrada para probar el desempeño de la RNA.
- Recepción y visualización del resultado. Una vez que RNA ha terminado de procesar un patrón de entrada, envía el resultado a la **Interfaz de usuario**, cuya función es recibirla y mostrarla al usuario en una forma fácil de interpretar.

El algoritmo de entrenamiento *backpropagation*, descrito en la sección 3.1, fue elegido para ser implementado en la **Interfaz de usuario**, debido a que es ideal para adaptar un MLP a una aplicación específica y a que se trata de uno de los algoritmos de entrenamientos más populares. El Algoritmo 4.1 muestra el algoritmo del *backpropagation* usado como base para su implementación en la **Interfaz de usuario**.

Algoritmo 4. 1. Algoritmo *backpropagation*.

Paso 1 Definir los patrones de entrenamiento, inicializar los Pesos sinápticos y umbral de cada neurona, Ritmo de aprendizaje ε, Error Cuadrático Medio obtenido old_mse, Error Cuadrático final E_f y Promedio de reducción de error final pre_f.

Paso 2 Inicializar $E=0$, $\Delta w'_{kj}=0$, $\Delta w'_{ji}=0$.

Paso 3 Para cada (x^{μ},t^{μ}) del conjunto de entrenamiento

Fase hacia adelante

- Calcular $y_j^{\mu}=f\left(\sum_i w_{ij}x^{\mu}{}_i-\theta_j\right)\ z_k^{\mu}=g\left(\sum_j w'_{kj}y_j^{\mu}-\theta'_j\right)$
- Actualizar $E=E+(\delta)^2$; $\delta=t^{\mu}-z^{\mu}$

Fase hacia atrás

- Actualización de pesos $w'_{kj}=w'_{kj}+\varepsilon\cdot\Delta w'_{kj}\quad w_{ji}=w_{ji}+\varepsilon\cdot\Delta w_{ji}$

Paso 4 Verificar el error $E=\frac{E}{numpat}\quad pre=\frac{old\_mse-E}{E}\ old\_mse=E$

Paso 5 Regresar al paso 2 si no se cumple que $(pre>pre\_f \parallel old\_mse>E\_f)$

Una vez que se ha definido la estructura del MLP que será utilizada para cierta aplicación y que ha sido implementada en el componente **RNA lógica reconfigurable**, la **Interfaz de usuario** permite generar los valores de los pesos sinápticos con solo indicarle los parámetros de configuración que definen la estructura del MLP, los cuales son:

- Número de capas ocultas
- Neuronas por capa
- Valor de los pesos sinápticos y umbrales iniciales
- Número de patrones
- Descriptores por patrón

- Clases de salida

Para el intercambio de información entre la **Interfaz de usuario** y el **Sistema embebido RNA-MLP** se estableció un simple protocolo de comunicación. La finalidad de este protocolo es permitir una transferencia de datos eficaz y segura, ofrecer una forma de distinguir la información que se está transfiriendo y abstraer al usuario de los procesos de envío y recepción de datos. El protocolo está formado por tres tramas de similar estructura (ver Figura 4. 3), pero la información contenida en sus campos tiene diferente significado en cada una de ellas, como se describe a continuación. Todas las tramas están formadas por tres campos, ID, CTR y DATA. El campo ID es de 8 bits y el campo CTR de 16 bits en las tres tramas, mientras que el campo DATA es de tamaño variable para cada una de las tramas. La función del campo ID es diferenciar a las tramas entre sí; el campo CTR tiene por función indicar cuantos datos (en bytes) están contenidos en el campo DATA; finalmente, el campo DATA contiene la información que intercambiaran los dos sub-sistemas que conforman al sistema propuesto.

- Trama de envío de pesos sinápticos. Mediante esta trama la **Interfaz de usuario** envía el resultado del algoritmo de aprendizaje al **Sistema embebido RNA-MLP**. En esta trama el campo ID tiene un valor fijo de 01 hex.
- Trama de envío de patrón de prueba. La **Interfaz de usuario** usa esta trama para enviar un patrón de prueba al **Sistema embebido RNA-MLP**. El valor de su campo ID es fijo, 02 hex.
- Trama de resultados. El resultado del procesamiento realizado por la RNA implementada en el **Sistema embebido RNA-MLP** es enviado a la **Interfaz de usuario** mediante esta trama. El valor de su campo ID es fijo, 03 hex.

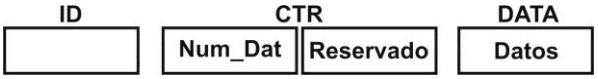

Figura 4. 3 Estructura de las tramas del protocolo de comunicaciones.

Así, cuando la **Interfaz de usuario** ha concluido la fase de entrenamiento, procede a enviar los pesos sinápticos obtenidos al **Sistema embebido RNA-MLP**, haciendo uso de la trama de envío de pesos sinápticos, donde son canalizados al componente **RNA modular** para darle la estructura final a la red. A partir de este momento, la **Interfaz de usuario** permite al usuario

enviar patrones de entrada, mediante la trama de envío de patrón de prueba, al componente **RNA modular** para su procesamiento. En respuesta, haciendo uso de la trama de resultados, la RNA modelada en el FPGA regresara el resultado del procesamiento mediante el cual el usuario podrá evaluar su desempeño. Antes de que la **Interfaz de usuario** envíe los datos al **Sistema embebido RNA-MLP** correspondientes a los pesos sinápticos w y umbral θ, se hace una conversión para representarlos en punto fijo con un tamaño en bits que el usuario define. Lo mismo para el proceso de recepción de datos, se debe hacer la conversión de los datos para que se puedan mostrar de forma gráfica.

4.3 Sistema embebido RNA-MLP

Considerando que los dispositivos programables son parte esencial de la mayoría de los sistemas de hoy en día, y que tienen un enfoque no sólo hacia diseño lógico programable, sino también a la integración de sistemas completos programables, el objetivo principal de esta investigación es ofrecer una herramienta que permita estructurar RNAs sobre lógica reconfigurable y evaluar sus características y desempeño. Por tal motivo, fue necesario construir un sistema embebido cuyo elemento central de procesamiento es un FPGA, a este sistema se le ha denominado **Sistema embebido RNA-MLP** y está integrado por los siguientes componentes:

- Un elemento de procesamiento denominado **RNA lógica reconfigurable**. En la actualidad la compañía Xilinx es líder mundial en investigación, desarrollo y comercialización de FPGAs, por tal motivo se decidió elegir como elemento central de procesamiento del **Sistema embebido RNA-MLP** a un FPGA de esta compañía. Con la finalidad de reducir el costo y agilizar el proceso de diseño, se decidió utilizar una tarjeta de evaluación comercial como elemento base en la construcción del sistema propuesto. La tarjeta elegida fue la Nexys 2 de la compañía Digilent Inc.

- Un elemento de comunicación denominado **Controlador DLP-USB**. El componente que complementa al **Sistema embebido RNA-MLP** es el circuito integrado DLP-USB245M que permite establecer una comunicación con la **Interfaz de usuario** a través del puerto USB.

La Figura 4. 4 muestra la estructura final del **Sistema para el modelado de RNA-MLP sobre lógica reconfigurable** formada por la **Interfaz de usuario** y el **Sistema embebido RNA-MLP**.

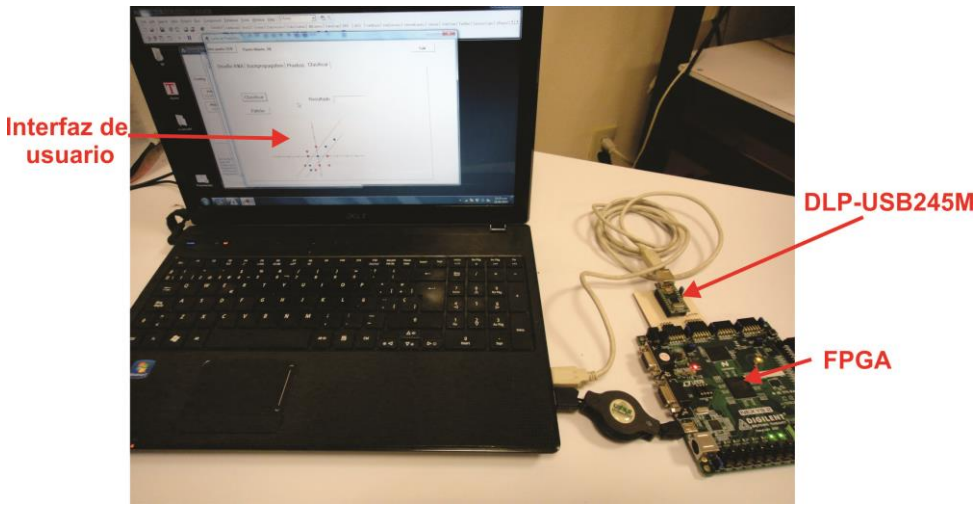

Figura 4. 4 Sistema embebido **RNA-MLP.**

4.3.1 Controlador DLP-USB

El componente **Controlador DLP-USB** provee un método efectivo de transferencia de información, de hasta 8 millones de bits por segundo, entre la **Interfaz de usuario** y el **Sistema embebido RNA-MLP** vía el puerto USB.

El componente **Controlador DLP-USB** está constituido por el sistema DLP-USB245M (ver Figura 4.4), éste es un módulo de interfaz, que convierte de USB a FIFO y viceversa, que utiliza al dispositivo de 4ª generación FT245R de la compañía FTDI y que es compatible con USB 2.0. El DLP-USB245M incluye un driver directo denominado D2XX, cuya arquitectura consiste de un driver USB que se comunica con el FT245R a través del *stack* USB de Windows y una biblioteca de vínculo dinámico (DLL, *Dynamic Link Library*) que facilita la interfaz de la aplicación software del usuario (que puede ser escrita en VC++, C++ Builder, Delphi, VB, etc.) con el driver USB.

Por su parte, el **Sistema embebido RNA-MLP** visualiza al DLP-USB245M como un simple dispositivo de entrada-salida, en forma de FIFO, lo que facilita su interfaz a través de pines de entrada-salida de propósito general.

4.3.2 RNA lógica reconfigurable

El componente **RNA lógica reconfigurable** representa al elemento central de procesamiento del sistema propuesto y, como ya se mencionó, se decidió utilizar la tarjeta de evaluación comercial Nexys 2 de la compañía Digilent Inc. (ver Figura 4.4). Esta tarjeta cuenta con un FPGA miembro de la familia Spartan-3E de la compañía Xilinx. Esta familia de FPGAs es ideal para aplicaciones que involucren co-procesamiento DSP, control embebido y en general integración de sistemas. El FPGA especifico con el que cuenta la tarjeta mencionada es un Spartan 3E-500 FG320 cuyos recursos principales son 1164 CLBs (soportan características de *carry logic* y RAM distribuida), 20 bloques RAM de 18 kbits, 20 multiplicadores de 18×18 bits y 4 administradores de señales de reloj (DCM).

La estructura del componente **RNA lógica reconfigurable** fue definida siguiendo la metodología de diseño digital descendente (*Top-Down*). Siguiendo esta metodología se determinó que el componente **RNA lógica reconfigurable** fuera compuesto por una serie de módulos que pueden ser clasificados en dos categorías, aquellos que son transparentes para el usuario y que cumplen funciones de control (**Unidad de control general** y **Driver DLP-USB**) y aquellos que son ofrecidos al usuario para construir la RNA (**RNA modular**). Todos y cada uno estos los módulos fueron modelados mediante el lenguaje VHDL utilizando los niveles de abstracción RTL y algorítmico. En las sub-secciones siguientes son descritos en detalle.

4.3.2.1 Driver DLP-USB

El módulo **Driver DLP-USB** utiliza el diagrama de tiempos del dispositivo DLP-USB245M para modelar el protocolo, a través de una simple máquina de estados finitos (FSM, *finite state machine*), que permite al componente **RNA lógica reconfigurable** acceder a los recursos del DLP-USB245M y establecer la transferencia de datos con la **Interfaz de usuario**.

Cuando la **Interfaz de usuario** envía información al componente **RNA lógica reconfigurable**, los pesos sinápticos o un patrón de prueba, este módulo la recibe y la entrega al módulo **Unidad de control general** para su procesamiento. Cuando la RNA termina de procesar la información, el resultado es enviado a la **Interfaz de usuario** mediante el módulo **Driver DLP-USB**.

La Figura 4.5 muestra el símbolo del módulo **Driver DLP-USB** y en la Tabla 4. 1 se describen brevemente la función de sus señales.

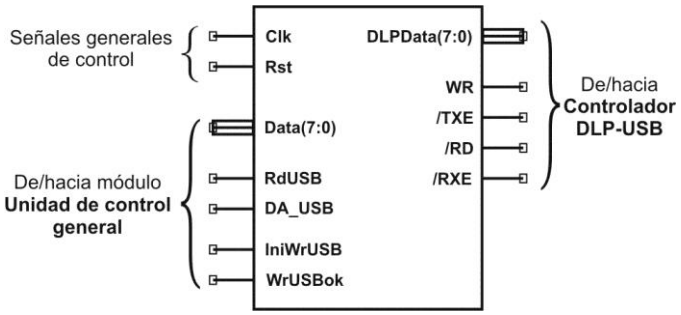

Figura 4.5 Símbolo del módulo **Driver DLP-USB**.

Tabla 4. 1. Señales del módulo **Driver DLP-USB**.

Nombre	Modo	Descripción
Clk	In	Señal de reloj global del sistema
Rst	In	Señal de inicialización global del sistema
Data	In/Out	Bus de datos usado para transferir la información entre el módulo **Driver DLP-USB** y el módulo **Unidad de control general**.
DA_USB	Out	Señal de control activa en alto utilizada por el **Driver DLP-USB** para indicar al módulo **Unidad de control general** que existe información disponible para él. Cumple la función de reconocimiento (*Strobe*). Esta señal es desactivada cuando ***RdUSB*** es activa.
RdUSB	In	Señal de control activa en alto utilizada para indicar al **Driver DLP-USB** que su bus de datos ha sido leído. Cumple la función de reconocimiento (*Acknowledge*).
IniWrUSB	In	Señal activa en alto, indica al **Driver DLP-USB** que debe transmitir la información presente en su bus ***Data*** hacia la **Interfaz de usuario**.
WrUSBok	Out	Señal activa en alto, cuando esta señal está activa indica que el **Driver DLP-USB** ha terminado su tarea de transmisión.
DLPData	In/Out	Contiene el dato que el **Controlador DLP-USB** ha recibido/debe transmitir de/hacia la **Interfaz de usuario**.
WR	Out	En la transición de nivel alto a bajo de esta señal, el **Controlador DLP-USB** lee el bus ***DLPData*** y lo transmite a la **Interfaz de usuario**.
/TXE	In	Cuando esta señal se encuentra en nivel alto indica que el buffer de transmisión, de 385 bytes, del **Controlador DLP-USB** está lleno y que no se le debe enviar otro dato
/RXE	In	Cuando esta señal se encuentra en nivel bajo indica que al menos un dato (byte) está presente en el buffer de recepción, de 128 bytes, del **Controlador DLP-USB** y está listo para ser leído. Esta señal está en alto cuando el buffer está vacío.
/RD	Out	Esta señal es activa en bajo y es utilizada para indicar al **Controlador DLP-USB** que saque de alta impedancia al bus ***DLPData*** y ponga disponible el primer byte del buffer de recepción.

4.3.2.2 Unidad de control general

El módulo **Unidad de control general** es la unidad que administra el funcionamiento del componente **RNA lógica reconfigurable**. Este módulo también fue descrito mediante una FSM que decodifica las tramas, de envío de pesos sinápticos y de envío de patrón de prueba, provenientes de la **Interfaz de usuario** y genera las señales que controlan la operación de los módulos que conforman al componente **RNA modular**; además, una vez que el módulo **Unidad de control general** recibe los resultados de la RNA, estructura la trama de resultados y habilita al módulo **Driver DLP-USB** para enviarla hacia la **Interfaz de usuario**.

La Figura 4. 6 muestra la estructura externa de la **Unidad de control general** y la Tabla 4.2 describe la función de sus señales

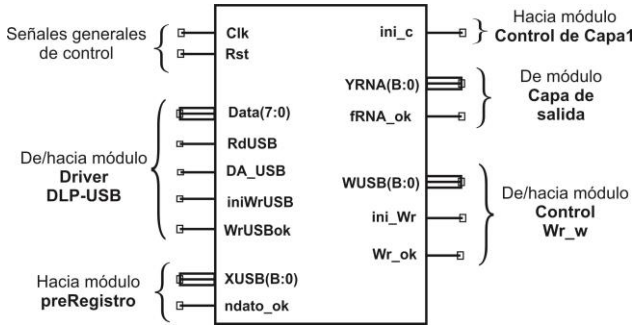

Figura 4. 6 Símbolo del módulo **Unidad de control general.**

Tabla 4.2 Señales del módulo **Unidad de control general.**

Nombre	Modo	Descripción
Clk	In	Señal de reloj global del sistema
Rst	In	Señal de inicialización global del sistema
Data	In/Out	Bus de datos usado para transferir la información entre el módulo **Driver DLP-USB** y el módulo **Unidad de control general**.
DA_USB	In	Señal de control activa en alto utilizada por el **Driver DLP-USB** para indicar al módulo **Unidad de control general** que existe información disponible para él.
RdUSB	Out	Señal de control activa en alto utilizada para indicar al **Driver DLP-USB** que su bus de datos ha sido leído.
IniWrUSB	Out	Señal activa en alto, indica al **Driver DLP-USB** que debe transmitir la información presente en su bus ***Data*** hacia la **Interfaz de usuario**.
WrUSBok	In	Señal activa en alto, cuando esta señal está activa indica que el **Driver DLP-USB** ha terminado su tarea de transmisión.

Nombre	Modo	Descripción
XUSB	out	Bus de datos utilizado para enviar información al **preRegistro** Contiene el dato que el **Controlador DLP-USB** ha recibido/debe transmitir hacia el módulo **preRegistro**.
ndato_ok	Out	Señal activa en alto para indicarle al módulo **preRegistro** que un nuevo descriptor del patrón de entrada se ha recibido y se debe almacenar (*Strobe*).
ini_c	Out	Señal de control activa en alto que le indica al módulo **Control de capa** de la capa de entrada de inicio su operación (este evento da inicio de operación de la RNA).
WUSB	Out	Contiene el dato correspondiente al pesos sináptico o umbral que la **Unidad de Control general** ha recibido y debe transmitir hacia **ControlWr_w** para que sea almacenado en la RAM (módulo **Pesos sinápticos**) de la **Neurona** correspondiente
ini_wr	Out	Señal activa en alto que indica al módulo **ControlWr_w** que existe un dato disponible para él en el bus ***WUSB*** para su canalización a la neurona correspondiente.
wr_ok	In	Cuando el módulo **ControlWR_w** ha concluido su tarea de actualización del peso sináptico o umbral, se lo hace saber a la **Unidad de Control general** mediante esta señal (activa en alto).
fRNA_ok	Out	Señal activa en alto que indica a la **Unidad de Control general** que la operación de la RNA para el patrón de entrada enviado ha concluido.
YRNA	In	Bus de datos que contiene el resultado final de la RNA

4.3.2.3 RNA modular

Hasta ahora los elementos mencionados que forman parte del sistema tienen por función acceder a sus recursos y administrar su funcionamiento. Los módulos que integran al componente **RNA modular** son los que permiten al usuario diseñar y construir una RNA sobre el sistema propuesto.

El componente **RNA modular** fue diseñado guardando una estructura modular, por lo que una vez que se identificaron los diferentes elementos que integran a una neurona artificial, se modelaron de forma independiente. Esta estructura hace de este componente un ambiente ideal para probar diversas opciones en cada uno de los elementos que forman a una RNA, lo que se consigue con solo cambiar y/o modificar el elemento que se desea estudiar.

Al analizar la estructura de una RNA, es fácil distinguir que se trata de un conjunto de elementos de procesamiento, neuronas, interconectados entre sí a través de enlaces

denominados conexiones sinápticas. Por lo tanto, el primer paso es definir la estructura que guardara la neurona dentro del sistema.

4.3.2.3.1 Definición de la estructura de la neurona artificial

La neurona artificial es el elemento básico en el diseño de una RNA, y por lo regular todas las neuronas que forman a la RNA tiene la misma estructura; por lo tanto, dentro de la herramienta propuesta, una vez modelada la neurona artificial puede ser instanciado tantas veces como el diseño lo requiera y es mediante su agrupación en capas e interconexiones que se puede estructurar la RNA.

La definición de la estructura externa de la neurona es mostrada en la Figura 4.7 y la descripción de sus señales se encuentra en la Tabla 4.3.

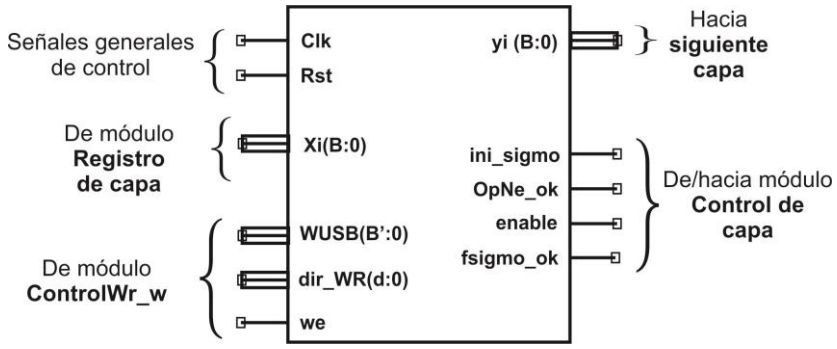

Figura 4.7 Símbolo del módulo **Neurona.**

Tabla 4.3 Descripción de las señales del módulo **Neurona.**

Nombre	Modo	Descripción
Clk	In	Señal de reloj global del sistema
Rst	In	Señal de inicialización global del sistema
Xi	In	Bus de datos utilizado para indicar el descriptor del patrón proviene del módulo **Registro de capa**
WUSB	In	Contiene el dato correspondiente al pesos sináptico o umbral que proviene del módulo **ControlWr_w** para que sea almacenado en la RAM (módulo **Pesos sinápticos**) de la **Neurona** correspondiente
dir_WR	In	Contiene la dirección de lectura o escritura de los pesos sinápticos y umbral, este bus es compartido por el **Control de capa** y **ControlWr_w**
we	In	Señal activa en alto que indica que habilita la escritura del pesos sináptico o umbral

Nombre	Modo	Descripción
ini_sigmo	In	Señal activa en alto que indica habilita el proceso del módulo **función de activación**
OpNe_ok	Out	Señal activa en alto que indica que las operaciones del módulo **Regla de propagación** terminó su operación
ndato_ok	Out	Señal activa en alto para indicarle al módulo **preRegistro** que un nuevo descriptor del patrón de entrada se ha recibido y se debe almacenar (*Strobe*).
enable	In	Señal activa en alto que habilita las operaciones de la **Neurona**
fsimo_ok	Out	Señal activa en alto que indica que las operaciones de la **Neurona** terminaron
yi	Out	Señal que contiene el dato resultado de la operación de la Neurona e indica el estado de esta

Para definir la estructura interna de la neurona, se considera que ésta debe contemplar dos procesos:

- Entrenamiento. Que consiste simplemente en almacenar los pesos sinápticos, provenientes de la **Interfaz de usuario**. Como administrador de este proceso existe un módulo **ControlWr_W** por capa.
- Clasificación. Contempla los módulos necesarios para que una vez presentado un patrón de entrada, la red genere su identificación.

Debido a que se diseñó un modelado que infiere neuronas de tamaño genérico, es decir al instanciar a la neurona artificial se debe indicar el tamaño de ésta, también es necesario considerar que este módulo presenta diversos parámetros de configuración acordes al diseño de la red. Estos parámetros de configuración son:

- Tamaño en bits, B, de las entradas w y x, esta configuración es para todas las neuronas de la RNA. Este mismo valor se toma para determinar el tamaño de la salida de la neurona, cuyo valor se propaga a la siguiente capa.
- El número de entradas a la neurona, siendo este valor igual para las neuronas de una misma capa y pudiendo variar para las neuronas de una capa diferente, esto debido a que una RNA puede presentar un número diferente de neuronas en cada capa.

Ahora, de acuerdo a la función desempeñada, existen tres elementos principales que integran a una neurona artificial y que son definidos como módulos independientes dentro del sistema propuesto (ver Figura 4.8):

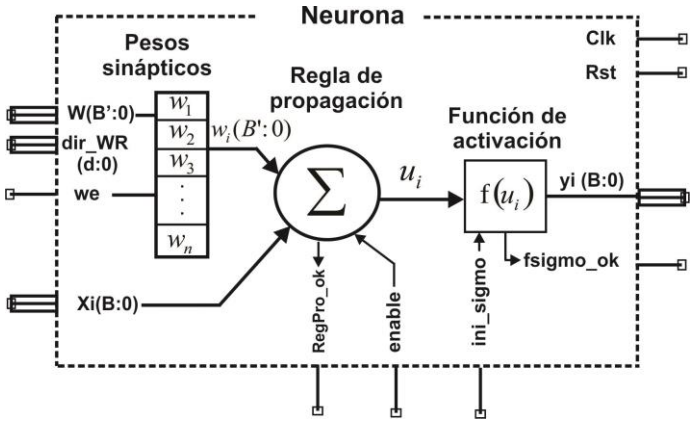

Figura 4.8 Estructura interna de la **Neurona Artificial**.

- El módulo **Pesos sinápticos**. Tiene por función almacenar la información correspondiente a los pesos sinápticos. La organización o estructura de este módulo depende de la arquitectura que se esté implementando para la regla de propagación.
- El módulo **Regla de propagación**. Su función es llevar a cabo el cálculo de la suma ponderada de las entradas por los pesos sinápticos (potencial sináptico).
- El módulo **Función de activación**. Determina el estado de la neurona aplicando la función de activación elegida sobre el potencial sináptico.

Definición del módulo Pesos sinápticos

La definición del módulo **Pesos sinápticos** se muestra en la Figura 4.9. Este módulo permite almacenar y leer los pesos sinápticos y el umbral en un momento determinado. Se compone de una RAM distribuida *single port* de tamaño $n \times B'$, donde

- n es el número de localidades con el que cuenta la RAM y depende de la arquitectura de la regla de propagación que se esté implementando.
- B' es el tamaño en bits de los pesos sinápticos que se esté empleando y permanece constante para todas las neuronas de la RNA. Este valor es acorde a la arquitectura que se esté implementando.
- Es el tamaño en bits de las salidas

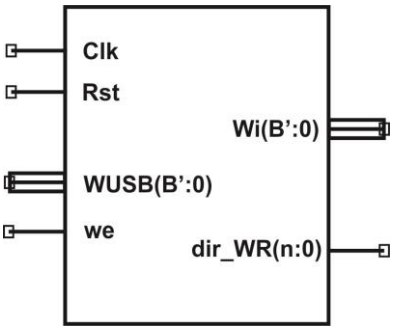

Figura 4.9 Símbolo del módulo **Pesos sinápticos**.

La operación de escritura es síncrona, mediante la cual se permite almacenar todos los pesos sinápticos y umbral de la neurona. Una vez que se lleva a cabo este proceso en todas las neuronas, esta operación queda deshabilitada y los datos almacenados quedan disponibles para su uso en las operaciones normales de la RNA. La lectura de los pesos sinápticos y umbral es asíncrona para que el módulo de la **Regla de propagación** pueda disponer del dato que se esté direccionando en ese momento.

En la Tabla 4.4 se muestra la descripción de las señales de entrada y salida del módulo **Pesos sinápticos**.

Tabla 4.4 Descripción de las señales del módulo **Pesos sinápticos.**

Nombre	Modo	Descripción
clk	In	Señal de reloj global del sistema
rst	In	Señal de inicialización global del sistema
dir_WR	In	Indica la dirección de memoria en donde se va a leer o escribir el pesos sináptico y umbral de la neurona
WUSB	In	Es el dato que se va a almacenar en la memoria, puede corresponder al peso sináptico o umbral y proviene de la PC.
we	In	Señal activa en alto, habilita la operación de escritura de la RAM
Wi	Out	Contiene el valor del peso sináptico o umbral en la operación de lectura de la RAM

Definición del módulo Regla de propagación

El módulo **Regla de propagación** contiene la implementación de la función de propagación, la cual permite calcular la sumatoria de las entradas ponderadas por su peso sináptico. En la Figura 4.10 se muestra la propuesta para el módulo **Regla de propagación**.

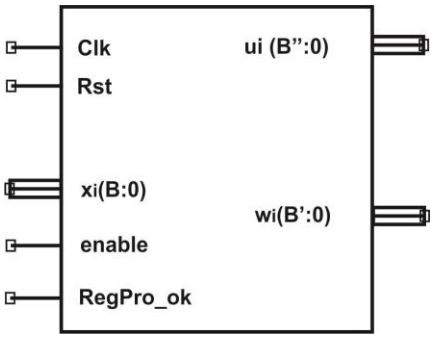

Figura 4.10 Símbolo del módulo **Regla de propagación.**

Mediante los parámetros de configuración es posible definir

- B', el número de bits para representar los pesos sinápticos ***wi***
- B, número de bits para representar las entradas ***xi***

El tamaño B'' de la salida ***ui*** es variable, el cual depende del número de entradas (n) a la neurona y desde que se realizan multiplicaciones y sumas, su valor queda definido como $(B+B')+n$, siendo n el número de patrones de entrada a la neurona.

En la Tabla 4.5 se muestra la descripción de las señales del módulo **Regla de propagación**.

Tabla 4.5 Descripción del módulo **Regla de propagación.**

Nombre	Modo	Descripción
Clk	In	Señal de reloj global del sistema
Rst	In	Señal de inicialización global del sistema
enable	In	Señal activa en alto, indica que se ha habilitado el módulo para realizar el cálculo del producto punto.
Xi	In	Representa el patrón de entrada
Wi	In	Representa los pesos sinápticos
RegPro_ok	Out	Indica que se realizaron correctamente las operaciones de la función de propagación correspondientes a la entrada ***xi*** y ***wi***
ui	Out	Contiene el resultado de las operaciones de la función de propagación

La estructura interna de este módulo depende de la arquitectura que se desee implementar, las cuales pueden ser

- Arquitectura Convencional (Arq/Conv)
- Arquitectura basada en Aritmética Distribuida (Arq.b/AD)

Arquitectura convencional

En esta arquitectura se implementa la función de propagación mediante un esquema Multiplicador-Acumulador (MAC, *Multiplier accumulator*), el cual realiza la suma de la multiplicación de las entradas x_i por los pesos sinápticos w_i.

$$y_i = (w_1 \times x_1) + (w_2 \times x_2) + ... + (w_n \times x_n) - \theta_m \tag{4.1}$$

donde n corresponde a las entradas que se hayan especificado para la neurona, por lo que este proceso se repite $n+1$ veces tomando en cuenta el umbral. En la iteración $n+1$ se lleva a cabo la suma del umbral correspondiente a la neurona i, de la ecuación (4.1) se puede ver que este no se multiplica por una entrada constante de -1, por lo que no es necesario contar con una entrada x para esta operación. En la Figura 4.11 se muestra el esquema del Multiplicador-Acumulador.

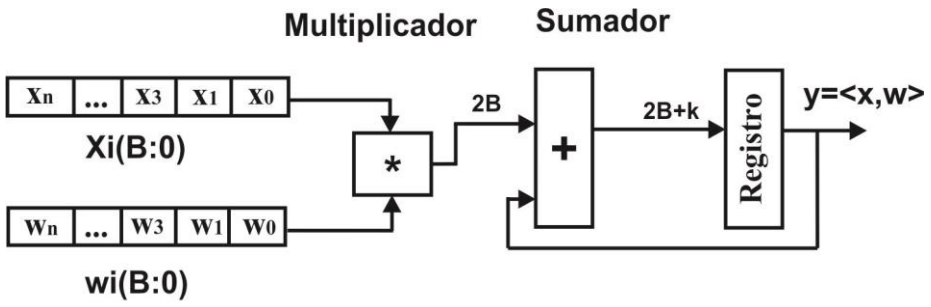

Figura 4.11 Esquema MAC.

En la Figura 4.12 se muestra la definición del módulo **Regla de propagación** para la implementación con base en la arquitectura convencional. La implementación de esta arquitectura se lleva a cabo a través de tres elementos internos

- Módulo **Multiplicador**, permite realizar las multiplicaciones de los registros de entrada ***xi*** y ***wi***.

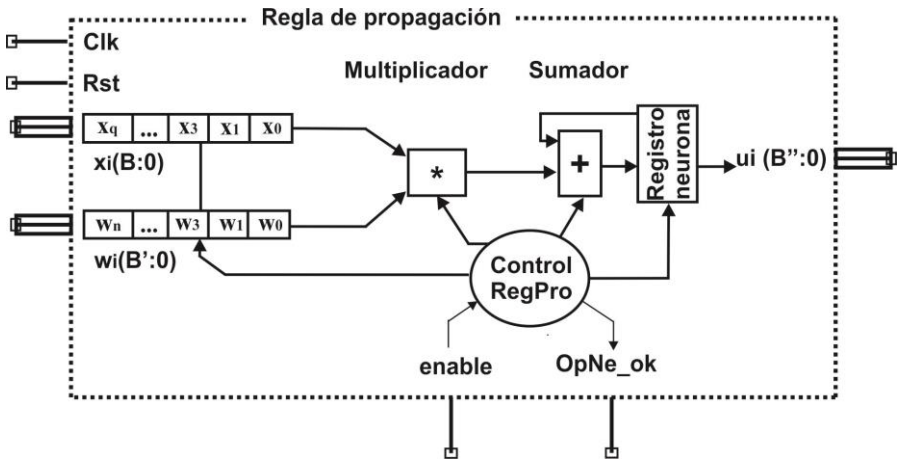

Figura 4.12 Definición de la arquitectura convencional.

- Módulo **Sumador**, realiza la suma de la salida del multiplicador con el resultado de la suma anterior.
- Módulo **Control RegPro**, se encarga de controlar la operación del módulo **Regla de propagación**.

En el módulo **Multiplicador** se realiza la multiplicación de los registros de entrada ***A***, ***B*** y el resultado se expresa en el registro de salida ***rMul***. Este módulo soporta operaciones con signo, el cual es definido por el MSBit y el tamaño de los registros de entrada y salida es definido previo a su funcionamiento. En la Figura 4.13 se muestra la definición externa del módulo **Multiplicador**.

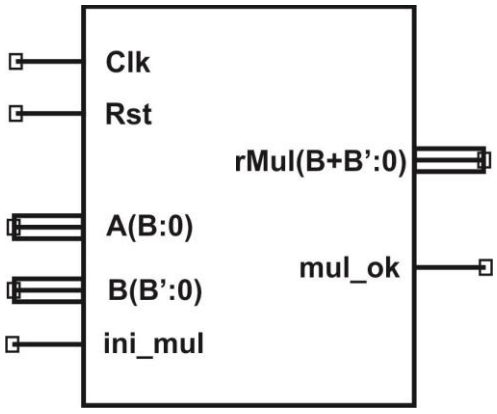

Figura 4.13 Símbolo del módulo **Multiplicador**

En la Tabla 4.6 se muestra la descripción de las señales del módulo **Multiplicador**.

Tabla 4.6 Descripción de las señales del módulo **Multiplicador**.

Nombre	Modo	Descripción
Clk	In	Señal de reloj global del sistema
Rst	In	Señal de inicialización global del sistema
A,B	In	Registros de entrada para la multiplicación
ini_mul	In	Señal activa en alto, habilita el módulo para que se realice la multiplicación de ***A*** con ***B***
rMul	Out	Registro de salida que contiene el resultado de la multiplicación
mul_ok	Out	Señal activa en alto, indica que la operación del módulo se llevó a cabo correctamente

El módulo Sumador realiza la suma de los registros de entrada ***A, B*** y el resultado se expresa en el registro de salida ***rSum***. En la Figura 4.14 se muestra la entidad para el módulo del sumador.

En la Tabla 4.7 se muestra la descripción de las señales del módulo **Suma**.

Tabla 4.7 Descripción de las señales del módulo **Suma**.

Nombre	Modo	Descripción
Clk	In	Señal de reloj global del sistema
Rst	In	Señal de inicialización global del sistema
A,B	in	Registros de entrada para la suma
ini_sum	In	Señal activa en alto, indica habilita el módulo para que realice la suma de los registros ***A*** y ***B***
rSum	Out	Registro de salida que contiene el resultado de la suma
sum_ok	Out	Señal activa en alto, indica que la operación del módulo se llevó a cabo correctamente

El **Registro neurona** permite mantener el valor del resultado del acumulador para que pueda ser empleado en cualquier momento determinado que la aplicación lo requiera.

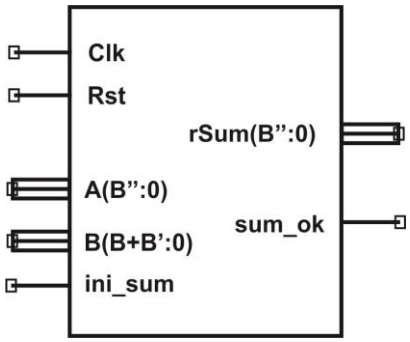

Figura 4.14 Símbolo para el módulo **Suma.**

Finalmente, **Control RegPro** es el módulo que administra la operación del módulo **Regla de propagación**; este módulo fue implementado mediante una simple FSM, que inicia su operación cuando la señal ***enable*** es activada, y que se encarga de controlar el flujo de información a través de los otros módulos que integran al de la **Regla de propagación**. Determina el momento en que los valores w_i, x_i son canalizados al multiplicador y cuando el resultado debe ser sumado con el anterior, finalmente, mediante la señal ***OpNe_ok***, indica cuando el valor del potencial sináptico está presente en la salida ui.

Bajo estas condiciones, tomando en cuenta que la primer multiplicación que se realiza en la $j-$ésima neurona, es $w_1 \times x_1$, ecuación (4.1), para facilitar la lectura, los pesos sinápticos, w_i, se organizan en el módulo **Pesos sinápticos** en el siguiente orden: $w_{j1}, w_{j2}, ..., w_{jn}, \theta_m$, este es el orden en que se acceden para su procesamiento.

Por su parte, las entradas x_i son proporcionadas por el módulo **Registro de capa**, el cual contiene todos los vectores de entrada de la capa y los va asignando a las neuronas en el orden $x_1, x_2, ..., x_n$.

Arquitectura basada en Aritmética distribuida

La arquitectura basada en aritmética distribuida presenta un enfoque alternativo al MAC para el cálculo del producto punto en el módulo **Regla de propagación** de la neurona artificial.

Partiendo de la función de propagación

$$y =< c, x >= \sum_{n=0}^{N-1} c[n] \cdot x[n] \tag{4.2}$$
$$= c[0]x[0] + c[1]x[1] + ... + c[N-1]x[N-1]$$

donde $c[n]$ representa los pesos sinápticos, que después del entrenamiento no se modifican por lo que pueden ser tomados como constantes, y $x[n]$ representa los patrones de entrada, los cuales pueden variar y N representa el número de patrones de entrada a la capa. Desde que los pesos sinápticos y los patrones de entrada pueden ser negativos, se adopta una representación en complemento a dos para estos valores. Por lo que la magnitud del número será representada por B bits y el signo será el MSB ($B+1$).

La variable $x[n]$ es representada como

$$x[n] = -2^B \times x_B[n] + \sum_{b=0}^{B-1} x_b[n] \times 2^b \quad \text{con} \quad x[n] \in [0,1] \tag{4.3}$$

donde $x_b[n]$ denota $b-$ésimo bit de $x[n]$. Si se sustituye (4.3) en (4.2) se tiene que

$$y =< c, x >= \sum_{n=0}^{N-1} c[n] \times x[n] = \sum_{n=0}^{N-1} c[n] \times (-2^B \times x_B[n] + \sum_{b=0}^{B-1} x_b[n] \times 2^b) \tag{4.4}$$
$$= -2^B \sum_{n=0}^{N-1} c[n] \times x_B[n] + \sum_{b=0}^{B-1} 2^b \times \sum_{n=0}^{N-1} c[n] \times x_b[n]$$
$$= -2^B \sum_{n=0}^{N-1} f(c[n], x_B[n]) + \sum_{b=0}^{B-1} 2^b \times \sum_{n=0}^{N-1} f(c[n], x_b[n]) \tag{4.5}$$

De la ecuación 4.5 podemos determinar la arquitectura hardware de un sistema con aritmética distribuida el cual se muestra en la Figura 4.15, donde la LUT contiene todos los coeficientes de $f(c[n], x_b[n])$ previamente calculados.

Las características de la LUT dependerán directamente del número de patrones de entrada y la resolución que se adopte para representar los datos (B bits). En la Tabla 4.8 se muestra como está organizado el contenido del módulo **Pesos sinápticos** de una neurona.

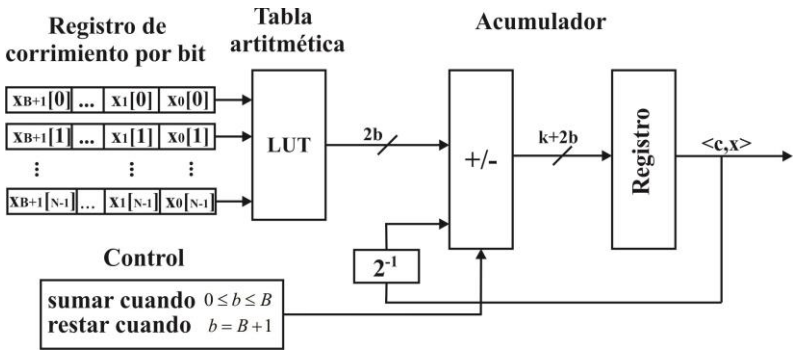

Figura 4.15 Arquitectura basada en aritmética distribuida.

Tabla 4.8 Memoria RAM con aritmética distribuida.

Localidad	Dato
0	0
1	$c[0]$
2	$c[1]$
3	$c[0]+c[1]$
…	…
N-1	$c[0]+c[1]+c[2]+...+c[N-1]$

Se puede observar que el número de localidades depende directamente del número de patrones de entrada y queda determinado como 2^N, donde cada localidad es de $locAD = ((B+1)+2N)$ bits, esto debido a que cada suma incrementa 1 bit la representación de $f(c[n], x_b[n])$. Este módulo es la principal ventaja de esta arquitectura al sustituir al multiplicador de la arquitectura convencional.

El resto de los módulos usados en la arquitectura convencional son reutilizados en el diseño de la arquitectura basada en aritmética distribuida, solamente se modifica el comportamiento de algunos de ellos para adecuarlos a la estructura de aritmética distribuida manteniendo siempre la misma estructura externa de los mismos.

Como se mencionó, de la Figura 4.15 se puede observar que los multiplicadores utilizados en el esquema MAC (ver Figura 4.11) son reemplazados por una LUT, debido a esto en el módulo **Regla de propagación** se omite el multiplicador y solo se implementa un registro de corrimiento al resultado del acumulador en cada iteración, de igual forma se contempla las reglas de control para que en la última operación se realice una resta.

En la Figura 4.16 se muestra la propuesta de la estructura interna de este módulo.

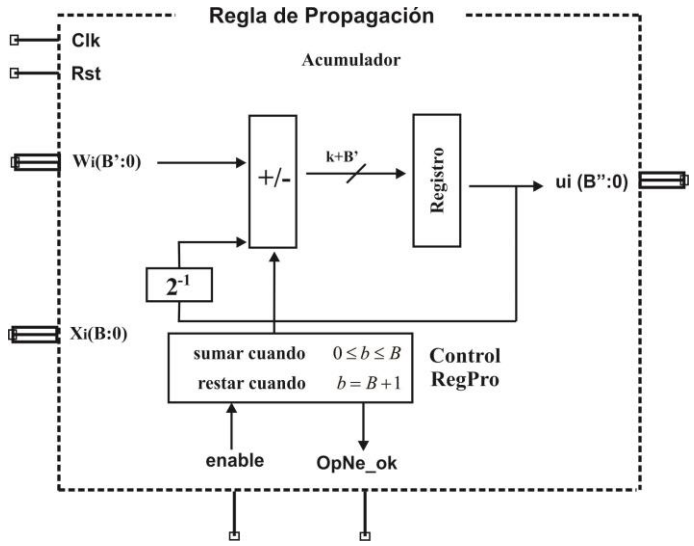

Figura 4.16 Definición de la arquitectura basada en **Aritmética Distribuida.**

En el módulo **Pesos sinápticos** (ver Figura 4.9) se implementó la LUT que reemplaza a los multiplicadores, la cual almacena todos los posibles valores que se pueden obtener de la suma de los pesos sinápticos. Según la Tabla 4.8 se requiere que los pesos sinápticos y umbral de cada neurona tengan un orden específico, por lo que la **Interfaz de usuario** envía está información en el siguiente formato

$$0, w_{11}, w_{11}+w_{12},..., w_{11}+w_{12}+...+w_{1i}, \theta_1,$$
$$0, w_{21}, w_{21}+w_{22},..., w_{21}+w_{22}+...+w_{2i}, \theta_2,...,$$
$$0, w_{j1}, w_{j1}+w_{j2},..., w_{j1}+w_{j2}+...+w_{ji}, \theta_m$$

Definición del módulo Función de activación

El módulo **Función de activación** contiene la implementación de un algoritmo de la aproximación de una función de activación tipo sigmoidea, en la Figura 4. 17 se muestra la definición de este módulo.

Cuando el módulo **Regla de propagación** realiza el cálculo de la suma ponderada de las entradas por los pesos sinápticos, el módulo **Función de activación** debe calcular el valor correspondiente de ui para determinar el estado de la neurona. Esté proceso está controlado por el módulo **Control capa** y se realiza en el mismo instante en todas las neuronas de la misma capa.

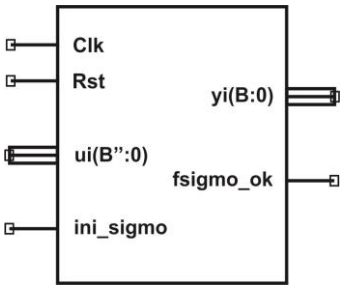

Figura 4. 17 Símbolo del módulo **Función de activación.**

En la Tabla 4.9 se muestra la descripción de las señales de entrada y salida del módulo en estudio.

Tabla 4.9 Descripción de las señales del módulo **Función de activación.**

Nombre	Modo	Descripción
Clk	In	Señal de reloj global del sistema
Rst	In	Señal de inicialización global del sistema
ui	In	Contiene el resultado de la función de propagación
ini_sigmo	In	Esta señal en nivel alto indica que se debe hacer el cálculo de la función de activación del valor que en ese momento se tenga de ui
yi	Out	Es el valor de la función sigmoidea correspondiente a la salida de la neurona
fsigmo_ok	Out	Indica con un nivel alto que la operación se llevó acabo correctamente

Los parámetros de configuración de este módulo permiten determinar

- El tamaño en bits de la señal de entrada, la cual está en función de la aplicación de la neurona
- El tamaño en bits de la señal de salida y_i, la cual se propaga a la siguiente capa por lo que es necesario que se adecue el tamaño para que pueda ser usado correctamente por los siguientes módulos

Los parámetros de configuración anteriores permanecen fijos para todas las neuronas de la RNA.

Para la implementación de diversos algoritmos de la función de activación, la estructura externa de este módulo permanece fija, sólo se modifica la estructura interna y los parámetros de configuración mencionados.

En el presente trabajo, se modelaron e implementaron varias aproximaciones lineales a tramos (PWL, *Piecewice Linear*) de la función de activación tipo sigmoidea, éstas son:

- Ley A
- Alipi
- Aproximación por Interpolación Recursiva Centrada
- PLAN

Cabe hacer mención que la mayoría de las implementaciones en hardware de aproximaciones a una función de activación tipo sigmoidea, explotan la simetría existente este tipo de funciones. Para una función logística, esta simetría se expresa formalmente mediante:

$$\begin{aligned} y_{x>0} &= 1 - y_{x\leq 0} \\ y_{x<0} &= 1 - y_{x\geq 0} \end{aligned} \tag{4.6}$$

Y para una función tangente hiperbólica se expresa mediante

$$\begin{aligned} y_{x>0} &= -y_{x\leq 0} \\ y_{x<0} &= -y_{x\geq 0} \end{aligned} \tag{4.7}$$

Ley A

La ley A (A-Law) fue creada por Smith D. R. en 1985. Es un sistema de cuantificación logarítmica de señales de audio, usado habitualmente con fines de compresión en aplicaciones de voz humana y transmisión de datos y es de baja complejidad computacional. El algoritmo ley A basa su funcionamiento en un proceso de compresión y expansión denominado *companding*.

La primer aplicación de la ley A (llamada *Scaled A-law*) consta de 13 segmentos y fue escalada para valores $x \in [-8,8]$ y $y \in [-1,1]$ o $[0,1]$. Pero no obtuvo un buen desempeño cuando fue usada para entrenar al MLP y ocasiono un entrenamiento inestable y problemas al converger, esto debido a que el gradiente de la aproximación de la ley A en la región cercana a $x = 0$ es mucho más grande que el de la función sigmoidea original. Para resolver esto Mayers y Hutchinson desarrollaron una variante de la ley A en la que cada gradiente de los segmentos es expresado como una potencia de 2 [60]. Con esta nueva propuesta la curva solo tiene 7 segmentos y se aproxima mejor en las regiones cercanas a 0. En la Tabla 4. 10 se muestra los valores de la propuesta para la función de activación de la ley A y la modificación.

En la Tabla 4.11 a) y b) se muestra una tabla de verdad de la Ley A modificada, con un vector de entrada I de 16 bits en el rango de [–8,+8] y salidas $(R_o - R_7)$ en el rango de 0 a 1.

Tabla 4. 10 Valores de la función propuesta.

Puntos de control			
Ley A		**Ley A modificada**	
x	**y**	**x**	**y**
-8.0	0.0	8.0	0.0
-4.0	0.0625	4.0	0.0625
-2.0	0.125	2.0	0.125
-1.0	0.1875	1.0	0.1875
-0.5	0.25	-	-
-0.25	0.3125	-	-
-0.125	0.375	-	-
0.125	0.625	-	-
0.25	0.6875	-	-
0.5	0.75	-	-
1.0	0.8125	1.0	0.75
2.0	0.875	2.0	0.875
4.0	0.9375	4.0	0.9375
8.0	1.0	8.0	1.0

Tabla 4.11a Tabla de verdad de las entradas de la Ley A modificada.

I_{15}	I_{14}	I_{13}	I_{12}	I_{11}	I_{10}	I_9	I_8	I_7	I_6	I_5	I_4	I_3	I_2	I_1	I_0
0	0	0	0	0	a	b	c	d	x	x	x	x	x	x	x
0	0	0	0	1	a	b	c	d	x	x	x	x	x	x	x
0	0	0	1	a	b	c	d	x	x	x	x	x	x	x	x
0	0	1	a	b	c	x	x	x	x	x	x	x	x	x	x
0	1	a	b	c	x	x	x	x	x	x	x	x	x	x	x

Tabla 4.12b. Tabla de verdad de las salidas de la Ley A modificada.

R_7	R_6	R_5	R_4	R_3	R_2	R_1	R_0
0	1	0	0	a	b	c	d
0	1	0	1	a	b	c	d
0	1	1	0	a	b	c	d
0	1	1	1	0	a	b	c
0	1	1	1	1	a	b	c

La implementación en hardware de la Tabla 4.11a se muestra en la Figura 4.18.

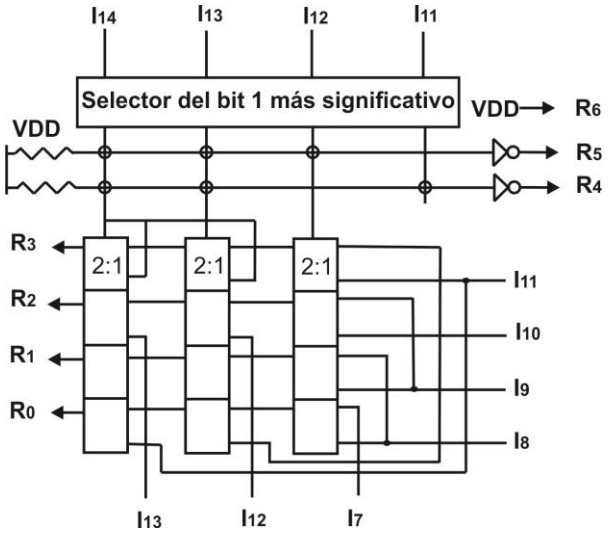

Figura 4.18 Selector del bit 1 más significativo.

Esta arquitectura consiste en un selector del bit más significativo con valor "1", para se consideran las entradas $(I_{11} - I_{14})$ para generar las salidas de 4 bits $(y_{11} - y_{14})$ cuyo valor será 1 en la posición donde se encuentre que el bit de más peso que tiene un valor de 1 y las posiciones restantes tendrá un valor de 0. Esta salida es decodificada y mostrada en $R_4 - R_5$, el bit $R_6 = 1$ por que las salidas siempre son > 0.5 y $R_7 = 0$ porque son positivas.

Finalmente, explotando la simetría existente en una función sigmoidea, la arquitectura final de la ley A modificada se muestra en la Figura 4.19.

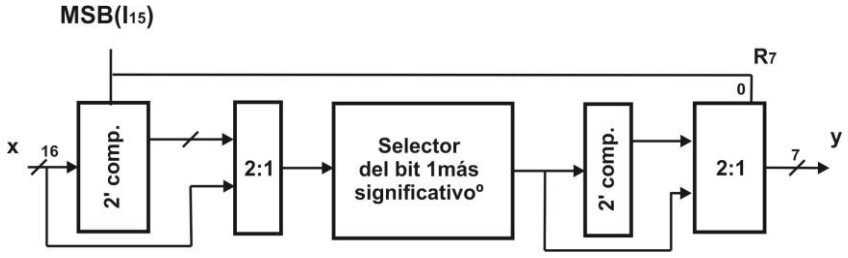

Figura 4.19 Arquitectura completa de la **Ley A** modificada.

Alippi

En esta propuesta la aproximación a la función de activación se basa en la selección de un conjunto de puntos de control (*Break-points*) de valor entero y expresar los valores de y correspondientes como números potencia de 2 [3].

Si se consideran valores negativos x<0 se pueden tomar dos puntos cualesquiera de la función sigmoidea (ver Figura 4.20),

$$P_n = \left(-n, \frac{1}{2^{n+1}} \right), \; P_{n-1} = \left(-(n-1), \frac{1}{2^n} \right)$$

donde n define los *break-points*.

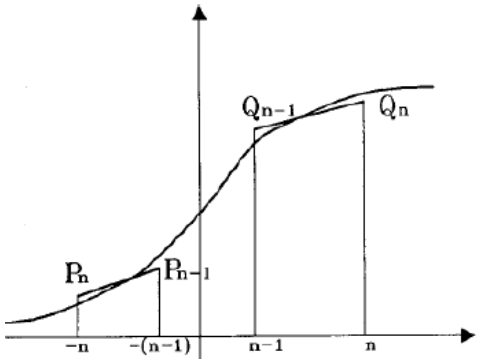

Figura 4.20 Función sigmoidea, breakpoints Pn,Qn.

Utilizando interpolación lineal se sabe que la ecuación definida por dos puntos es

$$\frac{y - y_1}{y2 - y_1} = \frac{x - x_1}{x_2 - x_1} \text{ por lo que } \frac{y - \frac{1}{2^{n+1}}}{\frac{1}{2^n} - \frac{1}{2^{n+1}}} = \frac{x - (-n)}{-(n-1) - (-n)}$$

Si se despeja y, se tiene que

$$y = \frac{x + (n+1)}{2^{n+1}} \text{ donde } n = 0,1,..., x < 0 \tag{4.8}$$

Para los valores de $x \geq 0$, se toman los puntos

$$Q_n = \left(n, 1 - \frac{1}{2^{n+1}} \right), \; P_{n-1} = \left(n-1, 1 - \frac{1}{2^n} \right)$$

Siguiendo el mismo procedimiento que para $x < 0$, se tiene que

$$y = \frac{2^{n+1} + x - (n+1)}{2^{n+1}} \text{ donde } n = 0,1,..., x \geq 0 \tag{4.9}$$

Debido a la función logística es simétrica en el punto (0,0.5), la aproximación se puede implementar usando solo una de las expresiones (4.8) y (4.9), considerando la expresión (4.6).

Para el caso de que $x<0$, los *breakpoints* pueden ser definidos como

$$\begin{array}{cc} -1\le x<0 & n=1 \\ -2\le x<-1 & n=2 \\ \cdots & \cdots \\ -m\le x<-(m-1) & n=m \end{array}$$

Si se defina a (x) como la parte entera de x, entonces n puede ser escrita como $n=|(x)|+1$. Sustituyendo en (4.8) se tiene que

$$y=\frac{x+|(x)|+2}{2^{|(x)|+2}} \tag{4.10}$$

Ahora si se define a $\hat{x}$ como la parte entera de x con su signo $\hat{x}=x+|(x)|$ y se sustituye esta expresión en (4.10) tenemos que

$$y=\frac{\frac{\hat{x}}{4}+\frac{1}{2}}{2^{|(x)|+2}} \tag{4.11}$$

Finalmente la propuesta de la implementación en hardware de la expresión (4.11) se muestra en la Figura 4.21.

EL registro de corrimiento inicia con el valor de $y=\frac{\hat{x}}{4}+\frac{1}{2}$ y se desplazan $|(x)|$ veces a la derecha y en cada desplazamiento se inserta $not(sign)$ en el MSBit.

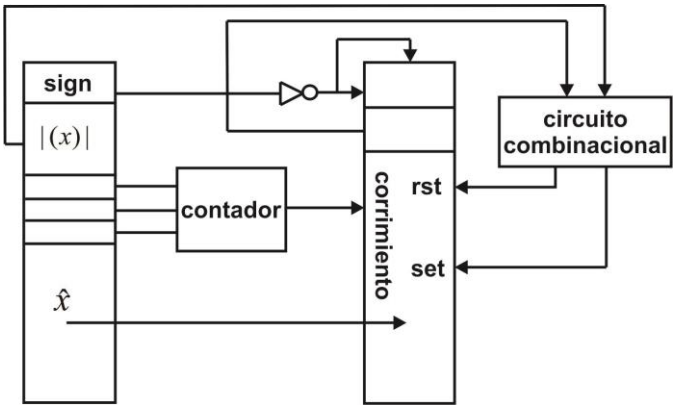

Figura 4.21 Arquitectura de la función de aproximación **Alippi.**

Aproximación por Interpolación Recursiva Centrada

Basterretxea *et al*. propusieron una aproximación PWL, mediante una estructura algebraica. Esta propuesta usa operadores máximos y mínimos para generar una PWL, los cuales son modificados mediante una interpolación linear recursiva (CRI, *Centred Recursive interpolation*) [8].

En la aproximación de funciones sigmoideas mediante el método CRI la estructura inicial queda definida mediante 3 líneas cuyas ecuaciones son

$$y_1 = 0$$
$$y_2 = \frac{1}{2}(1 + \frac{x}{2})$$
$$y_3 = 1$$

Al considerar la simetría de la función sigmoidea, solo se evalúa la parte negativa de esta función, evitando con esto la definición de $y_3 = 1$.

El algoritmo para la definición de estos parámetros es

$$g(x) = y_1(x) = 0;$$
$$h(x) = y_2(x) = \frac{1}{2}\left(1 + \frac{x}{2}\right);$$
$$for(i = 0; i = q; i++)\{$$
$$g'(x) = \max[g(x), h(x)];$$
$$h(x) = \frac{1}{2}(g(x) + h(x) + \Delta);$$
$$g(x) = g'(x);$$
$$\Delta = \frac{\Delta}{4};\}$$
$$g'(x) = \max[g(x), h(x)];$$

Los parámetros de configuración son

- q es el nivel de interpolación
- Δ es el parámetro de profundidad y depende de q
- $h(x)$ es la función de interpolación lineal
- $g(x)$ es la aproximación obtenida

Los valores óptimos de Δ para $q = 1,2,3$ y el número de segmentos de la aproximación se muestra en la Tabla 4.13.

Tabla 4.13 Tabla de valores de q y Δ.

Nivel de interpolación	Nivel de profundidad óptimo	Número de segmentos
$q = 0$		3
$q = 1$	$\Delta_{1,opt} = 0.30895$	5
$q = 2$	$\Delta_{2,opt} = 0.28094$	9
$q = 3$	$\Delta_{1,opt} = 0.26588$	17

Desde que este método es iterativo, requiere $q+1$ ciclos de reloj para obtener el valor de un dato de la aproximación. En la Figura 4. 22 se muestra la propuesta para la implementación en hardware de este método de aproximación.

PLAN

Amin *et al*. propusieron la aproximación denominada PLAN (*Piecewise Linear Approximation of a Nonlinear function*), la cual es una técnica que selecciona gradientes que permiten una implementación sencilla en hardware mediante la eliminación de multiplicadores. La representación de la función sigmoidea está definida en la Tabla 4.14 [5].

Tabla 4.14 Representación de la aproximación **PLAN.**

Rango	Operación	Condición	Bandera			
			z_1	z_2	z_3	z_4
1	$y = 1$	$\lvert x \rvert \geq 5$	0	0	0	1
2	$y = 0.03125 \times \lvert x \rvert + 0.84375$	$2.375 \leq \lvert x \rvert < 5$	0	0	1	0
3	$y = 0.125 \times \lvert x \rvert + 0.625$	$1 \leq \lvert x \rvert < 2.375$	0	1	0	0
4	$y = 0.25 \times \lvert x \rvert + 0.5$	$0 \leq \lvert x \rvert < 1$	1	0	0	0
5	$y = 1 - y$	$x < 0$	0	0	0	0

Las banderas son utilizadas en la implementación en hardware para saber a cuál rango pertenece *x*, donde cada salida representa una línea recta entre 2 *break-points*. La selección de estos *break-points* se hizo buscando que en el cálculo de la aproximación, las multiplicaciones se puedan sustituir por operaciones de corrimiento, en la Figura 4.23 se muestra el circuito combinacional que permite obtener el valor para las banderas.

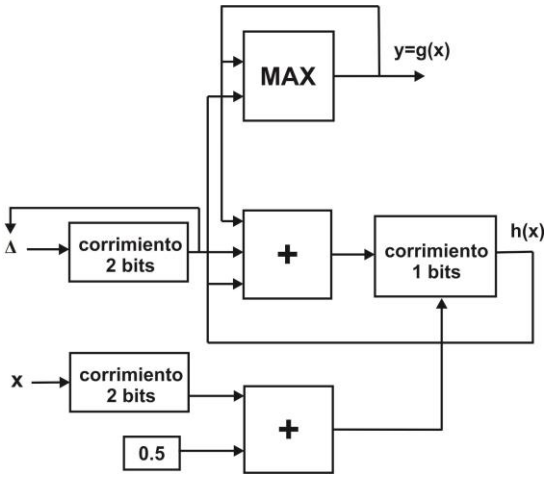

Figura 4. 22 Diagrama para la implementación en hardware de **CRI**

Una vez que se obtiene el rango al que pertenece el registro de entrada x, se deben de hacer el registro de corrimiento que puede ser 2,3 o 5 veces y posteriormente una suma para estimar el valor de la salida y, por lo que en el caso crítico se requieren de al menos 5 ciclos de reloj clk. Estas operaciones pueden ser evitadas aplicando una trasformación directa del registro x al registro y (DTXY).

De la Tabla 4.14 se observa que el rango 1 en el que $0 \leq |x| < 1; y = 0.25 \times |x| + 0.5; z_1 = 1$ el valor de $|x|$ debe correrse dos veces a la derecha y sumarle 0.5_{10}.

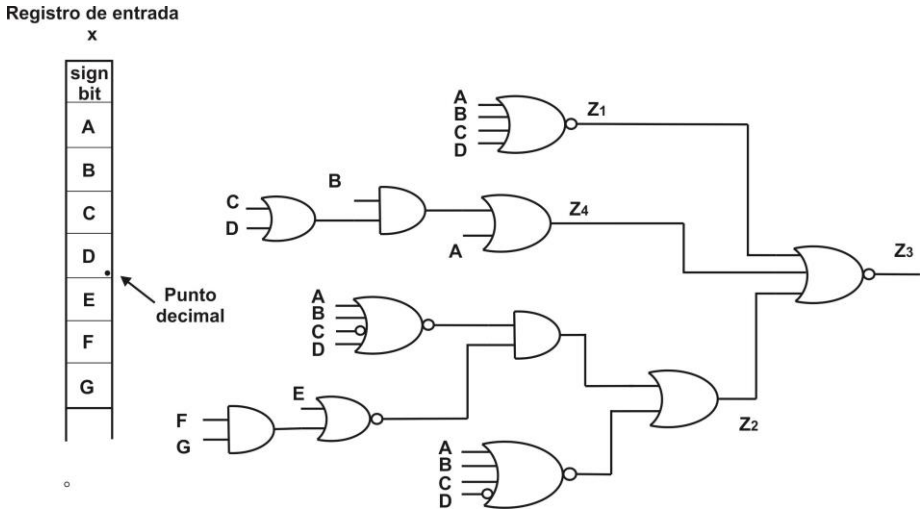

Figura 4.23 Obtención del rango del registro de entrada x.

Entonces las operaciones sobre x quedan definidas como

$$ABCD.EFGH \rightarrow 0000.EFGH$$
$$0000.EFGH \rightarrow 00.00EFGH$$
$$00.00EFGH \rightarrow 00.10EFGH$$
$$\downarrow\downarrow\downarrow\downarrow\downarrow\downarrow\downarrow$$
$$A'.B'C'D'E'F'G'$$

De esta forma para el rango 4, el resultado de la DTXY según lo anterior queda definido como

$$A' = 0; B' = 1; C' = 0; D' = E; E' = F; F' = G; G' = H;...$$

Siguiendo este mismo procedimiento se puede ver que la DTX para los otros rangos queda definida como

$$A' = 0; B' = 1; C' = 1; D' = C; E' = E; F' = F; G' = G;... \quad \text{Rango 3}$$

$$A' = 0; B' = 1; C' = 1; D' = 1; E' = (C \ \ XNOR \ \ D); F' = NOT(D); G' = E;... \quad \text{Rango 2}$$

$$A' = 1 \quad \text{Rango 1}$$

Al unir todos los rangos, los autores proponen la siguiente implementación en hardware para DTXY mostrado en la Figura 4.24.

Finalmente el diagrama de la arquitectura final para la implementación de la propuesta PLAN se muestra en la Figura 4.25. Se puede observar que el vector de entrada x puede representar valores negativos, por lo que previo a la etapa de comparador de magnitud y DTX, se obtiene el complemento para operar con el valor positivo y a la salida se vuelve a recuperar su representación negativa para el caso de que sea x negativo.

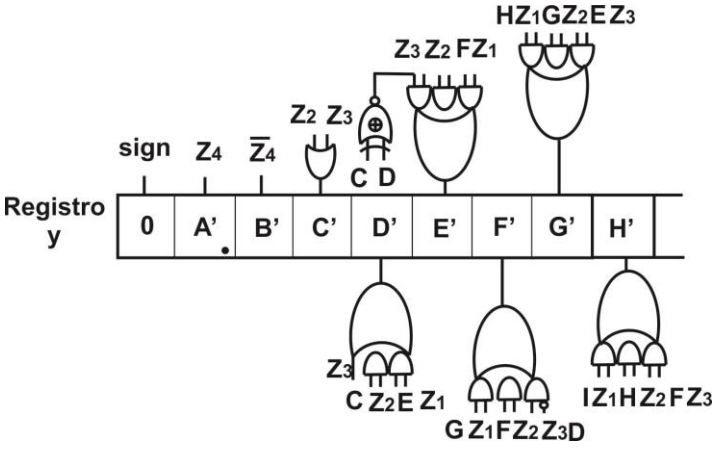

Figura 4.24 Propuesta del hardware para la **DTX.**

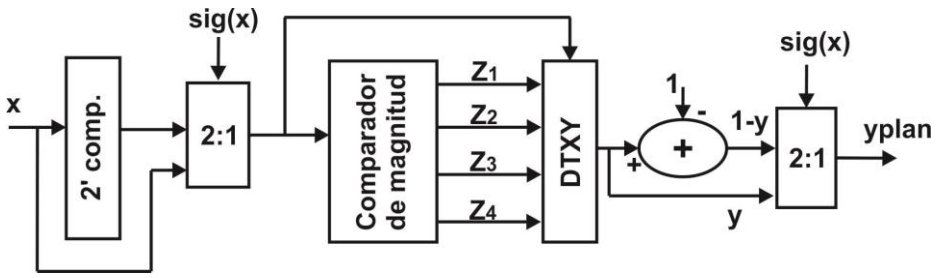

Figura 4.25 Propuesta en hardware de **PLAN.**

4.3.2.3.2 Estructura de la RNA

Considerando los módulos descritos y agregando algunos que complementan la funcionalidad de la des, la estructura en una capa queda definida como se muestra en la Figura 4.26. La conexión de este tipo de estructuras da como resultado a la RNA final.

El módulo **ControlWr_w** es una FSM que permite controlar las operaciones de escritura de los pesos sinápticos y umbral en la RAM distribuida contenida en cada neurona. Este módulo controla dichas operaciones de todas las neuronas contenidas en una capa de la RNA. El proceso es previo a la operación normal de la RNA con los patrones de entrada provenientes de la PC.

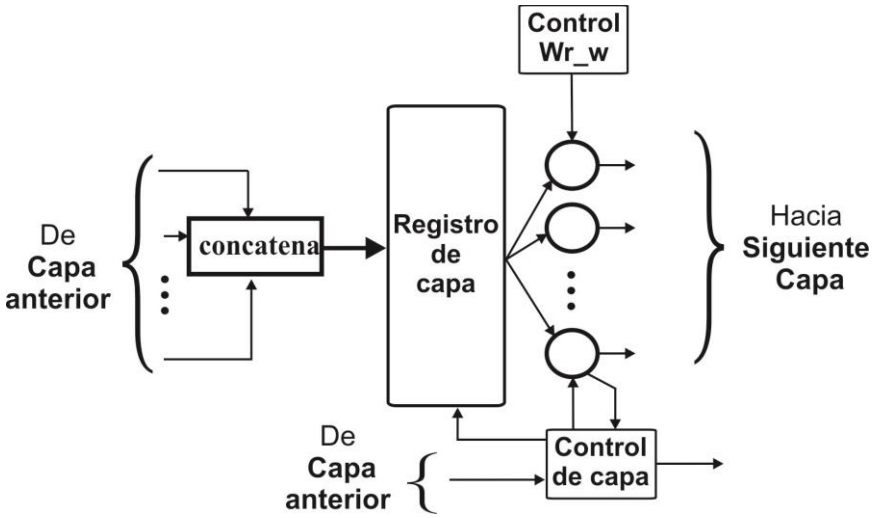

Figura 4.26 Conexión de una capa en una RNA.

En la Figura 4.27 se ve la arquitectura externa de dicho módulo.

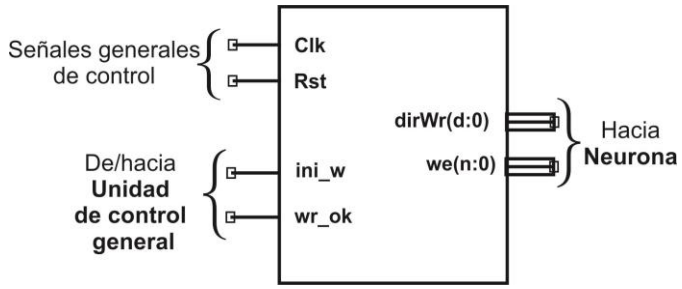

Figura 4.27 Símbolo del módulo **ControlWr_w.**

La descripción de las señales de entrada y salida se muestran en la Tabla 4.15.

Tabla 4.15 Descripción del módulo **ControlWr_w.**

Nombre	Modo	Descripción
clk	In	Señal de reloj global del sistema
rst	In	Señal de inicialización global del sistema
ini_w	In	Señal activa en alto, le indica al módulo cuando ya se ha recibido un nuevo dato correspondiente a los pesos sinápticos o umbral y que debe ser escrito en la RAM de la neurona
dirWr	Out	Indica la localidad de memoria en donde serán actualizados los pesos sináptico o umbral dentro de la neurona. El tamaño del bus está en función de las localidades que se desee direccionar. Para la arquitectura convencional d es igual al número de entradas y para la arquitectura basada en aritmética distribuida es igual 2^N , N es el número de entradas a la capa
we	Out	Permite habilita la memoria RAM de la neurona, donde m es el número de localidades de la RAM, donde r es el número de neuronas con que cuenta la capa, el tamaño depende del número de neuronas con los que cuente la capa, 1 bit para cada neurona.
wr_ok	In	Señal activa en alto indica que se realizó correctamente el proceso de escritura de todas las neuronas de esa capa

A cada neurona de la misma capa se le asigna $we(0)$ para la neurona0, $we(1)$ para la neurona1 y así sucesivamente con todas las neuronas de la misma capa.

Desde que todas las neuronas en una capa realizan las operaciones de forma concurrente, es posible enviar x_1 en el mismo instante de tiempo a todas las neuronas de esa capa para que realicen las operaciones de la función de propagación. Posteriormente se envía x_2 y así sucesivamente hasta terminar con todos los patrones almacenados en el módulo **Registro de capa**. En la Figura 4.28 se muestra su símbolo.

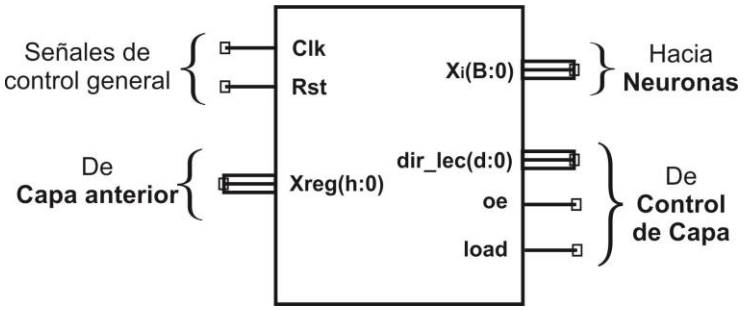

Figura 4.28 Símbolo del módulo **Registro de capa.**

El tamaño de las señales de entrada ***Xreg*** y ***Xi*** se configura a través de los parámetros de configuración del módulo y se ajustan al diseño de la capa en donde se esté instanciado este módulo. En la Tabla 4.16 se muestra la descripción de las señales del módulo Control de capa.

Tabla 4.16 Descripción de las señales del módulo **Registro de capa.**

Nombre	Modo	Descripción
clk	In	Señal de reloj global del sistema
rst	In	Señal de inicialización global del sistema
Xreg	In	Contiene el vector de todas las entradas a la capa, el tamaño depende del B y el número de entradas a la capa n; $h = B \times n$
load	In	Señal activa en alto, indica que se debe cargar los datos del vector de entrada de la capa anterior y guardarlo en una RAM distribuida
dir_lec	in	Indica la localidad de memoria de donde se va a leer el patrón
oe	In	Señal activa en alto, habilita la operación de lectura
Xi	out	Indica el patrón que será enviado a las neuronas de la misma capa

Una vez que se tiene precargado los pesos sinápticos en cada neurona y el vector de entrada de la red, se inicia la operación de la RNA habilitando al módulo **Control capa** mostrado en la Figura 4. 29. Se puede observar que el módulo **Control capa** tiene por función controlar todas las operaciones dentro de una capa. Genera señales de control que le permiten al módulo **Registro de capa** y las **Neuronas** en una capa llevar a cabo sus operaciones.

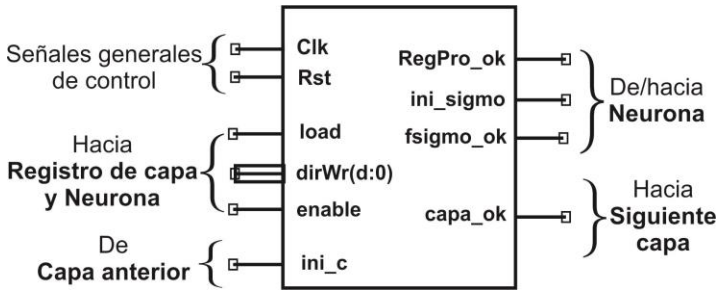

Figura 4. 29 Símbolo del módulo **Control de capa**

En la Tabla 4.17 se muestra la descripción de las señales de entrada y salida del módulo **Control de capa**.

Tabla 4.17 Descripción de las señales del módulo **Control de capa.**

Nombre	Modo	Descripción
clk	In	Señal de reloj global del sistema
rst	In	Señal de inicialización global del sistema
ini_c	In	Activa en alto la ejecución de este módulo
dirwr	Out	Para direccionar los patrones de la RAM del módulo **Registro de capa** y los pesos sinápticos de la RAM de las neuronas
enable	Out	Para habilita la lectura del módulo **Registro de capa** y RAM de la neurona del módulo **Pesos sinápticos**
RegPro_ok	In	Indica que las operaciones de la **Regla de propagación** se realizó correctamente y que es posible direccionar un nuevo dato
load	Out	Activa en alto la lectura del módulo la escritura del módulo **Registro de capa** para almacenar el vector de las entradas
ini_sigmo	Out	Señal para habilitar el módulo de la **Función de activación**
capa_ok	Out	Señal activa en alto para indicar el inicio de operación de la siguiente capa
fsigmo_ok	In	Activa en alto que la ejecución de la **Neuronas** de la capa se realizó correctamente.

Debido a que los patrones de entrada a la RNA son enviados desde la PC y la comunicación de esta con el FPGA es de forma serial, es necesario que se acomoden en un registro en el FPGA para que la RNA cuente con todos los datos disponibles para su uso cuando inicie sus operaciones. En la Figura 4.30 se muestra la estructura externa del módulo **preRegistro**. Este módulo solo se usa para la capa de entrada y es reemplazado para el resto de las capas con una operación de concatenación. Con lo que se permite tener todas las salidas de las neuronas de la capa anterior en un registro en un ciclo de reloj.

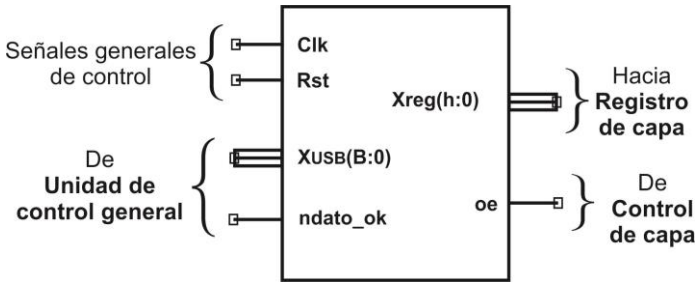

Figura 4.30 Símbolo del módulo **preRegistro**

Este módulo agrupa los descriptores del patrón de entrada en el siguiente formato $x_1, x_2, ..., x_n$

En la Tabla 4.18 se muestra la descripción de las señales de entrada y salida del módulo **preRegistro**.

Tabla 4.18 Descripción de las señales del módulo **preRegistro.**

Nombre	Modo	Descripción
clk	In	Señal de reloj global del sistema
rst	In	Señal de inicialización global del sistema
Xusb	In	Es el descriptor del patrón de entrada, proviene de la PC
load	In	Señal activa en alto para habilitar la escritura del registro ***Xreg***
ndato_ok	In	Indica que se ha recibido un nuevo descriptor del patrón de entrada de la PC y se debe almacenar en el registro
Xreg	out	Contiene todos descriptores del patrón de entrada a la RNA

Ejecución de una capa con Arquitectura Convencional

Una vez diseñada la RNA, el proceso de ejecución según la conexión mostrada en la Figura 4.26 en una capa de una RNA con Arquitectura convencional, sigue el siguiente orden

1 Almacenar los pesos sinápticos y umbral de cada neurona en la capa de la RNA. Para ello el módulo **ControlWr_w** permite habilitar y direccionar la RAM distribuida de cada neurona para actualizar sus campos. Este proceso es previo a la operación normal de la RNA.

2 El proceso de ejecución de la RNA está controlado por capas por el módulo **Control de capa**, el cual permite

2.1 Habilitar el módulo **Registro de capa** para cargar los vectores de salida de las neuronas de la capa anterior y poder disponer de ellos en la capa actual

2.2 Se direcciona las RAM distribuida de cada N**eurona** y **Registro de capa** para obtener los datos correspondientes w_i y x_i respectivamente

2.3 Cuando se realizan las operaciones de las entradas w_i y x_i en la **Neurona**, esta emite una señal activa en alto para indicarle al módulo de control que se requiere el siguiente par de entradas. Si aún no se han direccionado todas las localidades se regresa al paso 2.2

2.4 Se activa la señal de control que permite indicarle al módulo **Función de activación** que debe hacer el cálculo de la aproximación a la función sigmoidea. Una vez que se realiza la aproximación, se adecua el resultado para la siguiente capa.

2.5 En este punto todas las neuronas de la misma capa han realizado sus operaciones y emiten una señal activa en alto para indicarlo, las cuales se pasan por una compuerta AND y la salida de esta le indica al módulo **Control de capa** que las operaciones de todas las neuronas concluyeron.

2.6 Habilita con una señal activa en alto a la siguiente capa

3 Las salidas de las neuronas correspondientes a la función de activación, se concatenan en el orden $x_1, x_2, ..., x_n$ para adecuarlos a la siguiente capa.

4 Cuando se ejecutan todas las capas de la RNA, el módulo Control de capa de la capa de salida informa a la Unidad de control General que ya se cuentan con los datos de salda de la RNA y está envía la información a la PC.

Para el paso 2.1, si la capa que se está ejecutando es la primera, los vectores de entrada que se van a cargar corresponden a los patrones de entrada de la RNA y estos son tomados del módulo **preRegistro**.

Ejecución de una capa con Arquitectura basada en Aritmética Distribuida

La estructura externa del módulo **Registro de capa** de la Figura 4.28 permanece sin cambios y solamente se modifica su comportamiento. La principal función de este módulo consiste en presentar en un instante de tiempo t_0 el bit 0 de todos los patrones de entrada en el siguiente

formato $x_{N-1}[0],...,x_2[0],x_1[0],x_0[0]$ en el registro de salida x_i, posteriormente en un tiempo t_1 presenta el bit 1 y así sucesivamente hasta la salida del B bit $x_{N-1}[B],...,x_2[B],x_1[B],x_0[B]$.

El resto de los módulos presentados en la arquitectura convencional, permanecen sin cambios en la adaptación de la arquitectura basada en aritmética distribuida.

Una vez diseñada la RNA, el proceso de ejecución en una capa es similar al planteado en la arquitectura convencional y se muestra a continuación

1 Almacenar los pesos sinápticos y umbral de cada neurona en la capa de la RNA. Para ello el módulo **ControlWr_w** permite habilitar y direccionar la RAM distribuida de cada neurona para actualizar sus campos. Este proceso es previo a la operación normal de la RNA.

2 El proceso de ejecución de la RNA, está controlado por capas y está a cargo del módulo **Control de capa**, el cual permite

2.1 Habilitar el módulo **Registro de capa** para cargar los vectores de salida de las neuronas de la capa anterior y poder disponer de ellos en la capa actual

2.2 Se direcciona las RAM distribuida del **Registro de capa** cuyo valor es usado para direccionar la RAM del módulo **Pesos sinápticos**

2.3 Cuando se realizan las operaciones suma de la entrada actual con la anterior, la **Neurona** emite una señal activa en alto para indicarle al módulo de control que se requiere la siguiente entrada. Si aún no se han direccionado todas las localidades se regresa al paso 2.2

2.4 Se activa la señal de control que permite indicarle al módulo **Función de activación** que debe hacer el cálculo de la aproximación a la función sigmoidea. Una vez que se realiza la aproximación, se adecua el resultado para la siguiente capa.

2.5 En este punto todas las neuronas de la misma capa han llevado a cabo sus operaciones y emiten una señal activa en alto para indicarlo, las cuales se pasan por una compuerta AND y la salida de esta le indica al módulo **Control de capa** que las operaciones de todas las neuronas concluyeron.

2.6 Habilita con una señal activa en alto a la siguiente capa

El resto de las operaciones de la RNA se lleva a cabo de igual forma que en la arquitectura convencional.

Capítulo 5 Pruebas y resultados

En este capítulo se describen las pruebas que se realizaron al **Sistema para el modelado de RNA-MLP sobre lógica reconfigurable** y los resultados obtenidos de ellas. Se diseñaron RNA's de prueba para resolver dos problemas, que por sus características son frecuentemente usados para medir el desempeño de una red, la compuerta XOR y planta Iris.

Las implementaciones de las RNA's fueron realizadas combinando las diferentes arquitecturas modeladas para la regla de propagación y la función de activación, teniendo por objetivo comparar el desempeño de cada combinación y demostrar el fácil uso de la herramienta propuesta en el diseño de RNA's.

Para cada implementación se muestra un reporte que incluye:

- Porcentaje de reconocimiento.
- Recursos consumidos.
- Frecuencia máxima de operación.
- El diagrama esquemático resultante de las implementaciones

5.1 Problema XOR

5.1.1 Definición del problema XOR

El problema XOR (O exclusivo) es el clásico ejemplo de problema de clasificación no lineal en dos dimensiones, por lo que es típicamente usada para probar el desempeño de una RNA. Este problema cuenta con dos clases cuyos valores son 0 y 1, la Tabla 5.1 muestra los vectores de rasgos (descriptores) del problema XOR junto con su clasificación.

Tabla 5.1 Rasgos y clasificación del problema XOR.

Entradas		**Salida (clase)**
A	**B**	**Y**
1	1	0
1	0	1
0	1	1
0	0	0

El problema consiste en, dado un vector con dos rasgos, A y B, se debe determinar la clase, Y, de acuerdo a $Y = A'B + AB'$. El problema XOR puede ser solucionado mediante una RNA con dos neuronas en su capa de entrada, una capa oculta formada por dos neuronas y la capa de salida integrada por una neurona, tal como lo muestra la Figura 5.1.

5.1.2 Implementación en hardware de la RNA para el problema XOR

Una vez definida la estructura de la RNA a utilizar en el problema XOR, se realizaron implementaciones utilizando ambas arquitecturas de la regla de propagación, la convencional y la basada en Aritmética distribuida, combinándolas con todas las funciones de activación modeladas.

Estas implementaciones incluyen:

- Modelado e implementación de la RNA en el **Sistema embebido RNA-MLP**, mediante la instanciación de los diversos módulos que forman la herramienta propuesta.
- Adecuación de la **Interfaz de usuario**, para entrenar a la RNA y mostrar los resultados

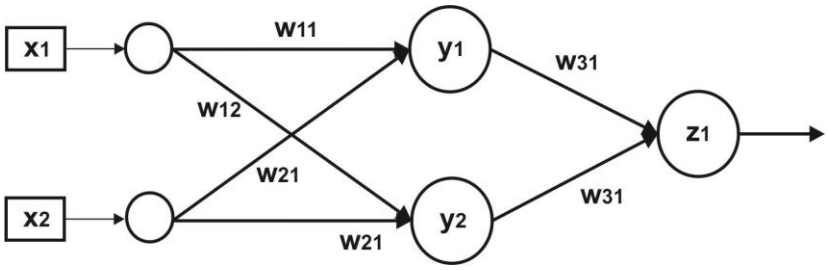

Figura 5.1 Definición de la RNA para el problema de la compuerta XOR.

5.1.2.1 Implementación RNA de la XOR en el sistema embebido RNA-MLP

El modelado e implementación de la RNA en el **Sistema embebido RNA-MLP** se lleva a cabo mediante la instanciación de los diferentes módulos ofrecidos por el sistema propuesto. En la instanciación, se definen los parámetros de configuración, que determinan las características del módulo, y las interconexiones, que determinan la interacción con los otros módulos que componen a la red. Los parámetros de configuración son los siguientes

- *Nw*, número de pesos sinápticos por neurona
- *Nx*, número de entradas a la red
- `Tamvecdat`, contiene el número de datos de entrada * el tamaño de los datos de entrada -1
- `tambuswe`, define el número de neuronas que contiene la red, se usa para saber el tamaño del bus para habilitar la RAM de cada neurona
- `Nneuronasc1`, número de neuronas de la capa oculta
- `Nneuronasc2` número de neuronas de la capa de salida
- `Nneuronas`, total de neuronas de la red
- `nbits`, número de bits que se van a emplear para representar las entradas a la red
- `nbitsw`, número de bits para representar los pesos sinápticos
- `direg`, para que el módulo **Control de capa** direccione la memoria del módulo **Registro de capa** (número de entradas a la red +1)
- `nlocr`, número de localidades de la memoria RAM para el módulo **Pesos sinápticos**; `2(Nx)+1`,

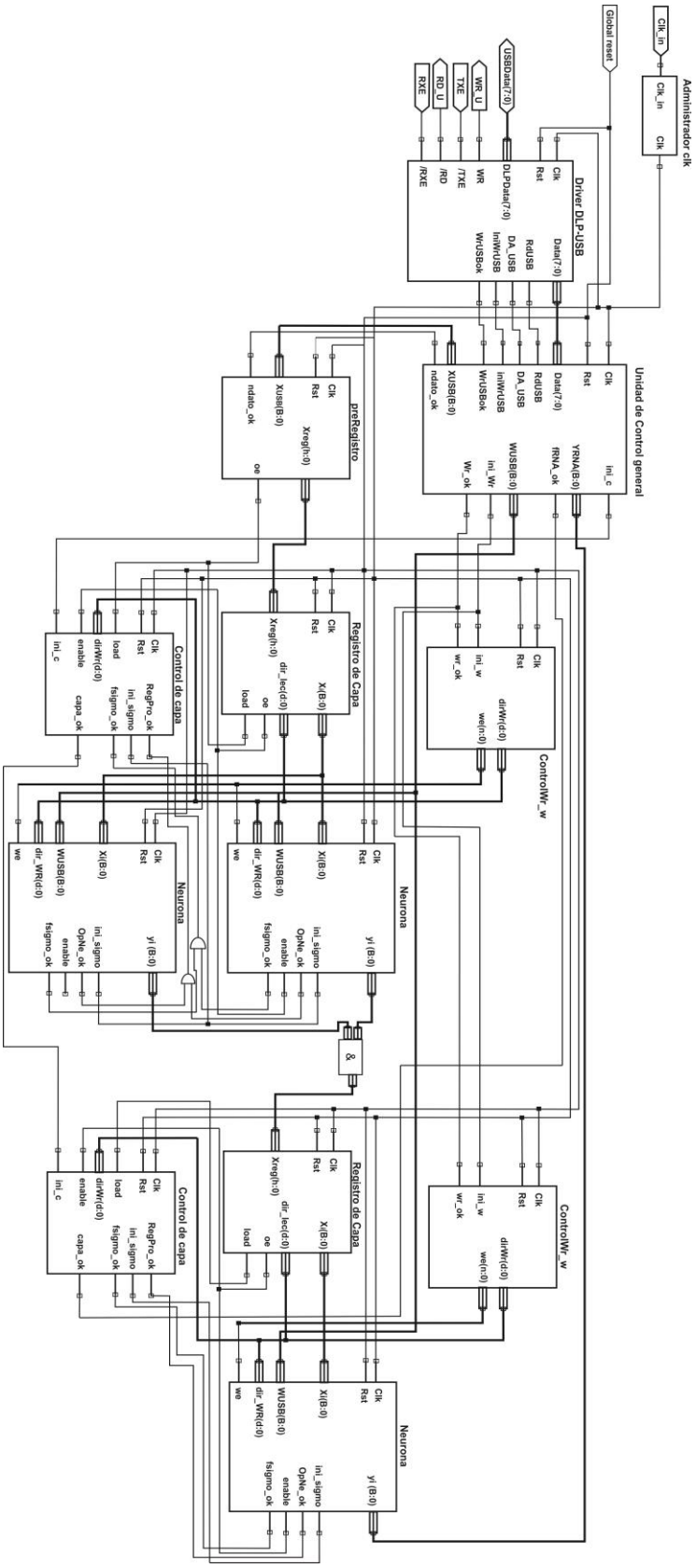

Figura 5.2 Diagrama de conexiones de la RNA para la compuerta XOR.

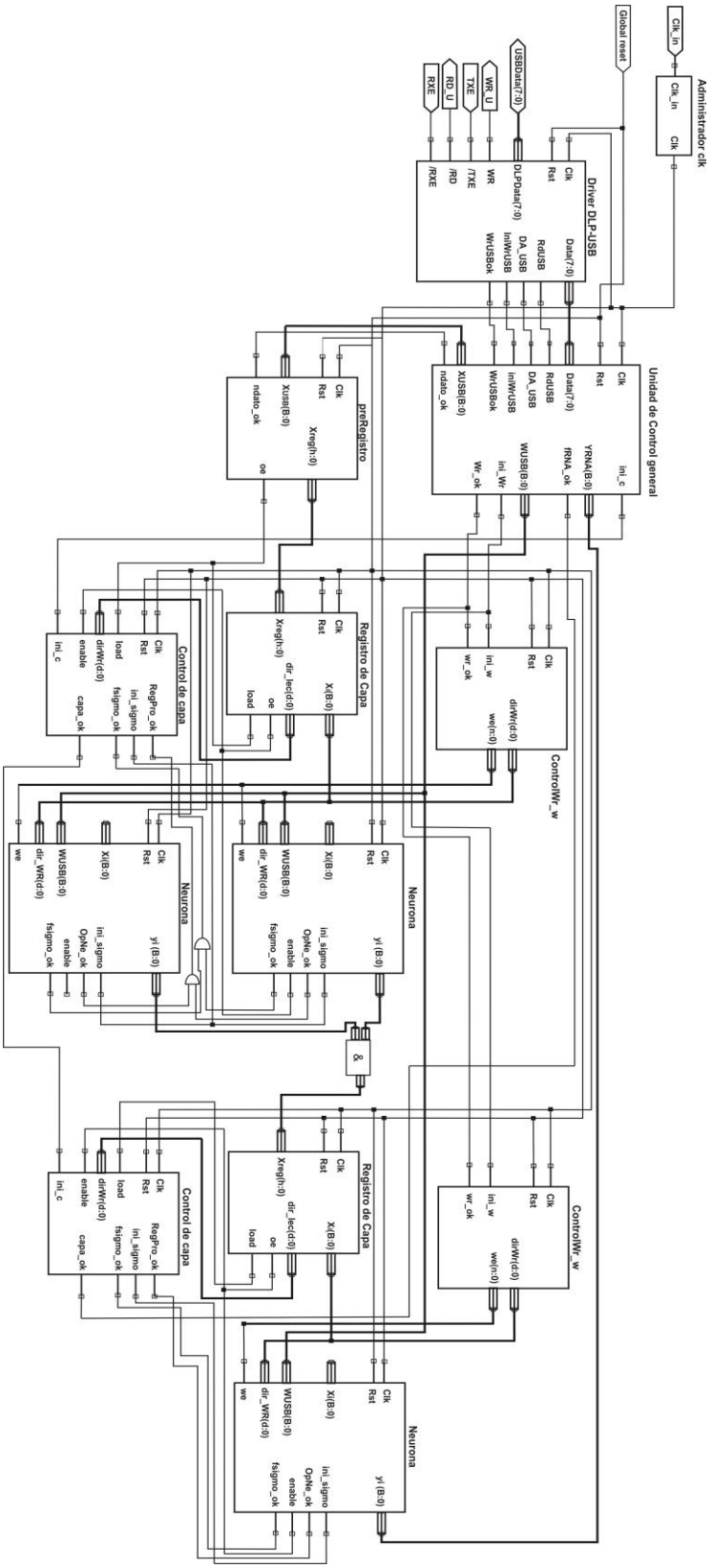

Figura 5.3 Conexiones de la RNA con Aritmética distribuida.

Es necesario aclarar que la instanciación de cada módulo permite introducir sus propios parámetros de configuración, por lo que las variables usadas para especificar ciertos campos, puede ser distintas para cada instancia. Para el caso de la RNA para la compuerta XOR, la definición de la neurona de la capa oculta es igual a las de la capa de salida, por lo que se usan las mismas variables para su configuración.

Una vez que se definieron los parámetros para cada elemento de la RNA para el problema de la compuerta XOR, se realiza la instancia de los diversos módulos para crear la RNA. En la Figura 5.2 se muestra las conexiones de la RNA empleando la Arquitectura convencional.

Las conexiones de los diversos módulos para crear la RNA empleando la Arquitectura basada en Aritmética distribuida se muestra en la Figura 5.3

Se observa que en ambas redes los módulos empleado, aunque representan diferentes arquitecturas (internamente son diferentes), sus entidades son iguales (externamente son iguales) y las interconexión entre los módulos permanecen igual para ambas redes. Esto permite fácilmente adecuar un nuevo modelado en cualquiera de los módulos que componen a la RNA. Los cambio se presentan en la señal de la dirección de la **Neurona**, ya que está no es controlada por el módulo **Control capa** si no por la salida del módulo **Registro de capa** y el registro de las entradas en la **Neurona** queda sin uso.

5.1.2.2 Adecuación de la Interfaz de usuario

La implementación y/o adecuación de las RNA's en la **Interfaz de usuario** se componen de tres fases, el *diseño de la RNA*, el *algoritmo de aprendizaje* y la *fase de pruebas*.

El diseño de la RNA tiene por finalidad realizar la configuración de la estructura de la RNA. La configuración se lleva a cabo mediante la especificación de los siguientes parámetros (ver Figura 5. 4):

- Número de capas de la RNA.
- Las neuronas por capa, se habilita un botón que permite introducir el número de neuronas por capa
- Número de entradas a cada capa, se realiza a través del botón *Cargar*
- Número de patrones con los que se va a entrenar la RNA
- Número de descriptores de cada patrón

- Número de salidas de la RNA

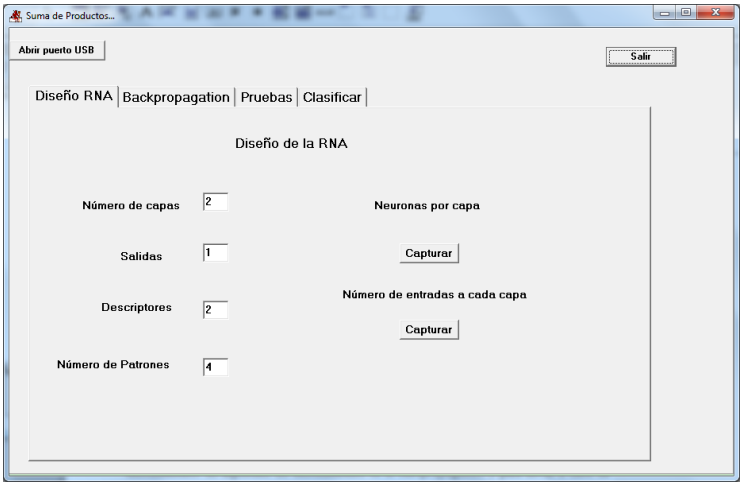

Figura 5. 4 Especificación de parámetros para el diseño de la RNA.

Una vez definida la estructura de la RNA se deben definir los parámetros del algoritmo de entrenamiento. El algoritmo de entrenamiento es el *backpropagation* y para llevar a cabo su función requiere la especificación de los siguientes parámetros:

- Ritmo de aprendizaje.
- Promedio de reducción de error final.
- Error cuadrático medio final.
- Valor inicial del valor cuadrático medio.
- Conjunto de entrenamiento.
- Inicialización de los pesos sinápticos.

La **Interfaz de usuario** permite la especificación de los primeros cuatro parámetros, mientas que los patrones de entrenamiento y los valores iniciales de los pesos sinápticos son pasados al algoritmo de entrenamiento a través de un archivo en formato "*.txt*".

En las Tabla 5. 2 y Tabla 5. 3 se muestran el conjunto de entrenamiento y valores iniciales de los pesos sinápticos respectivamente para la RNA del problema XOR.

Tabla 5. 2 Conjunto de entrenamiento para el problema XOR.

Patrón	Descriptor 1	Decriptor2	Valor deseado
1	1	1	0
2	0	0	0
3	1	0	1
4	0	1	0

Tabla 5. 3 Valores iniciales de pesos sinápticos y umbrales para el problema XOR.

Neurona	W1	W2	Umbral
1	0.1	-0.7	1
2	0.5	0.3	1
3	0.2	0.4	1

En la Figura 5.5 se muestra la sección de la **Interfaz de usuario** utilizada para la especificación de los parámetros para el entrenamiento de la RNA XOR.

Los resultados obtenidos para los pesos sinápticos y umbral después de que se ha ejecutado el algoritmo de entrenamiento para la RNA del problema XOR se muestran en la Tabla 5. 4

Tabla 5. 4 Resultado de los pesos sinápticos y umbral después del entrenamiento.

Neurona	W1	W2	Umbral
1	5.49	-4.33	3.29
2	-5.566	4.06	2.25
3	6.178	6.259	-3.059

Una vez concluido el proceso de entrenamiento, la **Interfaz de usuario** permite enviar los valores de los pesos sinápticos y umbrales obtenidos, con esta acción se termina de definir la estructura de la RNA modelada en el **sistema embebido RNA-MLP** y se adapta para dar solución al problema XOR.

Este proceso se realiza mediante el botón *Enviar*, presente en la pestaña de *Pruebas* de la interfaz (ver Figura 5.6), seleccionado previamente la arquitectura que se esté empleando, para que el envío de los datos sea en el orden adecuado.

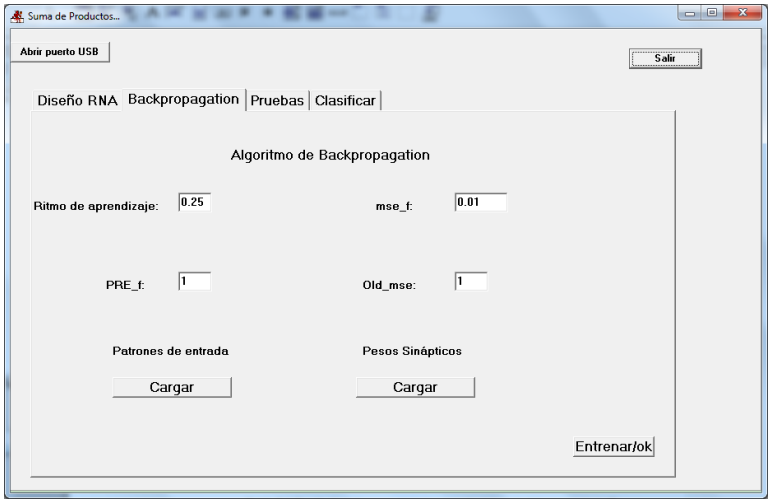

Figura 5.5 Especificación de los parámetros para el entrenamiento de la RNA.

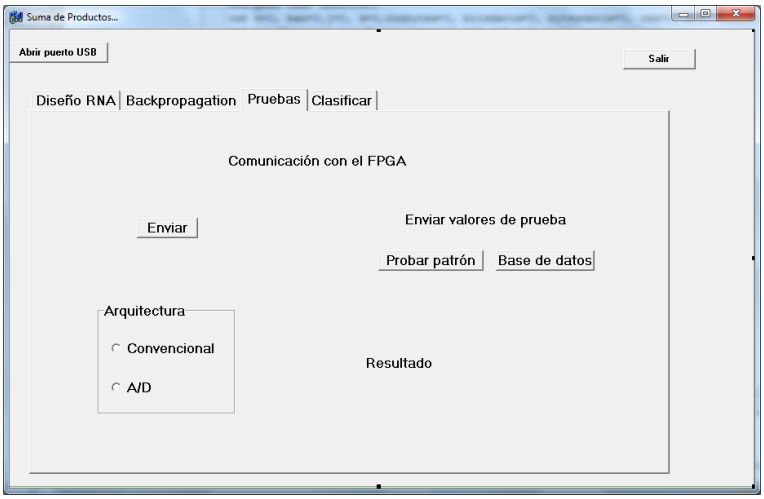

Figura 5.6 Pestaña para establecer fase de prueba de la RNA implementada en el FPGA.

5.1.2.3 Resultados para la RNA del problema XOR

Proceso de clasificación

Una vez definida y adaptada la RNA al problema XOR, y con la finalidad de determinar su desempeño, la **Interfaz de usuario** permite el envío de patrones de prueba, incluso aquellos no utilizados en el proceso de entrenamiento.

Para cumplir con esta tarea, se tiene como primera opción en envío de todos los patrones de entrenamiento, a la RNA implementada en el **Sistema embebido RNA-MLP**, para su evaluación, esta acción se lleva a cabo en forma automática usando el botón *Base de datos* presente en la interfaz mostrada en la Figura 5.6.

El resultado entregado por el **Sistema embebido RNA-MLP** es mostrado en el área de resultados, para el caso del problema XOR, la visualización de los resultados es en forma numérica y gráfica.

Una segunda opción es ofrecida al usuario, la cual se logra usando el botón *Probar patrón*, de la pestaña *Clasificar* (Figura 5.7), que permite presentar un patrón de prueba de cualquier valor a la RNA, esta acción solicita sean introducidos los descriptores, del patrón en estudio, mediante la ventana mostrada en la Figura 5.8

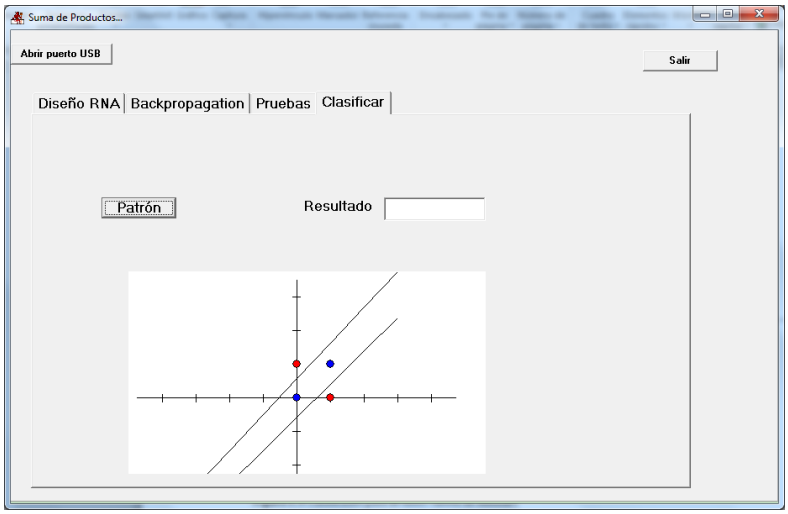

Figura 5.7 Resultados del clasificador de la RNA de la compuerta XOR.

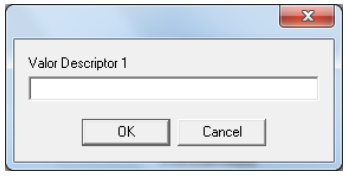

Figura 5.8 Ventana para introducir los descriptores de un patrón.

En la Figura 5.9 se muestran los resultados de diversos patrones de prueba.

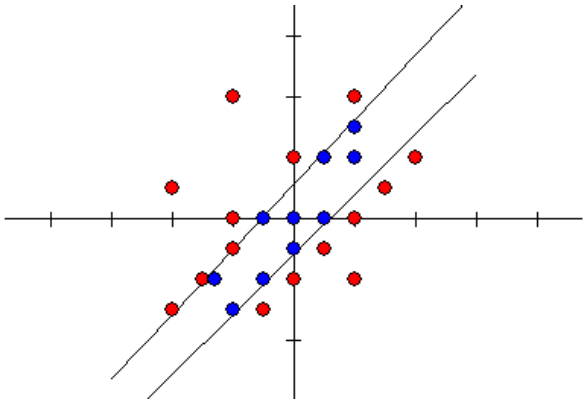

Figura 5.9 Clasificador para diversos valores de entrada.

Porcentaje de reconocimiento

Para establecer el porcentaje de reconocimiento de la RNA implementada en el **Sistema embebido RNA-MLP**, le fue presentado el conjunto de entrenamiento de la Tabla 5. 2, estos resultados pueden ser utilizados para determinar el desempeño de la RNA.

El porcentaje de reconocimiento obtenido cuando la estructura de la neurona está formada por una arquitectura convencional, en el cálculo de la regla de propagación, y las diferentes funciones de activación modeladas se muestra en la Tabla 5. 5, mientras que la Tabla 5. 6 lo muestra para el caso en que el cálculo de la regla de propagación se realiza mediante la arquitectura basada en aritmética distribuida.

Tabla 5. 5 Porcentaje de reconocimiento para la compuerta XOR con Arq. Convencional.

Técnica	Patrones			Porcentaje de reconocimiento
	Probados	Reconocidos	fallados	
Plan	4	4	0	100%
Alippi	4	4	0	100%
Ley A	4	4	0	100%
CRI	4	4	0	100%

Tabla 5. 6 Porcentaje de reconocimiento para la compuerta XOR con Arq. b/Aritmética Dist.

Técnica	Patrones			Porcentaje de reconocimiento
	Probados	Reconocidos	fallados	
Plan	4	4	0	100%
Alippi	4	4	0	100%
Ley A	4	4	0	100%
CRI	4	4	0	100%

Recursos utilizados

El reporte de los recursos consumidos por las implementaciones de la RNA para resolver los problemas planteados, se obtuvieron de la herramienta de diseño ISE Desing suit 12.1.

En la Tabla 5.7 se muestra los recursos utilizados por la Neurona y la implementación de la RNA con base en la arquitectura convencional para el problema de la compuerta XOR y en la

Tabla 5.8 se muestra cursos para la RNA implementada con la arquitectura basada en aritmética distribuida.

Tabla 5.7 Recursos utilizados por la implementación con Arq. Convencional de la XOR.

Recursos	Recursos utilizados				Recursos de la tarjeta
	Neurona	%	RNA	%	
Número de Slices	265	5	1099	23	4656
Número de Slices Flip Flops	276	2	1255	13	9312
Número de luts de 4 entradas	493	5	2002	21	9312
Lógicos	477		1954		
RAMs	16		48		

Tabla 5.8 Recursos utilizados por la implementación con Arq. b/Aritmética Dist.de la XOR

Recursos	Recursos utilizados				Recursos de la tarjeta
	Neurona	%	RNA	%	
Número de Slices	127	2	673	14	4656
Número de Slices Flip Flops	117	1	671	7	9312
Número de luts de 4 entradas	241	2	1235	13	9312
Lógicos	225		1187		
RAMs	16		48		

En la Tabla 5.9 se muestra el reporte de los recursos utilizados por las diferentes técnicas implementadas para la aproximación de la función de activación.

Tabla 5.9 Recursos utilizados por las funciones de activación.

Recursos	Técnica				Recursos de la tarjeta
	Plan	Alippi	Ley A	CRI	
Número de Slices	105	39	32	95	4656
Número de Slices Flip Flops	66	30	26	84	9312
Número de luts de 4 entradas	197	75	58	178	9312

Frecuencia de operación

La frecuencia máxima de operación y el tiempo de propagación del módulo **Función de activación** para las diferentes implementaciones de las técnicas para la función de activación se muestra en la Tabla 5.10.

Tabla 5.10 Reporte de tiempos de operación de la función de activación.

Técnicas implementadas	Frecuencia máxima de operación (MHz)	Periodo (nseg)	Ciclos de reloj
Plan	218.198	4.583	2
Alippi	126.936	7.878	3-9
Ley A	234.192	4.270	2
CRI	129.467	7.724	9

El reporte de la frecuencia máxima de operación y tiempo de propagación para la implementación de la RNA para el problema de la XOR se muestra en la Tabla 5.11.

Tabla 5.11 Tiempos de operación de la RNA para la XOR.

Arquitectura	Módulo	Frecuencia máxima de operación (MHz)	Periodo (nseg)	Ciclos de reoloj
Convencional	Neurona	82.686	12.09	5
	RNA	81.86	12.216	46
Basada en Aritmética distribuida	Neurona	132.398	7.553	3
	RNA	59.27	16.872	60

5.2 Problema de la planta Iris

5.2.1 Definición del problema de la planta Iris

R.A. Fisher propuso la base de datos que contienen tres clases que definen un tipo de planta llamado Iris [FIS36]. Estas clases son:

- Iris Setosa
- Iris Virgínica
- Iris Versicolor

La base de datos de la planta Iris consiste de 50 ejemplos de cada una de las tres clases que la conforman. Cada ejemplo o patrón está compuesto por cuatro rasgos que corresponden a las medidas de: largo y ancho de sépalo y largo y ancho de pétalo, dadas en centímetros. Una característica importante de esta base de datos es que dos de sus clases son linealmente separables y una es no linealmente separable. Cada patrón que representa una planta está formado por cuatro descriptores (ver Tabla 5.12).

Tabla 5.12 Descriptores de los patrones del problema de la Iris Planta.

Rasgo	Descripción (cm)
1	Largo del sépalo
2	Ancho del sépalo
3	Largo del pétalo
4	Ancho del pétalo

El problema de la planta Iris puede ser solucionado mediante una RNA con cuatro neuronas en su capa de entrada, una capa oculta formada por cuatro neuronas y la capa de salida integrada por dos neuronas, tal como lo muestra la Figura 5.10. Considerando el número de clases de la base de datos y la estructura de la RNA se definieron los valores de salida de acuerdo a como se muestra en la Tabla 5.13.

Tabla 5.13 Definición de las salidas para la RNA para el problema de la planta iris.

Clase	Descriptor				Valor esperado	
	LS	AS	LP	AP	t_1	t_2
Iris Setosa	$x[0]$	$x[1]$	$x[2]$	$x[3]$	0	0
Iris Versicolor	$x[0]$	$x[1]$	$x[2]$	$x[3]$	1	0
Iris Virgínica	$x[0]$	$x[1]$	$x[2]$	$x[3]$	0	1

5.2.2 Implementación de la RNA para el problema de la Iris Planta

Una vez definida la estructura de la RNA a utilizar en el problema de la planta iris, se realizaron implementaciones utilizando ambas arquitecturas de la regla de propagación, la convencional y la basada en Aritmética distribuida, combinándolas con todas las funciones de activación modeladas.

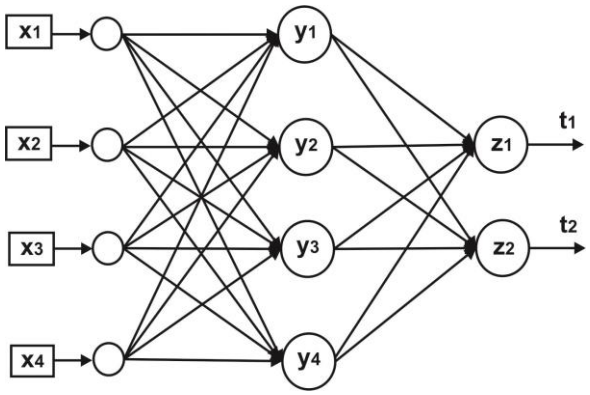

Figura 5.10 Definición de la RNA para el problema de la planta Iris.

Estas implementaciones incluyen:

- Modelado e implementación de la RNA en el **Sistema embebido RNA-MLP**, mediante la instanciación de los diversos módulos que forman la herramienta propuesta.
- Adecuación de la **Interfaz de usuario**, para entrenar a la RNA y mostrar los resultados

5.2.2.1 Implementación RNA de la planta Iris en el sistema embebido RNA-MLP

El modelado e implementación de la RNA en el **Sistema embebido RNA-MLP** se lleva a cabo mediante la instanciación de los diferentes módulos ofrecidos por el sistema propuesto. En la instanciación, se definen los parámetros de configuración, que determinan las características del módulo, y las interconexiones, que determinan la interacción con los otros módulos que componen a la red. Los parámetros que se deben de configurar son los mismos que para el RNA de la XOR.

Es necesario aclarar que la instanciación de cada módulo permite introducir sus propios parámetros de configuración, por lo que las variables usadas para especificar ciertos campos, puede ser distintas para cada instancia. Para el caso de la RNA para la compuerta XOR, la definición de la neurona de la capa oculta es igual a las de la capa de salida, por lo que se usan las mismas variables para su configuración. Una vez que se definieron los parámetros para cada elemento de la RNA para el problema de la planta Iris, se realiza la instancia de los diversos módulos para crear la RNA empleando la arquitectura basada en Aritmética distribuida, el diagrama de conexiones de la RNA queda definido como se muestra en la Figura 5.12.

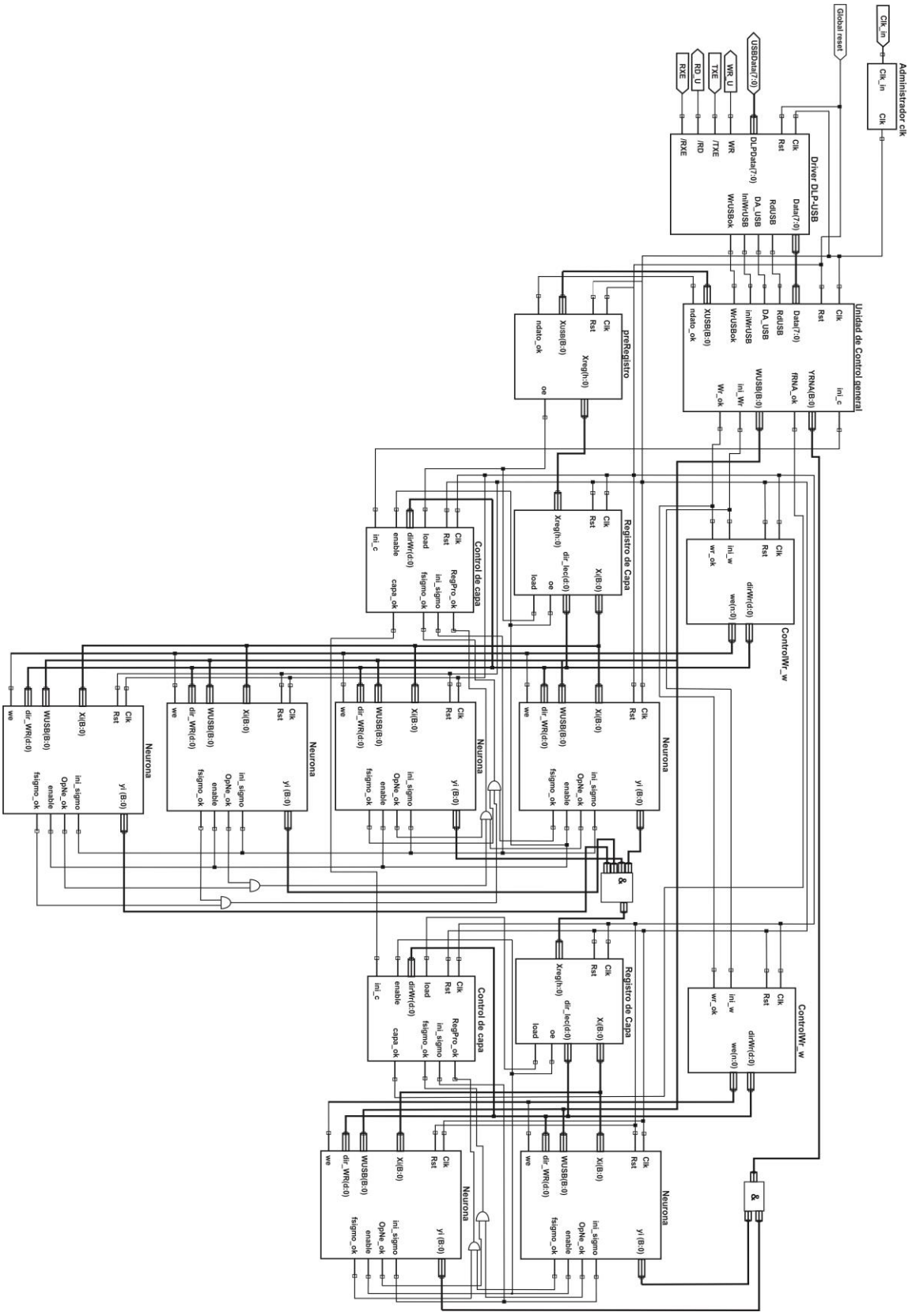

Figura 5.11 Diagrama de conexiones de la RNA con arquitectura convencional para la planta Iris.

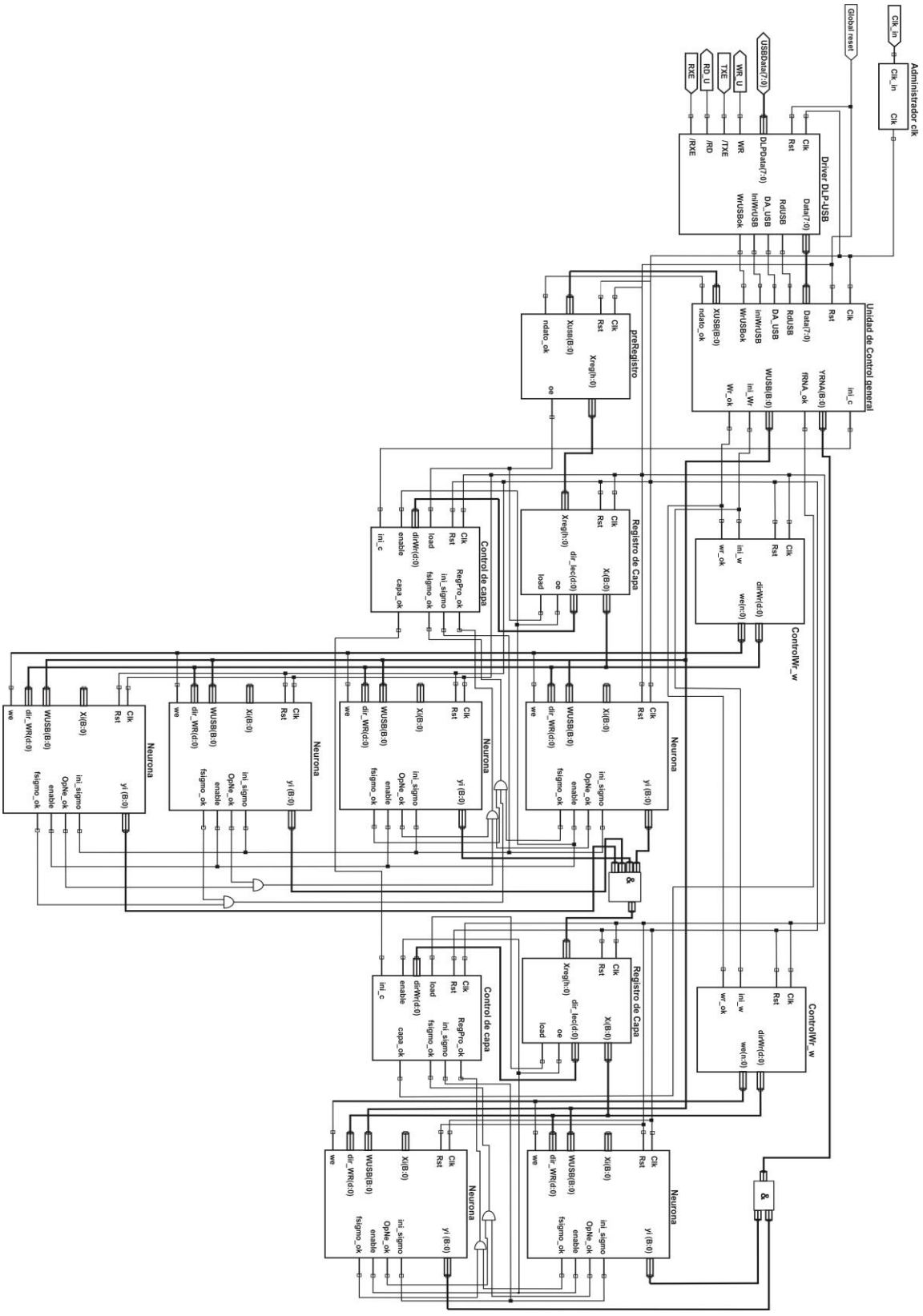

Figura 5.12 Conexiones para la RNA con aritmética distribuida para la planta Iris.

5.2.2.2 Adecuación de la interfaz de usuario para la planta Iris

La implementación y/o adecuación de las RNA's en la **Interfaz de usuario** se componen de tres fases, el *diseño de la RNA*, el *algoritmo de aprendizaje* y la *fase de pruebas*.

La configuración de la RNA de la planta Iris se realizó con los parámetros mostrados en la Figura 5.13.

Una vez definida la estructura de la RNA se deben definir los parámetros del algoritmo de entrenamiento. El algoritmo de entrenamiento es el *backpropagation* y para llevar a cabo su función requiere la especificación de los siguientes parámetros:

- Ritmo de aprendizaje.
- Promedio de reducción de error final.
- Error cuadrático medio final.
- Valor inicial del valor cuadrático medio.
- Conjunto de entrenamiento.
- Inicialización de los pesos sinápticos.

La **Interfaz de usuario** permite la especificación de los primeros cuatro parámetros, mientas que los patrones de entrenamiento y los valores iniciales de los pesos sinápticos son pasados al algoritmo de entrenamiento a través de un archivo en formato "*.txt*". En la Figura 5.14 se muestra la sección de la **Interfaz de usuario** utilizada para la especificación de los parámetros para el entrenamiento de la RNA de la planta Iris.7

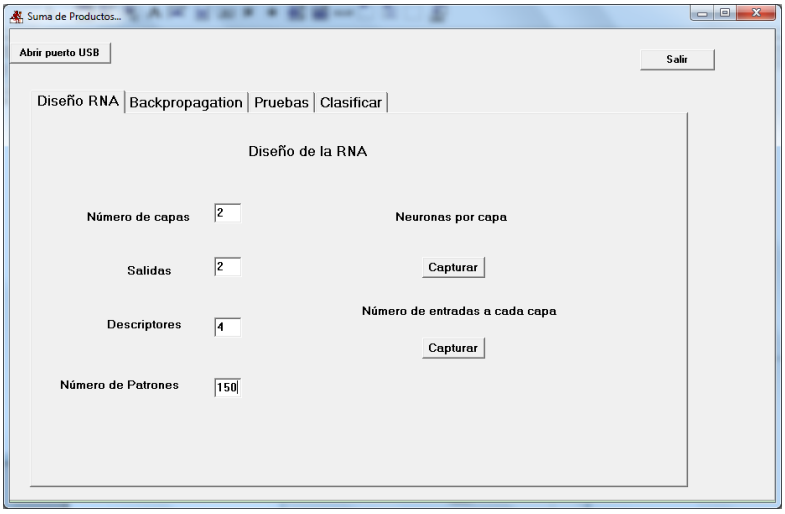

Figura 5.13 Especificación de parámetros para el diseño de la RNA de la panta Iris.

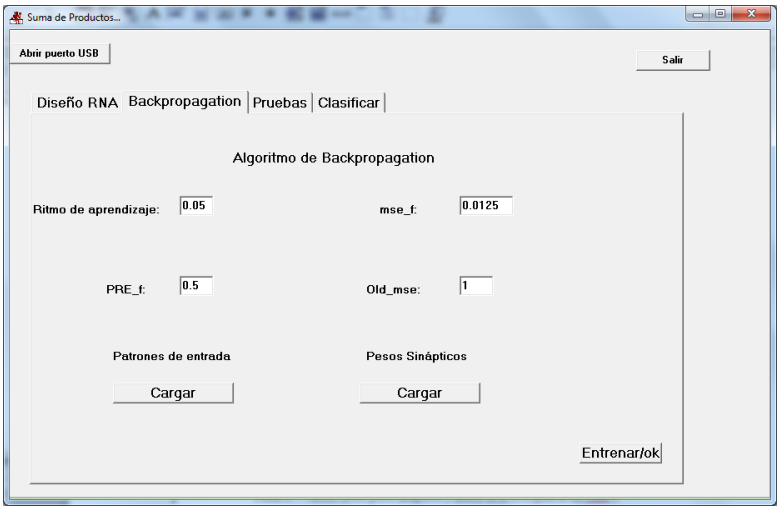

Figura 5.14 Especificación de los parámetros para el entrenamiento de la RNA de la planta iris.

Las pruebas de la RNA para el problema de la planta Iris se realizaron con una base de datos que contiene 50 patrones de prueba para cada clase, por lo que el total de patrones de entrada son de 150 patrones. Los resultados obtenidos para los pesos sinápticos y umbral después de que se ha ejecutado el algoritmo de entrenamiento para la RNA de la planta Iris se muestran en la Tabla 5.14.

Tabla 5.14 Valores de los pesos sinápticos y umbra de la RNA para la planta iris.

	Neurona	w_1	w_2	w_3	w_4	θ
Capa oculta	1	-0.402705	-0.402705	-0.402705	-0.402705	-0.402705
	2	-25.80956	-25.80956	-25.80956	-25.80956	-25.80956
	3	-0.407004	-0.407004	-0.407004	-0.407004	-0.407004
	4	-25.80956	-25.80956	-25.80956	-25.80956	-25.80956
Capa de salida	1	17.264579	17.264579	17.264579	17.264579	17.264579
	2	-17.80738	-17.80738	-17.80738	-17.80738	-17.80738

Una vez concluido el proceso de entrenamiento, la **Interfaz de usuario** permite enviar los valores de los pesos sinápticos y umbrales obtenidos, con esta acción se termina de definir la estructura de la RNA modelada en el **sistema embebido RNA-MLP** y se adapta para dar solución al problema de la planta Iris. Este proceso se realiza mediante el botón *Enviar*,

presente en la pestaña de *Pruebas* de la interfaz (ver Figura 5.15), seleccionado previamente la arquitectura que se esté empleando, para que el envío de los datos sea en el orden adecuado.

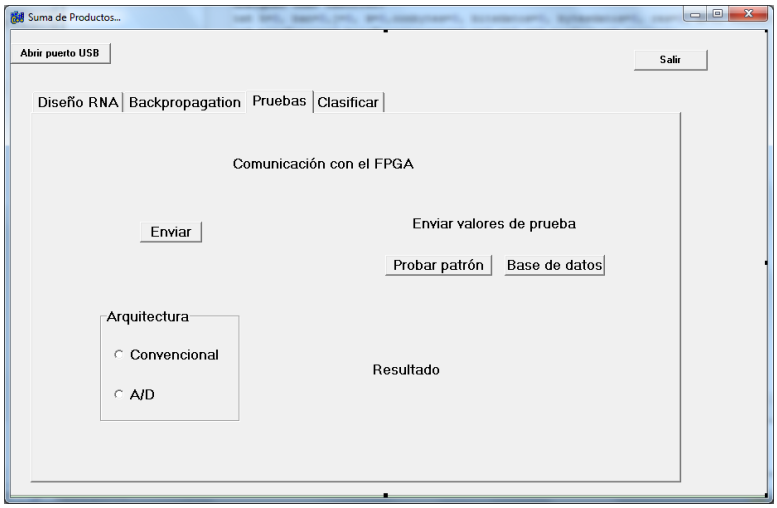

Figura 5.15 Pestaña para establecer fase de prueba de la RNA implementada en el FPGA.

5.2.2.3 Resultados para la RNA del problema de la planta iris

Una vez definida y adaptada la RNA al problema de la planta Iris, y con la finalidad de determinar su desempeño, la **Interfaz de usuario** permite el envío de patrones de prueba, incluso aquellos no utilizados en el proceso de entrenamiento.

Para cumplir con esta tarea, se tiene como primera opción en envío de todos los patrones de entrenamiento, a la RNA implementada en el **Sistema embebido RNA-MLP**, para su evaluación, esta acción se lleva a cabo en forma automática usando el botón *Base de datos* presente en la interfaz mostrada en la Figura 5.16.

El resultado entregado por el **Sistema embebido RNA-MLP** es mostrado en el área de resultados, e indica el número de clases reconocidas para la base de datos de los patrones enviados

Para el caso en el que se pruebe un patrón el resultado se indica en una ventana con la clase a la que pertenece dicho patrón. En la Figura 5.17 se muestra un ejemplo para la clase Versicolor.

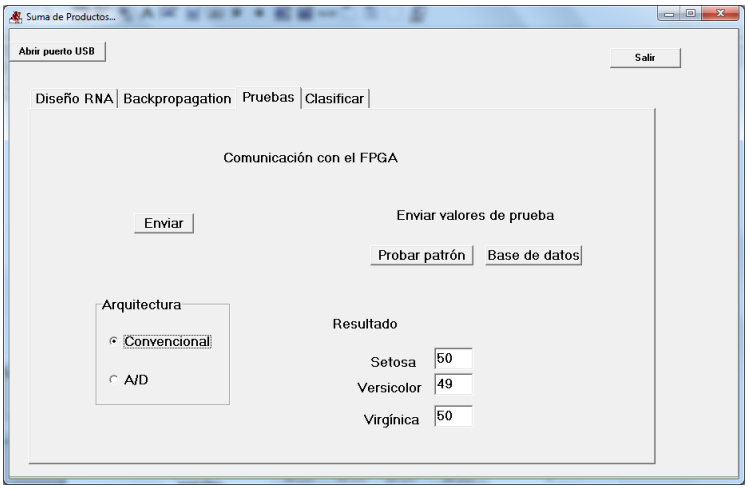

Figura 5.16 Pestaña de pruebas para la RNA con Aritmética distribuida.

Figura 5.17 Mensaje de salida para la prueba de un patrón.

Porcentajes de reconocimiento

En la Tabla 5.15 se muestran los resultados de la RNA con base en Arquitectura convencional para las diferentes técnicas empleadas para la aproximación de la función de activación, así como el porcentaje de reconocimiento.

Tabla 5.15 Porcentaje de reconocimiento la planta Iris con arquitectura convencional.

Clase	Técnica				Porcentaje de reconocimiento por clase
	Plan	Alippi	Ley A	CRI	
Iris Setosa	50	50	50	50	100%
Iris Versicolor	49	49	49	49	98%
Iris Virgínica	50	50	50	50	100%
Porcentaje de reconocimento total (%)	99.33	99.33	99.33	99.33	

Los resultados para la Arquitectura basada en aritmética distribuida con las diferentes técnicas empleadas para la aproximación de la función de activación, así como el porcentaje de reconocimiento se muestran en la Tabla 5.16.

Tabla 5.16 Porcentaje de reconocimiento la planta Iris con arquitectura b/Aritmética distribuida.

Clase	**Técnica**				**Porcentaje de reconocimiento por clase**
	Plan	**Alippi**	**Ley A**	**CRI**	
Iris Setosa	50	50	50	50	100%
Iris Versicolor	49	49	49	49	98%
Iris Virgínica	50	50	50	50	100%
Porcentaje de reconocimento total (%)	99.33	99.33	99.33	99.33	

En la Tabla 5.17 se muestra algunos patrones de prueba y las salidas que arrojaron la RNA's para el problema de la planta Iris con las dos arquitecturas desarrolladas

Tabla 5.17 Resultados de la RNA para el problema de la planta Iris .

Clase	**Descriptor**				**Valor esperado**		**Arq. Convencional**		**Arq. Basada en aritmética distribuida**	
	LS	AS	LP	AP	t_1	t_2	t_1'	t_2'	t_1''	t_2''
Iris Setosa	5.4	3.9	1.7	0.4	0	0	0	0	0	0
	5.4	3.4	1.7	0.2	0	0	0	0	0	0
Iris Versicolor	5.7	2.8	4.5	1.3	1	0	1	0	0.984	0
	5.9	3.2	4.8	1.8	1	0	0.984	0	1	0
Iris Virgínica	7.6	3.0	6.6	2.1	0	1	0.0625	0.906	0.0625	0.89
	6.9	3.2	5.7	2.3	0	1	0	1	0	1

Recursos utilizados

Los recursos utilizados por la implementación de la RNA para el problema de la planta Iris con arquitectura convencional se muestran en la Tabla 5.18 y en la Tabla 5.19 se muestra el reporte para la arquitectura basada en aritmética distribuida.

Tabla 5.18 Recursos utilizados por la implementación con Arq. Convencional de la planta Iris.

Recursos	Recursos utilizados				Recursos de la tarjeta
	Neurona	%	RNA	%	
Número de Slices	272	5	2061	44	4656
Número de Slices Flip Flops	266	2	2166	23	9312
Número de luts de 4 entradas	506	5	3818	41	9312
Lógicos	490		3722		
RAMs	16		96		

Tabla 5.19 Recursos utilizados por la implementación con Arq. b/Aritmética Dist. de la planta Iris.

Recursos	Recursos utilizados				Recursos de la tarjeta
	Neurona	%	RNA	%	
Número de Slices	220	4	1753	37	4656
Número de Slices Flip Flops	180	1	1589	17	9312
Número de luts de 4 entradas	419	4	3236	34	9312
Lógicos	371		2948		
RAMs	48		288		

Frecuencia de operación

El reporte de la frecuencia máxima de operación y tiempo de propagación para la implementación de la RNA para el problema de la planta Iris se muestra en la Tabla 5.20.

Tabla 5.20 Tiempos de operación de la RNA para la planta Iris

Arquitectura	Módulo	Frecuencia máxima de operación (MHz)	Periodo (nseg)	Ciclos de reloj
Convencional	Neurona	82.686	12.094	5
	RNA	72.233	13.844	92
Basada en Aritmética distribuida	Neurona	135.704	7.369	3
	RNA	43.287	23.01	154

Capítulo 6 Conclusiones y trabajos futuros

6.1 Conclusiones

Considerando la estructura que una RNA guarda y las principales técnicas utilizadas para el desarrollo de los diferentes elementos que componen una RNA, se diseñó e implementó un sistema modular que representa un ambiente que se puede utilizar en el desarrollo y evaluación de los diferentes elementos que componen a una RNA.

Además, el sistema desarrollado representa una herramienta que permite la implementación de un MLP en un FPGA y su adaptación a diversas aplicaciones, con lo cual se obtienen los siguientes beneficios

- Contar con una herramienta que proporcione los elementos necesarios para que el proceso de implementación y adaptación de una RNA a una aplicación en específico se realice de una manera simple y eficaz.
- Disminución del tiempo de desarrollo, ya que se ofrecen diversos módulos que permiten crear una RNA a partir del elemento básico, la **Neurona**

- Probar rápidamente diversos algoritmos para los elementos que componen una RNA, para lo cual se diseñaron elementos que conservan la misma estructura externa y se modifica solamente su comportamiento interno
- Hacer una comparativa en tiempo y recursos consumidos por cada una de las implementaciones, es posible verificar los recursos y tiempo consumido por los elementos que componen la RNA gracias a la modularidad del sistema
- Portabilidad del sistema, debido a que no se usan elementos propios de un FPGA
- Escalable, el sistema se puede implementar en un nuevo FPGA de nueva generación
- De fácil uso, debido a que en la implementación de una nueva RNA el usuario debe efectuar un mínimo de configuraciones

Con la finalidad de mostrar el uso del sistema propuesto, fue aplicado en la solución de problemas usados para determinar el desempeño de una RNA, estos problemas son el problema XOR y el problema de la planta Iris.

Al comparar los resultados obtenidos, en el parámetro de porcentaje de reconocimiento, con implementaciones en software que resolvían los mismos problemas, se obtuvieron los mismos resultados, es decir el porcentaje de reconocimiento entregado por las RNAs implementadas por el sistema propuesto coincide con el entregado por las RNAs implementadas en software.

Considerando las diferentes técnicas empleadas en el modelado de los elementos que integran a una neurona y al aplicarlas en la solución de los problemas mencionados, la combinación de una arquitectura convencional, en la regla de propagación, con la ley A, en la función de activación, es la que ofrece mejores prestaciones en el parámetro de velocidad. Mientras que la combinación de una arquitectura basada en aritmética distribuida, en la regla de propagación, con la ley A, en la función de activación, es la que consume menos recursos.

Durante el desarrollo de este trabajo se determinó que una simulación conceptual (en software) proporciona información del valor máximo y mínimo que pueden tomar los pesos sinápticos lo que permite determinar el número de bits utilizados para representarlos. De esta forma se hace más eficiente el uso de los recursos, ya que el reporte de recursos y tiempos se ve afectado por la representación que se adopte para definir los datos de la aplicación.

Finalmente, para el cálculo de la suma de productos, fue necesario adecuar un sistema que permite que la suma no se incremente 1 bit en cada operación. Esto permite reducir los recursos utilizados para la representación y se basa en un análisis para determinar el valor máximo y mínimo del registro del acumulador en la **Regla de propagación** para todas las neuronas y que está operación no se vea afectada en pérdidas de información.

6.2 Trabajos futuros

A partir de los resultados obtenidos y las limitantes de la herramienta se sugieren los siguientes trabajos de investigación

- Con la finalidad de optimizar al sistema propuesto, se plantea modelar, mediante técnicas alternativas, a los módulos propuestos e implementados en este trabajo de tesis. Esto también ampliara el repertorio de módulos del sistema
- Ampliar la herramienta para que permita diseñar y modelar otros tipos de RNA
- Extender esta herramienta hacia otros modelos neuronales como las memorias asociativas

Bibliografía

[1]Abad, J., Linares-Flores, J., Guzman-Ramirez, E., and Sira-Ramirez, H. "Generalized Proportional Integral Tracking Controller for a Single Phase Multilevel Cascade Inverter: A FPGA Implementation". IEEE Transactions on Industrial Informatics, Vol. 10. No. 99. PP. 256-266, 2013.

[2]Adetiba, E., Ibikunle, F.A., Daramola, S.A., Olajide, A.T., "Implementation of Efficiente Multilayer Perceptron on Field Progammable Gate Array Chip", Intenational Journal of Engineering & Technology IJET-IJENS, Vol. 14. PP. 152-159. 2014.

[3]Alippi, C., Storti-Gajani, G.,"Simple Approximation of Sigmoidal Sunctions: Realistic of Digital Neural Networks Capable of Learnig". Dipartimento di Elettronica del Politecnico di Milano. IEEE. PP.1505-1508. 1991.

[4]Amari S., "Learning patterns and pattern sequences by self-organizing nets of threshold elements", IEEE Transactions on Computers, Vol. C-21, 11, PP. 1197-1206. 1972.

[5]Amin, H., Curtis, K.M., Hayes-Gill, B.R., "Piecewise linear approcimation applied to nonlinear function of a neural network".IEEE Proceedings Vol. 144. No. 6. PP. 313-317. 1997.

[6]Anderson, J. A., "A simple neural network generating an interactive memory", Mathematical Biosciences, Vol. 4, PP. 197-220, 1972.

[7]Anderson, J. A., and Rosenfeld, E. (Eds.), Neurocomputing: foundations of research. Cambridge: MIT Press. 1988.

[8]Basterretxea, J.M., Tarela and I. del Campo,"Approximation of sigmoid function and the derivative for hardware implementation of artificial neurons". IEEE Proceedings No. 20030607. 2003.

[9]Batlle, J., Martí, J., Ridao, P., and Amat, J. "A New FPGA/DSP-Based Parallel Architecture for Real-Time Image Processing". Real-Time Imaging, Vol. 8. No. 5. PP. 345-356. 2002.

[10]Battezzati, N., Sterpone, L., Violante, M. "Reconfigurable Field Programmable Gate Arrays for Mission Critical Aplications". Springer. Berlin. 2011.

[11]Bhargav, H., Nataraj, K.R., "Mapping FPGA to Field Programmable Neural Network Array (Fpnna)". Intenational Journal of Engineering & Technology IJET-IJENS. Vol. 2. PP.21-27. 2012.

[12]Bishop, C.M., "Neural Networks and their aplications". Rev.Sci. Intrument.,65,6, PP. 1803-1832, 1994.

[13]Bonato, V., Marques, E., and Constantinides, G. A. "A Parallel Hardware Architecture for Scale and Rotation Invariant Feature Detection". IEEE Transactions on Circuits and Systems for Video Technology, Vol. 18. No. 12. PP. 1703-1712. 2008.

[14]Bryson A. E. and Ho Y. C. "Applied Optimal Control. Waltham". MA: Blaisdell. 1969.

[15]Caudill, M. y Butler, Ch. "Understanding neural networks: computer explorations". MIT Press. 1992.

[16]CHU, P. P. "FPGA Prototyping by VHDL Examples". Wiley-Interscience. 2008.

[17]Dinu A., Cirstea M. N., and Cirstea S. E. "Direct Neural-Network Hardware-Implementation Algorithm", IEEE Transactions on Industrial Electronics, Vol. 57. No 5. PP. 1845-1848. 2010.

[18]Eldredge G.J., Hutchings L.B., "RRANN: A Hardware Implementation of the Backpropagation Algorithm Using Reconfigurable FPGAs", IEEE international Conference on Neural Networks, 1994.

[19]Eklund, S.V. "A massively parallel architecture for distributed genetic algorithms". Parallel Computing, Vol. 30. No. 5-6. PP. 647-676. 2004.

[20]Farrugia, N., Mamalet, F., Roux, S., Fan Yang, and Paindavoine, M. "Fast and Robust Face Detection on a Parallel Optimized Architecture Implemented on FPGA". IEEE Transactions on Circuits and Systems for Video Technology, Vol. 19. No. 4. PP. 597-602. 2009.

[21]Feldman, J. A. y Ballard, D. H. "Connectionists models and its properties". Cognitive Science, Vol. 6. No. 3. PP.205-254. 1982.

[22]Freeman, J. A. and Skapura, D. M. Redes Neuronales. Algoritmos, aplicaciones y técnicas de propagación. Addison-Wesley. 1993.

[23]Fukushima K., "Neocognitron: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position", Biological Cybernetics, Vol. 36. PP. 193-202. 1980.

[24]Fukushima k., Miyake S., Ito T., "Neocognitron: A neural network model for a mechanism of visual pattern recognition". IEEE Transactions on Systems, Man, and Cybernetics. Vol. SMC-13. PP. 826–834. 1983.

[25]Gadea, R., Cerda, J., Ballerter, F., Mocholi, A.,"Artificial Neural Network Implementation on a single FPGA of a Pipelined On-Line Backpropagation", In Proceedings of the 13th international symposium on System synthesis, PP. 225-230. 2000.

[26]Gangadhar, M., and Bhatia, D. "FPGA based EBCOT architecture for JPEG 2000". Microprocessors and Microsystems, Vol. 29. No. 8–9. PP 363-373. 2005.

[27]Garcia I., Guzmán-Ramírez E. and Pacheco C. "CoLFDImaP: a web-based tool for teaching of FPGA-based digital image processing in undergraduate courses". Journal on Computer Applications in Engineering Education, John Wiley & Sons. Vol 23. PP. 92-108. 2013.

[28]Girau, B. "FPNA: concepts and properties". In FPGA Implementations of Neural Networks, Springer, PP. 62–101. 2006.

[29]Gomperts, A., "Development and Implementation of Parameterized FPGA Based General Purpose Neural Networks for Online Applications". Technische Universiteit Eindhoven. 2009.

[30]Gori M. and Tesi A. "On the problem of local minima in backpropagation". IEEE, Transactions on Patterns Analysis and Machine Intelligence, Vol. 1. No. 14. PP. 76-86. 1992.

[31]Guzmán-Ramírez, E., and García I. "Using the Project-Based Learning Approach for Incorporating an FPGA-Based Integrated Hardware/Software Tool for Implementing and Evaluating Image Processing Algorithms Into Graduate Level Courses". Journal on Computer Applications in Engineering Education, John Wiley & Sons, Vol. 21. No. S1. PP. E73-E88. 2012.

[32]Hebb, D. "The organization of Behaviour". Weley. 1949.

[33]Hecht-Nielsen, R. "Neurocomputing: picking the human brain". IEEE Spectrum, Vol. 25, No. 3, PP. 36-41 1988.

[34]Hecht-Nielsen R. "Theory of backpropagation neural network". In Proc. IEEE-IJCNN89, Vol I. PP. 593-605. 1989.

[35]Hecht-Nielsen, R. "Neurocomputing". Addison-Wesley. 1990.

[36]Hilera, G.J.R. y Martínez, H.V.J. "Redes Neuronales Artificiales: fundamentos, modelos y aplicaciones". Ra-Ma, Libreria y Editorial Microninfomatica. 1995.

[37]Himavathi, S., Anitha, D., and Muthuramalingam, A. "Feedforward Neural Network Implementation in FPGA Using Layer Multiplexing for Effective Resource Utilization". IEEE. Transactions on Neural Networks, Vol. 18. No. 3. PP. 1045-9227. 2007.

[38]Hinton G. E. and Sejnowski T. J., “Optimal perceptual inference”, in Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, PP. 448-453, 1983.

[39]Hiromoto, M., Sugano, H., and Miyamoto, R. “Partially Parallel Architecture for AdaBoost-Based Detection With Haar-Like Features”. IEEE Transactions on Circuits and Systems for Video Technology, Vol. 19. No 1. PP. 41-52. 2009.

[40]Hoogerheide,L.,”Essays on neural networks sampling methods and instrumental variables”. Erasmus University Rotterdam. PP. 31. 2006.

[41]Hopfield, J.J. “Neural networks and physical systems with emergent collective computational abilities”, in Proceedings of the National Academy of Sciences, Vol. 79. PP. 2554-2558. 1982.

[42]Hu H., Huang J., Xing J., Wang W., “Key Issues of FPGA Implementation of Neural Networks“, Second International Symposium on Intelligent Information Technology Application, 2008.

[43]Hubel, D. H. and Wiesel, T. N., “Receptive fields, binocular interaction and functional architecture in cat's visual cortex”, Journal of Physiology, Vol. 160, No. 9, PP. 106-154, 1962.

[44]Kohonen T., “Correlation Matrix Memories”, IEEE Transactions on Computers, Vol.C-21. PP. 353-359. 1972.

[45]Kohonen T. “Self-organized formation of topologically correct feature maps. Biological Cybernetics”, Helsinki university technology.Springer-verlag. 1982.

[46]Kohonen, T. “An introduction to neural computing”. Neural Networks, Vol.1. No. 1. PP.3-16. 1988.

[47]Kohonen T., “Self-Organizing Maps”, Ed. Springer-Verlag, third edition, Germany, 2001.

[48]Konar, A. Artificial Intelligence and soft Computing. Behavioral and cognitive modeling: behavioral and cognitive modeling of the human brain. CRC Press LLC 1999.

[49]Krös, P. y van der Smagt, P. An introduction to Neural Networks. The university of Amsterdam. 1996.

[50]Kumar, G.J., "Artificial Neural Network". Pusa, New Delhi-110. 2012.

[51]Le Cun Y. “A theoretical framework for backpropagation”. In Proc. 1988 Conectionist Models Summer Sch., Touresky D., Hinton G. and Sejnowski T., Eds. Morgan Kauffmann, PP. 21-28. 1988.

[52]Lingireddy, S.,Brion, G.M. “Artificial Neural Networks in water supply Engineering”.American Society of Civil Engineers. ISBN 0-7844-0765-7. PP. 11. 2005.

[53]Martín, B.B. y Sanz, M.A. “Redes Neuronales y Sistemas Borrosos”. Alfaomega. 3ra Edición. PP. 20-40. 2007.

[54]McBader, S., and Lee, P. “An FPGA implementation of a flexible, parallel image processing architecture suitable for embedded vision systems”. In: Proceedings of the International Parallel and Distributed Processing Symposium, PP. 228. 2003.

[55]McCulloch, W.S., Pitts, W. “A logical calculus of the ideas immanent in nervous activity”. Bulletin of Mathematical Biophysics, Vol. 5. PP.115-133, 1943.

[56]Mehrotra, K., Mohan C.K., Ranka S. “Elements of Artificial Neural Networks”. Massachusetts Institute of Technology. PP. 5-7. 2000.

[57]Minsky, M.L. “Theory of neural-analog reinforcement systems and its applicattion to the brain-model problema”. PhD thesis, Princenton University, Princenton, NJ, 1954.

[58]Minsky M. L. and Papert S. A.,“Perceptrons: An Introduction to Computational Geometry”. The MIT Press, Cambridge, MA. 1969.

[59]Moon, S.W., Kong, S.G., “Block-Based Neural Networks”. IEEE Transactions on Neural Networks. Vol. 12. PP. 307-317. 2001.

[60]Myers, D.J.,Hutchinson, R.A.,”Efficient implementation of piecewise linear activation function for digital VLSI neural network”. Martlesham Heath, Ipswich, Suffolk IP5 7RE, United Kingdom. 1989.

[61]Nakano K. “Associatron- a model of associative memory”. IEEE Transactions on Systems Man, and Cybernetics. Vol. SMC-12. PP.380-388.1972.

[62]Nedjah, N., Martins da Silva, R., Mourelle, L.”Compact yet efficient hardware implementation of artificial neural networks with customized topology”, Expert Systems with Applications, Vol. 39. PP. 9191-9206. 2012.

[63]Novikoff, A., " On convergence proof of perceptron",.Symp. on Mathematical Theory of Automata. 1963.

[64]Omondi, A. R., Rajapakse, J. C., and Bajger, M. "FPGA neurocomputers". In FPGA Implementations of Neural Networks, Springer, PP. 1–36. 2006.

[65]Oniga, S. "A new method for FPGA implementation artificial neural network used in smart devices". North University of Baia Mare. Romania.

[66]Ortigosa E.M., Cañas A., Ros E., Ortigosa P.M., Mota, S., Díaz J., "Hardware description of multi-layer perceptrons with different abstraction levels", Microprocessors and Microsystem, Vol. 30. PP. 435-444. 2006.

[67]Parker D. "Learning logic. Invention Report S81-64", File 1, Office of Technology Licensing, Stanford University, Stanford CA. 1982.

[68]Parker D. "Learning logic. Technical Report TR-47", Center for Computational Research in Economics and Management Science, MIT. 1985.

[69]Porrmann, M., Witkowski, U., and Ruckert, U. "A massively parallel architecture for self-organizing feature maps". IEEE Transactions on Neural Networks, Vol. 14. No. 5. PP. 1110-1121. 2003.

[70]Ranjan, T.J., Kumar, T.H, Nayak, M., "FPGA Implementation of a fully and partially connected MLP-A dedicated approach". IJCSI International Journal of Computer Science Issues, Vol. 10. 2013.

[71]Rosenblatt F., "The perceptron: a probablistic model for information storage and retrieval in the brain". Psych. Rev, Vol. 65. PP.386–408. 1958.

[72]Rumelhart D.E., Hinton G.E., R.J., "Learning internal representations by error propagation", in Parallel Distributed Processing: Explorations in the Microstructure of Cognition, vol l., Cambridge, MA: MIT Press, PP. 318-362. 1986.

[73]Sahin, S., Becerikli, Y., and Suleyman, Y. "Neural Network Implementation in Hardware Usign FPGAs". Springer-Verlag Berlin Heidelberg. PP. 1105–1112.2006.

[74]Swankoski, E.J., Brooks, R.R., Narayanan, V., Kandemir, M., and Irwin, M.J. "A parallel architecture for secure FPGA symmetric encryption". In: Proceedings of the International Parallel and Distributed Processing Symposium, PP. 132. 2004.

[75]Taylor, W. K. "Electrical simulation of some nervous system functional activities". Information Science, PP. 314-328. 1956.

[76]Uttley, A. M. "Conditional probability machines and conditional reflexes". Automata Studies. PP. 253-276,1956.

[77]Uttley, A. M. "The transmission of information and the effect of local feedback in theoretical and neural networks". Brain Research. 1966.

[78]Werbos P.J, "Beyond regression: New tools for prediction and analysis in the behavioral sciences", Ph.D dissertation, Committee on Appl. Math., Harvard Univ., Cambridge, MA, 1974.

[79]Werbos P.J. "Backpropagation Throught Time: What it does and how to do it". Proceedings of the IEE. Vol. 78. 1990.

[80]Widrow B. and Hoff M. E., "Adaptive switching circuits", IRE WESCON Convention Record, PP. 96-104. 1960.

[81]Wilson, D.R., Martinez, T. R., "The general inefficiency of batch training for gradient descent learning". Neural networks, 16, PP. 1429-1451. 2003.

[82]Yegnanarayana B. "Artificial Neural Network". Prentice Hall. PP. 21-24.2005.

[83]Zurada, M.J. Introduction to Artificial Neural system. West publishing company.1992.

Sitios de internet

[URL1] http://www.dlpdesign.com/usb/usb232.shtml. DLP Design. "Fabricante de hardware para comunicaciones vía USB". Última visita 20 de Enero de 2015.